AM3517 Linux User Guide

Software Documentation

Logic PD // Products
Published: July 2011
Last revised: October 2013

Revision History

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	ME, JA	Initial Release	JCA	07/01/11
B	JA	-Added Section 2.1, Section 2.2, Section 3.1, and Section 3.3; -Section 3.4.2: Updated procedures and commands throughout; -Added Section 3.5 and Section 4.1; -Section 4.2.2: Updated procedures and commands throughout; -Section 4.3.1: Changed locations of SPL, U-Boot, and U-Boot environment; added note regarding U-Boot prompt to be used throughout document for brevity; -Section 4.3.3: Updated procedures and commands throughout; -Section 4.4.2: Updated procedures and commands throughout; -Section 4.5.2: Changed definition of *nfsargs* variable in Step 2; changed value of bytes transferred in output from Step 3; -Section 4.6.2.1: Added new location of default U-Boot environment; -Section 4.6.3: Added new location of default U-Boot environment; -Section 4.6.3.2: Designated directory in which *boot.txt* file resides; -Section 4.6.4: Included reminder that procedure will erase NAND and reprogram it; added note that the procedure will erase the U-Boot environment variables created previously; -Section 4.6.4.1: Changed location of JFFS2 root file system image; -Section 4.6.4.2: Updated procedures and commands throughout; -Section 4.7.2: Updated procedures and commands throughout; changed location of source code; -Added Section 4.8 and Section 4.9; -Section 5.1.2: Updated procedures and commands throughout; -Section 5.2: Removed previous content; added note that section will be updated in next release; -Section 6: Updated procedures and commands throughout; -Section 6.2: Added note explaining that WEP security is not supported; -Section 7: Updated procedures and commands throughout; -Appendix A: Updated commands and procedures throughout; -Added Appendix B	SWE, SO	02/05/13
C	JA, SWE	-Added Section 1.1 to explain that this document applies to SOMs with 4-bit ECC NAND ; -Added Section 3.2 to explain how to apply the 4-bit ECC patch; -Section 4.6.3: Added note that instructions are for SOMs with 4-bit ECC NAND; -Section 4.6.3.2: Updated commands for 4-bit ECC in Step 2; -Section 4.7: Added note that instructions are for SOMs with 4-bit ECC NAND; -Section 4.7.2: Updated commands for 4-bit ECC in Step 9; -Section 4.8: Added note that instructions are for SOMs with 4-bit ECC NAND; -Section 4.8.2: Updated commands for 4-bit ECC in Step 6	SO	10/15/13

Table of Contents

1 Introduction

The purpose of this user guide is to assist with software development using the Zoom™ AM3517 Development Kit and a Linux operating system (OS).

1.1 Applicable AM3517 SOM-M2s

As detailed in PCN 452 and 479 of the *AM3517 SOM-M2 Product Change Notification*,[1] the AM3517 SOM-M2 was updated from 1-bit ECC NAND to 4-bit ECC NAND due to manufacturer part obsolescence. To determine the type of NAND on your AM3517 SOM-M2, identify the part number on the identification sticker affixed to the top of the SOM. (See the Logic PD Part Number and Serial Number Sticker web page[2] for help locating the part number.) Compare the part number to the part numbers identified in PCN 452 and 479 to determine which type of NAND is on your SOM

The instructions in this user guide are specific to SOMs with 4-bit ECC NAND; as a result, customers using SOMS with 1-bit ECC NAND may have difficulty with the procedures herein. Please contact Logic PD[3] for Linux support with SOMs that contain 1-bit ECC NAND.

2 Preparing for Development

There are two paths for creating a development environment. You can either download the Texas Instruments (TI) AM3517 Linux SDK (hereafter, SDK) into an existing Linux environment, as described in Section 2.1, or you can use the *Virtual Machine for the AM3517 Linux SDK* provided by Logic PD that has the development environment already installed, as described in Section 2.2. It is only necessary to complete one of these sections to begin the development process.

It is important to note that the examples provided in later sections of this document use the Virtual Machine (VM) environment. If you decide to build the development environment on your own, you may need to adjust the commands accordingly in the examples and the output may look slightly different.

2.1 Install SDK on Host PC for Development

The SDK is a self-contained installer and it handles the majority of the installation process. The SDK is first installed on the host PC and then a configuration script will be run to complete the process.

2.1.1 Prerequisites

Root or sudo access on a Ubuntu workstation is necessary6. Ubuntu version 10.04 is recommended.

2.1.2 Procedure

1. Switch to the *Downloads* directory.

```
logic@logic-desktop-am3517:~$ cd Downloads
logic@logic-desktop-am3517:~/Downloads$
```

[1] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=940
[2] http://www.logicpd.com/part-number-and-serial-number-sticker/
[3] http://support.logicpd.com/TechnicalSupport/AskAQuestion.aspx

2. Download the TI SDK installer using the *wget* command.

```
logic@logic-desktop-am3517:~/Downloads/temp$ wget http://software-
dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sdk/AM3517/latest/ex
ports/ti-sdk-am3517-evm-05.05.00.00-Linux-x86-Install
...
Saving to: `ti-sdk-am3517-evm-05.05.00.00-Linux-x86-Install'

100%[======================================================>]
1,845,743,768 1.99M/s   in 21m 21s 1

2012-08-13 11:34:49 (1.37 MB/s) - `ti-sdk-am3517-evm-05.05.00.00-Linux-
x86-Install' saved [1845743768/1845743768]

logic@logic-desktop-am3517:~/Downloads$
```

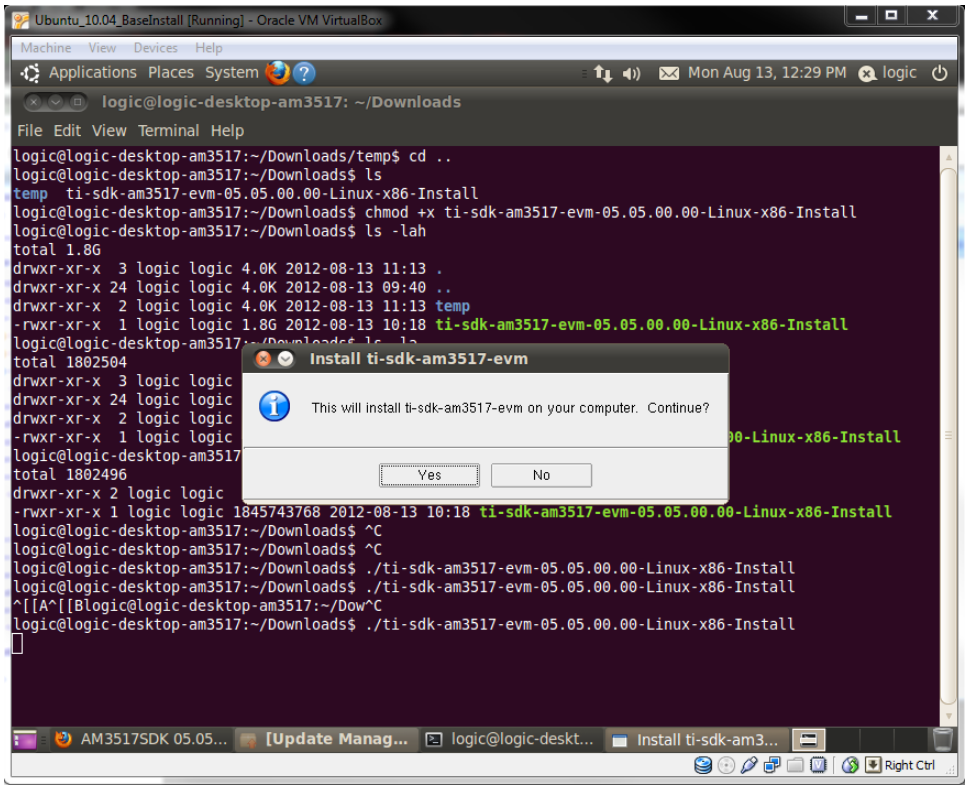
3. When the download completes, you will need to mark the installer as executable so you can run it.

```
logic@logic-desktop-am3517:~/Downloads$ chmod +x ti-sdk-am3517-evm-
05.05.00.00-Linux-x86-Install
logic@logic-desktop-am3517:~/Downloads$ ls -l
total 1802496
-rwxr-xr-x 1 logic logic 1845743768 2012-08-13 10:18 ti-sdk-am3517-evm-
05.05.00.00-Linux-x86-Install
```

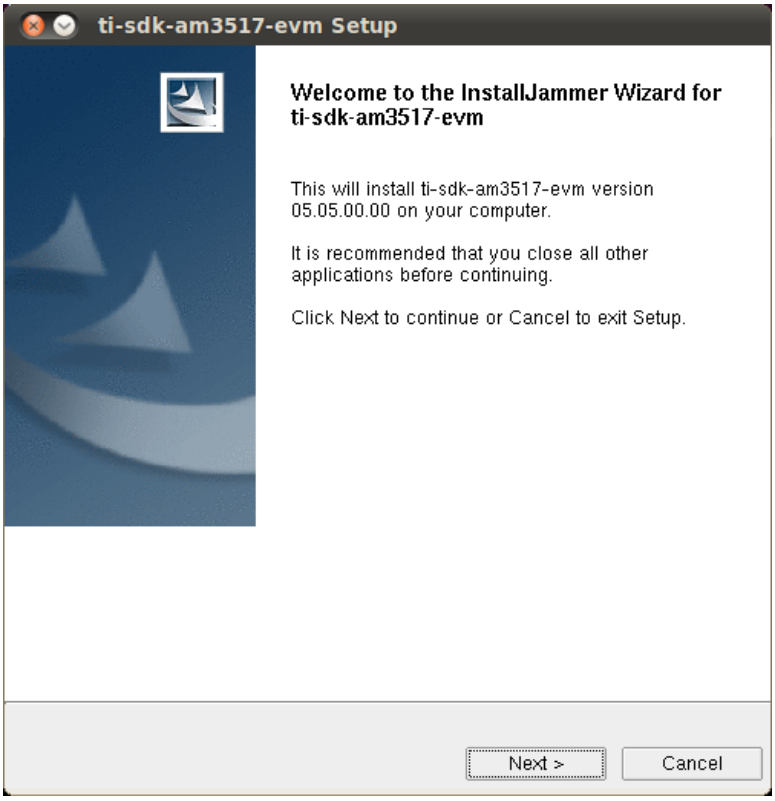
4. Run the TI SDK installer.

```
logic@logic-desktop-am3517:~/Downloads$ ./ti-sdk-am3517-evm-
05.05.00.00-Linux-x86-Install
```

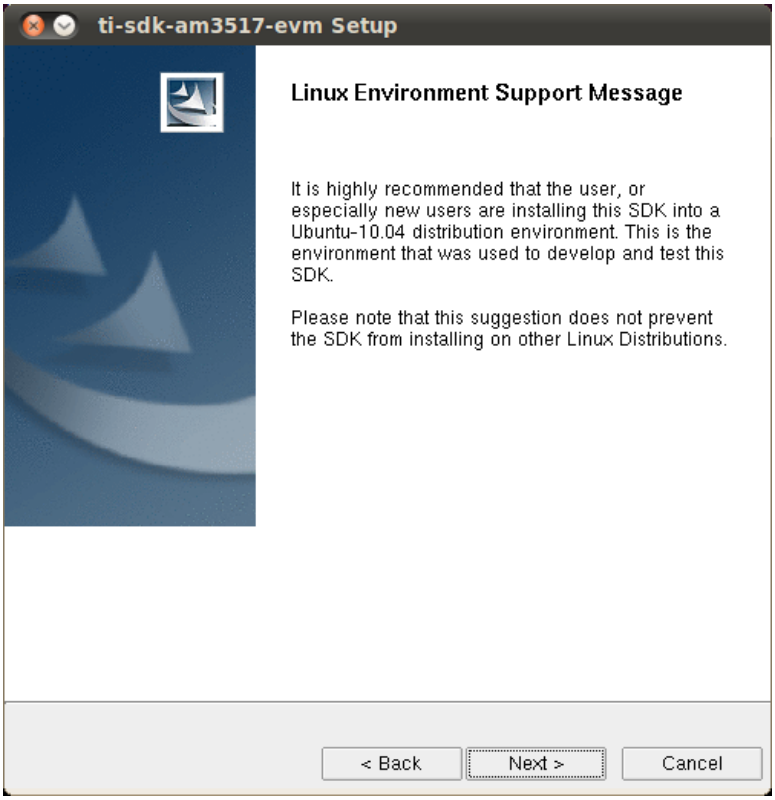

5. When a window appears to ask if you would like to continue the installation process, click Yes.



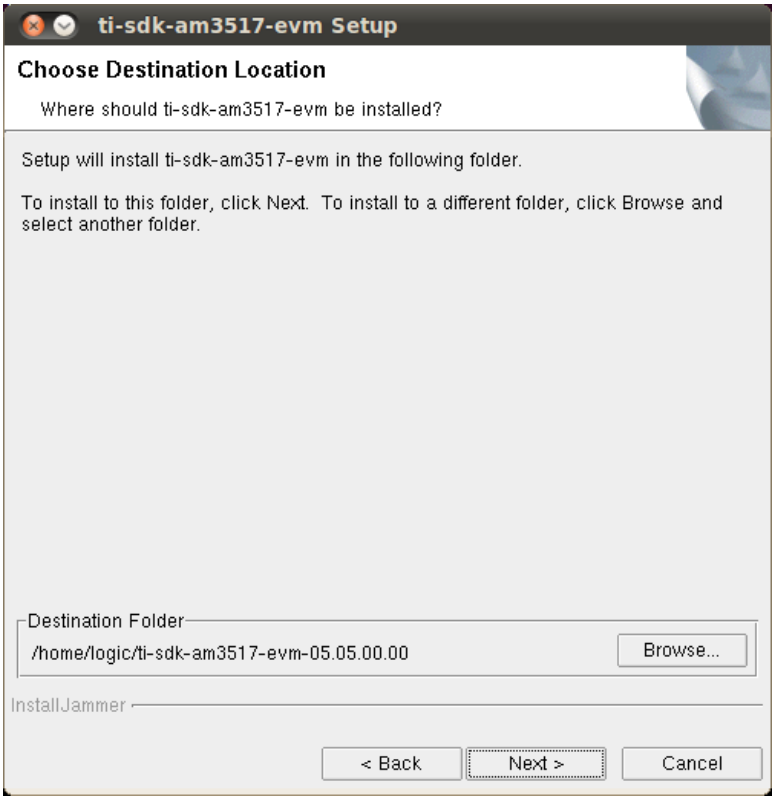
6. When the installation wizard welcome screen appears, click Next.



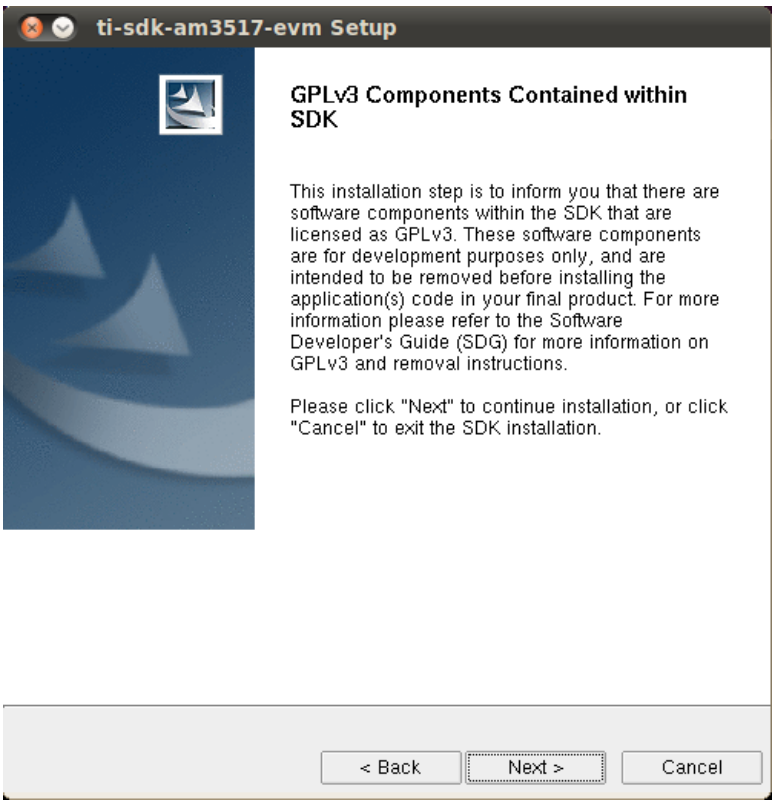
7. Ensure you are installing the SDK in Ubuntu 10.04 and click Next.



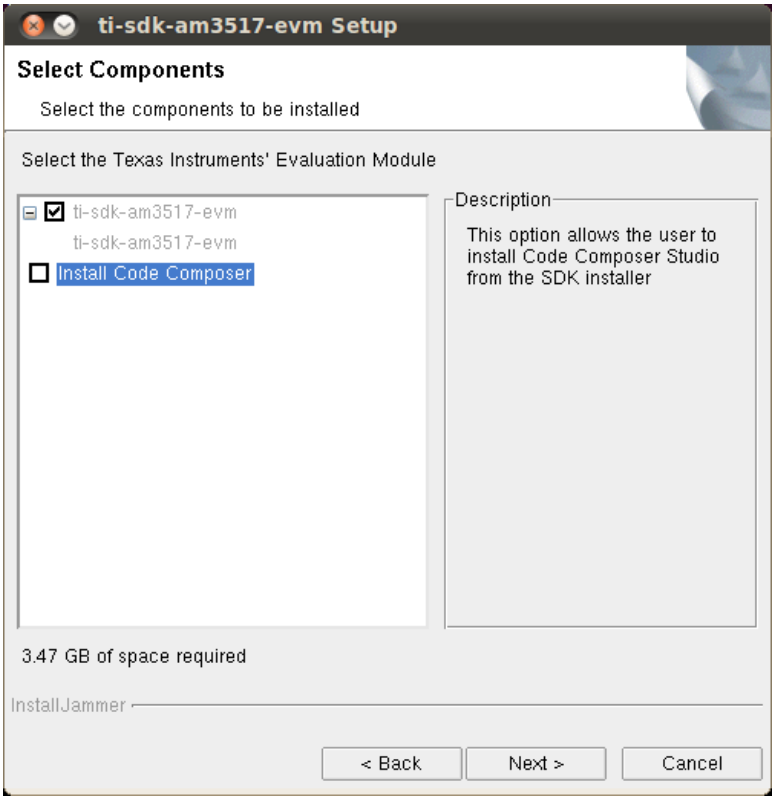
8. Click Next to select the default installation location as your home directory.



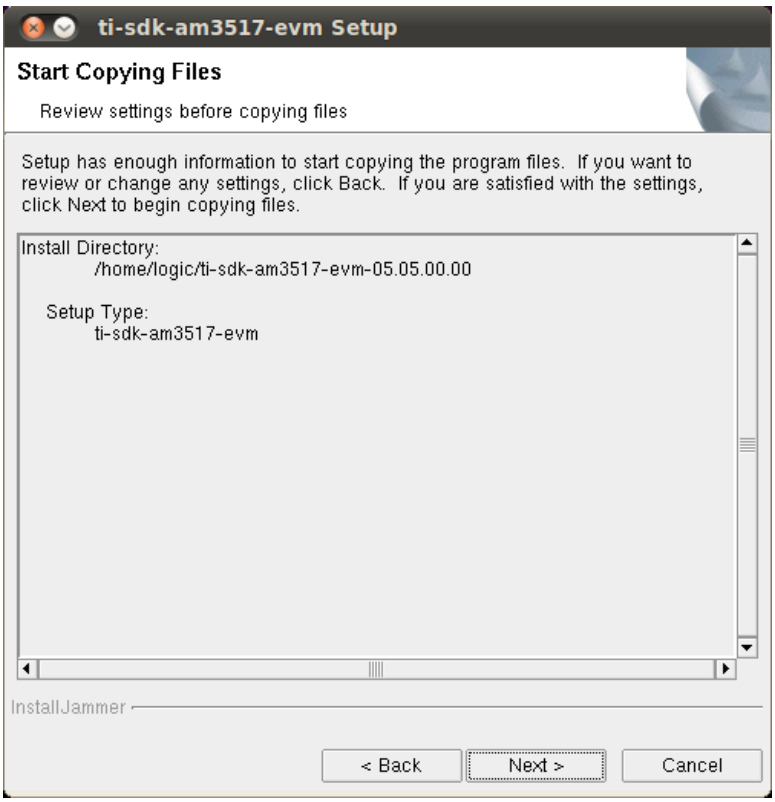
9. Read the GPLv3 message and click Next when finished.



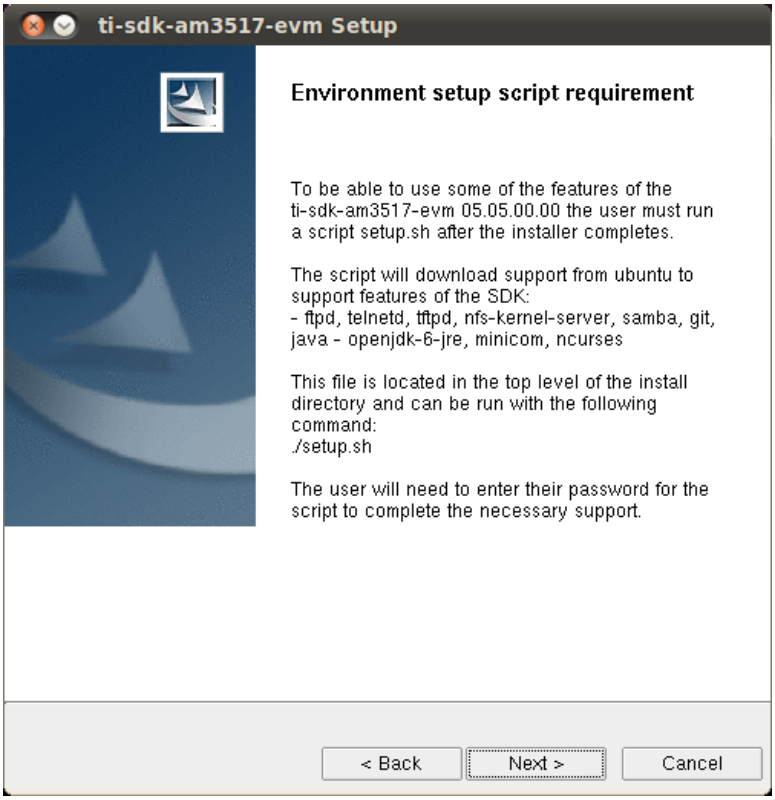
10. Deselect the checkbox labeled "Install Code Composer" and click Next.



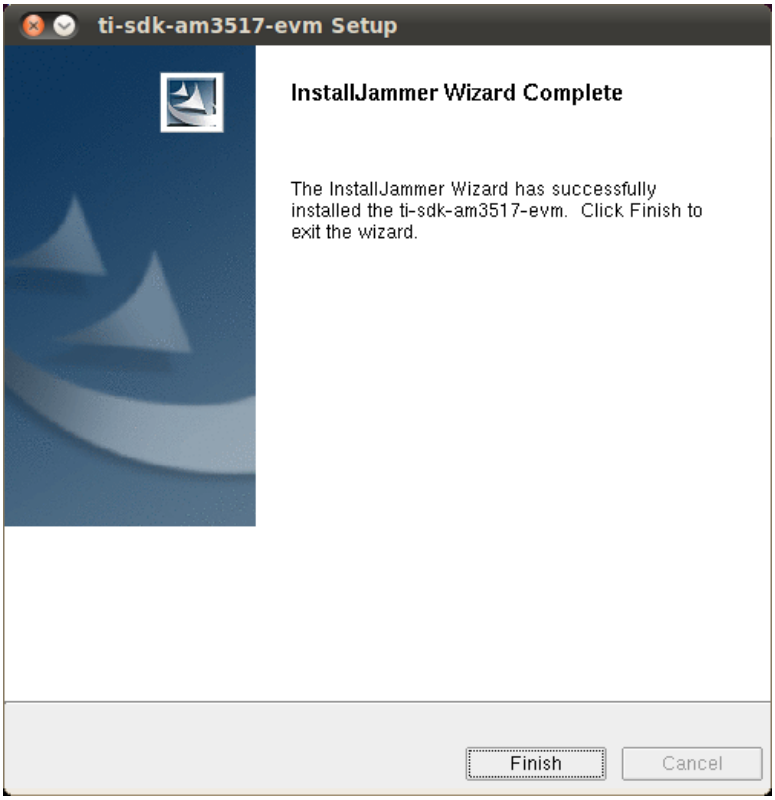
11. Click Next to begin copying the files to your PC.



12. Click Next to make the required changes to your environmental scripts.



13. Click Finish to close the installer.



14. Switch back to your home directory and you should see the installed SDK.

```
logic@logic-desktop-am3517:~/Downloads$ cd
logic@logic-desktop-am3517:~$ ls
Desktop    Downloads         Music     Public     ti-sdk-am3517-evm-
05.05.00.00
Documents  examples.desktop  Pictures  Templates  Videos
logic@logic-desktop-am3517:~$
```

15. Switch to the *SDK* directory.

```
logic@logic-desktop-am3517:~$ cd ti-sdk-am3517-evm-05.05.00.00/
```

16. Run the one-time setup script.

```
logic@logic-desktop-am3517:~/ti-sdk-am3517-evm-05.05.00.00$ ./setup.sh

------------------------------------------------------------------------
TISDK setup script

This script will set up your development host for dvsdk development.
Parts of this script require administrator priviliges (sudo access).
------------------------------------------------------------------------
...
```


The remaining output of the process is shown below. The areas in bold point out instances in which the user must provide input to continue.

a. There are many choices to be made here. The setup script will first ask if you have sudo access to your host PC. If you're using the standard Ubuntu 10.04 install, you do have access. The install process will ask you for your password and you should provide it. **NOTE:** Depending on how fast your machine is, you may be asked for your password multiple times.

```
...
------------------------------------------------------------------------
Verifying Linux host distribution
Ubuntu 10.04 LTS found successfully, continuing..
------------------------------------------------------------------------
------------------------------------------------------------------------
This step will make sure you have the proper host support packages
installed
using the following command: sudo apt-get install xinetd tftpd nfs-
kernel-server minicom build-essential libncurses5-dev uboot-mkimage
autoconf automake

Note! This command requires you to have administrator priviliges (sudo
access)
on your host.
Press return to continue

[sudo] password for logic:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  autotools-dev cpp-4.4 dpkg-dev fakeroot g++ g++-4.4 gcc-4.4 gcc-4.4-
base libgcc1 libgomp1 libgssglue1 libnfsidmap2
  librpcsecgss3 libstdc++6 libstdc++6-4.4-dev lrzsz m4 nfs-common patch
portmap xz-utils
Suggested packages:
  autoconf2.13 autoconf-archive gnu-standards autoconf-doc libtool
gettext gcc-4.4-locales debian-keyring
  debian-maintainers g++-multilib g++-4.4-multilib gcc-4.4-doc
libstdc++6-4.4-dbg gcc-4.4-multilib libmudflap0-4.4-dev
  libgcc1-dbg libgomp1-dbg libmudflap0-dbg libcloog-ppl0 libppl-c2
libppl7 libstdc++6-4.4-doc diffutils-doc
The following NEW packages will be installed:
  autoconf automake autotools-dev build-essential dpkg-dev fakeroot g++
g++-4.4 libgssglue1 libncurses5-dev libnfsidmap2
  librpcsecgss3 libstdc++6-4.4-dev lrzsz m4 minicom nfs-common nfs-
kernel-server patch portmap tftpd uboot-mkimage xinetd
  xz-utils
The following packages will be upgraded:
  cpp-4.4 gcc-4.4 gcc-4.4-base libgcc1 libgomp1 libstdc++6
6 upgraded, 24 newly installed, 0 to remove and 172 not upgraded.
Need to get 19.2MB of archives.
After this operation, 39.7MB of additional disk space will be used.
...
```


b. There may be several packages that need to be installed and the additional disk space required will be noted, as shown in the output above. Press the **Y** key and then press **Enter** to continue.

```
...
Do you want to continue [Y/n]? Y
Get:1 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main gcc-4.4-
base 4.4.3-4ubuntu5.1 [118kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main
libstdc++6 4.4.3-4ubuntu5.1 [348kB]
Get:3 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main libgomp1
4.4.3-4ubuntu5.1 [24.3kB]
Get:4 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main cpp-4.4
4.4.3-4ubuntu5.1 [3,759kB]
Get:5 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main gcc-4.4
4.4.3-4ubuntu5.1 [2,986kB]
Get:6 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main libgcc1
1:4.4.3-4ubuntu5.1 [55.3kB]
Get:7 http://us.archive.ubuntu.com/ubuntu/ lucid/main m4 1.4.13-3
[241kB]
Get:8 http://us.archive.ubuntu.com/ubuntu/ lucid/main autoconf 2.65-
3ubuntu1 [772kB]
Get:9 http://us.archive.ubuntu.com/ubuntu/ lucid/main autotools-dev
20090611.1 [64.1kB]
Get:10 http://us.archive.ubuntu.com/ubuntu/ lucid/main automake
1:1.11.1-1 [608kB]
Get:11 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main
libstdc++6-4.4-dev 4.4.3-4ubuntu5.1 [1,491kB]
Get:12 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main g++-4.4
4.4.3-4ubuntu5.1 [4,950kB]
Get:13 http://us.archive.ubuntu.com/ubuntu/ lucid/main g++ 4:4.4.3-
1ubuntu1 [1,442B]
Get:14 http://us.archive.ubuntu.com/ubuntu/ lucid/main xz-utils
4.999.9beta+20091116-1 [228kB]
Get:15 http://us.archive.ubuntu.com/ubuntu/ lucid/main patch 2.6-
2ubuntu1 [123kB]
Get:16 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main dpkg-dev
1.15.5.6ubuntu4.6 [653kB]
Get:17 http://us.archive.ubuntu.com/ubuntu/ lucid/main build-essential
11.4build1 [7,278B]
Get:18 http://us.archive.ubuntu.com/ubuntu/ lucid/main fakeroot 1.14.4-
1ubuntu1 [118kB]
Get:19 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main
libgssglue1 0.1-4ubuntu0.1 [22.7kB]
Get:20 http://us.archive.ubuntu.com/ubuntu/ lucid/main libncurses5-dev
5.7+20090803-2ubuntu3 [1,564kB]
Get:21 http://us.archive.ubuntu.com/ubuntu/ lucid/main libnfsidmap2
0.23-2 [29.1kB]
Get:22 http://us.archive.ubuntu.com/ubuntu/ lucid/main librpcsecgss3
0.19-2 [33.1kB]
Get:23 http://us.archive.ubuntu.com/ubuntu/ lucid/universe lrzsz
0.12.21-5 [99.9kB]
Get:24 http://us.archive.ubuntu.com/ubuntu/ lucid/universe minicom 2.4-
1 [312kB]
Get:25 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main portmap
6.0.0-1ubuntu2.2 [38.1kB]
```



```
Get:26 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main nfs-
common 1:1.2.0-4ubuntu4.2 [213kB]
Get:27 http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main nfs-
kernel-server 1:1.2.0-4ubuntu4.2 [159kB]
Get:28 http://us.archive.ubuntu.com/ubuntu/ lucid/main xinetd 1:2.3.14-
7ubuntu3 [146kB]
Get:29 http://us.archive.ubuntu.com/ubuntu/ lucid/universe tftpd 0.17-
17ubuntu1 [16.5kB]
Get:30 http://us.archive.ubuntu.com/ubuntu/ lucid/main uboot-mkimage
0.4build1 [9,876B]
Fetched 19.2MB in 12s (1,521kB/s)
Preconfiguring packages ...
(Reading database ... 120784 files and directories currently
installed.)
Preparing to replace gcc-4.4-base 4.4.3-4ubuntu5 (using .../gcc-4.4-
base_4.4.3-4ubuntu5.1_i386.deb) ...
Unpacking replacement gcc-4.4-base ...
Setting up gcc-4.4-base (4.4.3-4ubuntu5.1) ...
(Reading database ... 120784 files and directories currently
installed.)
Preparing to replace libstdc++6 4.4.3-4ubuntu5 (using
.../libstdc++6_4.4.3-4ubuntu5.1_i386.deb) ...
Unpacking replacement libstdc++6 ...
Setting up libstdc++6 (4.4.3-4ubuntu5.1) ...

Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
(Reading database ... 120784 files and directories currently
installed.)
Preparing to replace libgomp1 4.4.3-4ubuntu5 (using .../libgomp1_4.4.3-
4ubuntu5.1_i386.deb) ...
Unpacking replacement libgomp1 ...
Preparing to replace cpp-4.4 4.4.3-4ubuntu5 (using .../cpp-4.4_4.4.3-
4ubuntu5.1_i386.deb) ...
Unpacking replacement cpp-4.4 ...
Preparing to replace gcc-4.4 4.4.3-4ubuntu5 (using .../gcc-4.4_4.4.3-
4ubuntu5.1_i386.deb) ...
Unpacking replacement gcc-4.4 ...
Preparing to replace libgcc1 1:4.4.3-4ubuntu5 (using
.../libgcc1_1%3a4.4.3-4ubuntu5.1_i386.deb) ...
Unpacking replacement libgcc1 ...
Processing triggers for man-db ...
Setting up libgcc1 (1:4.4.3-4ubuntu5.1) ...

Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
Selecting previously deselected package m4.
(Reading database ... 120784 files and directories currently
installed.)
Unpacking m4 (from .../archives/m4_1.4.13-3_i386.deb) ...
Selecting previously deselected package autoconf.
Unpacking autoconf (from .../autoconf_2.65-3ubuntu1_all.deb) ...
Selecting previously deselected package autotools-dev.
Unpacking autotools-dev (from .../autotools-dev_20090611.1_all.deb) ...
Selecting previously deselected package automake.
Unpacking automake (from .../automake_1%3a1.11.1-1_all.deb) ...
Selecting previously deselected package libstdc++6-4.4-dev.
```



```
Unpacking libstdc++6-4.4-dev (from .../libstdc++6-4.4-dev_4.4.3-
4ubuntu5.1_i386.deb) ...
Selecting previously deselected package g++-4.4.
Unpacking g++-4.4 (from .../g++-4.4_4.4.3-4ubuntu5.1_i386.deb) ...
Selecting previously deselected package g++.
Unpacking g++ (from .../g++_4%3a4.4.3-1ubuntu1_i386.deb) ...
Selecting previously deselected package xz-utils.
Unpacking xz-utils (from .../xz-utils_4.999.9beta+20091116-1_i386.deb)
...
Selecting previously deselected package patch.
Unpacking patch (from .../patch_2.6-2ubuntu1_i386.deb) ...
Selecting previously deselected package dpkg-dev.
Unpacking dpkg-dev (from .../dpkg-dev_1.15.5.6ubuntu4.6_all.deb) ...
Selecting previously deselected package build-essential.
Unpacking build-essential (from .../build-
essential_11.4build1_i386.deb) ...
Selecting previously deselected package fakeroot.
Unpacking fakeroot (from .../fakeroot_1.14.4-1ubuntu1_i386.deb) ...
Selecting previously deselected package libgssglue1.
Unpacking libgssglue1 (from .../libgssglue1_0.1-4ubuntu0.1_i386.deb)
...
Selecting previously deselected package libncurses5-dev.
Unpacking libncurses5-dev (from .../libncurses5-dev_5.7+20090803-
2ubuntu3_i386.deb) ...
Selecting previously deselected package libnfsidmap2.
Unpacking libnfsidmap2 (from .../libnfsidmap2_0.23-2_i386.deb) ...
Selecting previously deselected package librpcsecgss3.
Unpacking librpcsecgss3 (from .../librpcsecgss3_0.19-2_i386.deb) ...
Selecting previously deselected package lrzsz.
Unpacking lrzsz (from .../lrzsz_0.12.21-5_i386.deb) ...
Selecting previously deselected package minicom.
Unpacking minicom (from .../minicom_2.4-1_i386.deb) ...
Selecting previously deselected package portmap.
Unpacking portmap (from .../portmap_6.0.0-1ubuntu2.2_i386.deb) ...
Selecting previously deselected package nfs-common.
Unpacking nfs-common (from .../nfs-common_1%3a1.2.0-
4ubuntu4.2_i386.deb) ...
Selecting previously deselected package nfs-kernel-server.
Unpacking nfs-kernel-server (from .../nfs-kernel-server_1%3a1.2.0-
4ubuntu4.2_i386.deb) ...
Selecting previously deselected package xinetd.
Unpacking xinetd (from .../xinetd_1%3a2.3.14-7ubuntu3_i386.deb) ...
Selecting previously deselected package tftpd.
Unpacking tftpd (from .../tftpd_0.17-17ubuntu1_i386.deb) ...
Selecting previously deselected package uboot-mkimage.
Unpacking uboot-mkimage (from .../uboot-mkimage_0.4build1_i386.deb) ...
Processing triggers for install-info ...
Processing triggers for man-db ...
Processing triggers for doc-base ...
Processing 28 changed 1 added doc-base file(s)...
Registering documents with scrollkeeper...
Processing triggers for ureadahead ...
ureadahead will be reprofiled on next reboot
Setting up libgomp1 (4.4.3-4ubuntu5.1) ...

Setting up cpp-4.4 (4.4.3-4ubuntu5.1) ...
Setting up gcc-4.4 (4.4.3-4ubuntu5.1) ...
```



```
Setting up m4 (1.4.13-3) ...
Setting up autoconf (2.65-3ubuntu1) ...
Setting up autotools-dev (20090611.1) ...
Setting up automake (1:1.11.1-1) ...
update-alternatives: using /usr/bin/automake-1.11 to provide
/usr/bin/automake (automake) in auto mode.

Setting up xz-utils (4.999.9beta+20091116-1) ...
Setting up patch (2.6-2ubuntu1) ...
Setting up dpkg-dev (1.15.5.6ubuntu4.6) ...
Setting up fakeroot (1.14.4-1ubuntu1) ...
update-alternatives: using /usr/bin/fakeroot-sysv to provide
/usr/bin/fakeroot (fakeroot) in auto mode.

Setting up libgssglue1 (0.1-4ubuntu0.1) ...

Setting up libncurses5-dev (5.7+20090803-2ubuntu3) ...
Setting up libnfsidmap2 (0.23-2) ...

Setting up librpcsecgss3 (0.19-2) ...

Setting up lrzsz (0.12.21-5) ...
Setting up minicom (2.4-1) ...

Setting up portmap (6.0.0-1ubuntu2.2) ...
portmap start/running, process 4754

Setting up nfs-common (1:1.2.0-4ubuntu4.2) ...

Creating config file /etc/idmapd.conf with new version

Creating config file /etc/default/nfs-common with new version
Adding system user `statd' (UID 115) ...
Adding new user `statd' (UID 115) with group `nogroup' ...
Not creating home directory `/var/lib/nfs'.
statd start/running, process 4979
gssd stop/pre-start, process 5002
idmapd stop/pre-start, process 5028

Setting up nfs-kernel-server (1:1.2.0-4ubuntu4.2) ...

Creating config file /etc/exports with new version

Creating config file /etc/default/nfs-kernel-server with new version
 * Exporting directories for NFS kernel daemon...
[ OK ]
 * Starting NFS kernel daemon
[ OK ]

Setting up xinetd (1:2.3.14-7ubuntu3) ...
 * Stopping internet superserver xinetd
[ OK ]
 * Starting internet superserver xinetd
[ OK ]

Setting up tftpd (0.17-17ubuntu1) ...
Note: xinetd currently is not fully supported by update-inetd.
```



```
      Please consult /usr/share/doc/xinetd/README.Debian and itox(8).

Setting up uboot-mkimage (0.4build1) ...
Setting up libstdc++6-4.4-dev (4.4.3-4ubuntu5.1) ...
Setting up g++-4.4 (4.4.3-4ubuntu5.1) ...
Setting up g++ (4:4.4.3-1ubuntu1) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++)
in auto mode.

Setting up build-essential (11.4build1) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place

Package verification and installation successfully completed
-----------------------------------------------------------------------


-----------------------------------------------------------------------
...
```

c. Next, you will be asked where you want to install the target file system. Press **Enter** to use the default value and continue.

```
...
In which directory do you want to install the target filesystem?(if
this directory does not exist it will be created)
[ /home/logic/ti-sdk-am3517-evm-05.05.00.00/targetNFS ]
-----------------------------------------------------------------------


-----------------------------------------------------------------------
...
```

d. The file system will now be extracted. You will be reminded about the requirement of having administrator privileges. Press **Enter** to continue.

```
...
This step will extract the target filesystem to /home/logic/ti-sdk-
am3517-evm-05.05.00.00/targetNFS

Note! This command requires you to have administrator priviliges (sudo
access)
on your host.
Press return to continue

Successfully extracted tisdk-rootfs-am3517-evm.tar.gz to
/home/logic/ti-sdk-am3517-evm-05.05.00.00/targetNFS
-----------------------------------------------------------------------


-----------------------------------------------------------------------
...
```


e. The binaries will now be installed in the target file system. Press **Enter** to use the default value and continue.

```
...
This step will set up the SDK to install binaries in to:
    /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/home/root/am3517-evm

The files will be available from /home/root/am3517-evm on the target.

This setting can be changed later by editing Rules.make and changing
the
EXEC_DIR or DESTDIR variable (depending on your SDK).

Press return to continue

Rules.make edited successfully..
------------------------------------------------------------------------

------------------------------------------------------------------------
...
```

f. The target file system directory will now be exported. You will be reminded about the requirement of having administrator privileges. Press **Enter** to continue.

```
...
This step will export your target filesystem for NFS access.

Note! This command requires you to have administrator priviliges (sudo
access)
on your host.
Press return to continue

 * Stopping NFS kernel daemon
[ OK ]
 * Unexporting directories for NFS kernel daemon...
[ OK ]
 * Exporting directories for NFS kernel daemon...
[ OK ]
 * Starting NFS kernel daemon
[ OK ]
------------------------------------------------------------------------

------------------------------------------------------------------------
...
```

g. The TFTP directory will be set up next. Press **Enter** to use the default value and continue. You will be reminded about the requirement of having administrator privileges. Press **Enter** to continue and re-enter your password if necessary.

```
...
Which directory do you want to be your tftp root directory?(if this
directory does not exist it will be created for you)
[ /tftpboot ]
------------------------------------------------------------------------
```



```
------------------------------------------------------------------------
This step will set up the tftp server in the /tftpboot directory.

Note! This command requires you to have administrator priviliges (sudo
access)
on your host.
Press return to continue

Successfully copied uImage-am3517-evm.bin to tftp root directory
/tftpboot

[sudo] password for logic:

/etc/xinetd.d/tftp successfully created

Restarting tftp server
 * Stopping internet superserver xinetd
[ OK ]
 * Starting internet superserver xinetd
[ OK ]
------------------------------------------------------------------------

------------------------------------------------------------------------
...
```

h. The script will attempt to determine which serial port you will use to connect to your kit. This will be discussed in more depth later in this document, so for now, press **Enter** to use the default value and continue.

```
...
This step will set up minicom (serial communication application) for
SDK development


For boards that contain a USB-to-Serial converter on the board
(BeagleBone),
the port used for minicom will be automatically detected. By default
Ubuntu
will not recognize this device. Setup will add will add a udev rule to
/etc/udev/ so that from now on it will be recognized as soon as the
board is
plugged in.

For other boards, the serial will defualt to /dev/ttyS0. Please update
based
on your setup.
------------------------------------------------------------------------


NOTE: For boards with a built-in USB to Serial adapter please press
      ENTER at the prompt below.  The correct port will be determined
      automatically at a later step.  For all other boards select
      the serial port that the board is connected to

Which serial port do you want to use with minicom?
[ /dev/ttyS0 ]
```



```
Configuration saved to /home/logic/.minirc.dfl. You can change it
further from inside
minicom, see the Software Development Guide for more information.
------------------------------------------------------------------------


------------------------------------------------------------------------
...
```

i. Press **Enter** to use the detected IP address and continue.

```
...
This step will set up the u-boot variables for booting the EVM.
Autodetected the following ip address of your host, correct it if
necessary
[ 10.0.2.15 ]
...
```

j. Press **Enter** to use the default Linux kernel location and continue.

```
...
Select Linux kernel location:
 1: TFTP
 2: SD card

[ 1 ]
...
```

k. Press **Enter** to use the default root file system location and continue.

```
...
Select root file system location:
 1: NFS
 2: SD card

[ 1 ]
...
```

l. Press **Enter** to use the default kernel image and continue.

```
...
Available kernel images in /tftproot:
    uImage-am3517-evm.bin

Which kernel image do you want to boot from TFTP?
[ uImage-am3517-evm.bin ]

Resulting u-boot variable settings:

setenv bootdelay 3
setenv baudrate 115200
setenv bootargs console=ttyO2,115200n8 rw noinitrd root=/dev/nfs
nfsroot=10.0.2.15:/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS,nolock,rsize=1024,wsize=1024 ip=dhcp
```



```
setenv bootcmd 'dhcp;setenv serverip 10.0.2.15;tftpboot;bootm'
setenv autoload no
setenv serverip 10.0.2.15
setenv bootfile uImage-am3517-evm.bin
------------------------------------------------------------------------


------------------------------------------------------------------------
...
```

m. When asked if you would like to create a Minicom script with the parameters, press the **N** key and then press **Enter** to continue.

```
...
Would you like to create a minicom script with the above parameters
(y/n)?
[ y ] n
------------------------------------------------------------------------
TISDK setup completed!
Please continue reading the Software Developer's Guide for more
information on
how to develop software on the EVM
logic@logic-desktop-am3517:~/ti-sdk-am3517-evm-05.05.00.00$
```

The installation process is now complete.

2.2 Use VM to Start Development

This section will explain how to use the Logic PD VM for the AM3517 Linux SDK to begin development. **NOTE:** If you followed the steps in Section 2.1 to create the development environment manually, you should skip this section.

2.2.1 Prerequisites

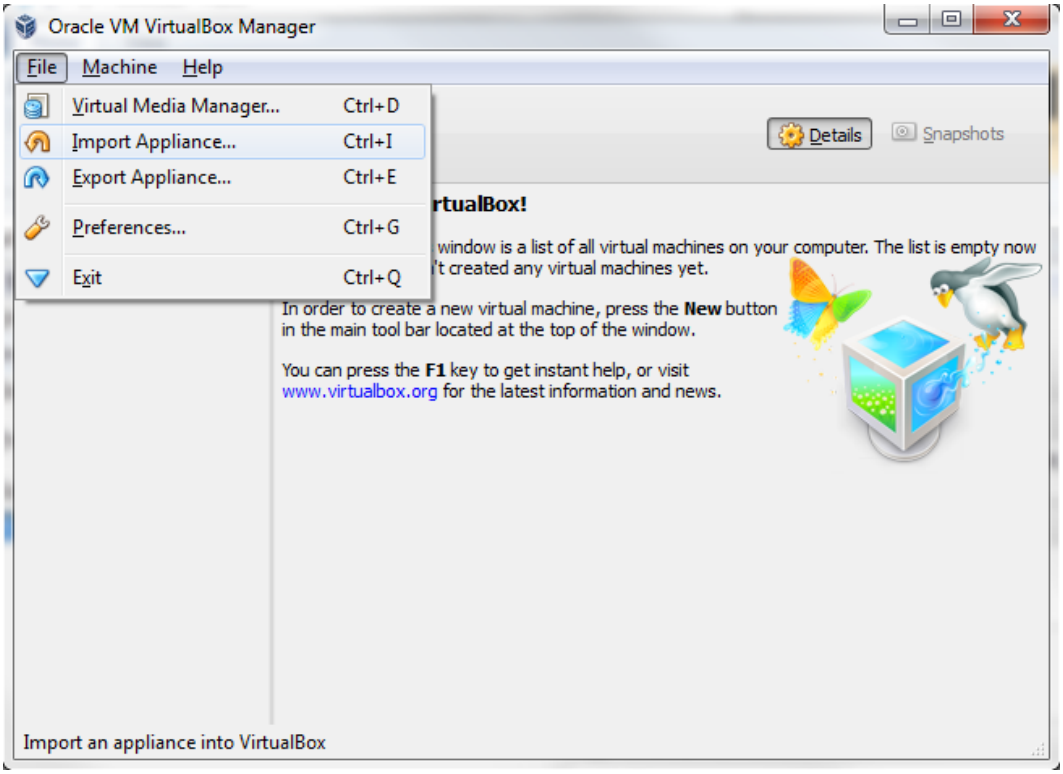
This example will use a Windows 7 PC running Virtual Box[4] version 4.1.0.

2.2.2 Procedure

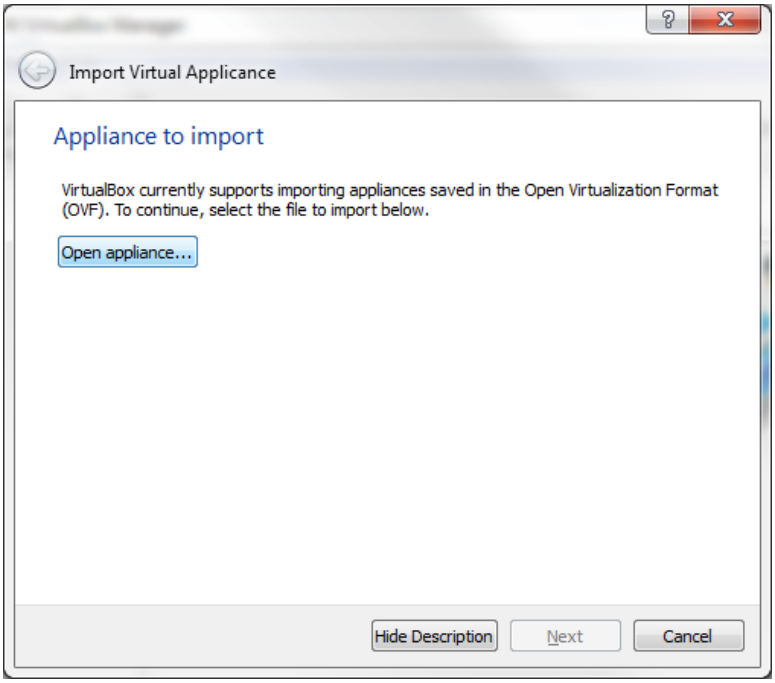
1. Download the Virtual Machine for the AM3517 Linux SDK[5] .ova file from the Logic PD support site.

[4] https://www.virtualbox.org/
[5] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1055

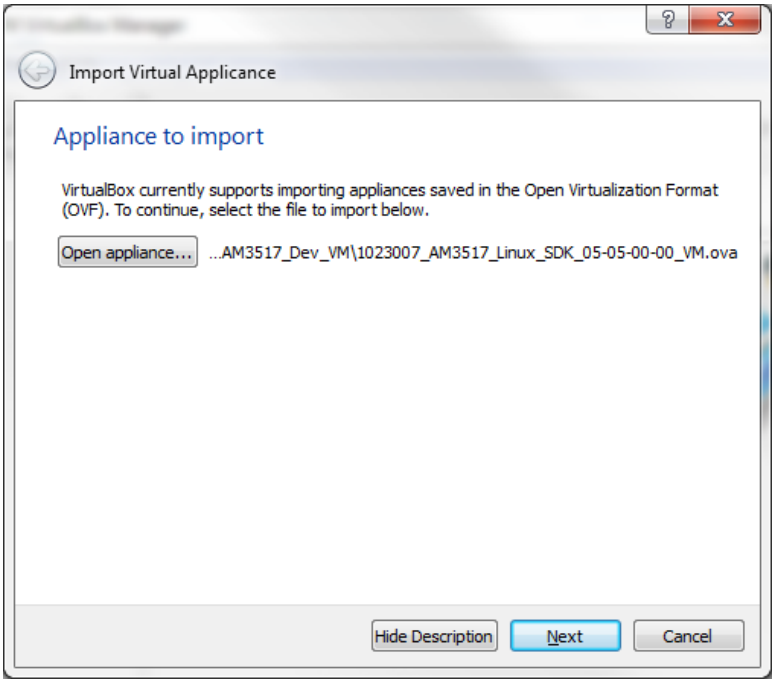
2. Open the Oracle VM VirtualBox Manager and select File > Import Appliance….



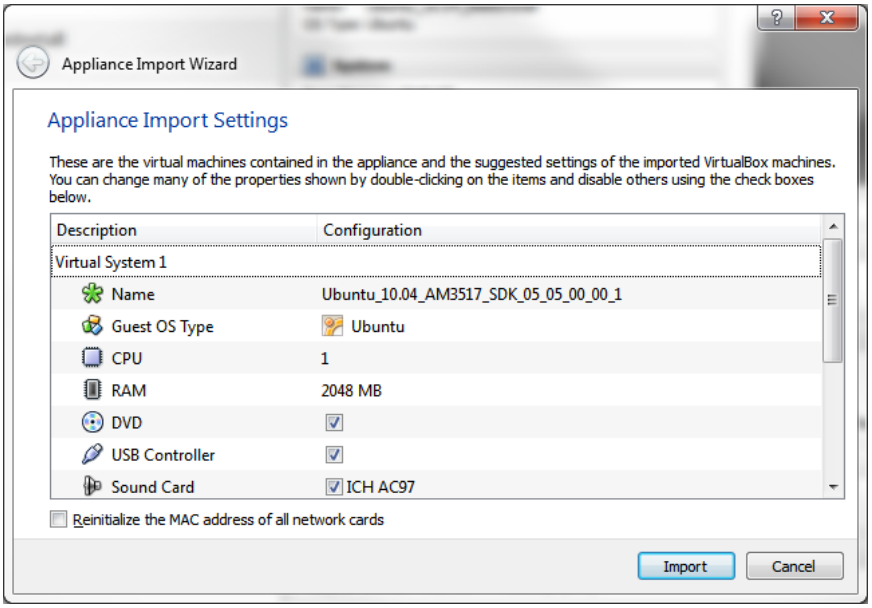
3. Click Open appliance… to select the .ova file from Step 1.



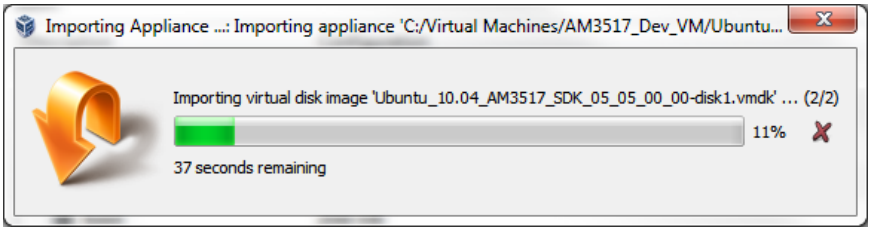
4. Navigate to the .ova file you downloaded in Step 1 and select it. Click Next.



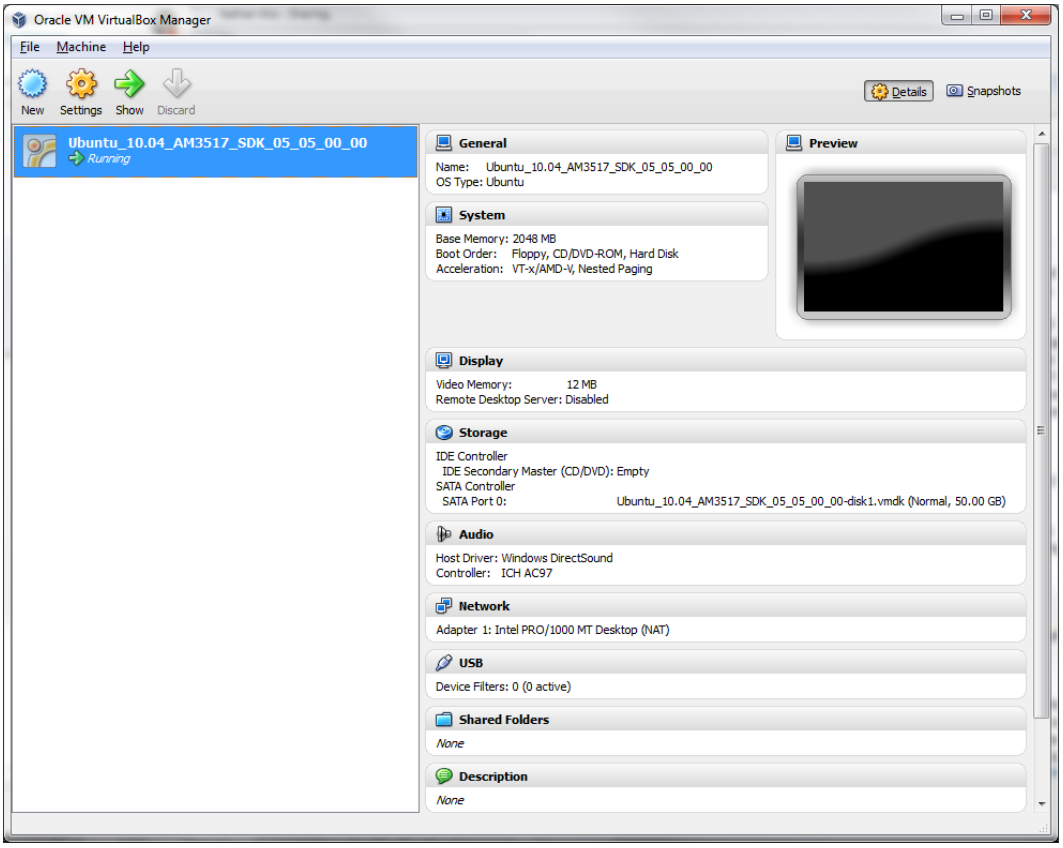
5. In the *Appliance Import Wizard* window that appears, click Import to begin the import/setup process.



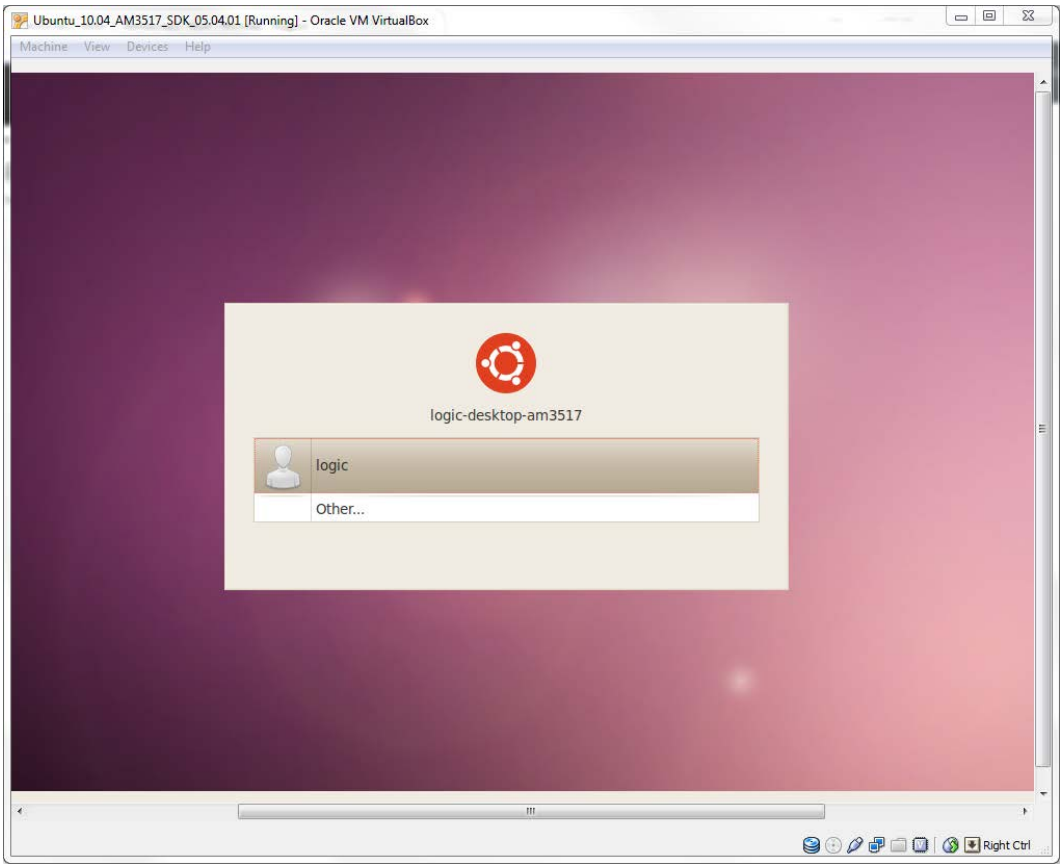
This will expand the compressed image and add it to your list of available VMs. The process will take several minutes and you should see a progress bar indicating the status.



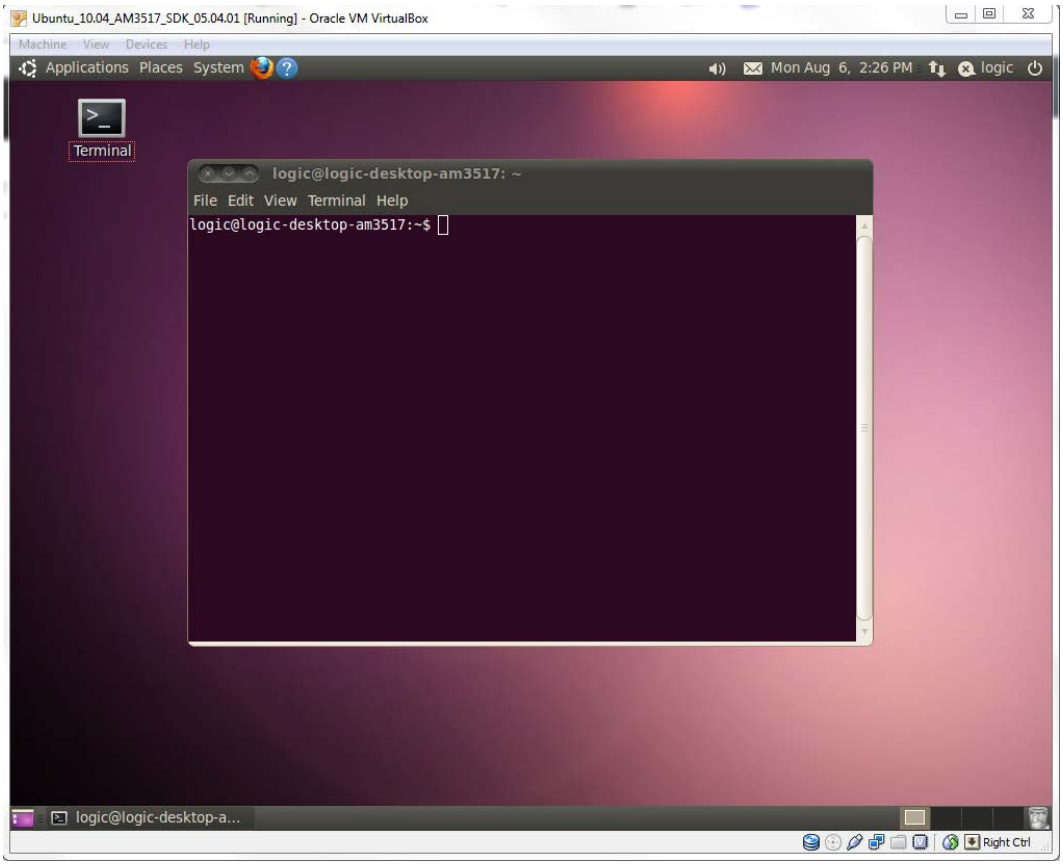
6. When the import process is complete, you will see the image has been added to your image manager. Double-click the image to start it.



7. In the Ubuntu login dialog box that appears, double-click the *logic* user name and enter the password *logic*. This will open the Ubuntu desktop.



8. Double-click the Terminal icon in the upper left corner of the screen to start the terminal prompt.



The VM you've just started is already configured with the TI AM3517 Linux SDK. Please proceed to the following sections that will detail how to use the SDK to build the bootloaders and Linux kernel.

3 Configure and Build Basic Software

The basic embedded GNU/Linux system will consist of a bootloader, Linux kernel, and root file system. TI's hardware splits the bootloader into two components, namely second- and third-stage loaders. This section explains how to configure and build the following components:

- SPL (second-stage bootloader)
- U-Boot (third-stage bootloader)
- Linux kernel
- Root file system

NOTE: The first-stage bootloader is ROM-based and is part of the processor. The only thing that can be changed about the first-stage bootloader by the user is to set the boot mode switches to configure it.

3.1 Set Up Development Environment

The SDK includes a script that configures the terminal session to properly access the build tools. However, the path names are extremely long, making documentation difficult. Let's first add some shortcuts to reduce this problem. Going forward we'll be able to refer to the short form of these common paths:

```
logic@logic-desktop-am3517:~$ ln -s ti-sdk-am3517-evm-05.05.00.00/
TI_SDK
logic@logic-desktop-am3517:~$ cd TI_SDK
logic@logic-desktop-am3517:~/TI_SDK$ ln -s board-support/linux-2.6.37-
psp04.02.00.07.sdk KERNEL
logic@logic-desktop-am3517:~/TI_SDK$ ln -s board-support/u-boot-
2011.09-psp04.06.00.08 UBOOT
```

Now, run the configuration script.

```
logic@logic-desktop-am3517:~/TI_SDK$ source linux-devkit/environment-
setup
[linux-devkit]:~/TI_SDK>
```

You will see that the terminal header is green. This process should be repeated for any terminal session that is opened to begin development work, including additional tabs in the same window.

3.2 Apply Patch to Source

AM3517 SOM-M2s with 4-bit ECC NAND may require a patch be added to the source before continuing to the build process. To determine the type of NAND on your SOM and whether these patches are necessary, please see Section 1.1.

The steps below will guide you through the patching process.

1. Download the Logic PD AM3517 Linux 4-bit ECC NAND Source Patch.[6]
2. Place the *4_bit_ecc.patch* file in the */home/logic/ti-sdk-am3517-evm-05.05.00.00* directory.

[6] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=2798

3. At the shell prompt, execute the command below.

```
logic@logic-desktop-am3517:~/TI_SDK$ patch -p1 < 4_bit_ecc.patch
patching file board-support/linux-2.6.37-
psp04.02.00.07.sdk/arch/arm/mach-omap2/board-flash.c
patching file board-support/linux-2.6.37-
psp04.02.00.07.sdk/drivers/mtd/nand/omap2.c
patching file board-support/u-boot-2011.09-
psp04.06.00.08/include/configs/am3517_evm.h
logic@logic-desktop-am3517:~/TI_SDK$
```

4. If you do not see any errors from the patch, the source files in the patch output have been patched and you can continue to Section 3.3.

NOTE: If your AM3517 SOM-M2 contains 4-bit ECC NAND according to Section 1.1, Logic PD recommends that you only use 4-bit ECC in the NAND driver. The 8-bit BCH ECC scheme has been observed to fail on these SOMs. For additional information, please contact Logic PD.

3.3 Build SPL and U-Boot

The second-stage bootloader in the current version of the SDK is the Second Program Loader (SPL), which replaced X-Loader. SPL is built at the same time as U-Boot and is tied very closely to it. SPL, as the second-stage bootloader, is configured to be small enough to fit into the 64k of static RAM included with the processor. It initializes DDR memory and enough of the peripheral devices to access and load U-Boot, the third-stage bootloader, into main memory.

3.3.1 Prerequisites

The procedures in this section require that you have the SDK downloaded and installed as described in Section 2.1 or Section 2.2.

3.3.2 Procedure

1. Switch to the *SPL/U-Boot* directory.

```
[linux-devkit]:~/TI_SDK> cd UBOOT
[linux-devkit]:~/TI_SDK/UBOOT>
```

2. Clean the build directories.

```
[linux-devkit]:~/TI_SDK/UBOOT> make CROSS_COMPILE=arm-arago-linux-
gnueabi- ARCH=arm distclean
awk '(NF && $1 !~ /^#/) { print $1 ": " $1 "_config; $(MAKE)" }'
boards.cfg > .boards.depend
[linux-devkit]:~/TI_SDK/UBOOT> rm -rf am3517
```

3. Commence the build of both SPL and U-Boot.

```
[linux-devkit]:~/TI_SDK/UBOOT> make O=am3517 CROSS_COMPILE=arm-arago-
linux-gnueabi- ARCH=arm am3517_evm
...
...
GP Header: Size 8d70 LoadAddr 40200800
```



```
make[2]: Leaving directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/board-support/u-boot-2011.09-psp04.06.00.08/spl'
make[1]: Leaving directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/board-support/u-boot-2011.09-psp04.06.00.08'
```

4. If you are not familiar with how ugly the output from a makefile build process can be, you may have difficulty being able to tell if the build was successful or not. A trick to tell for sure is to look at the return value that was passed to the shell from the build process. This is done by printing the '?' variable.

   NOTE: This will only give you an accurate answer if it's the FIRST command you give after the build process completes. Otherwise, the answer is lost forever.

```
[linux-devkit]:~/TI_SDK/UBOOT> echo $?
0
```

   If the value retuned by the *echo* command is 0, then the build was successful. Error values are usually negative numbers.

5. Check your results.

   The results of the build process are an *MLO* file for SPL and a *u-boot.img* file for U-Boot. You can find the files in the following locations:

   ```
   am3517/MLO
   am3517/u-boot.img
   ```

6. Copy the files to the *TFTP* directory for later use.

```
[linux-devkit]:~/TI_SDK/UBOOT> cp am3517/MLO /tftpboot/.
[linux-devkit]:~/TI_SDK/UBOOT> cp am3517/u-boot.img /tftpboot/.
```

3.4 Build Linux Kernel

This section explains how to configure and build the Linux kernel. At this point in time, Logic PD recommends that readers reuse the configuration provided and tested by TI. Once you have a full system up and running, you can go back and experiment with various kernel configurations.

3.4.1 Prerequisites

The procedures in this section require that you have the SDK downloaded and installed as described in Section 2.1 or Section 2.2.

3.4.2 Procedure

1. Open a new terminal session and start the configuration script.

```
logic@logic-desktop-am3517:~$ cd TI_SDK
logic@logic-desktop-am3517:~/TI_SDK$ source linux-devkit/environment-
setup
```

2. Switch to the Linux kernel directory.

```
[linux-devkit]:~> cd KERNEL
```


NOTE: The initial *cd* command is there because you may be starting from various places and the command will get you to a known point.

3. Clean the build directory.

```
[linux-devkit]:~/TI_SDK/KERNEL> make CROSS_COMPILE=arm-arago-linux-
gnueabi- ARCH=arm mrproper
  CLEAN   .
  CLEAN   arch/arm/kernel
  CLEAN   drivers/tty/vt
  CLEAN   drivers/video/logo
  CLEAN   kernel
  CLEAN   lib
  CLEAN   usr
  CLEAN   arch/arm/boot/compressed
  CLEAN   arch/arm/boot
  CLEAN   .tmp_versions
  CLEAN   vmlinux System.map .tmp_kallsyms1.o .tmp_kallsyms1.S
.tmp_kallsyms2.o .tmp_kallsyms2.S .tmp_kallsyms3.o .tmp_kallsyms3.S
.tmp_vmlinux1 .tmp_vmlinux2 .tmp_vmlinux3 .tmp_System.map
  CLEAN   scripts/basic
  CLEAN   scripts/genksyms
  CLEAN   scripts/kconfig
  CLEAN   scripts/mod
  CLEAN   scripts
  CLEAN   include/config include/generated
  CLEAN   .config .version include/linux/version.h Module.symvers
[linux-devkit]:~/TI_SDK/KERNEL>
```

4. Prepare the kernel by using the TI-provided configuration file.

```
[linux-devkit]:~/TI_SDK/KERNEL> make ARCH=arm CROSS_COMPILE=arm-arago-
linux-gnueabi- am3517_evm_defconfig
  HOSTCC  scripts/basic/fixdep
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-
...
...
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
[linux-devkit]:~/TI_SDK/KERNEL>
```

5. Build the Linux kernel.

```
[linux-devkit]:~/TI_SDK/KERNEL> make ARCH=arm CROSS_COMPILE=arm-arago-
linux-gnueabi- uImage
scripts/kconfig/conf --silentoldconfig Kconfig
  CHK     include/linux/version.h
  UPD     include/linux/version.h
  CHK     include/generated/utsrelease.h
  UPD     include/generated/utsrelease.h
  Generating include/generated/mach-types.h
  CC      kernel/bounds.s
```



```
  GEN     include/generated/bounds.h
  CC      arch/arm/kernel/asm-offsets.s
  GEN     include/generated/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  HOSTCC  scripts/genksyms/genksyms.o
...
...
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-2.6.37
Created:      Mon Aug  6 15:26:41 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3028736 Bytes = 2957.75 kB = 2.89 MB
Load Address: 0x80008000
Entry Point:  0x80008000
  Image arch/arm/boot/uImage is ready
[linux-devkit]:~/TI_SDK/KERNEL>
```

6. Locate the build output. As you can see from one of the last lines of the build, the completed Linux kernel image can be found at *arch/arm/boot/uImage*.
7. Copy the output to the *TFTP* directory for future use

```
[linux-devkit]:~/TI_SDK/KERNEL> cp arch/arm/boot/uImage /tftpboot/.
```

3.5 Prepare and Update Root FileSystem

Linux is only an OS kernel. It isn't very interesting without a corresponding root file system containing programs, libraries, files, etc. There are several tools and methods one can use to create a root file system for an embedded Linux device. Logic PD encourages you to use the provided root file systems images and treat them as a starting point; any desired components can be added to them.

The SDK created the */home/logic/ti-sdk-am3517-evm-05.05.00.00/targetNFS* directory to use as a file system sandbox. This directory is also used as the root of the NFS server hosted file system that can be used to boot the AM3517 via Ethernet. NFS is an extremely powerful and time-saving tool used for debugging a system. Since the file system actually resides on the host PC, the user may edit and change the file system in real-time while the AM3517 SOM-M2 is running. Booting the SOM from NFS will be covered later in this document.

In the following section, we will cover how to switch to one of the two TI-provided root file system images. You can see the images and their respective sizes by examining the file system directory of the SDK:

```
$ ls -lah $HOME/TI_SDK/filesystem
total 208M
drwxr-xr-x  2 logic logic 4.0K 2012-03-28 10:14 .
drwxr-xr-x 10 logic logic 4.0K 2012-07-10 14:44 ..
-rw-r--r--  1 logic logic  11M 2012-03-28 10:14 base-rootfs-am3517-
evm.tar.gz
-rw-r--r--  1 logic logic 197M 2012-03-28 10:13 tisdk-rootfs-am3517-
evm.tar.gz
```


As you can see, two file system images have been provided: a base/minimal image (*base-rootfs-am3517-evm.tar.gz)* that is small and a fully functioned image (*tisdk-rootfs-am3517-evm.tar.gz*) that is significantly larger. The user must decide which image they would like to start with for the purpose of their requirements. Most users will find that starting with the base file system and only adding the options they need is a good path. Alternatively, the fully functioned image may be chosen if the user is willing to take the time to strip out the items they do not need.

Both images are .tar files that have been compressed with gzip (*.tar.gz). The following example will explain how to extract these images into the *NFS* directory for future use.

NOTE: At various times, you will notice that the commands used in the following section are prefixed with the word *sudo*. This is because the NFS daemon that serves the NFS file system is a root process and thus the *targetNFS* directory is owned by root. Use the *sudo* prefix exactly as specified to avoid problems with permissions.

3.5.1 Prerequisites

The procedures in this section require that you have the SDK downloaded and installed as described in Section 2.1 or Section 2.2.

3.5.2 Procedure

1. Open a new terminal session and run the configuration setup.

```
logic@logic-desktop-am3517:~$ cd TI_SDK
logic@logic-desktop-am3517:~/TI_SDK$ source linux-devkit/environment-
setup
```

2. Switch to the *NFS* directory.

```
[linux-devkit]:~/TI_SDK> cd targetNFS
```

3. Create a backup of the current sandbox. These commands are the same commands you will use in the future to save or bundle the file system you have created.

   NOTE: Make sure you add the correct date to the file name. The file being created is large, so it may take several minutes.

```
[linux-devkit]:~/TI_SDK/targetNFS> sudo tar caf
$HOME/TI_SDK/targetNFS_2012-08-01.tar.gz .
[sudo] password for logic:
[linux-devkit]:~/TI_SDK/targetNFS>
```

4. Delete the current contents.

```
[linux-devkit]:~/TI_SDK/targetNFS> sudo rm -rf *
[linux-devkit]:~/TI_SDK/targetNFS>
```


5. Unpack the contents of the image provided by TI. In this example, we will load the base image.

```
[linux-devkit]:~/TI_SDK/targetNFS> sudo tar xf ../filesystem/base-
rootfs-am3517-evm.tar.gz
[linux-devkit]:~/TI_SDK/targetNFS>
```

6. Verify that the contents were extracted successfully.

```
[linux-devkit]:~/TI_SDK/targetNFS> ls
bin  boot  dev  etc  home  lib  linuxrc  media  mnt  proc  sbin  srv
sys  tmp  usr  var
[linux-devkit]:~/TI_SDK/targetNFS>
```

At this point, the root file system is prepared for use via NFS.

7. Future sections of this document will load the root file system into NAND as a JFFS2 image. To prepare for that, an additional step is needed to create the JFFS2 image. The *targetNFS* directory must be packaged as a JFFS2 file.

```
[linux-devkit]:~/TI_SDK/targetNFS> cd ..
[linux-devkit]:~/TI_SDK> sudo mkfs.jffs2 -lqn -e 128 -r targetNFS -o
rootfs.jffs2
[linux-devkit]:~/TI_SDK>
```

8. Move the output to the *TFTP* directory.

```
[linux-devkit]:~/TI_SDK> cp rootfs.jffs2 /tftpboot/.
[linux-devkit]:~/TI_SDK>
```

3.5.3 Upgrade Procedure

The larger, full-featured file system that was supplied with the kit is required for some examples in this document. The .tar file for that file system is named *tisdk-rootfs-am3517-evm.tar.gz* and can be found in the same directory as the base file system *base-rootfs-am3517-evm.tar.gz*. To update to the full file system image, follow the steps in Section 3.5.2 and substitute *tisdk-rootfs-am3517-evm.tar.gz* at Step 5.

4 Working with Development Kit

This section contains detailed procedures for several common activities that users will perform with their development kit.

4.1 First-time Boot of AM3517 SOM-M2

At the time this document was written, the Logic PD AM3517 SOM-M2 was being shipped with no bootloaders present in NAND. Because of this, you will not be able to proceed until you have something on the SOM to talk to in order to complete the configuration of the device.

Logic PD's *AN 506 AM3517 SOM-M2 Blank SOM Programming*[7] describes how to load a default set of bootloaders onto the AM3517 SOM-M2 so you may proceed. Please download the ZIP file and follow the instructions therein before continuing. If your SOM already has bootloaders on it, you may skip this step; however, it may be beneficial to complete this step anyway to guarantee that you have the correct environment.

NOTE: To condense this document, the command prompt has been shortened in some of the examples below. The full command prompt of `logic@logic-desktop-am3517` or `[linux-devkit]:~>` may simply be replaced with `host$`. Similarly, the prompt for examples showing output from the kit when it is running Linux may be shortened to `arago#`.

4.2 Create Serial Connection to Hardware

In this section, we will boot the development kit and talk to it.

4.2.1 Prerequisites

The procedures in the section require the following items:

- A working AM3517 Development Kit
- A null-modem serial cable (provided in the development kit)

4.2.2 Procedure

Minicom is a friendly serial communication program that we will use it to talk to our embedded hardware.

1. Which serial port you use to access your development kit is a large topic all to itself and has been given an entire section of this document. Please read Appendix B before proceeding and select the device name of the serial port you will use for your kit. This value will be critical for the next step.
2. Minicom reads configuration data out of the *$HOME/.minirc.dfl* file at startup. We will edit this file to set the serial port, baudrate, and mode (115200,8,N,1) using the Minicom variables port, baudrate, bits, parity, and stopbits.

   Also, Minicom has historically been used to create a serial connection to a modem. As such, its default behavior is to send initialization strings at start up. Similarly, it also sends strings when shutting down. Unfortunately, these strings can confuse embedded systems that aren't expecting them. By setting the Minicom variables *minit*, *mreset*, and *mhangup* to empty, Minicom will not send unnecessary communication over the serial line.

[7] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=927

Using your favorite text editor, edit or create the *$HOME/.minirc.dfl* file to include the lines shown below. If the file is already present, align the setting with what you see here. Be sure to adjust the *port* value as appropriate for your system. If you are using the VM, please see Section 0, which will help you select the proper port value.

```
# My custom minicom configuration file.
pu port         /dev/ttyS0
pu baudrate     115200
pu bits         8
pu parity       N
pu stopbits     1
pu rtscts       No
pu minit
pu mreset
pu mhangup
```

3. Connect a null-modem serial cable between the development kit and your host PC.
4. Start the Minicom program. **NOTE:** You may want to do this from a second command terminal tab to make it easier to switch between the host PC and the target.

```
[linux-devkit]:~> minicom

Welcome to minicom 2.4

OPTIONS: I18n
Compiled on Jan 25 2010, 06:49:09.
Port /dev/ttyS0

Press CTRL-A Z for help on special keys
```

5. Power on the development kit. You should see the bootloaders start with the output below.

```
U-Boot SPL 2011.09-00000-gd919f36-dirty (May 03 2012 - 15:04:36)
Texas Instruments Revision detection unimplemented
OMAP SD/MMC: 0
reading u-boot.img
reading u-boot.img


U-Boot 2011.09-00000-gd919f36-dirty (Aug 02 2012 - 16:02:46)

AM35XX-GP ES2.0, CPU-OPP2, L3-165MHz, Max CPU Clock 600 Mhz
AM3517EVM Board + LPDDR/NAND
I2C:   ready
DRAM:  256 MiB
WARNING: Caches not enabled
Now running in RAM - U-Boot at: 8ff70000
```

6. Press any key to stop U-Boot from automatically booting. You should now be at the U-Boot prompt.

```
AM3517_EVM #
```


NOTE: The output from your development kit will almost certainly differ from the text shown above. Variations will occur based on when the kit was built, what version of software is installed on it, what chip revision it is, etc. You may also find that U-Boot displays a different prompt. On the kit used to write this document, the prompt happens to be AM3517_EVM#. To avoid possibly confusing a U-Boot prompt with a Linux or host prompt, we will use `U-Boot >` throughout this document when we are referring to commands being entered into the U-Boot session running on the kit.

4.2.3 References

- *Embedded Linux Minicom* wiki page[8]
- Minicom Linux man page[9]
- Minicom configuration files such as:
  - */usr/share/doc/minicom/examples/minicom.users*
  - */usr/share/doc/minicom/examples/minirc.dfl*
  - */etc/minicom/minicom.users*

4.3 Working with U-Boot Environment

U-Boot keeps a list of environment variables stored in non-volatile flash memory. Some of the variables are just convenient ways to refer to file names and addresses. Others specify the system's boot delay and IP address. One variable of particular interest is named *bootargs*. The value of *bootargs* is passed as the command line to the Linux kernel.

In fact, the process of booting the Linux kernel from U-Boot can be broken down into roughly three steps:

1. Initialize *bootargs* to a proper kernel command line. Oftentimes the trickiest part of this step is setting up the root parameter which tells Linux where to find its root file system (/).
2. Load the Linux kernel into memory. This is normally a transfer of some sort (e.g., over a network via TFTP, copied from flash memory).
3. Start the kernel with the *bootm* command.

During this lab, we will configure several U-Boot environment variables that cover the steps above. We will explain the variables as we create and save them.

4.3.1 Understand Flash Memory Map

It will help you greatly to understand how the flash memory of the device is allocated, as well as how the allocation is determined.

At this point in time, we will concern ourselves with the location of three items, namely SPL, U-Boot, and the U-Boot environment. Once we have them sorted out, we can assign the rest of the flash to different uses, like storing the Linux kernel and a root file system.

```
----   0x2000,0000   ----

       Root File System (e.g. JFFS2)

----   0x0078,0000   ----

       Linux Kernel (uImage)
```

[8] http://elinux.org/Minicom
[9] http://linux.die.net/man/1/minicom


```
----   0x0028,0000   ----

       U-Boot Environment

----   0x0026,0000   ----

       U-Boot Code (u-boot.bin)

----   0x0008,0000   ----

       X-Loader (MLO)

----   0x0000,0000   ----
```

All good programmers have a natural aversion to magic numbers; unfortunately, the latest SDK is not as clear on where these values are defined as the previous SDK. Some of the values can be observed in the following files:

```
.../TI_SDK/UBOOT/include/configs/am3517_evm.h
.../TI_SDK/KERNEL/arch/arm/mach-omap2/board-am3517evm.c
```

Reviewing the contents of the files above and looking at the default environment variables will help you understand how the memory map was defined. Key areas to note are:

```
#define SMNAND_ENV_OFFSET            0x260000 /* environment starts here */
#define CONFIG_SYS_NAND_U_BOOT_OFFS    0x80000
#define CONFIG_EXTRA_ENV_SETTINGS      EXTRA_ENV_SETTINGS ….
#define CONFIG_JFFS2_PART_OFFSET       0x680000
```

For this last value, a keen observer will notice that there's a discrepancy between U-Boot and the kernel on the location of the start of the root file system area. The kernel trumps U-Boot in that it is the kernel that must actually load the root file system, and the true address used by U-Boot is specified in the environment variables.

NOTE: Because the actual U-Boot prompt is changeable and the default U-Boot prompt is long, it has been replaced in many places in this document with `U-Boot >` for clarity and brevity.

4.3.2 Prerequisites

The procedures in this section require a serial connection to a running AM3517 Development Kit as discussed in Section 4.2.

4.3.3 Procedure

1. Erase the U-Boot environment.

   To ensure everyone is on the same page, it may help to completely reset the U-Boot environment on the development kit to its factory settings. This step is optional and you may skip it if you have a new development kit or are comfortable with your current settings.

   The U-Boot environment is stored in flash memory from 0x26,0000 -> 0x28,0000. You may erase it using the commands below.

```
U-Boot > nand device 0
U-Boot > nand erase 0x260000 0x20000
```


For the curious, the related defines and default environment that was used to compile U-Boot is defined in *.../TI_SDK/UBOOT/include/configs/am3517_evm.h*. See the variables CONFIG_ENV_SIZE and SMNAND_ENV_OFFSET.

2. Cycle the power on the development kit and stop the auto boot countdown. You should see the message below.

```
*** Warning - bad CRC, using default environment.
```

This verifies that U-Boot did not find a valid environment stored in NAND because we erased it and U-Boot resorted to using the environment it was compiled with.

3. View the current state of the U-Boot environment using the *printenv* command.

```
U-Boot > printenv

baudrate=115200
bootargs_defaults=setenv bootargs console=${console} ${optargs}
bootcmd=if mmc rescan ${mmcdev}; then if run loadbootscript; then run
bootscript; else if run loadbootenv; then echo Loaded environment from
${bootenv};run i
mportbootenv;fi;if test -n $uenvcmd; then echo Running uenvcmd ...;run
uenvcmd;fi;if run loaduimage; then run mmcboot; else run nandboot; fi;
fi; else run na
ndboot; fi
bootdelay=3
bootenv=uEnv.txt
bootfile="uImage"
bootscript=echo Running bootscript from mmc ...; source ${loadaddr}
console=ttyO2,115200n8
dieid#=3cf400010000000001685a201600b00b
ethact=DaVinci-EMAC
ethaddr=40:98:4e:73:9a:f6
importbootenv=echo Importing environment from mmc ...; env import -t
$loadaddr $filesize
kloadaddr=0x80007fc0
loadaddr=0x82000000
loadbootenv=fatload mmc ${mmc_dev} ${loadaddr} ${bootenv}
loadbootscript=fatload mmc ${mmcdev} ${loadaddr} boot.scr
loaduimage=fatload mmc ${mmcdev} ${kloadaddr} uImage
mmcargs=run bootargs_defaults; setenv bootargs ${bootargs}
root=/dev/mmcblk0p2 rw rootfstype=ext3 rootwait
mmcboot=echo Booting from mmc ...; run mmcargs; bootm ${kloadaddr};
mmcdev=0
nandargs=run bootargs_defaults; setenv bootargs ${bootargs}
root=/dev/mtdblock4 rw rootfstype=jffs2
nandboot=echo Booting from nand ...; run nandargs; nand read
${kloadaddr} 280000 400000; bootm ${kloadaddr};
stderr=serial
stdin=serial
stdout=serial

Environment size: 1415/131068 bytes
```

Your environment may look different from the output above, depending on what you've been doing with your kit. You may have more than the variables shown above.

For now, you should simply take note of them and proceed with the rest of the section.

4. Erase the *bootcmd* variable.

   The *bootcmd* variable lists the commands that U-Boot will automatically execute at startup. The default setting on the author's kit was *if mmc init; then if run loadbootscript; ...* . This complex command causes the program to try to load scripts and a kernel image from SD/MMC or, failing that, to load from NAND flash.

```
U-Boot > setenv bootcmd
```

5. Erase the *bootargs* variable.

   The *bootargs* variable holds the arguments passed to the Linux kernel. On the author's kit, *bootargs* is set by other variables depending on the boot method. Thus, technically speaking, this step is not necessary, but it won't hurt anything and ensures old settings will not mess us up later.

```
U-Boot > setenv bootargs
```

6. Set up the networking variables using the commands below. Remember to adjust the values for your network settings as necessary.

   a. Set the IP address for your gateway.

```
U-Boot > setenv gateway 192.168.xxx.xxx
```

   b. Set the IP address for your development kit.

```
U-Boot > setenv ipaddr 192.168.xxx.xxx
```

   c. Set the IP address for your host PC.

```
U-Boot > setenv serverip 192.168.xxx.xxx
```

   d. Set the netmask.

```
U-Boot > setenv netmask 255.255.255.0
```

7. Set the location in RAM where images will be downloaded.

```
U-Boot > setenv loadaddr 0x83000000
```

8. Another convenient variable is *otherbootargs*. By including a reference to it in any script that sets *bootargs*, it is easy to insert temporary changes into the kernel command line (e.g., *ignore_loglevel*). For now, set it to an ASCII space so that it will not impact any command lines, yet won't be forgotten.

```
U-Boot > setenv otherbootargs ' '
```


9. Create variables to support an NFS-based root file system.

   The command line passed to a Linux kernel via the *bootargs* variable can easily become long and unwieldy. To simplify use and maintenance, we will create environment variables to hold different parts of the command line.

   a. Set a variable that holds the path to the NFS export where our device can mount its root file system. Ensure that the value you assign this variable matches the directory on your host PC where you are exporting the device's root file system via NFS. The path you see below fits with the VM setup that is used with this document :

```
U-Boot > setenv rootpath /home/logic/TI_SDK/targetNFS
```

   If you are not using the VM, you can use the following path after you update the *<INSTALL_PATH>* portion:

```
U-Boot > setenv rootpath /<INSTALL_PATH>/ti-sdk-am3517-evm-
05.05.00.00/targetNFS
```

   b. Set a variable that holds any options we pass to the kernel in respect to mounting its root file system via NFS. **NOTE:** Be sure to include the leading comma (,) in this definition.

```
U-Boot > setenv nfsoptions ',nolock'
```

   c. Create a variable that, when run, will assign *bootargs* to a proper command line for mounting root via NFS using the above variables.

   NOTE: It is important to enter the entire command below at once and to properly execute the placement of the single quotations (' '). If you do not do this, the command will fail.

```
U-Boot > setenv nfsargs 'setenv bootargs console=${console} noinitrd
root=/dev/nfs rw nfsroot=${serverip}:${rootpath}${nfsoptions} ip=dhcp
${otherbootargs}; '
```

   To help the reader, each component of the new value for *nfsargs* is listed individually below.

   - setenv bootargs
   - console=${console}
   - root=/dev/nfs rw nfsroot=${serverip}:${rootpath}${nfsoptions}
   - ip=dhcp
   - ${otherbootargs} ;

   As you can see, entering a long variable into U-Boot is somewhat tedious and the shell is pretty unforgiving. If you make a mistake, it's hard to go back and fix it. That's why we use intermediate variables like *rootpath* and *console*.

   NOTE: If your network does not have a DHCP server, or if the kit is to be assigned a static IP address for some other reason, you may substitute *ip=dhcp* for *ip=a:b:c:d:e:f:g* as defined in *$KERNEL/Documentation/filesystems/nfsroot.txt.*

10. Create variables to support an MTD-based root file system.

    During this step we will follow a similar process to support a proper kernel command line that will mount the root file system from a flash partition. Again, you need to enter the entire command at once.

    TIP: Your development kit may already have the following environment variable defined. If so, simply verify it and leave it alone.

```
U-Boot > setenv nandargs 'setenv bootargs console=${console} noinitrd
root=/dev/mtdblock4 rw rootfstype=jffs2 ${otherbootargs}; '
```

11. Create variables to support an MMC-based root file system

    Similar to the idea above, we can create a proper command line that will tell the Linux kernel to mount its root file system on an SD/MMC card. Again, this environment variable may already be defined for you.

```
U-Boot > setenv mmcargs 'setenv bootargs console=${console}
root=/dev/mmcblk0p2 rw rootfstype=ext3 rootwait ${otherbootargs}; '
```

    NOTES ABOUT THE ABOVE COMMAND LINE:

    - */dev/mmcblk0p2* indicates the second partition on the SD/MMC card. Adjust if necessary.
    - *rootfstype=ext3* specifies that the root partition be formatted as a third extended file system. Adjust if necessary.
    - *rootwait* instructs the kernel to wait until the *udev* subsystem fully populates the */dev* directory. If this parameter is not present, the kernel may panic because root exists on a device that has yet to be enumerated.

12. Create a variable to download a Linux kernel over a network.

```
U-Boot > setenv loadnet 'tftpboot ${loadaddr} ${serverip}:${bootfile};'
```

13. Create a variable to copy a Linux kernel from flash.

```
U-Boot > setenv loadfl 'nand device 0; nand read ${loadaddr} 0x280000
0x500000;'
```

14. Create a variable to copy a Linux kernel from MMC.

```
U-Boot > setenv loadmmc 'mmc init; fatload mmc 0 ${loadaddr}
${bootfile};'
```

15. Save the environment to flash so it persists across power cycles.

```
U-Boot > saveenv

Saving Environment to NAND...
Erasing Nand...
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
```


16. Cycle the power on the development kit and verify that the environment variables you have created are still there. You should also notice that, without *bootcmd* defined, U-Boot automatically stops at the prompt.
17. At this point, loading and running a Linux kernel is simply a matter of running the variables above. For example, should you want to load your kernel over the network and mount root via NFS, you would enter the commands below.

```
U-Boot > run nfsargs
U-Boot > run loadnet
U-Boot > bootm ${loadaddr}
```

Eventually you can string the above steps together into a final *bootcmd* variable that will execute automatically whenever the kit is powered on (after *bootdelay* seconds have passed, of course).

4.3.4 References

- *DENX U-Boot Environment Variables* wiki page[10]
- *DENX U-Boot Memory Commands* wiki page[11]

4.4 Update SPL and U-Boot

At some point you are likely to want to replace SPL and U-Boot on the hardware. This process consists roughly of the following steps:

1. Boot U-Boot.
2. Load the new SPL into main memory.
3. Erase the SPL section of flash.
4. Copy the new SPL from main memory into flash.
5. Repeat Steps 2 through 4 for U-Boot.
6. Possibly erase the U-Boot environment so the new program will not try and do the same thing the old one did.

It will help you greatly if you understand how the flash memory is allocated. Please review Section 4.3.1 before proceeding.

4.4.1 Prerequisites

The procedures in this section require the following items:

- A serial connection to a running AM3517 Development Kit as discussed in Section 4.2.
- Built versions of SPL and U-Boot.

4.4.2 Procedure

The procedure below loads the new versions of SPL and U-Boot into main memory over a network via TFTP.

IMPORTANT NOTE: Please take note of the switching between ECC algorithms below. SPL (*MLO*) is loaded by the processor's ROM bootloader. The ROM loader will calculate ECC in hardware as it reads NAND because that's all it can do. However, U-Boot is loaded by SPL, which will calculate an ECC using software.

[10] http://www.denx.de/wiki/DULG/UBootEnvVariables
[11] http://www.denx.de/wiki/DULG/UBootCmdGroupMemory

1. Update SPL.
    a. Erase the RAM buffer by filling it with 0xFF.

```
U-Boot > mw.b ${loadaddr} 0xFF 0x240000
```

    b. Load the new SPL binary into the RAM buffer.

```
U-Boot > tftp ${loadaddr} ${serverip}:MLO

Using DaVinci EMAC device
TFTP from server 192.168.1.42; our IP address is 192.168.1.10
Filename 'MLO'.
Load address: 0x83000000
Loading: ####
done
Bytes transferred = 18888 (49c8 hex)
```

    c. Erase the area of flash where the current SPL resides.

```
U-Boot > nand erase 0x0 0x50000
NAND erase: device 0 offset 0x0, size 0x50000
Erasing at 0x40000 -- 120% complete.
OK
```

    d. Set the ECC calculation to *hw 2*.

```
U-Boot > nandecc hw 2

HW ECC [X-loader/U-boot layout] selected
```

    e. Copy the new SPL binary from main memory into flash.

```
U-Boot > nand write ${loadaddr} 0x0 0x50000

NAND write: device 0 offset 0x0, size 0x50000
 327680 bytes written: OK
```

2. Update U-Boot.
    a. Erase the RAM buffer by filling it with 0xFF.

```
U-Boot > mw.b ${loadaddr} 0xFF 0x240000
```

    b. Load the new U-Boot binary into the RAM buffer.

```
U-Boot > tftp ${loadaddr} ${serverip}:u-boot.img

Using DaVinci EMAC device
TFTP from server 192.168.1.42; our IP address is 192.168.1.10
Filename 'u-boot.img'.
Load address: 0x83000000
```



```
Loading: ###############################################
done
Bytes transferred = 224820 (36e34 hex)
```

c. Erase the area of flash where the current U-Boot resides.

```
U-Boot > nand erase 0x80000 0x1C0000

NAND erase: device 0 offset 0x80000, size 0x1c0000
Erasing at 0x220000 -- 100% complete.
OK
```

d. Set the ECC calculation to *hw 2*.

```
U-Boot > nandecc hw 2

HW ECC [X-loader/U-boot layout] selected
```

e. Copy the new U-Boot binary from main memory into flash.

```
U-Boot > nand write.i ${loadaddr} 0x80000 0x1C0000

NAND write: device 0 offset 0x80000, size 0x1c0000
 1835008 bytes written: OK
```

3. Reset the development kit and verify that the new versions of SPL and U-Boot load and execute.

NOTE: You may also load the files above from a FAT-formatted SD/MMC card. To do so, first activate the MMC card interface with the following command:

```
U-Boot > mmc rescan
```

Then follow the procedure above but replace the *tftp* command with TFTP loads using the following format:

```
U-Boot > fatload mmc 0 ${loadaddr} <filename>
```

4.4.3 References

- *TI Flashing U-Boot* wiki article[12]

[12] http://processors.wiki.ti.com/index.php/AMSDK_u-boot_User%27s_Guide#Installing_MLO_and_u-boot

4.5 Boot Linux Using NFS-Based Root File System

This section will explain how to load and run the Linux kernel. We have chosen to start out by mounting the device's root file system via a NFS share. This is a common method for development because it makes sharing files between the host PC and embedded device very easy.

4.5.1 Prerequisites

The procedures in this section require the following items:

- A properly exported NFS share as described in Section 2.1 or 2.2.
- A root file system as built in Section 3.5 and extracted into the NFS share identified above.
- A serial connection to the device as described in Section 4.2.
- A U-Boot environment as described in Section 4.3, or enough familiarity with U-Boot to follow along and fill in the gaps.

4.5.2 Procedure

1. Create a serial connection to your development kit and power it on. Interrupt any default boot command before it executes so that you are sitting at the U-Boot prompt.

```
U-Boot >
```

2. Run the *nfsargs* variable that we previously created.

```
U-Boot > run nfsargs
```

For convenience, the definition of this variable is shown below. This will set *bootargs* to a proper command line for the kernel.

```
setenv nfsargs 'setenv bootargs console=${console} noinitrd
root=/dev/nfs rw nfsroot=${serverip}:${rootpath}${nfsoptions} ip=dhcp
${otherbootargs}; '
```

3. Run the *loadnet* variable to download the Linux kernel to the device.

```
U-Boot > run loadnet

Using DaVinci EMAC device
TFTP from server 192.168.1.42; ...
Filename 'uImage'.
Load address: 0x83000000
Loading: #######################
         #######################
         ##############...
done
Bytes transferred = 3028992 (2e3800 hex)
```


Again, the definition of this variable is repeated below for convenience.

```
loadnet='tftpboot ${loadaddr} ${serverip}:${bootfile};'
```

4. Boot the downloaded kernel.

```
U-Boot > bootm ${loadaddr}

## Booting kernel from Legacy Image at 83000000 ...
   Image Name:   Linux-2.6.32
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2473736 Bytes =  2.4 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux......

  ... Many Lines of Output from the Linux Kernel ...
  ... and the Init Process                      ...
 _____                  _____           _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                   |___|

Arago Project http://arago-project.org arago ttyO2

Arago 2009.11 arago ttyO2

arago login:
```

5. Log in to the device as *root* - no password should be required.

```
arago login: root
arago#
```

At this point, you should feel free to explore the device. Play around in the root file system, find out what applications are available, etc.

6. To make things a little easier for our application development, we will create a directory with global access on the NFS share. That way we can easily move files and applications between our host PC and development kit without having to remember to *sudo* certain commands.

```
host$ cd /home/logic/TI_SDK
host$ sudo mkdir targetNFS/home/root/share
host$ sudo chmod 777 targetNFS/home/root/share
host$ echo "Hello, world" >> targetNFS/home/root/share/hello
arago# cat /home/root/share/hello
```



```
Hello, world
```

4.6 Working with SD / MMC Cards

4.6.1 Create Bootable SD / MMC Card

The AM3517 hardware can automatically mount a FAT-formatted SD or MMC card and boot system software from it. In general, the SD/MMC card must have:

- A FAT16 or FAT32 partition marked *bootable*
- A geometry of:
  - 255 heads
  - 63 sectors
  - appropriate number of cylinders
- The *MLO* file located in the root directory (This file must also be the first file copied to the card.)

4.6.1.1 Prerequisites

The procedures in this section require the ability to execute the host PC's *fdisk*, *mkfs*, and *umount* programs.

4.6.1.2 Procedure

1. Insert an SD/MMC card into your host PC.
2. Use the *mount* command to find the device entry that corresponds to the SD/MMC card.

```
host$ mount

    ...
/dev/sdb1 on /media/disk type vfat
    ...
```

   On the author's system, the SD/MMC card was automatically mounted as */media/disk*. By reviewing the output above, we can see that the card corresponds to */dev/sdb*.

3. Unmount the SD/MMC card.

```
host$ sudo umount /media/disk
```

4. Start the *fdisk* utility. **NOTE:** You may need to become root to do this.

```
host$ sudo fdisk /dev/sdb

GNU Fdisk 1.2.1
Copyright (C) 1998 - 2006 Free Software Foundation
    ...
Using /dev/sdb
Command (m for help):
```


5. Clear the existing partition table.

```
Command (m for help): o
```

6. Display information about the card.

```
Command (m for help): p

Disk /dev/sdb: 1 GB, 1965841920 bytes
xxx heads, yy sectors/track, zzz cylinders
Units = cylinders of ..... * ... = ....... bytes

 Device Boot  Start    End   Blocks   Id  System
```

Note the number of bytes on the card. The card in this example had *1965841920* bytes.

7. Calculate the number of cylinders the card will have based on the number of bytes in the output above. The calculation is:

bytes / heads / sectors / 512 = cylinders

In the working example, this reduces to:

1965841920 / 255 / 63 / 512 = 239

If your calculation includes a fraction, round down to the nearest integer.

8. Enter *expert* mode.

```
Command (m for help): x
```

9. Set the number of heads to 255.

```
Expert command (m for help): h

Number of heads (default xxx):  < 255 >
```

10. Set the number of sectors to 63.

```
Expert command (m for help): s

Number of sectors (default yy):  < 63 >
```

11. Set the number of cylinders to the value calculated above.

```
Expert command (m for help): c

Number of cylinders (default zzz):  < 239 >
```

12. Leave *expert* mode.

```
Expert command (m for help): r
```


13. Create a new primary partition from cylinder 0 to the last.

```
Command (m for help): n

Partition type
   e   extended
   p   primary partition (1-4)
< p >
First cylinder  (default 0cyl):   < 1 >
Last cylinder or +size or +sizeMB or +sizeKB (default xx): <enter>
```

14. Mark the newly created partition bootable.

```
Command (m for help): a

Partition number (1-1):  < 1 >
```

15. Mark the partition as a FAT file system.

```
Command (m for help): t

Partition number (1-1):  < 1 >
Hex code (type L to list codes):  < c >
Changed type of partition 1 to c (FAT32 LBA)
```

16. Double check all of our changes before writing them to the card.

```
Command (m for help): p

Disk /dev/sdb: 1 GB, 1965841920 bytes
255 heads, 63 sectors/track, 239 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot  Start  End  Blocks   Id  System
/dev/sdb1   *       1  239 1919736    c  FAT32 LBA
```

17. Commit the changes to the card.

```
Command (m for help): w

Information: Don't forget to update /etc/fstab

Writing all changes to /dev/sdb.
```

18. Sometimes a host will automatically remount the card once the changes have been made. We need to make sure that the card is *not* mounted so we can format it.

```
host$ mount
```


If the card has been mounted again, un-mount it using the *umount* command.

```
host$ sudo umount /media/disk
```

19. Format the newly partitioned card.

```
host$ sudo mkfs.vfat -F 32 /dev/sdb1

mkfs.vfat 3.0.1 (23 Nov 2008)
```

TIP: If you have trouble with your development kit booting the SD/MMC card, try formatting it as FAT16 instead of FAT32. This involves changing the *partition type* to "FAT16 LBA" in *fdisk* as well as using a different formatting command (*mkfs.vfat -F 16*).

IMPORTANT NOTE: On the author's Debian Lenny system, the *fdisk* command seems to have some peculiar behavior by which you must start the new partition at Cylinder 0. This doesn't make much sense because the partition table goes there. However, *fdisk* seems to perform some behind the scenes magic.

4.6.1.3 References

- *TI SD/MMC Format for OMAP3 Boot* wiki page[13]
- Fdisk Linux man-page[14]
- Mkfs Linux man-page[15]

4.6.2 Create SD/MMC Boot Script

4.6.2.1 Automatically Program System using SD/MMC Boot

As configured by TI, if booting from SD/MMC, U-Boot will automatically try to load and run a file named *boot.scr* from the SD/MMC card. For more information regarding this, take a look at the default U-Boot environment provided in *.../TI_SDK/UBOOT/include/configs/am3517_evm.h*. The environment is defined in the *C pre-processor macro CONFIG_EXTRA_ENV_SETTINGS*. Specifically, note the definitions of *CONFIG_BOOTCOMMAND* and *loadbootscript*.

The file *boot.scr* is a simple text file containing U-Boot commands, wrapped in a header using the *mkimage* program. During this section, we will string together the commands necessary to:

1. Completely erase the device's flash memory
2. Program X-Loader
3. Program U-Boot
4. Program the Linux kernel
5. Program a JFFS2-formatted root file system

These items are the basic building blocks for recovering a bricked board and programming a finished product during manufacturing.

4.6.3 Prior U-Boot Environments

NOTE: These instructions are for an AM3517 SOM-M2 with 4-bit ECC NAND. If you are using an older SOM with 1-bit ECC NAND, please contact Logic PD for support.

[13] http://processors.wiki.ti.com/index.php/SD/MMC_format_for_OMAP3_boot
[14] http://linux.die.net/man/8/fdisk
[15] http://linux.die.net/man/8/mkfs

The above script will only work if the original/default U-Boot environment is present. Namely, the environment defined in *.../TI_SDK/UBOOT/include/configs/am3517_evm.h*. If you have been using your development kit and have stored a U-Boot environment in its flash, that environment will be used. If you want to start fresh, simply erase the U-Boot environment using the procedure in Step 1 of Section 4.3.3.

4.6.3.1 Prerequisites

The procedures in this section require that U-Boot is compiled and built as described in Section 3.3. Specifically, you will need the *mkimage* program which is created during this process.

4.6.3.2 Procedure

1. Create a file named *boot.txt* in the */tftpboot* directory.

```
host$ touch /tftpboot/boot.txt
```

2. Using your favorite editor, add the following U-Boot commands into */tftpboot/boot.txt*.

   TIP: If you have forgotten how the addresses and offsets below are calculated, please review Section 4.3.1. If you are unsure of the commands used to program SPL and U-Boot, please review Section 4.4.

```
# This file will be loaded by U-Boot into RAM at
# 0x8300,0000 ( the value of ${loadaddr} ).
setenv ram_buf 0x83000000
setenv buf_sz  0x01000000

echo "Initializing MMC"
mmc rescan

echo "Initializing and erasing NAND"
nand device 0
nand erase.chip

echo "loading MLO"
mw.b ${ram_buf} 0xff ${buf_sz}
fatload mmc 0 ${ram_buf} MLO
nandecc hw 2
nand write.i ${ram_buf} 0x00 0x20000
nand write.i ${ram_buf} 0x20000 0x20000
nand write.i ${ram_buf} 0x40000 0x20000
nand write.i ${ram_buf} 0x60000 0x20000

echo "loading u-boot"
mw.b ${ram_buf} 0xff ${buf_sz}
fatload mmc 0 ${ram_buf} u-boot.img
nandecc hw 2
nand write.i ${ram_buf} 0x80000 0x1C0000

echo "loading kernel"
setenv kern_addr  0x00280000
mw.b ${ram_buf} 0xff ${buf_sz}
nand device 0
nand erase ${kern_addr} ${buf_sz}
fatload mmc 0 ${ram_buf} uImage
nandecc bch4_sw
```



```
nand write.i ${ram_buf} ${kern_addr} ${buf_sz}

echo "loading rootfs"
echo "  Clearing buffer (This takes a couple minutes and there are no
status updates.)"
mw.b ${ram_buf} 0xff ${buf_sz}
echo "  loading from SD (This takes about 5 minutes and there are no
status updates.)"
fatload mmc 0 ${ram_buf} rootfs.jffs2
setenv filesize 0x00D74000
nand erase 0x00780000 0x1f880000
nandecc bch4_sw
echo "  writing NAND (This takes about 1.5 minutes and there are no
status updates.)"
nand write.i ${ram_buf} 0x00780000 ${filesize}

echo "setting up boot process"
setenv console ttyO2,115200
setenv nandargs  'setenv bootargs mem=128M console=${console}n8
noinitrd root=/dev/mtdblock4 rw rootfstype=jffs2 ip=dhcp
${otherbootargs}; '
setenv nandboot 'echo Booting from nand ...; run nandargs; nand read
0x83000000 280000 500000; bootm 0x83000000'
nandecc hw 1
saveenv
```

Read the above script to understand what it does. Readers should take note of the explicit erasing of the RAM buffer before each copy from MMC. This is necessary to properly program the flash. The offsets and sizes are all congruent with the memory map defined in Section 4.3.1.

3. As mentioned before, the boot script must be wrapped in a header that U-Boot understands.

   NOTE: The angled bracket characters (>) added by bash have been removed to make this easy to copy.

```
host$ $HOME/TI_SDK/UBOOT/am3517/tools/mkimage \
 -A arm \
 -O linux \
 -T script \
 -C none \
 -a 0 \
 -e 0 \
 -n 'my boot script' \
 -d /tftpboot/boot.txt \
 /tftpboot/boot.scr
```

If the script file was properly created, you should see the following output:

```
Image Name:   my boot script
Created:      Tue Jan  11 15:49:18 2011
Image Type:   ARM Linux Script (uncompressed)
Data Size:    913 Bytes = 0.89 kB = 0.00 MB
Load Address: 00000000
Entry Point:  00000000
Contents:
```



```
   Image 0: 905 Bytes = 0.88 kB = 0.00 MB
```

4. You can also verify the type of file created with the *file* command.

```
host$ file /tftpboot/boot.scr

boot.scr: u-boot/PPCBoot image
```

4.6.3.3 References

- *DENX U-Boot and Linux Guide* wiki page[16]
- *U-Boot Memory Commands* wiki article[17]

4.6.4 Populate and Boot SD/MMC Card

This section explains how to boot software from an SD/MMC card on the development kit. The script that will be put on the SD/MMC card in this procedure will erase the entire NAND and program it with bootloaders, a kernel, and a root file system that will run at boot.

IMPORTANT NOTE: This procedure is not intended to be run sequentially with the other sections in Section 4. In order to continue with the other sections after running this procedure, you will need to repeat Section 4.3 to restore the U-Boot environment variables. To state it explicitly, this section will erase the U-Boot environment variables created in Section 4.3.

4.6.4.1 Prerequisites

The procedures in this section require the following items:

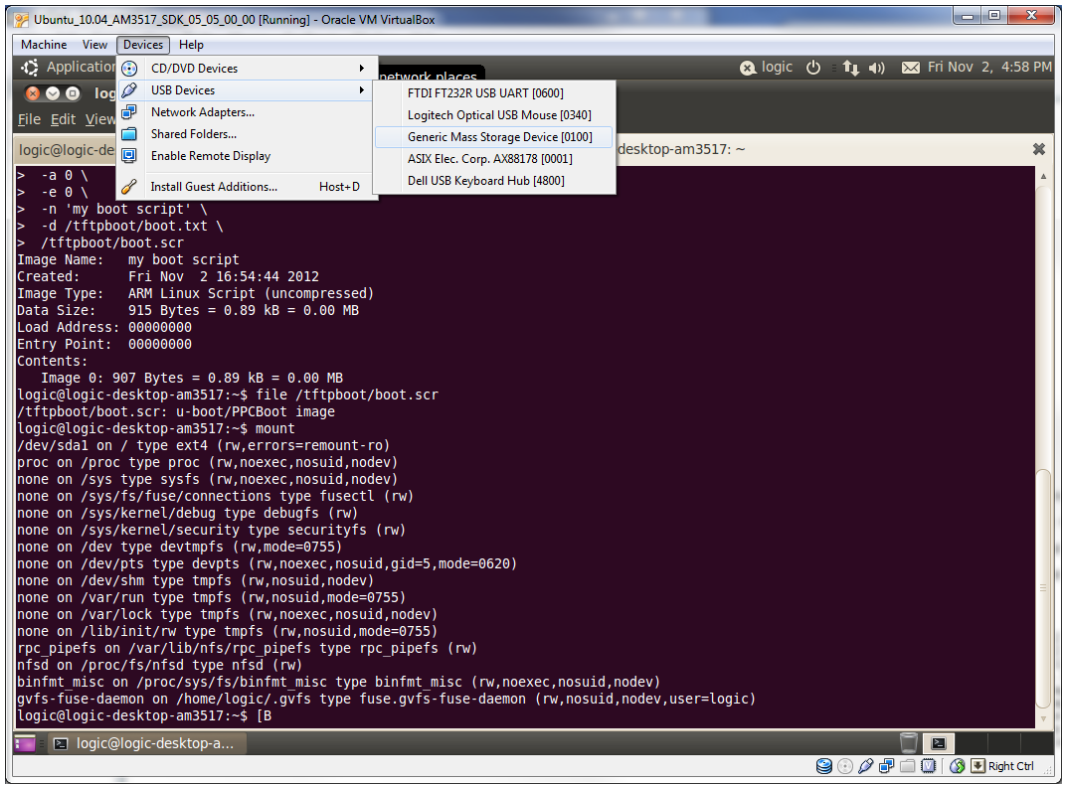
- A properly formatted SD/MMC card as described in Section 4.6.1.
- A compiled and built SPL and U-Boot as described in Section 3.3.
- A compiled and built Linux kernel as described in Section 3.4.
- A JFFS2 root file system image like one that might be found in */tftpboot* after completing the steps in Section 3.5.
- A *boot.scr* script as described in Section 4.6.2.

4.6.4.2 Procedure

1. Insert the SD/MMC card into the host PC and mount it.

[16] http://www.denx.de/wiki/DULG/WebHome
[17] http://www.denx.de/wiki/DULG/UBootCmdGroupMemory

NOTE: If you're using the VM, you must connect the SD/MMC card to the VM. To do this, use the options menu on the VirtualBox window to select Devices > USB Devices and then choose your card. It may only show up as "Generic Mass Storage Device."



2. Determine which device the SD/MMC card is on the host system.

   NOTE: This is a very critical step. If you do not correctly identify your card, you could damage your system. If you are unsure, ask another person proficient in Linux.

3. Check the system log to see which device was created.

```
logic@logic-desktop-am3517:~$ dmesg | tail
[ 3390.476991] sd 4:0:0:0: [sdb] Mode Sense: 03 00 00 00
[ 3390.476993] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 3390.756410] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 3390.756414]  sdb:
[ 3390.801852] sd 4:0:0:3: [sde] Attached SCSI removable disk
[ 3390.810390]  sdb1
[ 3390.861444] sd 4:0:0:1: [sdc] Attached SCSI removable disk
[ 3390.876331] sd 4:0:0:2: [sdd] Attached SCSI removable disk
[ 3390.945056] sd 4:0:0:0: [sdb] Assuming drive cache: write through
[ 3390.945059] sd 4:0:0:0: [sdb] Attached SCSI removable disk
logic@logic-desktop-am3517:~$
```

   In this case, */dev/sdb* is the device that was created with a single partition */dev/sdb1*.

4. Check to see if the card was mounted.

```
logic@logic-desktop-am3517:~$ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
```



```
none on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
none on /dev type devtmpfs (rw,mode=0755)
none on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
none on /dev/shm type tmpfs (rw,nosuid,nodev)
none on /var/run type tmpfs (rw,nosuid,mode=0755)
none on /var/lock type tmpfs (rw,noexec,nosuid,nodev)
none on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
rpc_pipefs on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
nfsd on /proc/fs/nfsd type nfsd (rw)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc
(rw,noexec,nosuid,nodev)
gvfs-fuse-daemon on /home/logic/.gvfs type fuse.gvfs-fuse-daemon
(rw,nosuid,nodev,user=logic)
/dev/sdb1 on /media/F6C6-9D68 type vfat
(rw,nosuid,nodev,uhelper=udisks,uid=1000,gid=1000,shortname=mixed,dmask
=0077,utf8=1,flush)
logic@logic-desktop-am3517:~$
```

As you can see on the last line of the output, the card was mounted as */media/F6C6-9D68.* However, this information is card specific, so pay close attention to the value on your host PC. For the rest of this example, we will refer to this mount point as */media/disk*.

5. Copy the built objects onto the card. As noted before, we want to copy the *MLO* file first to ensure it is listed in the partition's root directory.

   NOTE: Be sure to replace */media/disk* in the command below with the correct mount point for your system that was determined in Step 4 above.

```
host$ sudo cp /tftpboot/MLO /media/disk/.
host$ sudo cp /tftpboot/boot.scr /media/disk/.
host$ sudo cp /tftpboot/u-boot.img /media/disk/.
host$ sudo cp /tftpboot/uImage /media/disk/.
host$ sudo cp /tftpboot/rootfs.jffs2 /media/disk/.
```

6. Synchronize and unmount the card.

```
host$ sync
host$ umount /media/disk
```

7. Remove the SD/MMC card from the host PC and insert it into the powered-off development kit.

8. Set DIP switches S7:1 and S7:4 to the ON position to boot the development kit from SD/MMC first. Make a serial connection to the development kit, power it on, and verify that the commands you inserted into *boot.scr* run.

   If the script doesn't run, you may have a valid U-Boot environment already programmed into flash that is being used. Please review Section 4.3.3.

   TIP: If you have trouble with your development kit booting the SD/MMC card, try formatting it as FAT16 instead of FAT32. This involves changing the partition type in *fdisk* as well as using a different formatting command (*mkfs.**).

Assuming you have been successful, your system should now be ready to run completely from flash-memory. You should script U-Boot to:

- Copy the Linux kernel (*uImage*) from NAND to RAM
- Create a *bootargs* string pointing the kernel to an appropriate MTD block device where it can find its root file system
- Run the kernel using the *bootm* command

For a hint, look at the definition of U-Boot's default *nandboot* environment variable.

4.7 Use MTD-based Root File System

Storing a root file system in flash is relatively easy, so let's work on that now.

NOTE: These instructions are for an AM3517 SOM-M2 with 4-bit ECC NAND. If you are using an older SOM with 1-bit ECC NAND, please contact Logic PD[18] for support.

4.7.1 Prerequisites

- A working knowledge of the U-Boot environment from Section 4.3.
- An understanding of the flash memory map as discussed in Section 4.3.1.
- A JFFS2-formatted root file system like that from Section 3.5.

4.7.2 Procedure

1. Determine the MTD layout of the device.

   This can be done in several ways. The author encourages readers to check all of them to ensure they agree.

   a. Review the source code in *~/TI_SDK/KERNEL/arch/arm/mach-omap2/board-am3517evm.c.* The MTD layout of the file used in this example is shown below for reference.

```
struct mtd_partition am3517evm_nand_partitions[] = {
/* All the partition sizes are listed
 * in terms of NAND block size */
{
   .name        = "xloader-nand",
   .offset      = 0,
   .size        = 4*(SZ_128K),
   .mask_flags  = MTD_WRITEABLE
},
{
   .name        = "uboot-nand",
   .offset      = MTDPART_OFS_APPEND,
   .size        = 14*(SZ_128K),
   .mask_flags  = MTD_WRITEABLE
},
{
   .name        = "params-nand",
   .offset      = MTDPART_OFS_APPEND,
   .size        = 2*(SZ_128K)
},
{
   .name        = "linux-nand",
   .offset      = MTDPART_OFS_APPEND,
```

[18] http://support.logicpd.com/TechnicalSupport/AskAQuestion.aspx


```
   .size        = 40*(SZ_128K)
},
{
   .name        = "jffs2-nand",
   .size        = MTDPART_SIZ_FULL,
   .offset      = MTDPART_OFS_APPEND,
},
};
```

Counting the partitions from the top down and starting at zero, we can determine the area defined as *jffs2-nand* is MTD partition number 4.

b. If you have a Linux kernel already running on your hardware (for example, via NFS), you can review the MTD partition layout using the command below.

```
root@am3517-evm:~# cat /proc/mtd

dev:    size   erasesize  name
mtd0: 00080000 00020000 "xloader-nand"
mtd1: 001c0000 00020000 "uboot-nand"
mtd2: 00040000 00020000 "params-nand"
mtd3: 00500000 00020000 "linux-nand"
mtd4: 1f880000 00020000 "jffs2-nand"
```

c. Finally, you should review the flash layout defined by SPL and U-Boot as discussed in Section 4.3.1, checking the source code as necessary.

All of the above items should agree. If they don't, you are likely going to have problems in the future. In that case, it is likely that you are using components from more than one PSP. If you have verified that you are using the desired PSP components, you must adjust the partition sizes in the source of each component so they all agree.

2. Boot the development kit and stop at the U-Boot prompt.

3. Erase the MTD partition.

```
U-Boot > nand device 0

Device 0: NAND 512MiB 1,8V 16-bit... is now current device
```

4. Determine the amount of RAM you can erase. It is unlikely that the device has more RAM than flash, so we can't just use the same size from the last step.

```
U-Boot > bdinfo

    ...
DRAM bank   = 0x00000000
-> start    = 0x80000000
-> size     = 0x10000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000000
    ...
```


Based on the output above, this hardware has one bank of RAM starting at 0x8000,0000 with a size of 0x1000,0000 (256 MiB).

5. Display the load address.

```
U-Boot > echo ${loadaddr}

0x83000000
```

Do the math:

0x9000,0000 - 0x8300,0000 = 0x0D00,0000

We get a value of 0x0D00,0000; however, there seems to be an issue with the current version of U-Boot that prevents the full area from being accessed. Limit the access to 0x0CF30CFF.

6. Erase the RAM buffer by filling it with 0xff. **NOTE:** This takes about forty seconds. You may consider using a stop watch so you can tell if something actually went wrong, but it is important to be patient. If you get excited and hit **Enter** to check if it's awake, it will repeat the command and will take even more time.

```
U-Boot > mw.b ${loadaddr} 0xff 0x0CF30CFF
```

7. Copy the JFFS2 image onto the device. This step relies on the JFFS2 image that was created in Section 3.5 and will take approximately one minute with the example image.

```
U-Boot > tftp ${loadaddr} ${serverip}:rootfs.jffs2
```

After the transfer completes, the variable *filesize* is updated automatically by U-Boot to contain the size of the transferred image.

8. In order to burn the image properly, you must round the value up to the next 2K page boundary. Print the value, update it, and save it back.

```
U-Boot > printenv filesize
filesize=D73F88

U-Boot > setenv filesize 0x00D74000
```

9. Make sure the proper ECC algorithm is being used.

```
U-Boot > nandecc bch4_sw
```

10. Copy the data into flash.

```
U-Boot > nand erase 0x00780000 0x1f880000

U-Boot > nand write.i ${loadaddr} 0x00780000 ${filesize}
```


11. Load and boot a Linux kernel, passing it the following parameters (among others) on the command line:

    - root=/dev/mtdblock4 rw
    - rootfstype=jffs2

    At the time of writing, the default U-Boot environment supplied by TI came with the variable *nandargs.* This variable properly configures a kernel command line for this situation. For example, you could run the command sequence below.

```
U-Boot > run nandargs
U-Boot > run loadnet
U-Boot > bootm ${loadaddr}
```

    You may notice the first time you boot the device from an MTD-based root file system it takes longer to get to a prompt. This is expected and has to do with the journaling aspects of the JFFS2 file system. The delay will go away with subsequent boots.

IMPORTANT NOTICE: Once your root file system begins to persist in flash, it is important that you gracefully shut down the system whenever possible. You should get in the habit of using the *shutdown* command, as shown below.

```
arago# shutdown -hP -t 40 now
```

4.8 Use MTD-based Kernel

Storing the kernel in flash is the last piece of the puzzle for booting the device entirely from flash.

NOTE: These instructions are for an AM3517 SOM-M2 with 4-bit ECC NAND. If you are using an older SOM with 1-bit ECC NAND, please contact Logic PD for support.

4.8.1 Prerequisites

The procedures in this section require the following items:

- A working knowledge of the U-Boot environment from Section 4.3.
- An understanding of the flash memory map as discussed in Section 4.3.1.
- A kernel like that from Section 3.4.

4.8.2 Procedure

1. Review the MTD layout of the device. Refer to Section 4.7.2 for the location and size of the kernel area in flash.

   For this example we'll use a similar buffer to that of Section 4.6.3.2 for staging the kernel in RAM before loading it into flash. This buffer matches the size of the kernel area in NAND.

2. Create variables for the RAM buffer and size.

```
U-Boot > setenv ram_buf 0x83001000
U-Boot > setenv buf_sz  0x0500000
U-Boot > setenv kern_addr  0x00280000
U-Boot > saveenv
```


3. Erase the RAM buffer.

```
U-Boot > mw.b ${ram_buf} 0xff ${buf_sz}
```

4. Erase the area of NAND where the kernel will be loaded.

```
U-Boot > nand device 0

Device 0: NAND 512MiB 1,8V 16-bit... is now current device

U-Boot > nand erase ${kern_addr} ${buf_sz}

NAND erase: device 0 offset 0x280000, size 0x500000
Erasing at 0x760000 -- 100% complete.
OK
```

5. Load the kernel into the RAM buffer.

   To load from TFTP:

```
U-Boot > tftp ${ram_buf} ${serverip}:uImage
```

   To load from MMC:

```
U-Boot > mmc init

mmc1 is available

U-Boot > fatload mmc 0:1 ${ram_buf} uImage

reading uImage
2473488 bytes read
```

6. Write the RAM buffer to NAND.

```
U-Boot > nandecc bch4_sw

4 BIT SW ECC selected

U-Boot > nand write.i ${ram_buf} ${kern_addr} ${buf_sz}

NAND write: device 0 offset 0x280000, size 0x500000
 5242880 bytes written: OK
```


7. Boot your loaded kernel.

   a. You'll want to review your environment variables *nandargs* and *nandboot*. Especially the *root* and *rootfstype* values. For this example, you may want to set them to the values below.

      IMPORTANT NOTE: Take great care if you copy and paste these lines into your terminal. Missing spaces or accidentally introducing characters will cause the boot to fail. Take a moment to understand each part of these variables.

```
U-Boot > setenv nandargs 'setenv bootargs console=${console}
root=/dev/mtdblock4 rw rootfstype=jffs2 ${otherbootargs};'

U-Boot > setenv nandboot 'echo Booting from nand ...; run nandargs;
nand read ${loadaddr} ${kern_addr} ${buf_sz}; bootm ${loadaddr}'
```

   b. Run the *saveenv* command once they are properly set.

```
U-Boot > saveenv
```

   c. Launch the kernel.

```
U-Boot > run nandboot

Booting from nand ...

NAND read: device 0 offset 0x280000, size 0x400000
 4194304 bytes read: OK
## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-2.6.37
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3028952 Bytes = 2.9 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Linux version 2.6.37 (logic@logic-desktop-am3517)....
```

4.9 Boot Automatically at Power-Up

4.9.1 Prerequisites

The procedures in this section require the following items:

- A working knowledge of the U-Boot environment from Section 4.3.
- The flash must be loaded with SPL, U-Boot, a root file system and a valid kernel, as described in Section 4.4, Section 4.7, and Section 4.8.
- The U-Boot environment variables must be properly set to launch the kernel as described in Section 4.8.2.

4.9.2 Procedure

When U-Boot first boots, the initial commands it runs are stored in an environment variable called *bootcmd*. These commands are only run if the *bootdelay* environment variable is also set to a value greater than 0. For the example below, we will set U-Boot to load and run from flash automatically at power-up.

1. Set the environment variables below.

```
U-Boot > setenv bootcmd 'run nandboot'
U-Boot > setenv bootdelay 5
```

2. Save the environment variables that were just set.

```
U-Boot > saveenv
```

3. Restart the system by pressing the reset button.

```
U-Boot SPL 2011.09 (Aug 14 2012 - 16:54:37)
Texas Instruments Revision detection unimplemented


U-Boot 2011.09 (Aug 14 2012 - 16:54:37)

AM35XX-GP ES2.0, CPU-OPP2, L3-165MHz, Max CPU Clock 600 Mhz
AM3517EVM Board + LPDDR/NAND
I2C:   ready
DRAM:  256 MiB
WARNING: Caches not enabled
NAND:  HW ECC [Kernel/FS layout] selected
512 MiB
MMC:   OMAP SD/MMC: 0
In:    serial
Out:   serial
Err:   serial
Die ID #33d20001000000001685a201600f00f
Net:   Ethernet PHY: GENERIC @ 0x00
DaVinci-EMACHit any key to stop autoboot:  5 4 3 2 1 0
Booting from nand ...
```


5 Software Development

This section covers a few simple examples to help the reader quickly develop software for their hardware.

5.1 Build and Debug User Application

During this section we will build, deploy, and debug a simple user application. Because we have chosen to build the application standalone, we have some housekeeping to do regarding system libraries. Specifically, the C-libraries that our application links to won't be exactly the same as the C-libraries that were generated when our root file system was created. Luckily, the CodeSourcery tools come with several *sysroot* options and instructions about how to install them onto our device.

5.1.1 Prerequisites

The procedures in this section require the following items:

- A project workspace as described in Section 3.1.
- A connection to a development kit running Linux with its root file system mounted via NFS as described in Section 4.5.
- A familiarity with "Chapter 3: Sourcery G++ Lite for ARM GNU/Linux" of the *Sourcery G++ Lite Getting Started Guide*.[19]
- The full-featured *tisdk-rootfs-am3517-evm* root file system as described in Section 3.4.3.

5.1.2 Procedure

NOTE: As throughout this guide, the author's NFS mount point is */home/logic/TI_SDK/targetNFS*.

NOTE: The output from your development kit will almost certainly differ from the output in the example below. Throughout this section, we will shorten the prompt to host$ for clarity.

1. Start out in the root of our project directory tree.

```
host$ cd
host$ cd TI_SDK
```

2. Create a directory for our sample application in the *Projects* directory.

```
host$ mkdir Projects
host$ mkdir Projects\Hello
host$ cd Projects\Hello
```

3. Create shortcuts to the NFS source and *Projects* directories:

```
host$ export NFSE=/home/logic/TI_SDK/targetNFS
host$ export PROJECTS=/home/logic/TI_SDK/Projects
```

[19] https://sourcery.mentor.com/sgpp/lite/arm/portal/doc4337

4. We need to pass some special options to *arm-arago-linux-gnueabi-gcc* so the linker on our target will know which *sysroot* to use. This will be easier if we create a simple Makefilefor our project.

   Using your favorite editor, create the file Makefile and add the lines shown below.

```
CC=arm-arago-linux-gnueabi-gcc
CFLAGS+=-Wl,--rpath=/lib:/usr/lib
CFLAGS+=-Wl,--dynamic-linker=/lib/ld-linux.so.3
CFLAGS+=-g -ggdb

hello: hello.c
	$(CC) $(CFLAGS) -o hello hello.c
```

5. Create a simple user program to play around with. Using your favorite editor, create the file *hello.c* and enter the source code shown below.

```
#include <stdio.h>

int
main(int argc, char **argv)
{
	int i;
	printf("Hello, world\n");
	printf("cmd line:\n");
	for (i = 0; i < argc; ++i)
		printf("\t %i : %s\n", i, argv[i]);

	return 0;
}
```

6. Finally, there are a few things that we need to tell GDB so it can find the proper system libraries based on the way that we will build our program. It is easier to add these to a .gdbinit file so we don't have to enter them each time we debug the program.

   Using your favorite editor, create the *$PROJECTS/Hello/.gdbinit* file and add the lines shown below.

```
# This corresponds to the directory on the device where we copied
# the "sysroot" to.
#echo set sysroot on target \n
#set sysroot-on-target /sourcery

# This is the directory on our host machine where the cross
# compilation tools are installed.  Specifically, it is the
# "sysroot" of the proper multi-lib.
echo set sysroot on host \n
set sysroot /FULLPATH/arm-2009q1/arm-arago-linux-gnueabi/libc

# It's always convenient to start with a breakpoint at the
# entry to our program.
echo setting a breakpoint at main() \n
b main

echo use <target remote ip:port> to connect \n
```


IMPORTANT NOTE: GDB cannot access variables defined in the calling shell. Thus, *$TOOLS* cannot be used in the definition of *sysroot* above. Therefore, you *must* enter the fully-qualified pathname in this instance.

NOTE: The *echo* commands shown in the .gdbinit file above are not necessary. They are just there to remind you that this script is being executed each time you start GDB in the *$PROJECTS/Hello* directory.

7. Build the sample program.

```
host$ make hello
```

8. Copy the program to a place where our device can see it.

```
host$ sudo cp hello $NFSE/home/root/.
```

9. Now, on the target make the program executable.

```
root@am3517-evm:~# sudo chmod +x /home/root/hello
```

10. Try running the program on the device to make sure that it works.

```
root@am3517-evm:~#  /home/root/hello foo bar

Hello, world
cmd line:
         0 : /home/root/hello
         1 : foo
         2 : bar
```

11. Run the program again, but this time under the control of *gdbserver*. Please note, since we linked our program under a separate *sysroot*, we need to invoke *gdbserver* using the dynamic linker in the same *sysroot*.

```
root@am3517-evm:~#  /lib/ld-linux.so.3 \
> --library-path /lib:/usr/lib \
> /usr/bin/gdbserver :10000 /home/root/hello foo

Process /home/share/hello created; pid = xxx
Listening on port 10000
```

TIP: Each time you restart the *hello* program using *gdbserver*, you will need to include the specific call to the appropriate linker: */sourcery/lib/ld-linux.so.3*. If you try and rerun the last command using BASH history (up arrow), you will get errors.

12. Launch the GNU Debugger on the host PC. Note, this is done from the *${PROJECTS}/Hello* directory.

```
host$ arm-arago-linux-gnueabi-gdb hello
```


13. Tell GDB to connect to *gdbserver* running on our remote target. Please fill in the appropriate IP address for your system.

```
(gdb) target remote 192.0.0.201:10000
```

On the terminal connected to your device, you should see a message from *gdbserver* that a remote connection has been made.

```
Remote debugging from host 192.0.0.200
```

14. Continue running the program until we hit our breakpoint.

```
(gdb) continue


Continuing.

Breakpoint 1, main (argc=3, argv=0xbed03e34) at hello.c:7
7               printf("Hello, world\n");
```

15. Look at the source code around the current break point.

```
(gdb) list

2
3       int
4       main(int argc, char **argv)
5       {
6               int i;
7               printf("Hello, world\n");
8               printf("cmd line:\n");
9               for (i = 0; i < argc; ++i)
10                      printf("\t %i : %s\n", i, argv[i]);
11
```

16. Look at the value of some of the program's variables.

```
(gdb) print argc

$1 = 3

(gdb) print *argv

$2 = 0xbed03f18 "/home/share/hello"

(gdb) print argv[1]

$3 = 0xbed03f2a "foo"
```

17. Examine the local variables.

```
(gdb) info locals
```



```
i = 1073892864
```

18. Step through a few lines of the program statement-by-statement.

```
(gdb) next

8               printf("cmd line:\n");

(gdb) next

9       for (i = 0; i < argc; ++i )

(gdb) next

10              printf("\t %i : %s\n", i, argv[i]);
```

19. Examine the variable *i* now that it has been initialized.

```
(gdb) print i

$4 = 0
```

20. Examine where we are and how we got here.

```
(gdb) backtrace

#0  main (argc=3, argv=0xbed03e34) at hello.c:10
```

21. Continue the program and let it exit.

```
(gdb) continue

Continuing.

Program exited normally.
```

You should see the program output on the device's terminal as well as *gdbserver* displaying the program's exit code.

```
Hello, world
cmd line:
         0 : /home/share/hello
         1 : foo

Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
```


22. Exit GDB.

```
(gdb) quit
```

5.1.3 References

- *Sourcery G++ Lite Getting Started Guide* (Chapter 3)[20]
- GDB Documentation[21]
- GDB Debugger Tutorial[22]
- GDB Debugging Under Unix Tutorial[23]

5.2 Debug the Linux Kernel Using KGDB

This section will be updated in the next release of the document when the compatibility issues with the current SDK have been resolved.

[20] https://sourcery.mentor.com/sgpp/lite/arm/portal/doc4337
[21] http://www.gnu.org/software/gdb/documentation/
[22] http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html
[23] http://www.cs.cmu.edu/~gilpin/tutorial/

6 Using Wi-Fi

The AM3517 SOM-M2 includes the WL1271L Wi-Fi radio. At the time this document was written, TI didn't provide support for this radio in its standard SDK. This section details the process for adding the additional components to the kernel and file system to set up and test wireless network access on the AM3517 SOM-M2.

6.1 Prerequisites

The procedures in this section require the following items:

- Completion of Section 3.
- Completion of Sections 4.1 through 4.4 to prepare the development kit for use.
- Completion of Section 4.5 to try the wireless files on the kit via NFS.
- Access to the AM3517 WLS1271L Wi-Fi Addition Files[24] from the Logic PD website.

6.2 Procedure

The following procedure is broken down into subsections to make the process more clear. Due to the large number of sections and steps, it may be hard to identify where a problem originates. If there is any doubt about the output after any of the steps below, the user can verify that the step was successful by checking the return code with the following command:

```
logic@logic-desktop-am3517:~$ echo $?
0
logic@logic-desktop-am3517:~$
```

If the return code is 0, as shown above, then the step was successful.

NOTE: Due to size restrictions of the NAND on the AM3517 SOM-M2, this example uses the *base-rootfs-am3517-evm.tar.gz* file system from the SDK.

NOTE: There is a known issue with the SD/MMC timing that is related to the use of NFS. For this document, the initial setup is done with NFS and then the build output is programmed into NAND. If you experience intermittent issues with kernel opps/panics, try your project booting from NAND.

NOTE: Due to a known issue, WEP security is not currently supported. The following example demonstrates WPA security that overcomes the security vulnerabilities of WEP.

6.2.1 Download Wireless Update

1. Download the AM3517 WLS1271L Wi-Fi Addition Files from the Logic PD website to the *Downloads* directory.
2. Switch to the *Downloads* directory and unpack the archive.

```
logic@logic-desktop-am3517:~$ cd ~/Downloads
logic@logic-desktop-am3517:~/Downloads$ ls
1023690A_TI_05_05_00_00_PSP_AM3517_WL1271L_Addition.tar.gz temp
logic@logic-desktop-am3517:~/Downloads$ tar xf
1023690A_TI_05_05_00_00_PSP_AM3517_WL1271L_Addition.tar.gz
logic@logic-desktop-am3517:~/Downloads$
```

[24] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1440

3. Set up a symbolic link to shorten the paths.

```
logic@logic-desktop-am3517:~/Downloads $ cd
logic@logic-desktop-am3517:~$ ln -s
Downloads/TI_05_05_00_00_PSP_AM3517_WL1271L_Addition
logic@logic-desktop-am3517:~$ cd WIRELESS
logic@logic-desktop-am3517:~/WIRELESS$ ls
helpfulFiles  libnl  openssl  patches  WL1271L_modules  wpa_supplicant
logic@logic-desktop-am3517:~/WIRELESS$
```

6.2.2 Switch NFS Home to Minimal Root File System

1. Switch to the *targetNFS* directory.

```
logic@logic-desktop-am3517:~/Downloads $ cd $HOME/TI_SDK/targetNFS
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

2. Back up the current *targetNFS* directory. This may take several minutes depending on the speed of your host PC.

```
logic@am3517-vm:~/am3517-sdk/targetNFS$ sudo tar caf
../backupTargetNFS.tar.gz .
[sudo] password for logic:
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

3. Remove the current root file system in the *targetNFS* directory.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ sudo rm -rf *
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

4. Unpack the minimal root file system image.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ sudo tar xf
../filesystem/base-rootfs-am3517-evm.tar.gz
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

5. Verify the contents of the *targetNFS* directory.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ ls
bin  boot  dev  etc  home  lib  linuxrc  media  mnt  proc  sbin  srv  sys  tmp
usr  var
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ du -sh
36M    .
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

6.2.3 Apply Root File System Patch

1. Check the patch for errors.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ sudo git apply --check
$HOME/WIRELESS/patches/rootfs/ti-sdk-am3517-evm-05.05.00.00_rootfs.diff
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```


2. Apply the patch (whitespace errors are acceptable).

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ sudo git apply
$HOME/WIRELESS/patches/rootfs/ti-sdk-am3517-evm-05.05.00.00_rootfs.diff
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/rootfs/ti-sdk-
am3517-evm-05.05.00.00_rootfs.diff:28: trailing whitespace.
then
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/rootfs/ti-sdk-
am3517-evm-05.05.00.00_rootfs.diff:108: trailing whitespace.
then
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/rootfs/ti-sdk-
am3517-evm-05.05.00.00_rootfs.diff:290: trailing whitespace.
then
warning: 3 lines add whitespace errors.
logic@logic-desktop-am3517:~/ti-sdk-am3517-evm-05.05.00.00/targetNFS$
```

6.2.4 Apply Kernel Patch

1. Switch to the *kernel* directory.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ cd $HOME/TI_SDK/KERNEL
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$
```

2. Check the patch for errors.

```
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$ git apply --check
$HOME/WIRELESS/patches/kernel/linux-2.6.37-
psp04.02.00.07.sdk_WL1271.diff
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$
```

3. Apply the patch (whitespace errors are acceptable).

```
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$ git apply
$HOME/WIRELESS/patches/kernel/linux-2.6.37-
psp04.02.00.07.sdk_WL1271.diff
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/kernel/linux-
2.6.37-psp04.02.00.07.sdk_WL1271.diff:856: space before tab in indent.
            printk(KERN_WARNING "Failed to gpio_request %d for
wlan_irq\n", OMAP_AM3517EVM_WIFI_IRQ_GPIO);
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/kernel/linux-
2.6.37-psp04.02.00.07.sdk_WL1271.diff:993: trailing whitespace.
#undef ID_KEY_END
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/kernel/linux-
2.6.37-psp04.02.00.07.sdk_WL1271.diff:1203: trailing whitespace.
struct __attribute__ ((packed)) id_header {
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/kernel/linux-
2.6.37-psp04.02.00.07.sdk_WL1271.diff:1210: trailing whitespace.
struct __attribute__ ((packed)) id_checksums {
/home/logic/Downloads/AM3517_WL1271L_Addition/patches/kernel/linux-
2.6.37-psp04.02.00.07.sdk_WL1271.diff:1281: trailing whitespace.

warning: squelched 18 whitespace errors
warning: 23 lines add whitespace errors.
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$
```


6.2.5 Build Kernel

1. Apply the environment setup script.

```
logic@logic-desktop-am3517:~/TI_SDK/KERNEL$ source ../../../ti-sdk-
am3517-evm-05.05.00.00/linux-devkit/environment-setup
[linux-devkit]:~/TI_SDK/KERNEL>
```

2. Clean the build.

```
[linux-devkit]:~/TI_SDK/KERNEL> make CROSS_COMPILE=arm-arago-linux-
gnueabi- ARCH=arm mrproper
  CLEAN   .
  CLEAN   arch/arm/kernel
 …
  CLEAN   include/config include/generated
  CLEAN   .config .config.old .version include/linux/version.h
Module.symvers
[linux-devkit]:~/TI_SDK/KERNEL>
```

3. Apply the new Wi-Fi kernel configuration.

```
[linux-devkit]:~/TI_SDK/KERNEL> make ARCH=arm CROSS_COMPILE=arm-arago-
linux-gnueabi- am3517_evm_WIFI_defconfig
  HOSTCC  scripts/basic/fixdep
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/stdlib.h:955,
                 from scripts/basic/fixdep.c:112:
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/stdlib.h:65: warning: no previous prototype
for ‘ptsname_r’
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/sys/socket.h:40,
                 from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/netinet/in.h:25,
                 from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/arpa/inet.h:23,
                 from scripts/basic/fixdep.c:116:
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/socket.h:427: warning: ‘struct mmsghdr’
declared inside parameter list
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/socket.h:427: warning: its scope is only this
definition or declaration, which is probably not what you want
  HOSTCC  scripts/basic/docproc
  HOSTCC  scripts/kconfig/conf.o
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/stdlib.h:955,
                 from scripts/kconfig/conf.c:9:
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/stdlib.h:65: warning: no previous prototype
for ‘ptsname_r’
  HOSTCC  scripts/kconfig/kxgettext.o
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/stdlib.h:955,
                 from scripts/kconfig/kxgettext.c:7:
```



```
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/stdlib.h:65: warning: no previous prototype
for ‘ptsname_r’
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/lex.zconf.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/usr/include/stdlib.h:955,
                 from scripts/kconfig/zconf.tab.c:87:
/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-
gnueabi/usr/include/bits/stdlib.h:65: warning: no previous prototype
for ‘ptsname_r’
  HOSTLD  scripts/kconfig/conf
arch/arm/configs/am3517_evm_WIFI_defconfig:634:warning: override:
reassigning to symbol CRYPTO_DES
arch/arm/configs/am3517_evm_WIFI_defconfig:635:warning: override:
reassigning to symbol CRYPTO_TWOFISH
arch/arm/configs/am3517_evm_WIFI_defconfig:636:warning: override:
reassigning to symbol CRYPTO_TWOFISH_COMMON
arch/arm/configs/am3517_evm_WIFI_defconfig:637:warning: override:
reassigning to symbol CRYPTO_DEFLATE
arch/arm/configs/am3517_evm_WIFI_defconfig:638:warning: override:
reassigning to symbol CRYPTO_LZO
#
# configuration written to .config
#
[linux-devkit]:~/TI_SDK/KERNEL>
```

4. Build the kernel.

```
[linux-devkit]:~/TI_SDK/KERNEL> make ARCH=arm CROSS_COMPILE=arm-arago-
linux-gnueabi- uImage
scripts/kconfig/conf --silentoldconfig Kconfig
  CHK     include/linux/version.h
  UPD     include/linux/version.h
  CHK     include/generated/utsrelease.h
  UPD     include/generated/utsrelease.h
  Generating include/generated/mach-types.h
…
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-2.6.37
Created:      Wed Oct 31 12:38:42 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3067552 Bytes = 2995.66 kB = 2.93 MB
Load Address: 0x80008000
Entry Point:  0x80008000
  Image arch/arm/boot/uImage is ready
[linux-devkit]:~/TI_SDK/KERNEL>
```

5. Copy the output to the TFTP server directory.

```
[linux-devkit]:~/TI_SDK/KERNEL> cp arch/arm/boot/uImage /tftpboot/.
[linux-devkit]:~/TI_SDK/KERNEL>
```


6.2.6 Build and Install Kernel Wireless Modules

1. Switch to the kernel modules directory.

```
[linux-devkit]:~/TI_SDK/KERNEL> cd $HOME/WIRELESS/WL1271L_modules
[linux-devkit]:~/WIRELESS/WL1271L_modules>
```

2. Clean the build. **NOTE:** If you have not built the wireless modules yet, you do not need to perform this step.

```
[linux-devkit]:~/WIRELESS/WL1271L_modules> make
DEST=/home/logic/TI_SDK/targetNFS KERNEL_LOC=/home/logic/TI_SDK/KERNEL
wl12xx_compat_clean
make -C logicpd-generic/hardware/ti/wlan/WL1271_compat/drivers ARCH=arm
CROSS_COMPILE=/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/bin/arm-arago-linux-gnueabi- \
            KLIB=/home/logic/TI_SDK/KERNEL
KLIB_BUILD=/home/logic/TI_SDK/KERNEL clean
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/WL1271L_modules/logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers'
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/WL1271L_modules/logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers'
sudo rm
/home/logic/TI_SDK/targetNFS/lib/modules/2.6.37/updates/drivers/net/wir
eless/wl12xx/wl12xx.ko
sudo rm
/home/logic/TI_SDK/targetNFS/lib/modules/2.6.37/updates/drivers/net/wir
eless/wl12xx/wl12xx_sdio.ko
sudo rm
/home/logic/TI_SDK/targetNFS/lib/modules/2.6.37/updates/net/mac80211/ma
c80211.ko
sudo rm
/home/logic/TI_SDK/targetNFS/lib/modules/2.6.37/updates/net/wireless/cf
g80211.ko
sudo rm
/home/logic/TI_SDK/targetNFS/lib/modules/2.6.37/updates/compat/compat.k
o
#sudo rm logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/drivers/net/wireless/wl1
2xx/wl12xx.ko
#sudo rm logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/drivers/net/wireless/wl1
2xx/wl12xx_sdio.ko
#sudo rm logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/net/mac80211/mac80211.ko
#sudo rm logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/net/wireless/cfg80211.ko
#sudo rm logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/compat/compat.ko
[linux-devkit]:~/WIRELESS/WL1271L_modules>
```


3. Build/install the kernel wireless modules.

```
[linux-devkit]:~/WIRELESS/WL1271L_modules> make
DEST=/home/logic/TI_SDK/targetNFS \
KERNEL_LOC=/home/logic/TI_SDK/KERNEL wl12xx_compat
echo /home/logic/ti-sdk-am3517-evm-05.05.00.00/board-support/linux-
2.6.37-psp04.02.00.07.sdk
/home/logic/ti-sdk-am3517-evm-05.05.00.00/board-support/linux-2.6.37-
psp04.02.00.07.sdk
make -C logicpd-generic/hardware/ti/wlan/WL1271_compat/drivers ARCH=arm
CROSS_COMPILE=/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/bin/arm-arago-linux-gnueabi- \
            KLIB=/home/logic/ti-sdk-am3517-evm-05.05.00.00/board-
support/linux-2.6.37-psp04.02.00.07.sdk KLIB_BUILD=/home/logic/ti-sdk-
am3517-evm-05.05.00.00/board-support/linux-2.6.37-psp04.02.00.07.sdk
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/WL1271L_modules/logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers'
echo KLIB_BUILD is /home/logic/ti-sdk-am3517-evm-05.05.00.00/board-
support/linux-2.6.37-psp04.02.00.07.sdk
KLIB_BUILD is /home/logic/ti-sdk-am3517-evm-05.05.00.00/board-
support/linux-2.6.37-psp04.02.00.07.sdk
…
make[2]: Leaving directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/board-support/linux-2.6.37-psp04.02.00.07.sdk'
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/WL1271L_modules/logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers'
sudo mkdir -p /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/drivers/net/wireless/
wl12xx
sudo mkdir -p /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/net/mac80211
sudo mkdir -p /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/net/wireless
sudo mkdir -p /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/compat
sudo cp logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/drivers/net/wireless/wl1
2xx/wl12xx.ko /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/drivers/net/wireless/
wl12xx/.
sudo cp logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/drivers/net/wireless/wl1
2xx/wl12xx_sdio.ko /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/drivers/net/wireless/
wl12xx/.
sudo cp logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/net/mac80211/mac80211.ko
/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/net/mac80211/.
sudo cp logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/net/wireless/cfg80211.ko
/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/net/wireless/.
sudo cp logicpd-
generic/hardware/ti/wlan/WL1271_compat/drivers/compat/compat.ko
```



```
/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS//lib/modules/2.6.37/updates/compat/.
[linux-devkit]:~/WIRELESS/WL1271L_modules>
```

6.2.7 Build and Install Netlink

1. Switch to the Netlink directory.

```
[linux-devkit]:~/WIRELESS/WL1271L_modules> cd ../libnl
[linux-devkit]:~/WIRELESS/libnl>
```

2. Extract the files and switch to the project directory.

```
[linux-devkit]:~/WIRELESS/libnl> tar xf libnl-2.0.tar.gz
[linux-devkit]:~/WIRELESS/libnl> cd libnl-2.0/
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

3. Add the link to the patch folder.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> ln -s ../../patches/libnl/
patches
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

4. Apply the patches.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> quilt -a push
File series fully applied, ends at patch libnl-2.0-relink.patch
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

5. Configure the build. The output is not abbreviated here.

   NOTE: If you have problems running the Wi-Fi connection script later, check the output here for differences.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> ./configure --prefix=/ --
host=arm-linux --build=i686-pc-linux-gnu
configure: loading site script /home/logic/ti-sdk-am3517-evm-
05.05.00.00/linux-devkit/site-config
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for arm-linux-strip... arm-arago-linux-gnueabi-strip
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for arm-linux-gcc... arm-arago-linux-gnueabi-gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... yes
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether arm-arago-linux-gnueabi-gcc accepts -g... yes
```



```
checking for arm-arago-linux-gnueabi-gcc option to accept ISO C89... 
none needed
checking for style of include used by make... GNU
checking dependency style of arm-arago-linux-gnueabi-gcc... gcc3
checking whether arm-arago-linux-gnueabi-gcc and cc understand -c and -
o together... yes
checking build system type... i686-pc-linux-gnu
checking host system type... arm-unknown-linux-gnu
checking for a sed that does not truncate output... /bin/sed
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for fgrep... /bin/grep -F
checking for ld used by arm-arago-linux-gnueabi-gcc... /home/logic/ti-
sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-linux-gnueabi/bin/ld
checking if the linker (/home/logic/ti-sdk-am3517-evm-
05.05.00.00/linux-devkit/arm-arago-linux-gnueabi/bin/ld) is GNU ld... 
yes
checking for BSD- or MS-compatible name lister (nm)... arm-arago-linux-
gnueabi-nm
checking the name lister (arm-arago-linux-gnueabi-nm) interface... BSD 
nm
checking whether ln -s works... yes
checking the maximum length of command line arguments... 1572864
checking whether the shell understands some XSI constructs... yes
checking whether the shell understands "+="... yes
checking for /home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-
devkit/arm-arago-linux-gnueabi/bin/ld option to reload object files... 
-r
checking for arm-linux-objdump... arm-arago-linux-gnueabi-objdump
checking how to recognize dependent libraries... pass_all
checking for arm-linux-ar... arm-arago-linux-gnueabi-ar
checking for arm-linux-strip... (cached) arm-arago-linux-gnueabi-strip
checking for arm-linux-ranlib... arm-arago-linux-gnueabi-ranlib
checking command to parse arm-arago-linux-gnueabi-nm output from arm-
arago-linux-gnueabi-gcc object... ok
checking how to run the C preprocessor... arm-arago-linux-gnueabi-gcc -
E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for dlfcn.h... yes
checking for objdir... .libs
checking if arm-arago-linux-gnueabi-gcc supports -fno-rtti -fno-
exceptions... no
checking for arm-arago-linux-gnueabi-gcc option to produce PIC... -fPIC
-DPIC
checking if arm-arago-linux-gnueabi-gcc PIC flag -fPIC -DPIC works... 
yes
checking if arm-arago-linux-gnueabi-gcc static flag -static works... 
yes
```



```
checking if arm-arago-linux-gnueabi-gcc supports -c -o file.o... yes
checking if arm-arago-linux-gnueabi-gcc supports -c -o file.o...
(cached) yes
checking whether the arm-arago-linux-gnueabi-gcc linker
(/home/logic/ti-sdk-am3517-evm-05.05.00.00/linux-devkit/arm-arago-
linux-gnueabi/bin/ld) supports shared libraries... yes
checking whether -lc should be explicitly linked in... no
checking dynamic linker characteristics... GNU/Linux ld.so
checking how to hardcode library paths into programs... immediate
checking whether stripping libraries is possible... yes
checking if libtool supports shared libraries... yes
checking whether to build shared libraries... yes
checking whether to build static libraries... yes
checking for flex... flex
checking lex output file root... lex.yy
checking lex library... -lfl
checking whether yytext is a pointer... yes
checking for bison... bison -y
checking for an ANSI C-conforming const... yes
checking for inline... inline
checking for pow in -lm... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating doc/Doxyfile
config.status: creating doc/Makefile
config.status: creating lib/Makefile
config.status: creating include/Makefile
config.status: creating src/Makefile
config.status: creating src/lib/Makefile
config.status: creating libnl-2.0.pc
config.status: creating include/netlink/version.h
config.status: creating lib/defs.h
config.status: executing depfiles commands
config.status: executing libtool commands
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

6. Build Netlink.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> make CFLAGS+="-
DYY_BUF_SIZE=16384"
Making all in include
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/include'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/include'
Making all in lib
…
make[2]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/src'
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/src'
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0'
make[1]: Nothing to be done for `all-am'.
```



```
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0'
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

7. Install Netlink.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> sudo env PATH=$PATH make
install DESTDIR=/home/logic/TI_SDK/targetNFS
Making install in include
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/include'
make[2]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0/include'
…
make[2]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0'
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/libnl/libnl-2.0'
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0>
```

6.2.8 Build and Install OpenSSL

1. Switch to the *OpenSSL* directory.

```
[linux-devkit]:~/WIRELESS/libnl/libnl-2.0> cd ../../openssl
[linux-devkit]:~/WIRELESS/openssl>
```

2. Extract the files and switch to the project directory.

```
[linux-devkit]:~/WIRELESS/openssl> tar xf openssl-1.0.0d.tar.gz
[linux-devkit]:~/WIRELESS/openssl> cd openssl-1.0.0d/
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d>
```

3. Configure the build.

   NOTE: If you have problems running the Wi-Fi connection script later, check the output here for differences.

```
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d> ./Configure linux-
generic32 --prefix=/home/logic/TI_SDK/targetNFS shared no-asm -DL_ENDIA
Configuring for linux-generic32
    no-asm          [option]   OPENSSL_NO_ASM
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-jpake        [experimental] OPENSSL_NO_JPAKE (skip dir)
    no-krb5         [krb5-flavor not specified] OPENSSL_NO_KRB5
    no-md2          [default]  OPENSSL_NO_MD2 (skip dir)
    no-rc5          [default]  OPENSSL_NO_RC5 (skip dir)
    no-rfc3779      [default]  OPENSSL_NO_RFC3779 (skip dir)
    no-store        [experimental] OPENSSL_NO_STORE (skip dir)
    no-zlib         [default]
    no-zlib-dynamic [default]
IsMK1MF=0
CC            =arm-arago-linux-gnueabi-gcc
```



```
CFLAG         =-fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTRANT -
DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIA -DTERMIO -O3 -fomit-frame-pointer -
Wall
EX_LIBS       =-ldl
CPUID_OBJ     =mem_clr.o
BN_ASM        =bn_asm.o
DES_ENC       =des_enc.o fcrypt_b.o
AES_ENC       =aes_core.o aes_cbc.o
BF_ENC        =bf_enc.o
CAST_ENC      =c_enc.o
RC4_ENC       =rc4_enc.o rc4_skey.o
RC5_ENC       =rc5_enc.o
MD5_OBJ_ASM   =
SHA1_OBJ_ASM  =
RMD160_OBJ_ASM=
CMLL_ENC=     =camellia.o cmll_misc.o cmll_cbc.o
PROCESSOR     =
RANLIB        =arm-arago-linux-gnueabi-ranlib
ARFLAGS       =
PERL          =/usr/bin/perl
THIRTY_TWO_BIT mode
DES_UNROLL used
DES_INT used
BN_LLONG mode
RC4 uses uchar
RC4_CHUNK is unsigned long
BF_PTR used
e_os2.h => include/openssl/e_os2.h
making links in crypto...
…
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/openssl/openssl-
1.0.0d/test'
make[1]: Nothing to be done for `generate'.
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/openssl/openssl-
1.0.0d/test'

Configured for linux-generic32.
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d>
```

4. Build OpenSSL.

```
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d> make -j1
making all in crypto...
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/openssl/openssl-
1.0.0d/crypto'
…
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/openssl/openssl-
1.0.0d/tools'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/openssl/openssl-
1.0.0d/tools'
```



```
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d>
```

5. Install OpenSSL.

```
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d> sudo env PATH=$PATH
make install_sw
created directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/engines'
created directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/include/openssl'
created directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/ssl'
created directory `/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/ssl/misc'
…
cp libcrypto.pc /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig
chmod 644 /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig/libcrypto.pc
cp libssl.pc /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig
chmod 644 /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig/libssl.pc
cp openssl.pc /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig
chmod 644 /home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS/lib/pkgconfig/openssl.pc
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d>
```

6.2.9 Build and Install WPA Supplicant

1. Switch to the *WPA Supplicant* directory.

```
[linux-devkit]:~/WIRELESS/openssl/openssl-1.0.0d> cd
../../wpa_supplicant
[linux-devkit]:~/WIRELESS/wpa_supplicant>
```

2. Extract the files and switch to the project directory.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant> tar xf wpa_supplicant-
0.7.3.tar.gz
[linux-devkit]:~/WIRELESS/wpa_supplicant> cd wpa_supplicant-0.7.3
```

3. Move the patch files from *wpa_supplicant-0.7.3.tar.gz* aside, as they will be not be used.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3> mv
patches unusedPatches
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3>
```


4. Add the link to the patch folder.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3> ln -s
../../patches/wpa_supplicant/ patches
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3>
```

5. Apply the patches.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3> quilt -a
push
File series fully applied, ends at patch wpa_supplicant-0.7.3-03-
libnl.patch
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3>
```

6. Switch to the sub-project.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-0.7.3> cd
wpa_supplicant
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant>
```

7. Clean the build.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant> make clean
make -C ../src clean
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src'
…
make[2]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src/wps'
rm -f *~
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src'
rm -f core *~ *.o *.d eap_*.so wpa_supplicant wpa_passphrase wpa_cli
wpa_supplicant.exe wpa_cli.exe wpa_passphrase.exe win_if_list.exe
eapol_test preauth_test
rm -f wpa_priv
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant>
```

8. Build the WPA Supplicant.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant> make -j1 V=1 ARCH=arm  CROSS_COMPILE=
arm-arago-linux-gnueabi-gcc -c -o config.o -MMD -O2 -Wall -g -I../src -
I../src/utils -D_GNU_SOURCE -I//usr/include/libnl-tiny -
DCONFIG_BACKEND_FILE -DCONFIG_PEERKEY -DCONFIG_DRIVER_WEXT -
DCONFIG_DRIVER_ATMEL -DCONFIG_WIRELESS_EXTENSION -DCONFIG_DRIVER_HOSTAP
-DCONFIG_DRIVER_WIRED -DCONFIG_DRIVER_NL80211 -DCONFIG_LIBNL20 -
```



```
DEAP_TLS -DEAP_PEAP -DEAP_TTLS -DEAP_MD5 -DEAP_MSCHAPv2 -DEAP_GTC -
DEAP_OTP -DEAP_LEAP -DIEEE8021X_EAPOL -DPKCS12_FUNCS -DCONFIG_SMARTCARD
-DEAP_TLS_OPENSSL -DCONFIG_CTRL_IFACE -DCONFIG_CTRL_IFACE_UNIX  -
DCONFIG_SME config.c
…
arm-arago-linux-gnueabi-gcc -c -o ../src/common/wpa_ctrl.o -MMD -O2 -
Wall -g -I../src -I../src/utils -D_GNU_SOURCE -I//usr/include/libnl-
tiny -DCONFIG_BACKEND_FILE -DCONFIG_PEERKEY -DCONFIG_DRIVER_WEXT -
DCONFIG_DRIVER_ATMEL -DCONFIG_WIRELESS_EXTENSION -DCONFIG_DRIVER_HOSTAP
-DCONFIG_DRIVER_WIRED -DCONFIG_DRIVER_NL80211 -DCONFIG_LIBNL20 -
DEAP_TLS -DEAP_PEAP -DEAP_TTLS -DEAP_MD5 -DEAP_MSCHAPv2 -DEAP_GTC -
DEAP_OTP -DEAP_LEAP -DIEEE8021X_EAPOL -DPKCS12_FUNCS -DCONFIG_SMARTCARD
-DEAP_TLS_OPENSSL -DCONFIG_CTRL_IFACE -DCONFIG_CTRL_IFACE_UNIX  -
DCONFIG_SME ../src/common/wpa_ctrl.c
arm-arago-linux-gnueabi-gcc  -o wpa_cli wpa_cli.o
../src/common/wpa_ctrl.o ../src/utils/os_unix.o
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant>
```

9. Install the WPA Supplicant.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant> sudo env PATH=$PATH make ARCH=arm CROSS_COMPILE=
DESTDIR=/home/logic/TI_SDK/targetNFS install
mkdir -p /home/logic/ti-sdk-am3517-evm-05.05.00.00/targetNFS/usr/sbin/
for i in wpa_supplicant wpa_passphrase wpa_cli; do cp $i
/home/logic/ti-sdk-am3517-evm-05.05.00.00/targetNFS/usr/sbin//$i; done
make -C ../src install
make[1]: Entering directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src'
…
make[2]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src/wps'
make[1]: Leaving directory
`/home/logic/Downloads/AM3517_WL1271L_Addition/wpa_supplicant/wpa_suppl
icant-0.7.3/src'
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant>
```

10. Link the *wpa_supplicant* binaries to the expected location.

    This step is necessary to alow the Wi-Fi connection script, which is used by several kits, to find the *wpa_supplicant* binary in an expected place. Symbolic links are used.

```
[linux-devkit]:~/WIRELESS/wpa_supplicant/wpa_supplicant-
0.7.3/wpa_supplicant> cd $HOME/TI_SDK/targetNFS
[linux-devkit]:~/ti-sdk-am3517-evm-05.05.00.00/targetNFS> cd usr/sbin
[linux-devkit]:~/ti-sdk-am3517-evm-05.05.00.00/targetNFS/usr/sbin> sudo
ln -s ../local/sbin/wpa_supplicant wpa_supplicant
[linux-devkit]:~/TI_SDK/targetNFS/usr/sbin> sudo ln -s
../local/sbin/wpa_passphrase wpa_passphrase
[linux-devkit]:~/TI_SDK/targetNFS/usr/sbin> sudo ln -s
../local/sbin/wpa_cli wpa_cli
```


6.2.10 Configure U-Boot for NFS Boot

1. Press reset to reboot the development kit and stop the auto-countdown.

```
U-Boot SPL 2011.09 (Oct 30 2012 - 13:42:19)
Texas Instruments Revision detection unimplemented


U-Boot 2011.09 (Oct 30 2012 - 13:42:19)

AM35XX-GP ES2.0, CPU-OPP2, L3-165MHz, Max CPU Clock 600 Mhz
AM3517EVM Board + LPDDR/NAND
I2C:   ready
DRAM:  256 MiB
WARNING: Caches not enabled
NAND:  HW ECC [Kernel/FS layout] selected
512 MiB
MMC:   OMAP SD/MMC: 0
In:    serial
Out:   serial
Err:   serial
Die ID #3cf400010000000001685a201600b00b
Net:   Ethernet PHY: GENERIC @ 0x00
DaVinci-EMAC
Hit any key to stop autoboot:  0
AM3517_EVM #
```

2. Set up the U-Boot commands to boot the kit via NFS as the default boot commands.

```
AM3517_EVM # setenv bootcmd 'run nfsargs; run loadnet; bootm
${loadaddr}'
AM3517_EVM # saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
AM3517_EVM #
```

3. Press the reset button and let the kit boot via NFS.

```
U-Boot SPL 2011.09 (Oct 30 2012 - 13:42:19)
Texas Instruments Revision detection unimplemented


U-Boot 2011.09 (Oct 30 2012 - 13:42:19)

AM35XX-GP ES2.0, CPU-OPP2, L3-165MHz, Max CPU Clock 600 Mhz
AM3517EVM Board + LPDDR/NAND
I2C:   ready
DRAM:  256 MiB
WARNING: Caches not enabled
NAND:  HW ECC [Kernel/FS layout] selected
512 MiB
MMC:   OMAP SD/MMC: 0
In:    serial
```



```
Out:   serial
Err:   serial
Die ID #3cf400010000000001685a201600b00b
Net:   Ethernet PHY: GENERIC @ 0x00
DaVinci-EMAC
Hit any key to stop autoboot:  0
Using DaVinci-EMAC device
TFTP from server 10.1.4.25; our IP address is 10.1.4.96
Filename 'uImage'.
Load address: 0x83000000
Loading:
#################################################################

#################################################################

#################################################################

#################################################################

#################################################################

#################################################################

#################################################################

#################################################################
         ###############
done
Bytes transferred = 3067616 (2ecee0 hex)
## Booting kernel from Legacy Image at 83000000 ...
   Image Name:   Linux-2.6.37
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3067552 Bytes = 2.9 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Linux version 2.6.37 (logic@logic-desktop-am3517) (gcc
version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Oct 31 12:38:29 CDT
2012
[    0.000000] CPU: ARMv7 Processor [411fc087] revision 7 (ARMv7),
cr=10c53c7f
[    0.000000] CPU: VIPT nonaliasing data cache, VIPT nonaliasing
instruction cache
[    0.000000] Machine: OMAP3517/AM3517 EVM
[    0.000000] Reserving 4194304 bytes SDRAM for VRAM
[    0.000000] Memory policy: ECC disabled, Data cache writeback
[    0.000000] AM3517 ES1.1 (l2cache iva sgx neon isp )
[    0.000000] SRAM: Mapped pa 0x40200000 to va 0xfe400000 size:
0x10000
```



```
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 64000
[    0.000000] Kernel command line: console=ttyO2,115200n8
root=/dev/nfs rw nfsroot=10.1.4.25:/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS,nolock ip=dhcp
…
Arago Project http://arago-project.org am3517-evm ttyO2

Arago 2011.09 am3517-evm ttyO2

am3517-evm login:
```

6.2.11 Configure Kernel to Access Wireless Modules

1. Log in to the kit as root.

```
am3517-evm login: root
root@am3517-evm:~#
```

2. Configure the locations of the wireless modules.

```
root@am3517-evm:~# depmod
root@am3517-evm:~#
```

3. Reboot the kit with the *reboot* command.and let it boot back to the login prompt. Ensure the wireless modules loaded properly; the key indicators have been highlighted in the output below.

```
root@am3517-evm:~# reboot

Broadcast message from root (ttyO2) (Sun Jul 29 06:15:59 2012):

The system is going down for reboot NOW!
…
Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Linux version 2.6.37 (logic@logic-desktop-am3517) (gcc
version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Oct 31 12:38:29 CDT
2012
[    0.000000] CPU: ARMv7 Processor [411fc087] revision 7 (ARMv7),
cr=10c53c7f
[    0.000000] CPU: VIPT nonaliasing data cache, VIPT nonaliasing
instruction cache
[    0.000000] Machine: OMAP3517/AM3517 EVM
[    0.000000] Reserving 4194304 bytes SDRAM for VRAM
[    0.000000] Memory policy: ECC disabled, Data cache writeback
[    0.000000] AM3517 ES1.1 (l2cache iva sgx neon isp )
[    0.000000] SRAM: Mapped pa 0x40200000 to va 0xfe400000 size:
0x10000
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 64000
```



```
[    0.000000] Kernel command line: console=ttyO2,115200n8
root=/dev/nfs rw nfsroot=10.1.4.25:/home/logic/ti-sdk-am3517-evm-
05.05.00.00/targetNFS,nolock ip=dhcp
…
```

You should still see the kernel command line for booting from NFS. You should also see the correct WLAN MAC address and the message that the WL1271 interface loaded.

```
…
[   13.830657] Compat-wireless backport release: compat-wireless-2011-
07-27
[   13.837829] Backport based on wl12xx.git v2.6.39-1152-g0c08ddf
[   17.196197] cfg80211: Calling CRDA to update world regulatory domain
[   19.792663] wl1271: WL1271 mac_addr: 00:08:ee:06:14:f5
[   19.792663]
[   19.936065] wl1271: loaded
...
```

6.2.12 Verify Wireless Operation

1. Log in as root.

```
am3517-evm login: root
root@am3517-evm:~#
```

2. Run the wireless configuration script to connect to the access point of your choice. In this example, the access point is named *RedPine*. WPA security is used and the WPA passphrase is *pbzBxobVuz*.

```
root@am3517-evm:~# /etc/rc.d/init.d/network-wifi-station init
Importing configuration variables from /etc/rc.d/rc.conf
Bring down wlan0
ifdown: interface wlan0 not configured
Stopping wpa_supplicant:
rc.restart: args 'wpa_supplicant' 'init'
killall: wpa_supplicant: no process killed
Removing /etc/wpa_supplicant.conf
wpa_supplicant: Missing /etc/wpa_supplicant.conf; recreating
wpa_supplicant: Missing SSID/Passphrase
wpa_supplicant: Enter WiFi SSID to connect to: RedPine
wpa_supplicant: Enter Encription mode (NONE, WEP40, WEP128, WPA, WPA2):
WPA
wpa_supplicant: Enter WiFi WPA passphrase: pbzBxobVuz
/usr/sbin/wpa_passphrase: /lib/libcrypto.so.1.0.0: no version
information available (required by /usr/sbin/wpa_passphrase)
loading wl12xx_sdio
Starting wpa_supplicant. Args:  -B -Dnl80211 -iwlan0 -C/var/run -
P/var/run/wpa_supplicant.pid -c/etc/wpa_supplicant.conf
/usr/sbin/wpa_supplicant: /lib/libssl.so.1.0.0: no version information
available (required by /usr/sbin/wpa_supplicant)
/usr/sbin/wpa_supplicant: /lib/libcrypto.so.1.0.0: no version
information available (required by /usr/sbin/wpa_supplicant)
[  333.218841] wl1271: firmware booted (Rev 6.3.1.0.79)
Bouncing WiFi interface (workaround).
```



```
[  333.319702] wl1271: down
[  333.951324] wl1271: firmware booted (Rev 6.3.1.0.79)
udhcpc (v1.13.2) started
Sending discover...
[  336.434631] wl1271: Association completed.
Sending discover...
Sending select for 192.168.1.128...
Lease of 192.168.1.128 obtained, lease time 86400
adding dns 192.168.1.1
root@am3517-evm:~#
```

NOTE: You may see error messages due to an *ERROR ELP wakeup timeout*. The system should recover automatically from this situation. You should only be concerned if the following step shows a failure.

3. Verify the WLAN interface has obtained an IP address.

```
root@am3517-evm:~# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 00:08:EE:06:14:F5
          inet addr:192.168.1.128  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1568 (1.5 KiB)  TX bytes:1508 (1.4 KiB)

root@am3517-evm:~#
```

4. Telnet to the kit via its wireless IP address to verify operation. For this step, the author switched to another system that was on the same wireless subnet as the access point to which the kit connected.

```
user@DEV274:~$ telnet 192.168.1.128
Trying 192.168.1.128...
Connected to 192.168.1.128.
Escape character is '^]'.


 _____                    _____          _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                   |___|

Arago Project http://arago-project.org am3517-evm

Arago 2011.09 am3517-evm

login: root
Last login: Sun Jul 29 07:20:50 on pts/0
root@am3517-evm:~#
```


6.2.13 Boot Wireless System from NAND

This section is optional. If you are experiencing issues with SD/MMC timing causing errors with communication to the Wi-Fi chip, you can move your root file system into NAND to eliminate NFS from the equation.

1. Use the commands in Step 7 and Step 8 of Section 3.5.2 to create a JFFS2 image of the file system that was created for the Wi-Fi. What we're doing here is creating a JFFS2 image of the file system that we created and patched earlier in this chapter.
2. Use the steps in Section 4.7 to load the Wi-Fi file system into NAND. That section covers the process for loading a JFFS2 file from the TFTP directory. In Step 7, make sure to use the correct file name. Be sure to use the JFFS2 file that you just created in Step 1 of this section for the *tftp* command.

7 Java

The AM3517 SOM-M2 includes enough memory and the Cortex-A8-based TI processor has enough horsepower to run applications written in Java. This section will explain how to set up a Java Development Environment on your host PC that matches the Java Runtime Environment (JRE) that will be deployed to the development kit.

7.1 Prerequisites

The procedures in this section require the following items:

- Completion of Section 3.
- Completion of Sections 4.1 through 4.5.

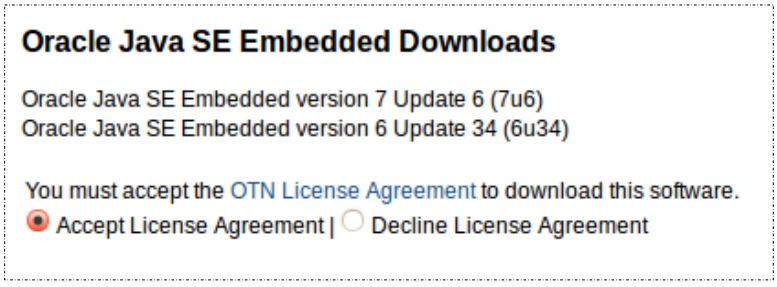
7.2 Procedure

Due to the licensing policies of Oracle, the Java Virtual Machine (JVM) cannot be included with the VM or SDK. The following steps will guide you through the process of downloading and installing the JVM.

1. Start a web browser and open the following Oracle Java SE Embedded Downloads web page.[25]

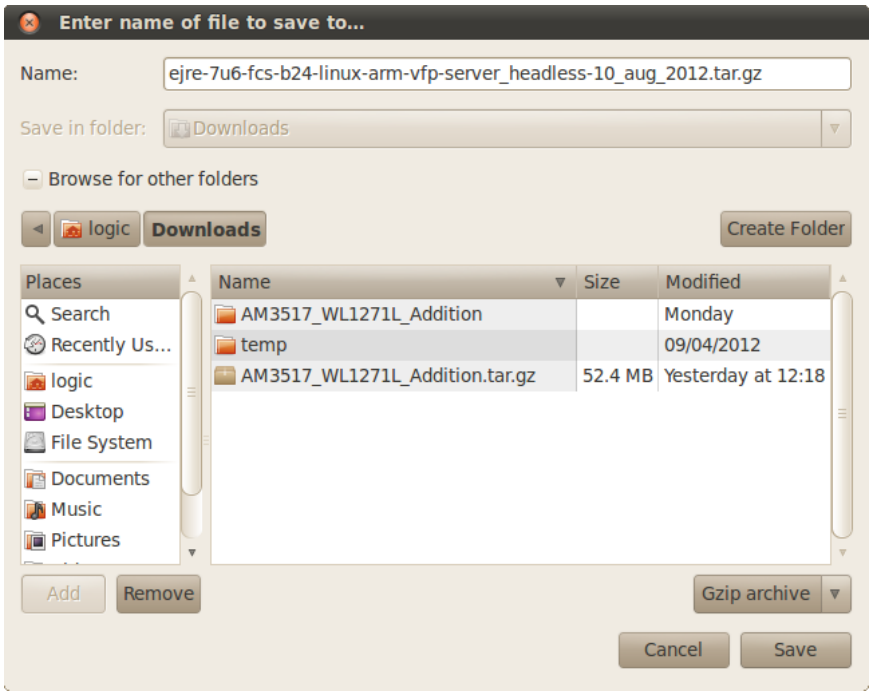
   NOTE: This example, which was completed on the VM, used Firefox as the web browser.

2. Select the radio button labeled "Accept the License Agreement."



[25] http://www.oracle.com/technetwork/java/embedded/downloads/javase/index.html

3. Select the *ARMv7 Linux - Headless - Server Compiler EABI, VFP, SoftFP ABI, Little Endian* version for download and save it to the *Downloads* directory. **NOTE:** If you do not have an account, you'll need to create one to download the file.



4. Open a command terminal and switch to the *Downloads* directory.

```
logic@logic-desktop-am3517:~$ cd ~/Downloads
logic@logic-desktop-am3517:~/Downloads$
```

5. Verify the file name is *ejre-7u6-fcs-b24-linux-arm-vfp-server_headless-10_aug_2012.tar.gz*.

```
logic@logic-desktop-am3517:~/Downloads$ ls -l
…
-rw-r--r-- 1 logic logic 33579311 2012-11-01 14:07 ejre-7u6-fcs-b24-
linux-arm-vfp-server_headless-10_aug_2012.tar.gz
…
logic@logic-desktop-am3517:~/Downloads$
```

6. Switch to the *targetNFS* directory.

```
logic@logic-desktop-am3517:~/Downloads$ cd $HOME/TI_SDK/targetNFS
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$
```

7. Create a home for the JRE and switch into the directory.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$ sudo mkdir -p opt/java
[sudo] password for logic:
logic@logic-desktop-am3517:~/TI_SDK/targetNFS$logic@logic-desktop-
am3517:~/TI_SDK/targetNFS$ cd opt/java
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/opt/java$
```


8. Extract the contents of the JRE download.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/opt/java$ sudo tar xf
$HOME/Downloads/ejre-7u6-fcs-b24-linux-arm-vfp-server_headless-
10_aug_2012.tar.gz
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/opt/java$
```

9. Set up the root accounts profile so that the path to the JVM is automatically added at boot. This is done by editing */etc/profile* for the target root file system.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/opt/java$ vim
../../etc/profile
```

When the editor opens, add the highlighted lines below to the file in the same location. For clarity, the line numbers have been turned on.

```
 22 if [ -d /etc/profile.d ]; then
 23   for i in /etc/profile.d/*.sh; do
 24     if [ -r $i ]; then
 25       . $i
 26     fi
 27   done
 28   unset i
 29 fi
 30
 31 #Add the path to Java JRE:
 32 PATH=$PATH:/opt/java/ejre1.7.0_06/bin
 33
 34 export PATH PS1 OPIEDIR QPEDIR QTDIR EDITOR TERM
 35
 36 umask 022
```

10. Boot the kit using the NFS file system and log in as root.

```
…
AM3517_EVM # run nfsargs; run loadnet; bootm ${loadaddr}
Using DaVinci-EMAC device
TFTP from server 10.1.4.25; our IP address is 10.1.4.96
Filename 'uImage'.
Load address: 0x83000000
Loading: #################################
…
am3517-evm login: root
root@am3517-evm:~#
```

11. Test the access to the JVM by checking the version.

```
root@am3517-evm:~# java -version
java version "1.7.0_06"
Java(TM) SE Embedded Runtime Environment (build 1.7.0_06-b24, headless)
Java HotSpot(TM) Embedded Server VM (build 23.2-b09, mixed mode)
root@am3517-evm:~#
```


12. Switch back to the host PC and create a Java program to test the system. Switch to the target device's root account *home* directory.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/opt/java$ cd
$HOME/TI_SDK/targetNFS/home/root
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/home/root$
```

13. Launch the editor to create the following *HelloWorldApp.java* file.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/home/root$ vim
HelloWorldApp.java
```

14. Add the following lines to the file:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

15. Save the file and close the editor.

16. Compile the java program and verify the output.

```
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/home/root$ sudo env
PATH=$PATH javac HelloWorldApp.java
[sudo] password for logic:
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/home/root$ ls
HelloWorldApp.class  HelloWorldApp.java
logic@logic-desktop-am3517:~/TI_SDK/targetNFS/home/root$
```

17. Switch back to the target device which should still be running. Switch to the *home* directory and verify that you see the Java program that was just compiled.

```
root@am3517-evm:~# cd
root@am3517-evm:~# ls
HelloWorldApp.class  HelloWorldApp.java
root@am3517-evm:~#
```

18. Execute the java program.

```
root@am3517-evm:~# java HelloWorldApp
Hello World!
root@am3517-evm:~#
```


Appendix A: Alternative SD/MMC Setup

Sometimes fdisk is difficult and, for no discernible reason, will not let you create a bootable SD/MMC card. This is one of the more frustrating experiences you are likely to have. In the unfortunate event that this happens to you, this section provides three alternative processes.

Before beginning any of the procedures below, you need to determine the number of cylinders that your disk is supposed to have based on a geometry of 255 heads and 63 sectors per track.

1. Use the *fdisk* command to read the size of your disk. In this example the disk is */dev/sdb,* but be sure to verify the correct device on your system.

```
host$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 4025 MB, 4025483264 bytes
255 heads, 63 sectors/track, 489 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x276e9ddc
…
```

   Note the number of bytes reported by *fdisk*. In the example output above, the number of bytes is 4025483264.

2. Divide this number by the number of bytes per cylinder to obtain the number of cylinders this disk should have. Round down to the nearest integer if you have a remainder.

   4025483264 of bytes / 8,225,280 = 489 cylinders

   Make a note of the value derived from this calculation, as you will use it later.

Option 1: Use *cfdisk*

If your system has the *cfdisk* program, you can use it to create a bootable SD/MMC card. *Cfdisk* is part of the *gpart* package and can be installed on a Debian or Debian varient by using the following command:

```
host$ sudo apt-get install gpart
```

Procedure

1. Erase the first chunk of the disk to clear out any old partition tables.

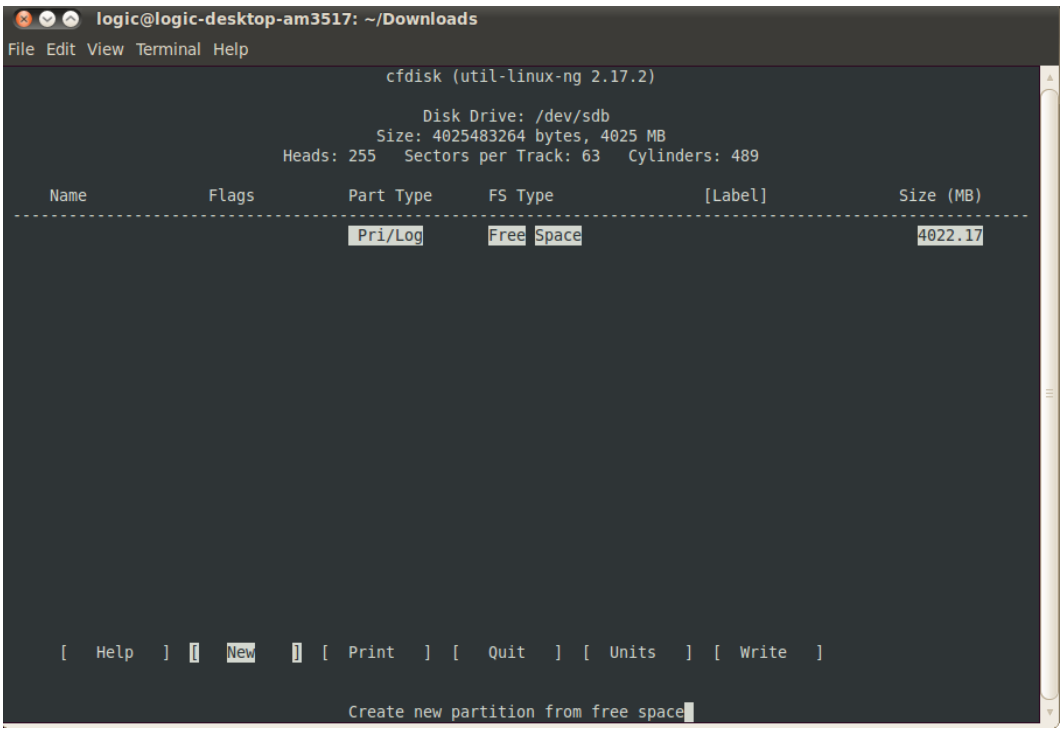
   NOTE: For your safety, the output device below is listed as *sdX*. You must change this to match the device on your system.

```
host$ sudo dd if=/dev/zero of=/dev/sdX bs=1024 count=1024
```

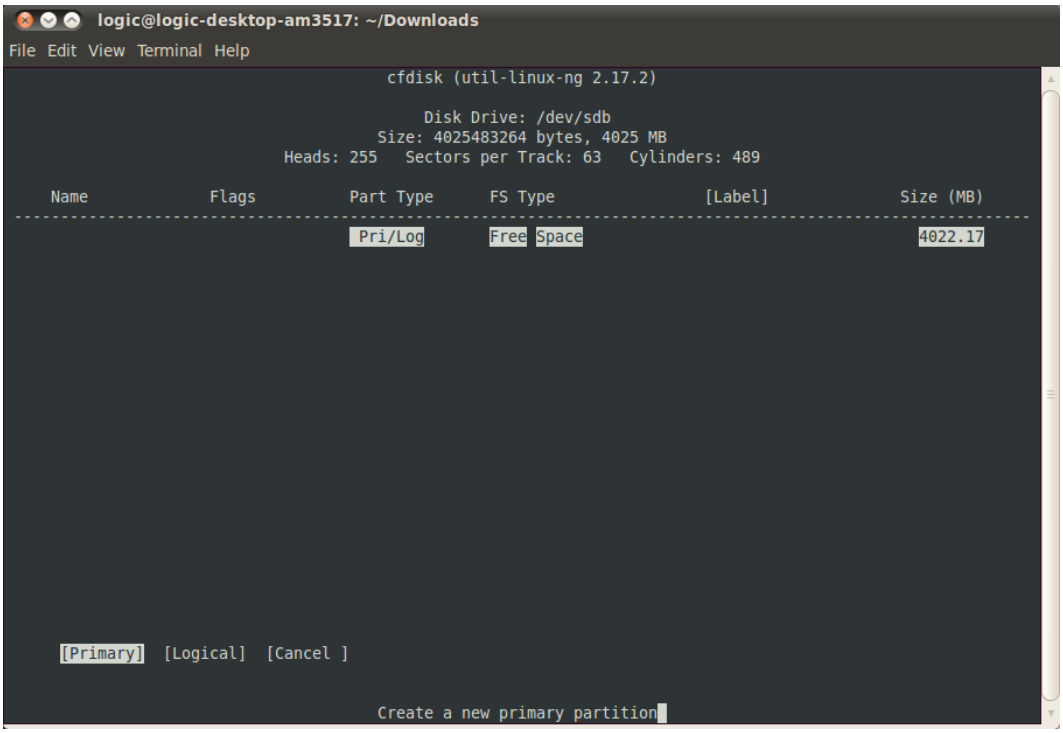
2. Launch the *cfdisk* program, telling it to ignore any partition information that it finds.

```
host$ sudo cfdisk -z /dev/sdX
```

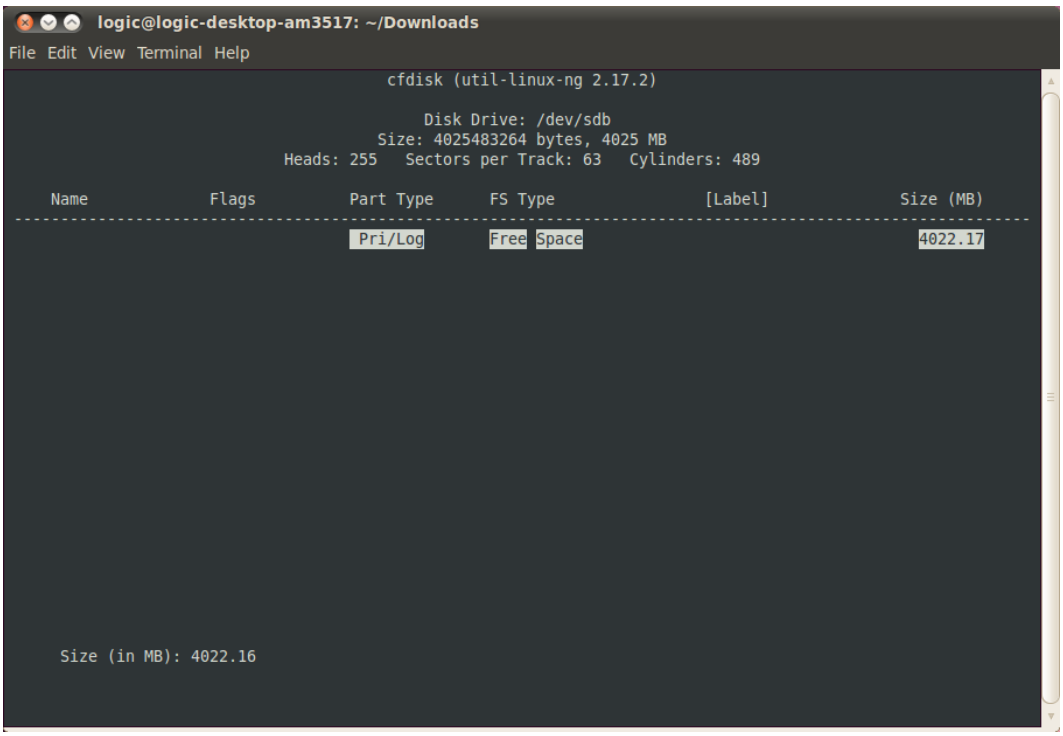

3. Select [New] and press **Enter**.



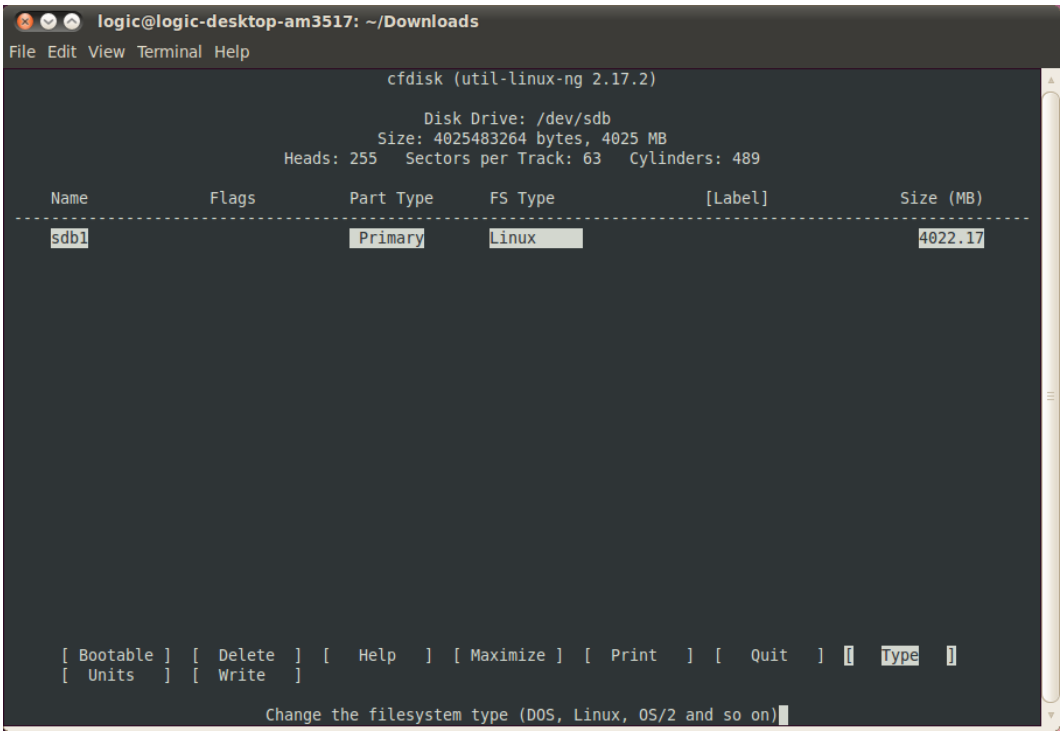
4. Select [Primary] and press **Enter**.



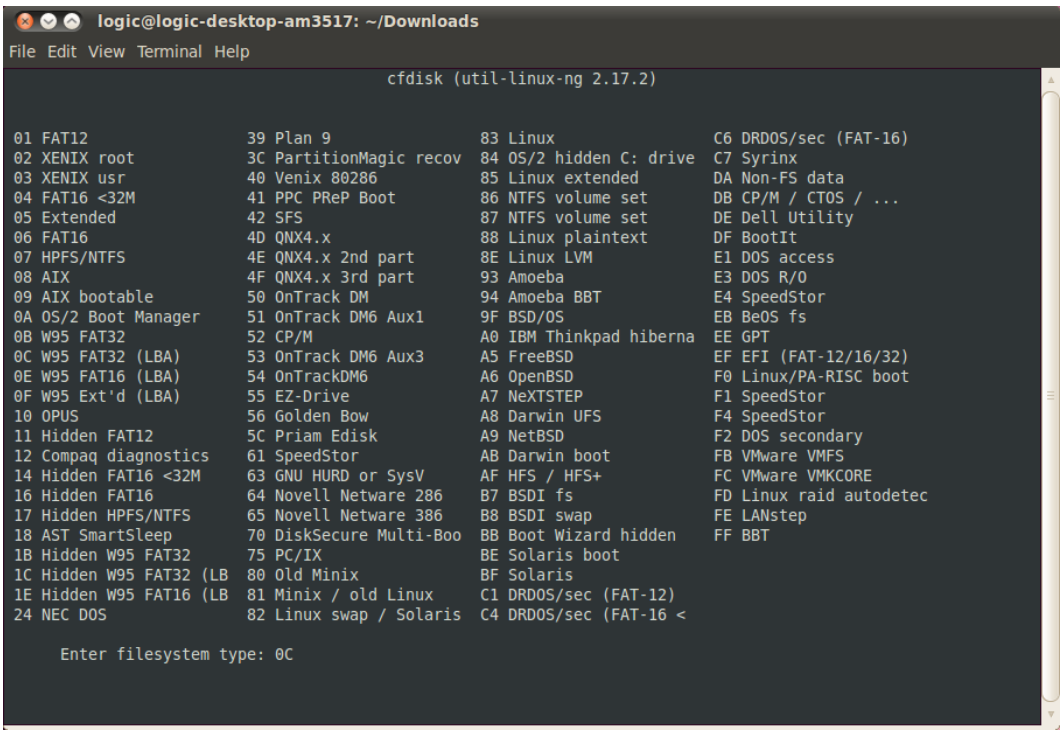
5. Accept the default size by pressing **Enter**.



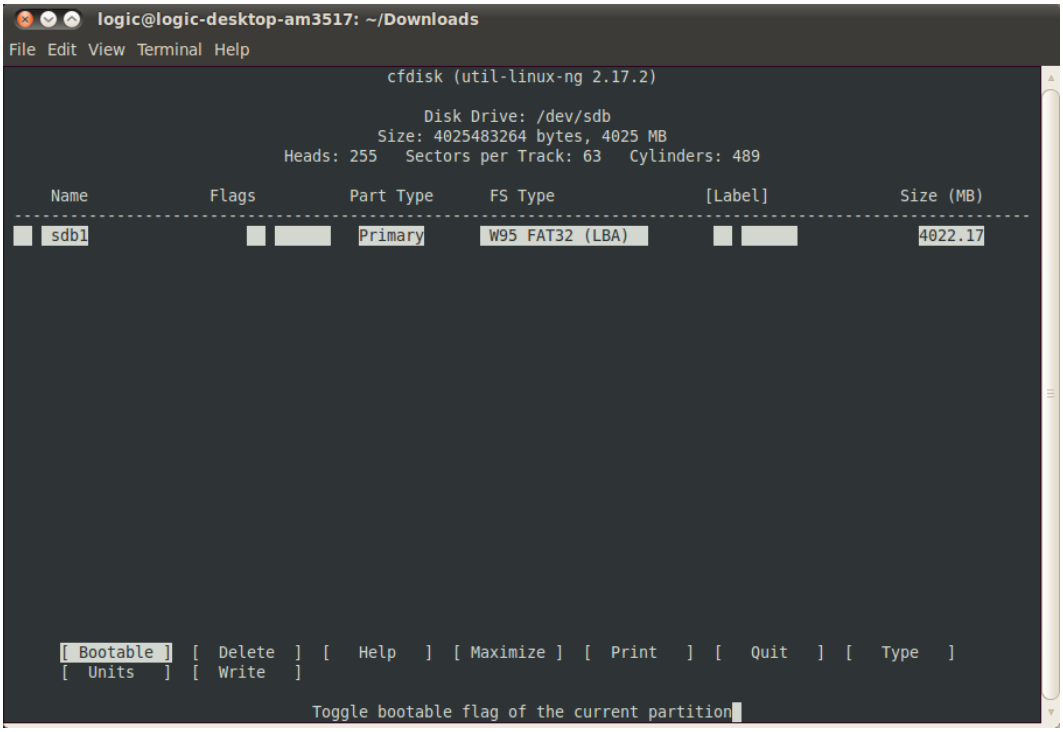
6. Select [Type] and press **Enter**.



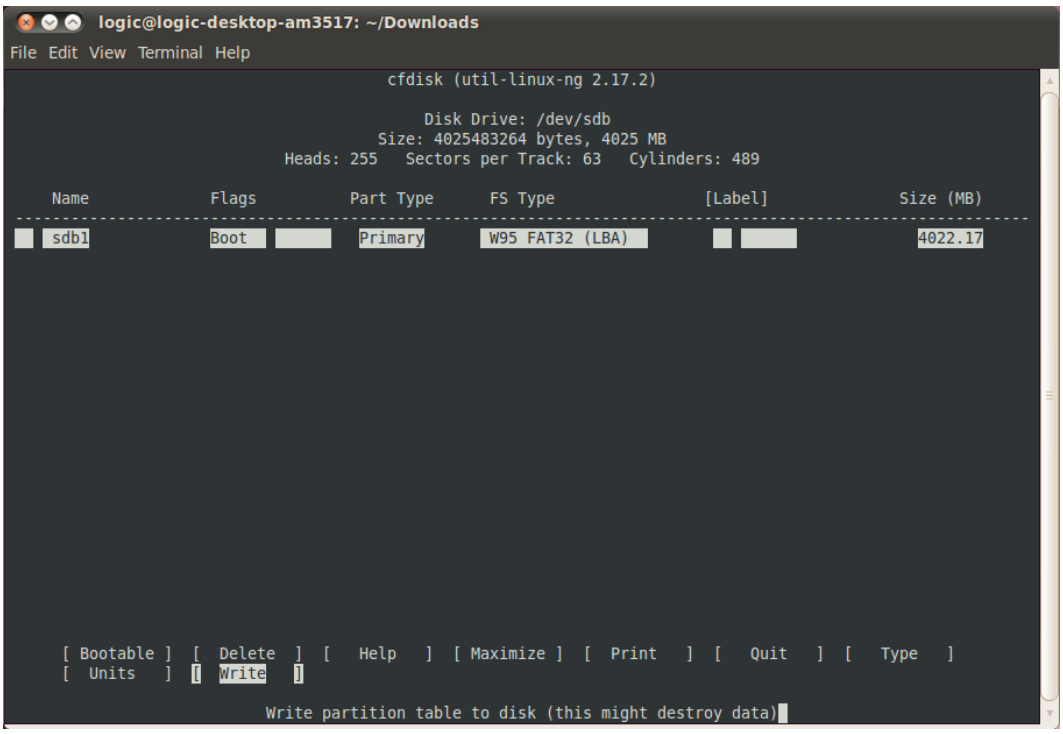
7. Select 0C W95 FAT32 (LBA) as the file system type and press **Enter**.



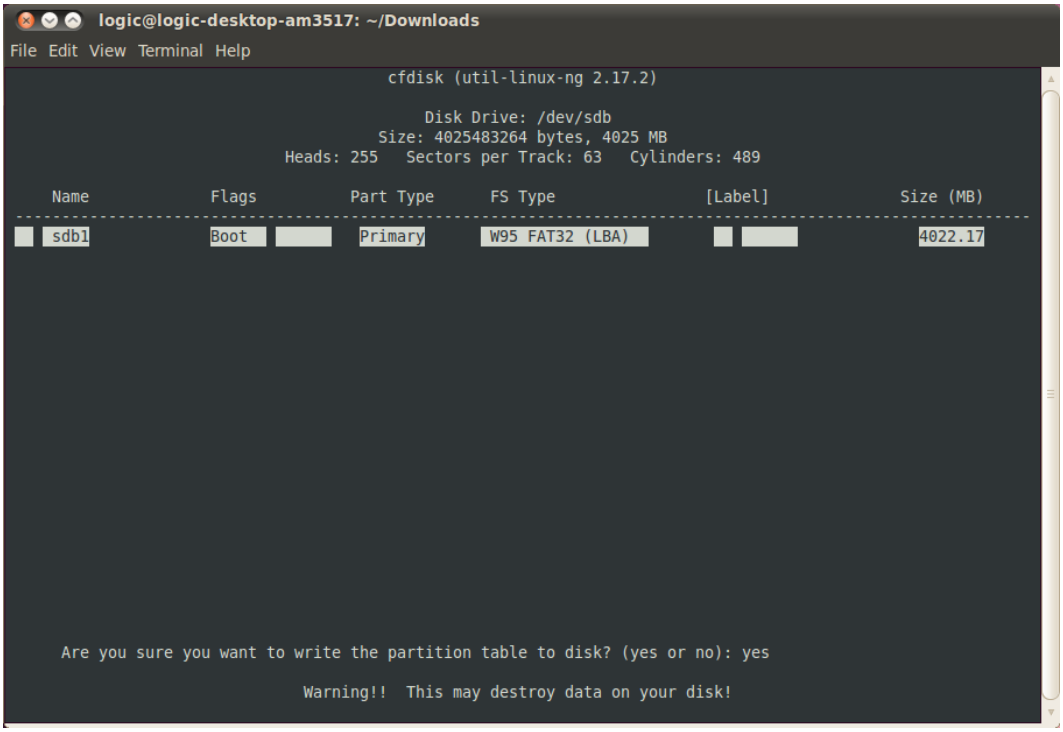
8. Select [Bootable] and press **Enter**.



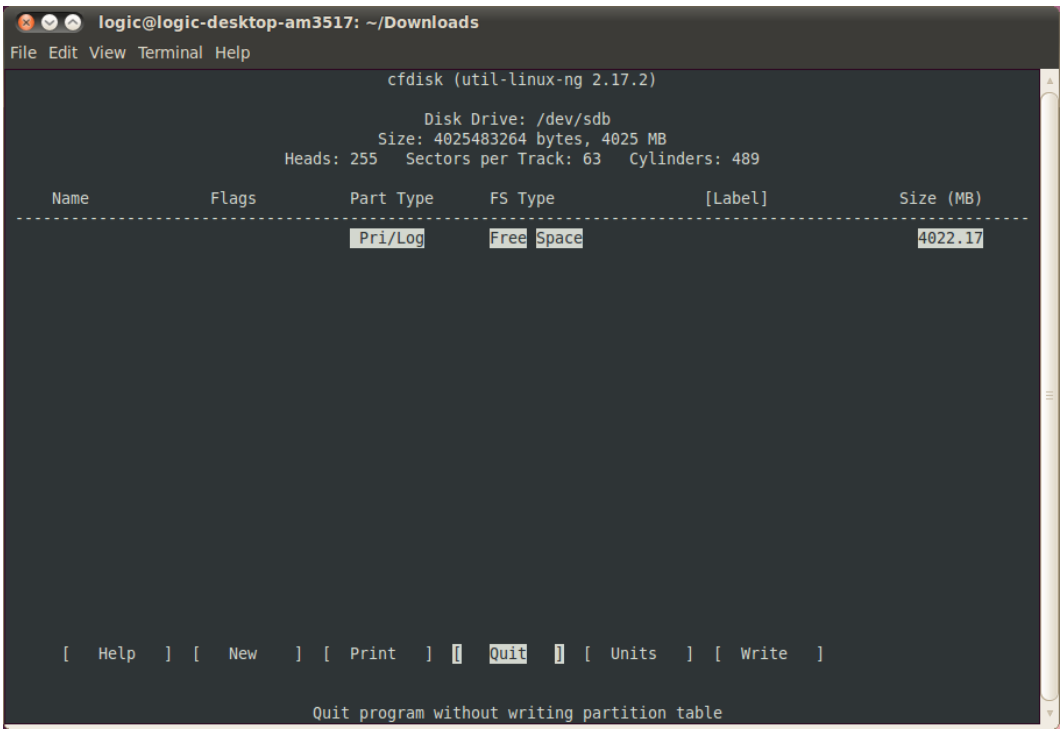
9. Select [Write] and press **Enter**.



10. Type **yes** and press **Enter**.



11. Select [Quit] and press **Enter**. You will be returned to the promt.



12. Create the file system on the new card.

```
logic@logic-desktop-am3517:~$ sudo mkfs.vfat -F 32 /dev/sdb1
mkfs.vfat 3.0.7 (24 Dec 2009)
logic@logic-desktop-am3517:~$
```

13. Mount the disk and copy files as you normally would. Refer to Section 4.6.4 for the steps to properly populate your SD card.

Option 2: Use *sfdisk*

If *cfdisk* didn't work or you need to script the process, you can use the *sfdisk* program. Rather than walking you through each step, we will include a copy of the script *mkcard.sh* shell script from the OpenEmbedded Project.[26] Readers are encouraged to check the link for the latest/greatest version of this script - the copy below is included for completeness.

NOTE: The script shown below will actually create two partitions on your SD/MMC card, one named *boot* and another named *Angstrom*. Should you only want one partition, or if you prefer different names, adjust accordingly.

```
#! /bin/sh
# mkcard.sh v0.5
# (c) Copyright 2009 Graeme Gregory <dp@xora.org.uk>
# Licensed under terms of GPLv2
#
# Parts of the procedure base on the work of Denys Dmytriyenko
# http://wiki.omap.com/index.php/MMC_Boot_Format
```

[26] http://git.openembedded.org/cgit.cgi/openembedded/tree/contrib/angstrom/omap3-mkcard.sh


```
export LC_ALL=C

if [ $# -ne 1 ]; then
    echo "Usage: $0 <drive>"
    exit 1;
fi

DRIVE=$1

dd if=/dev/zero of=$DRIVE bs=1024 count=1024

SIZE=`fdisk -l $DRIVE | grep Disk | grep bytes | awk '{print $5}'`

echo DISK SIZE - $SIZE bytes

CYLINDERS=`echo $SIZE/255/63/512 | bc`

echo CYLINDERS - $CYLINDERS

{
echo ,9,0x0C,*
echo ,,,-
} | sfdisk -D -H 255 -S 63 -C $CYLINDERS $DRIVE

sleep 1

if [ -b ${DRIVE}1 ]; then
    umount ${DRIVE}1
    mkfs.vfat -F 32 -n "boot" ${DRIVE}1
else
    if [ -b ${DRIVE}p1 ]; then
            umount ${DRIVE}p1
            mkfs.vfat -F 32 -n "boot" ${DRIVE}p1
    else
            echo "Cant find boot partition in /dev"
    fi
fi

if [ -b ${DRIVE}2 ]; then
    umount ${DRIVE}2
    mke2fs -j -L "Angstrom" ${DRIVE}2
else
    if [ -b ${DRIVE}p2 ]; then
            umount ${DRIVE}p2
            mke2fs -j -L "Angstrom" ${DRIVE}p2
    else
            echo "Cant find rootfs partition in /dev"
    fi
fi
```

The script starts out as before, erasing the first chunk of the disk so as to remove any existing partition table. It then automatically calculates the number of cylinders the disk should have in the same manner as we did manually. The end of the script finds the two new partitions and calls the *mkfs.vfat* and *mke2fs* to format the new file systems.

The middle part of the script may call for some explanation; each line is numbered below for ease of reference.

```
{
echo ,9,0x0C,*
echo ,,,-
} | sfdisk -D -H 255 -S 63 -C $CYLINDERS $DRIVE
```

In line 0 above, placing the *echo* commands in between the curly braces executes them one at a time without invoking a subshell. If you are confused by this, try running the following command sequence:

```
host$ {
> echo foo
> echo bar
> } > tmp_file
host$ cat tmp_file
```

Essentially, the commands between the curly braces are going to create a temporary, two-line file that will be read by the *sfdisk* program.

Line 0 is a partition descriptor for the FAT32 boot partition. *Sfdisk* partition descriptors use the following format and not all fields need to be filled:

format <start> <size> <id> <bootable> <c,h,s> <c,h,s>

Thus the entry *,9,0x0C,** defines a partition starting at the beginning, going for nine cylinders (9 x 8225280 = 70 MiB), marked as FAT32 (0x0C in *fdisk*-speak), and marked bootable.

Line 0 also describes a partition. In this case, defaults are used for start, end, and type. That will cause *sfdisk* to use the rest of the disk. The '-' character marks this partition as not bootable.

Line 0 is the call to the *sfdisk* program using DOS mode to manually set the disk geometry to the desired values (head, sectors, cylinders).

To use the script, follow the procedure below.

Procedure

1. Make the script executable.

```
host$ chmod +x mkcard.sh
```

2. Execute the scrip as root, passing it the device entry of your disk.

```
host$ sudo ./mkcard.sh /dev/sdX
```

3. Mount the partitions and copy files as you normally would.

Option 3: Use Sector Math

If *fdisk* still doesn't seem to behave for you, but it is your only option, this section may be for you. It is highly recommended that you employ the two methods above first. In the author's

experience, once *fdisk* runs off the rails, there isn't a fool-proof procedure for getting it right, although the method described here may do so.

Background

Throughout the course of development, various versions of the *fdisk* program have seemed to take it upon themselves to perform bits of “black-magic” on behalf of the user, whether requested to or not. For example, the Debian 5.0 (Lenny) system used to write this document uses a version of *fdisk* that starts the first partition at cylinder zero by default, then secretly moves it to cylinder one because the Master Boot Record (MBR) needs to be written into cylinder zero.

When this happens, switching to “sector math” can sometimes get your SD/MMC card sorted out and booted properly. To accomplish this, we will continue to use *fdisk*, but we will switch it over to use units of sectors, not cylinders.

First, we will need a quick refresher in disk geometry. In days of yore, disks actually had spinning platters so the terms *sector, track, head,* and *cylinder* represented real things in the physical world. Now, *sector* is the only disk geometry term likely to refer to anything real (such as the smallest chunk of data that can be read from, written to, or erased on a storage medium like NAND). *Tracks* and *heads* are almost always just logical terms referring to historically-determined fixed values used by software. *Cylinders* are also a logical value determined by the size of the disk and the values assigned to *heads* and *tracks*.

To wit:

- Disks are made up of *cylinders*.
- *Cylinders* are a logical value. The number of bytes per cylinder is equal to the number of heads times the number of bytes per track *(heads x bytes-per-track).*
- *Heads* are a logical value. For historical reasons, disks have 255 heads.
- *Tracks* are a logical value. For historical reasons, each track is made up of 63 sectors.
- *Sectors* are 512 bytes and represent the basic building block of a disk.

Based on the above information:

- 1 sector = 512 bytes
- 1 track = 32,256 bytes
  - 512 bytes/sector x 63 sectors/track
- 1 cylinder = 8,225,280 bytes
  - 255 heads x 32,256 bytes/track

Despite all of the information above, our goal here remains quite simple. We want to get *fdisk* to create a partition of a given size that starts at the disk's second cylinder (actually Cylinder 1, as they are numbered starting at zero), have it proceed for as many cylinders as we want (depending on how large we want the partition to be), and end on a nice cylinder boundary. We just want to express all of this using *sectors* rather than *cylinders*.

Procedure

Before doing anything with the disk, do the following calculations:

1. Decide on the size, in bytes, of the partition you want:

   Partition size in bytes = _____ (A)

2. Divide the desired partition size by the number of bytes per cylinder to determine the number of cylinders your partition requires. If you get a remainder, round up to the next integer:

   (A) / 8,225,280 = _____ (B)

3. Add one (1) to the number of cylinders calculated in Step 2 to adjust for the fact that cylinder-0 holds the MBR:

   (B) + 1 = _____ (C)

4. Multiply the result of Step 3 by the number of sectors per cylinder to calculate the number of sectors in the desired partition:

   (C) * (255 * 63) = _____ (D)

5. Subtract one (1) from the result of Step 4 to determine the number of the last sector in your partition:

   (D) - 1 = _____ (E)

Cylinder 0, which holds the MBR, is 63 sectors long, spanning sectors 0 to 62. Our partition starts at the very next cylinder, spanning sectors 63 to (E). For example, a 256 MiB (268,435,356 byte) partition will span sectors 63 to 546,209.

Now, follow the steps below.

1. Insert an SD/MMC card into your host PC.
2. Use the *mount* command to find the device entry that corresponds to the SD/MMC card.

```
host$ mount
    ...
/dev/sdb1 on /media/disk type vfat

    ...
```

   On the system used in this example, the SD/MMC card was automatically mounted as */media/disk*. By reviewing the output above, we can see that the card corresponds to */dev/sdb*.

3. Unmount the SD/MMC card.

```
host$ sudo umount /media/disk
```

4. Destroy everything on (or at least a good chunk near the beginning of) the disk. You don't need it and don't want anything getting in the way. Do adjust the *count* variable accordingly. Please note that, depending on the size of your disk, this can take some time.

```
host$ sudo dd if=/dev/zero of=/dev/sdb bs=512 count=524288
```

   If you want to check on the command's progress, send '-USR1' to the process ID and the *dd* command will print out some progress statistics. **NOTE:** You will need to do this from another terminal session.

```
host$ sudo kill -USR1 <PID_of_dd>
```


5. Synchronize all disks.

```
host$ sudo sync
```

6. Start the *fdisk* utility. **NOTE:** You may need to become root to do this.

```
host$ sudo fdisk /dev/sdb

GNU Fdisk 1.2.1
Copyright (C) 1998 - 2006 Free Software Foundation
    ...
Using /dev/sdb
Command (m for help):
```

Depending on the version of *fdisk* you are using, you may get a warning/notification that no partition table was found and that *fdisk* has created a default one for you. That should be fine, as it will be deleted in the next step.

7. Clear the existing partition table.

```
Command (m for help): o
```

8. Display information about the card.

```
Command (m for help): p

Disk /dev/sdb: 1 GB, 1965841920 bytes
xxx heads, yy sectors/track, zzz cylinders
Units = cylinders of ..... * ... = ....... bytes

Device Boot  Start    End   Blocks   Id  System
```

Note the number of bytes on the card. The card used in this example had *1965841920* bytes.

9. Calculate the number of cylinders the card will have based on the number of bytes we just noted. The calculation is:

    cylinders = bytes / (bytes per cylinder)

    In the working example, this reduces to:

    1,965,841,920 / 8,225,280 = 239

    If your calculation includes a fraction, round down to the nearest integer.

10. Enter *expert* mode.

```
Command (m for help): x
```

11. Set the number of heads to 255.

```
Expert command (m for help): h

Number of heads (default xxx):  < 255 >
```


12. Set the number of sectors to 63.

```
Expert command (m for help): s

Number of sectors (default yy):  < 63 >
```

13. Set the number of cylinders to the value calculated above.

```
Expert command (m for help): c

Number of cylinders (default zzz):  < 239 >
```

14. Leave *expert* mode.

```
Expert command (m for help): r
```

15. Switch the units *fdisk* is using from cylinders to sectors.

```
Command (m for help): u

Changing display/entry units to sectors
```

16. Create a new primary partition that extends from sector 63 to the number determined in Step 5 (value E) at the beginning of this section.

```
Command (m for help): n

Partition type
   e   extended
   p   primary partition (1-4)
< p >
First sector  (default xxxs):   < 63 >
Last sector or +size or +sizeMB or +sizeKB (default xxxs): < (E) >
```

17. Mark the newly created partition bootable.

```
Command (m for help): a

Partition number (1-1):  < 1 >
```

18. Mark the partition as a FAT file system.

```
Command (m for help): t

Partition number (1-1):  < 1 >
Hex code (type L to list codes):  < c >
Changed type of partition 1 to c (FAT32 LBA)
```


19. Double check all of the changes before writing them to the card.

```
Command (m for help): p

Disk /dev/sdb: 1 GB, 1965841920 bytes
255 heads, 63 sectors/track, 239 cylinders
Units = sectors of 1 * 512 = 512 bytes

   Device Boot  Start     End   Blocks  Id  System
/dev/sdb1   *      63  546209   273073   c  FAT32 LBA
```

20. Commit the changes to the card.

```
Command (m for help): w

Information: Don't forget to update /etc/fstab

Writing all changes to /dev/sdb.
```

21. Synchronize the disks.

```
host$ sudo sync
```

22. Sometimes a host will automatically remount the card once the changes have been made. We need to make sure that the card is *not* mounted so we can format it.

```
host$ mount
```

    If the card has been mounted again, unmount it using the *umount* command.

```
host$ sudo umount /media/disk
```

23. If you have trouble, you may wish to completely erase the new partition before proceeding. This can be done by piping */dev/zero* into the new partition. Divide the size of your partition in bytes by 512 and then adjust the *count* parameter of the command shown below.

    IMPORTANT NOTE: The target of this command is the partition *(/dev/sdxX*) *not* the disk (*/dev/sdx*).

```
host$ sudo dd if=/dev/zero of=/dev/sdb1 bs=512 count=524288
```

24. Format the newly partitioned card.

```
host$ sudo mkfs.vfat -F 32 /dev/sdb1
mkfs.vfat 3.0.7 (24 Dec 2009)
```

25. Mount the partition and copy files as you normally would.

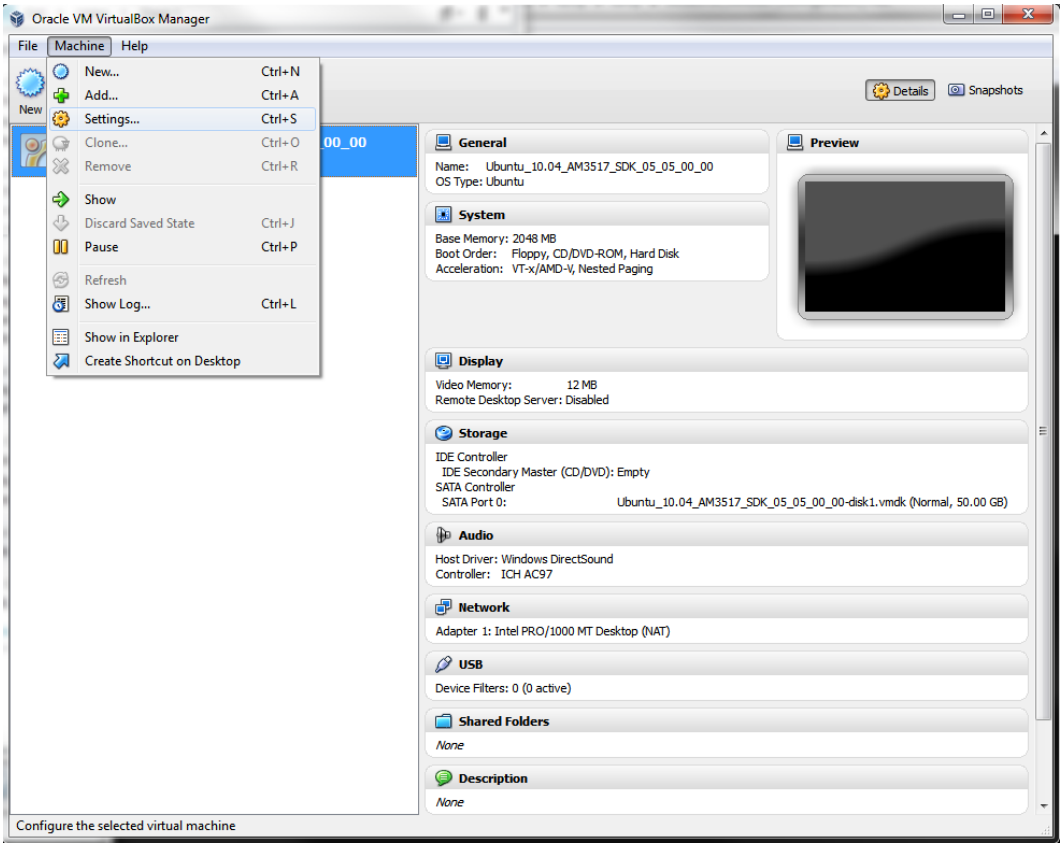
Appendix B: Connect to Serial Port in VM

If you are using the Logic PD-provided VM, you will need to decide if you are going to make a connection to your kit/project with your host PC's serial port or use a USB serial port. We'll explain both options below.

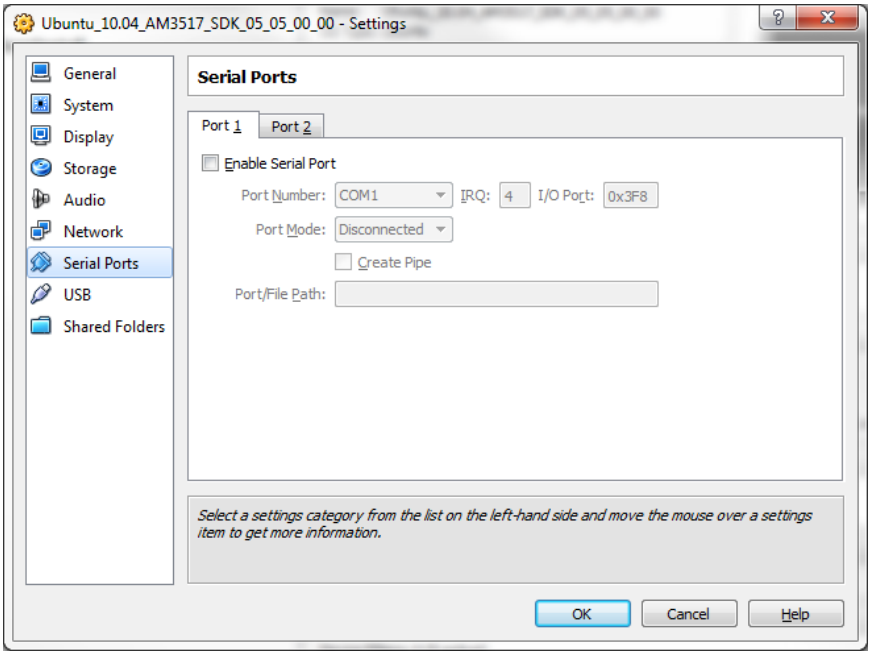
Use Host Serial Port

In this option, you will instruct the VM to use the serial port hardware of the host PC.

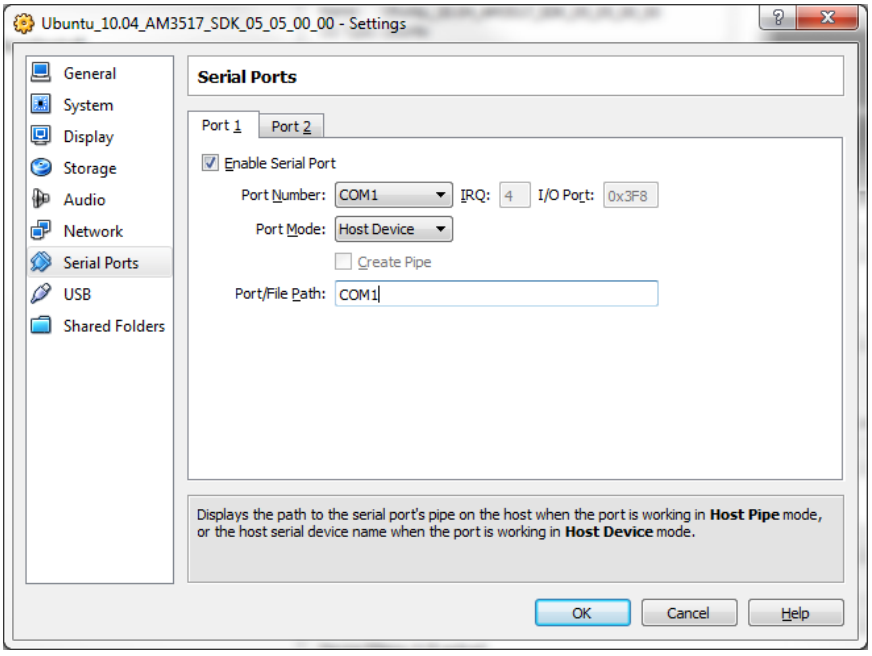
1. Shut down the VM and return to the VM manager window.
2. Select the VM and select Machine > Settings.



3. Select Serial Ports on the toolbar at the left side of the screen.



4. On the *Port 1* tab, select the checkbox labeled "Enable Serial Port" and select the correct port name for an available serial port on your host PC. For the *Port Mode* field, select Host Device. In the *Port/File Path* field, re-enter the port name for your selected serial port.

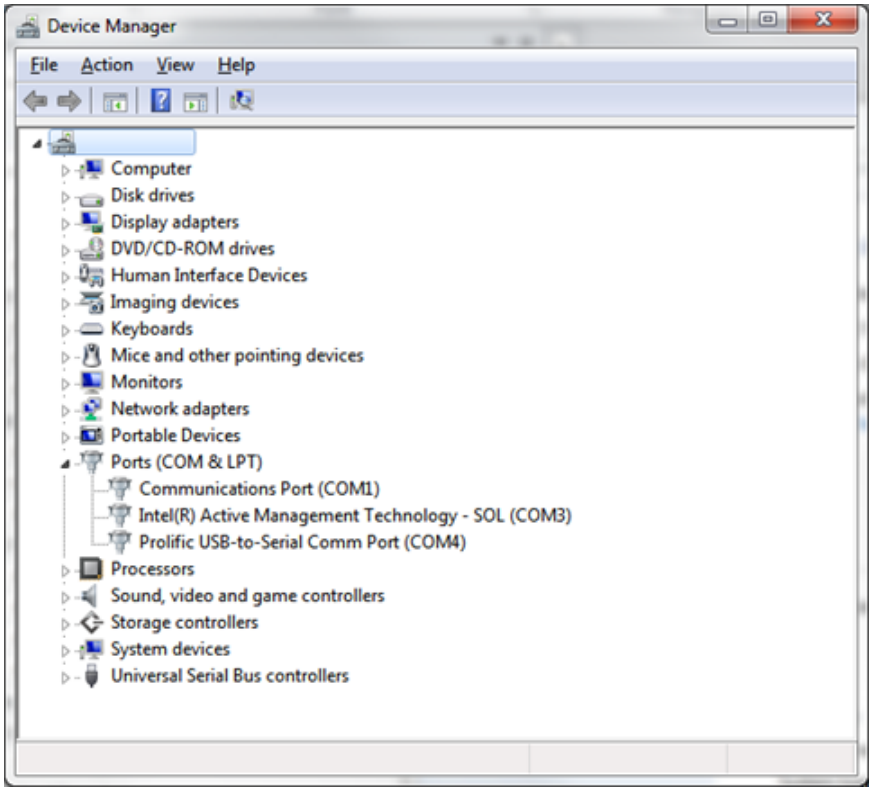


5. Click OK to close.
6. Restart the VM.
7. The serial port will be connected as */dev/ttyS0*. Use this path for serial port settings elsewhere in this document.

Use USB Serial Port

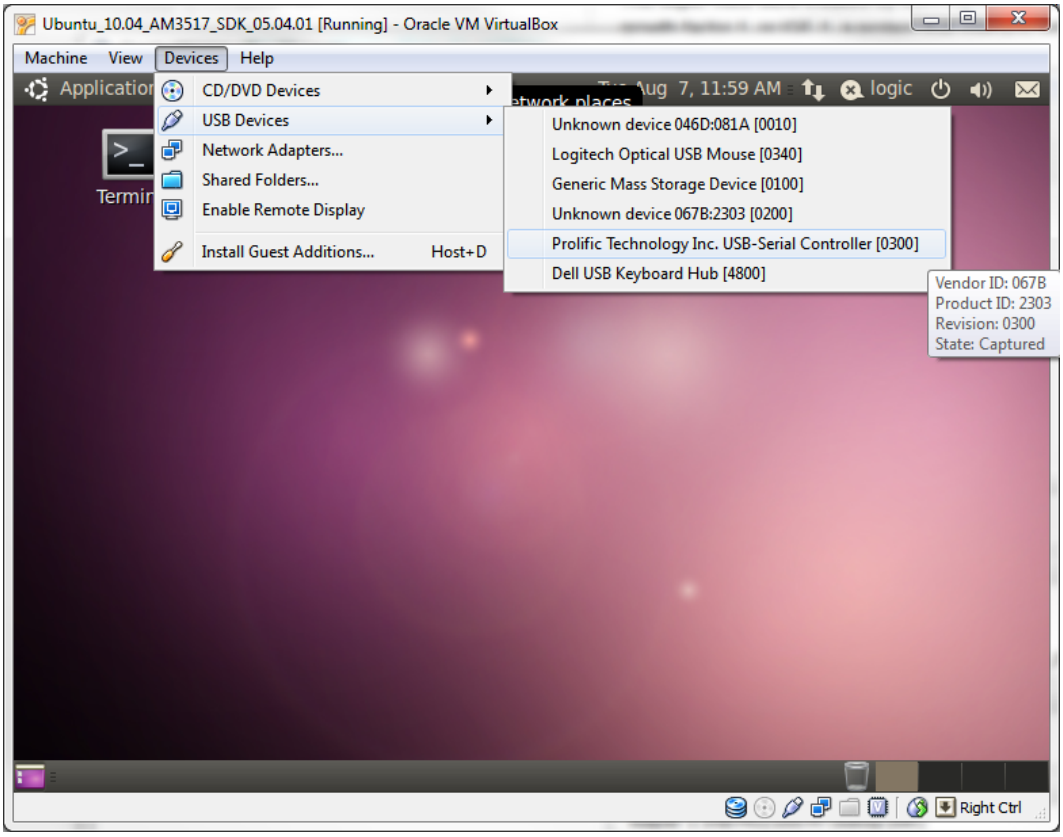
In this option, you will instruct the VM to use a USB serial port that has been connected to the host PC.

1. Shut down the VM.
2. Plug the USB serial port into the host PC. Follow the standard procedures for installing drivers; reboot your machine if necessary. Identify your USB serial port in the Device Manager of your host PC. In this example, the device is *Prolific USB-to-Serial Comm Port (COM4)*.



3. Start the VM and log in.

4. In the *Oracle VM VirtualBox* window, select Devices > USB Devices. You should see a reference to your USB serial device. Select it to connect the device to the VM.



5. Open a terminal window and check the recent log activity to identify the Linux device name of the USB serial port that was just added to the system.

```
logic@logic-desktop-am3517:~$ dmesg | tail
[   96.189457] usb 1-2: configuration #1 chosen from 1 choice
[   96.276769] usbcore: registered new interface driver usbserial
[   96.277497] USB Serial support registered for generic
[   96.278222] usbcore: registered new interface driver
usbserial_generic
[   96.278225] usbserial: USB Serial Driver core
[   96.287197] USB Serial support registered for pl2303
[   96.287907] pl2303 1-2:1.0: pl2303 converter detected
[   96.333804] usb 1-2: pl2303 converter now attached to ttyUSB0
[   96.333815] usbcore: registered new interface driver pl2303
[   96.333817] pl2303: Prolific PL2303 USB to serial adaptor driver
logic@logic-desktop-am3517:~$
```

You can see that the device was added as *ttyUSB0*. Use the path */dev/ttyUSB0* for serial port settings elsewhere in this document.

