

# Sitara Linux Training: Boot Time Reduction

From Texas Instruments Wiki

Jump to: [navigation](#), [search](#)



## Contents

- [1 Lab Configuration](#)
  - [1.1 Hardware](#)
    - [1.1.1 Optional](#)
  - [1.2 Software](#)
- [2 Setting up a Minimal System](#)
  - [2.1 Description](#)
  - [2.2 Key Points](#)
  - [2.3 Lab Steps](#)
- [3 Measuring Boot Time](#)
  - [3.1 Description](#)
  - [3.2 Key Points](#)
  - [3.3 Lab Steps](#)
- [4 Finding SPL/U-Boot Optimizations](#)
  - [4.1 Description](#)
  - [4.2 Key Points](#)
  - [4.3 Lab Steps](#)
- [5 Finding Linux Kernel Optimizations](#)
  - [5.1 Description](#)
  - [5.2 Key Points](#)
  - [5.3 Lab Steps](#)
- [6 Finding File System Optimizations](#)
  - [6.1 Description](#)
  - [6.2 Key Points](#)
  - [6.3 Lab Steps](#)
- [7 Optimizing the Root File System](#)

- [7.1 Description](#)
- [7.2 Key Points](#)
- [7.3 Lab Steps](#)
- [8 Writing the File System to SD](#)
  - [8.1 Description](#)
  - [8.2 Key Points](#)
  - [8.3 Lab Steps](#)
- [9 Optimizing the Linux Kernel](#)
  - [9.1 Description](#)
  - [9.2 Key Points](#)
  - [9.3 Lab Steps](#)
- [10 Writing the Linux Kernel to SD](#)
  - [10.1 Description](#)
  - [10.2 Key Points](#)
  - [10.3 Lab Steps](#)
- [11 Optimizing the U-Boot/SPL](#)
  - [11.1 Description](#)
  - [11.2 Key Points](#)
  - [11.3 Lab Steps](#)
- [12 Writing the U-Boot/SPL to SD](#)
  - [12.1 Description](#)
  - [12.2 Key Points](#)
  - [12.3 Lab Steps](#)
- [13 Measuring the Final Boot Time](#)
  - [13.1 Description](#)
  - [13.2 Key Points](#)
  - [13.3 Lab Steps](#)
- [14 Further Notes](#)
- [15 Additional Methods](#)
  - [15.1 Flashing the File System to NAND with UBIFS](#)
    - [15.1.1 Description](#)
    - [15.1.2 Key Points](#)
    - [15.1.3 Lab Steps](#)
  - [15.2 Flashing the Linux Kernel to NAND](#)
    - [15.2.1 Description](#)
    - [15.2.2 Key Points](#)
    - [15.2.3 Lab Steps](#)
      - [15.2.3.1 From Linux](#)
      - [15.2.3.2 From U-Boot](#)
  - [15.3 Flashing the U-Boot/SPL to NAND](#)
    - [15.3.1 Description](#)
    - [15.3.2 Key Points](#)
    - [15.3.3 Lab Steps](#)
      - [15.3.3.1 From Linux](#)
      - [15.3.3.2 From U-Boot](#)

# Introduction

---

This lab is going to give you a hands on tutorial of how to reduce the boot time of a Linux system. There are many techniques to reduce Linux boot time and not all of them will be covered in this lab so please refer to the presentation for more ideas.

## GOALS:

- Provide the user with a simple display on the LCD and await touchscreen input
- Provide the above in **3** seconds

## FEEDBACK:

If you have questions or feedback please e-mail the [sdk\\_feedback@list.ti.com](mailto:sdk_feedback@list.ti.com) mailing list.

**NOTE:** In this guide commands to be executed for each step will be marked in **BOLD**

## Lab Configuration[[edit](#)]

The following are the hardware and software configurations for this lab. The steps in this lab are written against this configuration. The concepts of the lab will apply to other configurations but will need to be adapted accordingly.

### Hardware[[edit](#)]

1. beaglebone (<http://beagleboard.org/buy>)
2. 7" LCD cape ([http://circuitco.com/support/index.php?title=BeagleBone\\_Capes](http://circuitco.com/support/index.php?title=BeagleBone_Capes))
3. 5V power supply

## Optional[\[edit\]](#)

1. 16-bit NAND cape  
([http://circuitco.com/support/index.php?title=BeagleBone\\_Capes](http://circuitco.com/support/index.php?title=BeagleBone_Capes))
  - This hardware is required if you wish to do the steps for flashing the kernel or file system into NAND

### IMPORTANT

In this use case the boards should be stacked as beaglebone -> NAND cape -> LCD Cape. The NAND cape cannot be placed on the cape expander of the LCD cape because it cannot pull the SYSBOOT pins to the right settings from there.

## Software[\[edit\]](#)

1. A Linux host system configured as per the [Linux Host Configuration](#) page
2. Sitara Linux SDK installed. This lab assumes the SDK is installed in `/home/sitara`. If you use a different location please modify the below steps accordingly.
3. `setup.sh` script in the SDK run to configure the target device for NFS and TFTP.
  - For help configuring the NFS and TFTP setup see [Sitara Linux Training: Hands on with the SDK](#)
4. SD card with Sitara Linux SDK installed.
  - For help creating a 2 partition SD card with the SDK content see the [create\\_sdcards.sh script page](#)
5. The [Sitara Linux Training: Linux Board Port](#) and [Sitara Linux Training: U-Boot Board Port](#) lab trees built with support for the 16-bit NAND cape (tag 05.05.00.00-nand16). These files should be save in the `/tftpboot` directory as:
  - `uImage-board-port`
  - `MLO-board-port`
  - `u-boot-board-port`

### NOTE

If using a TI laptop during the labs these images are already created and saved in the `/tftpboot` directory

# Setting up a Minimal System[\[edit\]](#)

## Description[\[edit\]](#)

This section will cover how to setup the Sitara Linux SDK and the target board to boot the minimal `base-roots-<machine>.tar.gz` file system found in the Sitara Linux SDK. During this lab you will also be using the Linux kernel and U-boot from the [Linux board port](#) and [U-Boot board port](#) labs. These serve as a good starting point for optimization

since many of the extra features enabled for development by the *tisdk-rootfs-<machine>.tar.gz* file system and the default kernel and u-boot configurations are not necessary in a boot time optimized system.

## Key Points[\[edit\]](#)

- Configuring the system for a typical development environment. This includes:
  - Loading the u-boot boot loader from MMC
  - Transferring the kernel over TFTP
  - Booting the target root file system via NFS

## Lab Steps[\[edit\]](#)

1. Close any open minicom sessions you may have by pressing **CTRL+A** then **Z** and then **X**. When prompted select **Yes** to exit minicom
2. Insert the Sitara Linux SDK SD card into the beaglebone and boot the board.

### NOTE

You will not see output on the LCD because the SDK does not currently support capes. This is why we will be using the board port kernel during this lab

3. Once the board boots you should see the "boot" partition mounted to your Linux host. You can use the **mount** command to verify that the boot partition is mounted and you should see output like:

```
/dev/sdb on /media/boot type vfat
(rw,nosuid,nodev,uid=1000,gid=1000,shortname=mixed,dmask=0077,utf8=1,showexec,flush,uhelper=udisks)
```

### IMPORTANT

If the SD card you are using already has a uEnv.txt file on it for NFS boot (possibly from a previous lab) and the board fails to boot (Perhaps because the NFS server is no longer available) you will need to connect to the serial console using minicom and can run the following commands to boot the board one time from the SD card.

**env default -f** (This sets the default environment)

**mmc rescan** (This enables the mmc device)

**run mmc\_boot** (This will boot from mmc)

You can now continue with the steps below which will reconfigure the NFS settings

4. Copy the board port u-boot and MLO files (listed as prerequisites) to the SD card boot partition. This is so that the default images on the MMC/SD card will have support for the 16-bit NAND cape.

#### NOTE

If you are using a TI laptop these files are already available at **/tftpboot/MLO-board-port** and **/tftpboot/u-boot-board-port**

- **cp /tftpboot/MLO-board-port /media/boot/MLO**
  - **cp /tftpboot/u-boot-board-port /media/boot/u-boot.img**
5. Change directory into the Sitara Linux SDK installation directory

```
cd /home/sitara/<SDK install>
```

6. Run the setup.sh script

```
./setup.sh
```

#### NOTE

These steps assume that you have already run the setup.sh script once before on your host system and therefore already have an NFS share and kernel uImage exported

7. When prompted provide the following responses
  - If prompted enter your sudo password. If using a TI laptop the password is **sitara**
  - When prompted to select a directory to install the target filesystem press **ENTER** to take the default
  - When asked whether you want to rename, overwrite, or skip the file system extraction enter **s** to skip the extraction
  - Press **ENTER** until prompted for the TFTP directory
  - When prompted for the tftp root directory press **ENTER** to accept the default */tftpboot* location
  - When asked whether to rename, overwrite, or skip the copy of the uImage enter **s** to skip the copy
  - When prompted for the serial port to use press **ENTER** because the serial port will be detected automatically

#### NOTE

The automatic detection only works on boards like the beaglebone and EVM-SK. For the full EVM you should enter your serial port here

- When prompted to enter your host IP address press **ENTER** to take the default

- When prompted to select the Linux kernel location press **ENTER** to take the default of TFTP
  - When prompted to select the root file system location press **ENTER** to take the default of NFS
  - When prompted for the kernel image to use press **ENTER** to take the default value
  - When asked if you are using a Beaglebone or EVM-SK enter **y** to configure for a beaglebone (unless you are using a full EVM)
  - When asked if you would like to restart the device now press **ENTER** to restart the device.
8. You should now see a minicom terminal open and the device begin to boot. You should see that the kernel is loaded over TFTP and the file system is booted from NFS
    - Your system is now configured for a TFTP kernel boot and and NFS root file system
  9. At the login prompt login as **root** with a blank password
  10. Shutdown the board using the following command. This is done so that you can swap the file system being used with the base image from the SDK.

#### **init 0**

#### **IMPORTANT**

Wait until you see the **System halted** message before going on

11. On your host go to the SDK directory and perform the following commands. These commands will move the default NFS file system out of the way and extract a new NFS file system using the base image.
  1. **cd /home/sitara/<sdk install dir>**
  2. **sudo mv targetNFS targetNFS.orig**
  3. **mkdir targetNFS**
  4. **cd targetNFS**
  5. **sudo tar xzf ../filesystem/base-rootfs-<machine>.tar.gz**

#### **NOTE**

**sudo** is used when extracting the target file system because special files such as device nodes require root permission to be created

#### **NOTE**

The steps above are done to preserve your original file system and so that you can re-use the export created for the targetNFS file system with you new file system

12. Now on the host you will change the TFTP kernel image to the image from the [Sitara Linux board port](#) lab. If you have not already done this lab please refer to the [Sitara Linux board port](#) lab for instructions on building the kernel.

## IMPORTANT

If you are using a training laptop provided by TI these images have already been compiled and stored in the /tftpboot directory for you. The following instructions will tell you how to copy these images. If you are not using a TI laptop then you will need to build the kernel image yourself and overwrite the existing image in the /tftpboot directory

0. **cd /tftpboot**
1. **sudo cp uImage-am335x-evm.bin uImage-am335x-evm.bin.orig**
2. **sudo cp uImage-board-port uImage-am335x-evm.bin**
13. Reboot the board. You should see the system come up to a login prompt. You have now finished configuring your system to boot a minimal file system and simple kernel which was ported to your hardware over NFS and TFTP.
14. Once the board is booted shut the board down with **init 0**

**init 0**

## IMPORTANT

Wait until you see the **System halted** message before going on

15. Exit minicom by pressing **CTRL+A** then **Z** and then **X**. When prompted select **Yes** to exit minicom

# Measuring Boot Time[[edit](#)]

## Description[[edit](#)]

This section will cover how to measure the boot time and the time used by different components of the system during boot. This measurement will be used in the following sections to help identify the areas that can be optimized.

## Key Points[[edit](#)]

1. Use the `tstamp.c` program to measure boot time on Linux systems
  - This program can be found in the `sitara-training-helper-files` git repository available at <https://gitorious.org/sitara-training> and can be cloned by doing **git clone [git://gitorious.org/sitara-training/sitara-training-helper-files.git](https://gitorious.org/sitara-training/sitara-training-helper-files.git)**
2. After capturing the boot time output you can use that output to find areas to optimize

## Lab Steps[[edit](#)]

1. Checkout the `sitara-training-helper-files` repository.



## IMPORTANT

If you are using a TI training laptop this has already been done for you and you can find the files in **/home/sitara/sitara-training-helper-files**

```
cd /home/sitara
git clone git://gitorious.org/sitara-training/sitara-training-helper-files.git
```

2. Compile the tstamp.c program

```
cd /home/sitara/sitara-training-helper-files
gcc tstamp.c -o tstamp
```

3. Copy the tstamp executable to the /usr/bin directory so it will be available in your default PATH

```
sudo cp tstamp /usr/bin/
```

4. Set your serial port to the proper baud rate. For these instructions we will assume /dev/ttyUSB1 but you may have a different port based on your setup.

## IMPORTANT

If you are using a device like the beaglebone with a built in USB-to-Serial adapter you can find the serial port by doing **dmesg**

```
stty -F /dev/ttyUSB1 115200
```

5. You can now use the tstamp program to measure boot times. To do this use the command

```
cat /dev/ttyUSB1 | tstamp
```

6. Now **press the Reset button** on the beaglebone (located on at the top right corner when using and LCD 7" cape) and let the system boot. You should see output on the screen with timestamps for each line like

```
37.920 0.080: Starting thttpd.
```

## IMPORTANT

The tstamp program works by capturing time elapsed between new lines on the screen, which is why you do not see the login prompt. It also keeps the counter from the first line which means that if you want to restart the base time counter (the first field) you will need to use **ctrl+c** to exit the application and then restart it running

7. The total boot time in this setup was around **39.252** seconds. We can do much better than that and the rest of this lab will be focused on doing just that to hit our 3 second goal.

## Finding SPL/U-Boot Optimizations[[edit](#)]

### Description[[edit](#)]

Now that you have an output from the boot process we can go through that output by scrolling back in history and reviewing the pieces of the boot that took the most time. This section will highlight areas in the boot loaders that can be optimized as well as a estimate of the difficulty in reducing that time.

The following difficulty classifications will be used during this lab:

- Easy: These changes can be as simple as modifying bootargs and environment settings.
- Medium: These changes usually involve modifying/rebuilding a component but do not require changing the system configuration
- Hard: These changes usually involve modifying/rebuilding a components and changing the system configuration

### Key Points[[edit](#)]

1. Identify steps in the boot loader that consume a large amount of time
2. Identify the begin and end of the boot loader steps
3. Build a list of the optimizations to make in later steps

### Lab Steps[[edit](#)]

1. Identify the beginning of the boot loader in your output. This should be one of the first lines and should look like the following

```
column1 is elapsed time since first message
column2 is elapsed time since previous message
column3 is the message
0.000 0.000:
0.004 0.004: U-Boot SPL 2011.09 (Sep 04 2012 - 09:40:54)
0.008 0.004: Texas Instruments Revision detection unimplemented
0.444 0.436: OMAP SD/MMC: 0
0.512 0.068: reading u-boot.img
0.516 0.004: reading u-boot.img
0.544 0.028:
0.544 0.000:
0.548 0.004: U-Boot 2011.09 (Sep 04 2012 - 09:40:54)
0.548 0.000:
0.600 0.052: I2C:   ready
0.600 0.000: DRAM:  256 MiB
0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND:  HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC:   OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net:   cpsw
4.512 3.004: Hit any key to stop autoboot:  3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0
```

2. Scroll down in the output until you find the beginning of the U-Boot boot. This should look like the following

```
column1 is elapsed time since first message
column2 is elapsed time since previous message
column3 is the message
0.000 0.000:
0.004 0.004: U-Boot SPL 2011.09 (Sep 04 2012 - 09:40:54)
0.008 0.004: Texas Instruments Revision detection unimplemented
0.444 0.436: OMAP SD/MMC: 0
0.512 0.068: reading u-boot.img
0.516 0.004: reading u-boot.img
0.544 0.028:
0.544 0.000:
0.548 0.004: U-Boot 2011.09 (Sep 04 2012 - 09:40:54)
0.548 0.000:
0.600 0.052: I2C:   ready
0.600 0.000: DRAM:  256 MiB
0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND:  HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC:   OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net:   cpsw
4.512 3.004: Hit any key to stop autoboot:  3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0
```

3. The lines located between this start line and end line represent the steps of the SPL boot loader. We will now focus on those lines to identify optimizations.
4. We see that the *OMAP SD/MMC* device is being initialized

```
column1 is elapsed time since first message
column2 is elapsed time since previous message
column3 is the message
0.000 0.000:
0.004 0.004: U-Boot SPL 2011.09 (Sep 04 2012 - 09:40:54)
0.008 0.004: Texas Instruments Revision detection unimplemented
0.444 0.436: OMAP SD/MMC: 0
0.512 0.068: reading u-boot.img
0.516 0.004: reading u-boot.img
0.544 0.028:
0.544 0.000:
0.548 0.004: U-Boot 2011.09 (Sep 04 2012 - 09:40:54)
0.548 0.000:
0.600 0.052: I2C:   ready
0.600 0.000: DRAM:  256 MiB
0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND:  HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC:   OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net:   cpsw
4.512 3.004: Hit any key to stop autoboot:  3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0
```

- Impact: This step is taking about 0.436 seconds
  - Removable?: We are currently booting from SD card and would need to find a different boot source to remove this initialization
  - Difficulty: Hard. Making this change will require modifying the boot loader and changing the system configuration to boot from a different interface than SD/MMC.
5. We have now reached the end of the SPL boot loader and need to identify the start of the U-Boot boot loader. This should look like the following

```

column1 is elapsed time since first message
column2 is elapsed time since previous message
column3 is the message
0.000 0.000:
0.004 0.004: U-Boot SPL 2011.09 (Sep 04 2012 - 09:40:54)
0.008 0.004: Texas Instruments Revision detection unimplemented
0.444 0.436: OMAP SD/MMC: 0
0.512 0.068: reading u-boot.img
0.516 0.004: reading u-boot.img
0.544 0.028:
0.544 0.000:
0.548 0.004: U-Boot 2011.09 (Sep 04 2012 - 09:40:54)
0.548 0.000:
0.600 0.052: I2C:   ready
0.600 0.000: DRAM:  256 MiB
0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND:  HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC:   OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net:   cpsw
4.512 3.004: Hit any key to stop autoboot:  3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0

```

6. Scroll down in the output until you find the beginning of the Linux kernel boot. This should look like the following

```

25.356 1.992: #####
##
26.352 0.996: #####
26.352 0.000: done
26.356 0.004: Bytes transferred = 3160568 (3039f8 hex)
26.368 0.012: ## Booting kernel from Legacy Image at 82000000 ...
26.372 0.004:   Image Name:   Linux-3.2.0-gddb1881-dirty
26.380 0.008:   Image Type:   ARM Linux Kernel Image (uncompressed)
26.380 0.000:   Data Size:    3160504 Bytes = 3 MiB
26.384 0.004:   Load Address: 80008000
26.388 0.004:   Entry Point:  80008000
27.304 0.916:   Verifying Checksum ... OK
28.732 1.428:   Loading Kernel Image ... OK
28.744 0.012: OK
28.744 0.000:
28.744 0.000: Starting kernel ...
28.744 0.000:
29.308 0.564: Uncompressing Linux... done, booting the kernel.
29.352 0.244: [    0.000000] Linux version 3.2.0-gddb1881-dirty (chase@chase-ubu
ntu) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Aug 29 15:37:28 CDT
2012
29.560 0.008: [    0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7),
cr=10c53c7d

```

7. As with the SPL the lines between these sections represent the U-Boot output and where we will focus our optimizations
8. The first optimization point concerns reading the u-boot environment from NAND. Because the NAND is blank this generates an ECC error.

```
0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND: HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC: OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net: cpsw
4.512 3.004: Hit any key to stop autoboot: 3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0
4.584 0.072: SD/MMC found on device 0
4.588 0.004: reading uEnv.txt
4.592 0.004:
4.592 0.000: 264 bytes read
4.596 0.004: Loaded environment from uEnv.txt
4.600 0.004: Importing environment from mmc ...
4.604 0.004: Running uenvcmd ...
4.608 0.004: Booting from network...
6.148 1.540: failed to read bmcr
6.152 0.004: link up on port 0, speed 100, full duplex
6.156 0.004: BOOTP broadcast 1
7.388 1.232: DHCP client bound to address 192.168.10.131
7.388 0.000: Using cpsw device
7.396 0.008: TFTP from server 192.168.10.130; our IP address is 192.168.10.131
7.400 0.004: Filename 'uImage-am335x-evm.'
```

- Impact: This step is taking about 0.764 seconds
  - Removable?: When creating your own product you will may store your environment in a persistent storage like NAND, but you can also modify u-boot to code your default environment directly into the executable.
  - Difficulty: Medium. To remove this you would need to modify u-boot to set your default boot environment.
9. The next step taking a large amount of time is the boot delay

```

0.624 0.024: WARNING: Caches not enabled
0.680 0.056: NAND: HW ECC Hamming Code selected
0.680 0.000: 512 MiB
0.684 0.004: MMC: OMAP SD/MMC: 0
1.448 0.764: *** Warning - bad CRC, using default environment
1.448 0.000:
1.508 0.060: Net: cpsw
4.512 3.004: Hit any key to stop autoboot: 3 ^H^H^H 2 ^H^H^H 1 ^H^H^H 0
4.584 0.072: SD/MMC found on device 0
4.588 0.004: reading uEnv.txt
4.592 0.004:
4.592 0.000: 264 bytes read
4.596 0.004: Loaded environment from uEnv.txt
4.600 0.004: Importing environment from mmc ...
4.604 0.004: Running uenvcmd ...
4.608 0.004: Booting from network...
6.148 1.540: failed to read bmcr
6.152 0.004: link up on port 0, speed 100, full duplex
6.156 0.004: BOOTP broadcast 1
7.388 1.232: DHCP client bound to address 192.168.10.131
7.388 0.000: Using cpsw device
7.396 0.008: TFTP from server 192.168.10.130; our IP address is 192.168.10.131
7.400 0.004: Filename 'uImage-am335x-evm.bin'.

```

- Impact: This step takes about 3.004 seconds
- Removable?: Yes. Although if/when we set this to zero you will be unable to stop the autoboot from the serial console
- Difficulty: Easy/Medium. This is given a rating of easy to medium because you cannot change this value using the uEnv.txt file support on the MMC card. You can change this value if you are using a NAND environment. So for NAND this is easy because the value can be set and saved. For MMC you will need to change the U-Boot configuration and rebuild U-Boot.

10. The next large section is where the network interface is defined and the kernel is transferred



```

4.604 0.004: Running uenvcmd ...
4.608 0.004: Booting from network...
6.148 1.540: failed to read bmcr
6.152 0.004: link up on port 0, speed 100, full duplex
6.156 0.004: BOOTP broadcast 1
7.388 1.232: DHCP client bound to address 192.168.10.131
7.388 0.000: Using cpsw device
7.396 0.008: TFTP from server 192.168.10.130; our IP address is 192.168.10.131
7.400 0.004: Filename 'uImage-am335x-evm.bin'.
7.400 0.000: Load address: 0x82000000
9.404 2.004: Loading: *^H#####
#####
11.404 2.000: #####
##
13.400 1.996: #####
##
15.392 1.992: #####
##
17.384 1.992: #####
##
19.376 1.992: #####
##
21.372 1.996: #####
##
23.364 1.992: #####
##
25.356 1.992: #####
##
26.352 0.996: #####
26.352 0.000: done

```

- Impact: All told this step is taking about 21.7 seconds
- Removable?: Yes, although this will require transferring the kernel from elsewhere
- Difficulty: Medium/Hard. This will require changing the boot configuration which is why it has received a hard rating. However, since changing where the kernel is loaded from is usually a trivial task and in fact supported by many of the default environment commands such as **nand\_boot** and **mmc\_boot** this is also listed as medium.

11. The next big time consumer is verifying the image checksum



```

25.356 1.992: #####
##
26.352 0.996: #####
26.352 0.000: done
26.356 0.004: Bytes transferred = 3160568 (3039f8 hex)
26.368 0.012: ## Booting kernel from Legacy Image at 82000000 ...
26.372 0.004: Image Name: Linux-3.2.0-gddb1881-dirty
26.380 0.008: Image Type: ARM Linux Kernel Image (uncompressed)
26.380 0.000: Data Size: 3160504 Bytes = 3 MiB
26.384 0.004: Load Address: 80008000
26.388 0.004: Entry Point: 80008000
27.304 0.916: Verifying Checksum ... OK
28.732 1.428: Loading kernel image ... OK
28.744 0.012: OK
28.744 0.000:
28.744 0.000: Starting kernel ...
28.744 0.000:
29.308 0.564: Uncompressing Linux... done, booting the kernel.
29.552 0.244: [ 0.000000] Linux version 3.2.0-gddb1881-dirty (chase@chase-ubu
ntu) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Aug 29 15:37:28 CDT
2012
29.560 0.008: [ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7),
cr=10c53c7d

```

- Impact: About 0.916 seconds
- Removable?: Maybe. Since we do nothing when an image checksum fails to match then we can remove this. If however you had a fallback mechanism for a bad kernel you would want to keep this in place.
- Difficulty: Easy. This verification can be controlled with a simple environment flag so it is an easy one to make.

12. Next you will see that we are spending time loading the kernel image to the proper location in DDR

```

25.356 1.992: #####
##
26.352 0.996: #####
26.352 0.000: done
26.356 0.004: Bytes transferred = 3160568 (3039f8 hex)
26.368 0.012: ## Booting kernel from Legacy Image at 82000000 ...
26.372 0.004: Image Name: Linux-3.2.0-gddb1881-dirty
26.380 0.008: Image Type: ARM Linux Kernel Image (uncompressed)
26.380 0.000: Data Size: 3160504 Bytes = 3 MiB
26.384 0.004: Load Address: 80008000
26.388 0.004: Entry Point: 80008000
27.304 0.916: Verifying Checksum ... OK
28.732 1.428: Loading Kernel Image ... OK
28.744 0.012: OK
28.744 0.000:
28.744 0.000: Starting kernel ...
28.744 0.000:
29.308 0.564: Uncompressing Linux... done, booting the kernel.
29.552 0.244: [ 0.000000] Linux version 3.2.0-gddb1881-dirty (chase@chase-ubu
ntu) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Aug 29 15:37:28 CDT
2012
29.560 0.008: [ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7),
cr=10c53c7d

```

- Impact: About 1.428 seconds
- Removable?: Yes. This step is being done because the default boot command is configured to load the kernel at a different address than where the kernel is configured to run from. Therefore u-boot must relocate the kernel in memory before booting.
- Difficulty: Easy. This can be fixed by changing the environment variables to use **kloadaddr** for the load address and bootm addresses rather than **loadaddr**

13. All told we have identified about 27.812 seconds of time that we can optimize. We will not get all of this time back because we will be trading some things like the long network load time for booting from NAND which will still have a copy time associated. But as a rough target this is a good estimate. If our total boot time is about 39.2 seconds and we can remove almost 27.8 seconds we will already have our boot time down to 11.4 seconds with just boot loader optimizations.

### Additional Notes

- If you are using the default SDK u-boot you will see that u-boot scans for any daughter cards on the I2C bus
  - Impact: These step is taking about 0.352 seconds
  - Removable?: When creating your own product you likely will not bother with I2C EEPROMs to scan to identify daughter cards. So this should be removable.

- Difficulty: Medium. We will need to remove the code that does the scan from the bootloader and recompile.

#### NOTE

In this lab we use the U-Boot from the [U-Boot Board Port lab](#) which already has this I2C scan removed.

## Finding Linux Kernel Optimizations[[edit](#)]

### Description[[edit](#)]

Now that you have an output from the boot process we can go through that output by scrolling back in history and reviewing the pieces of the boot that took the most time. This section will highlight areas in the Linux kernel that can be optimized as well as a estimate of the difficulty in reducing that time.

### Key Points[[edit](#)]

1. Identify steps in the Linux kernel that consume a large amount of time
2. Identify the begin and end of the Linux kernel steps
3. Build a list of the optimizations to make in later steps
4. `initcall_debug` is not used in this lab but can be used to help you better refine your kernel boot time optimization search
5. This is not a comprehensive list of all the optimizations that are possible. We are starting from a rather minimal kernel and it would be possible to cut this kernel down even further if needed but for the sake of this lab we will leave most of the kernel options alone and target the large ticket items and easy items.

### Lab Steps[[edit](#)]

1. Scroll up in your output to identify the beginning of the kernel boot process. This should look like the following

```

25.356 1.992: #####
##
26.352 0.996: #####
26.352 0.000: done
26.356 0.004: Bytes transferred = 3160568 (3039f8 hex)
26.368 0.012: ## Booting kernel from Legacy Image at 82000000 ...
26.372 0.004: Image Name: Linux-3.2.0-gddb1881-dirty
26.380 0.008: Image Type: ARM Linux Kernel Image (uncompressed)
26.380 0.000: Data Size: 3160504 Bytes = 3 MiB
26.384 0.004: Load Address: 80008000
26.388 0.004: Entry Point: 80008000
27.304 0.916: Verifying Checksum ... OK
28.732 1.428: Loading Kernel Image ... OK
28.744 0.012: OK
28.744 0.000:
28.744 0.000: Starting kernel ...
28.744 0.000:
29.308 0.564: Uncompressing Linux... done, booting the kernel.
29.352 0.244: [ 0.000000] Linux version 3.2.0-gddb1881-dirty (chase@chase-ubu
ntu) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Wed Aug 29 15:37:28 CDT
2012
29.560 0.008: [ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7),
cr=10c53c7d

```

2. Scroll down in the output until you find the beginning of the end of the Linux kernel boot. This should look like the following

```

33.812 2.980: [ 4.447113] PHY: 0:00 - Link is Up - 100/Full
35.068 1.256: [ 4.476531] Sending DHCP requests ., OK
35.076 0.008: [ 5.707000] IP-Config: Got DHCP answer from 0.0.0.0, my address
is 192.168.10.131
35.080 0.004: [ 5.714996] IP-Config: Complete:
35.088 0.008: [ 5.718383] device=eth0, addr=192.168.10.131, mask=255.255
.255.0, gw=192.168.10.1,
35.092 0.004: [ 5.726470] host=192.168.10.131, domain=, nis-domain=(none
),
35.101 0.009: [ 5.732849] bootserver=0.0.0.0, rootserver=192.168.10.130,
rootpath=
35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^MINIT: version 2.86 booting
35.324 0.032: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount

```

3. You have now identified the kernel boot step and can begin to look for areas of optimization.

### IMPORTANT

Since the kernel uses some buffers during output sometimes the tstamp tool is not optimal for finding the time taken by a kernel step since the tool tracks the time between new lines being printed. Luckily the kernel also has time stamping on its messages and you can use those time stamps to help you find the amount of time taken by a step. However, if you see a large delay in tstamp that is still a good area to focus your attention

4. The first optimization is the time taken to calibrate the delay loop

```
29.652 0.008: [ 0.000000] fixmap : 0xffff0000 - 0xffffe000 ( 896 kB)
29.656 0.004: [ 0.000000] vmalloc : 0xd0800000 - 0xff000000 ( 744 MB)
29.664 0.008: [ 0.000000] lowmem : 0xc0000000 - 0xd0000000 ( 256 MB)
29.668 0.004: [ 0.000000] modules : 0xbf000000 - 0xc0000000 ( 16 MB)
29.676 0.008: [ 0.000000] .text : 0xc0008000 - 0xc059c000 (5712 kB)
29.680 0.004: [ 0.000000] .init : 0xc059c000 - 0xc05da000 ( 248 kB)
29.688 0.008: [ 0.000000] .data : 0xc05da000 - 0xc0640998 ( 411 kB)
29.692 0.004: [ 0.000000] .bss : 0xc06409bc - 0xc066d764 ( 180 kB)
29.696 0.004: [ 0.000000] NR_IRQS:396
29.704 0.008: [ 0.000000] IRQ: Found an INTC at 0xfa200000 (revision 5.0) with
128 interrupts
29.708 0.004: [ 0.000000] Total of 128 interrupts on 1 active controller
29.716 0.008: [ 0.000000] OMAP clockevent source: GPTIMER2 at 24000000 Hz
29.720 0.004: [ 0.000000] OMAP clocksource: GPTIMER1 at 32768 Hz
29.728 0.008: [ 0.000000] sched_clock: 32 bits at 32kHz, resolution 30517ns,
wraps every 131071999ms
29.732 0.004: [ 0.000000] Console: colour dummy device 80x30
29.740 0.008: [ 0.000152] Calibrating delay loop... 718.02 BogoMIPS (lpj=3596
144)
29.744 0.004: [ 0.058563] pid_max: default: 32768 minimum: 301
29.748 0.004: [ 0.058685] Security Framework initialized
29.752 0.004: [ 0.058807] Mount-cache hash table entries: 512
29.760 0.008: [ 0.059173] CPU: Testing write buffer coherency: ok
```

- Impact: About 0.000152 seconds
  - Removable?: Yes
  - Difficulty: Easy. This does not save much time as it did in some previous devices, but this is a value that only needs to be calculated once in most cases. So adding `lpj=xxxxxxx` to the kernel bootargs is an easy way to save this time.
5. The next area of improvement is further down in the output and revolves around the networking required for NFS

```

30.820 0.004: [ 1.457122] PHY 0:01 not found
30.824 0.004: [ 1.461364] mmcblk0: mmc0:1234 SA04G 3.63 GiB
30.832 0.008: [ 1.468353] mmcblk0: p1 p2
33.812 2.980: [ 4.447113] PHY: 0:00 - Link is Up - 100/Full
35.068 1.256: [ 4.476531] Sending DHCP requests ., OK
35.076 0.008: [ 5.707000] IP-Config: Got DHCP answer from 0.0.0.0, my address
  is 192.168.10.131
35.080 0.004: [ 5.714996] IP-Config: Complete:
35.088 0.008: [ 5.718383]     device=eth0, addr=192.168.10.131, mask=255.255
.255.0, gw=192.168.10.1,
35.092 0.004: [ 5.726470]     host=192.168.10.131, domain=, nis-domain=(none
),
35.101 0.009: [ 5.732849]     bootserver=0.0.0.0, rootserver=192.168.10.130,
  rootpath=
35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^M^MINIT: version 2.86 booting
35.524 0.032: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
  Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes

```

- Impact: About 4.2 seconds
  - Removable?: Yes. By not bringing up the network interface in the kernel this time can be saved. A static IP would at least save about 1.252 seconds of time.
  - Difficulty: Medium/Hard. This change is listed as hard because it requires changing the system configuration to boot from something other than NFS. It is given a medium rating as well because this is not too difficult of a task depending on the location being used for the root file system. For example using a MMC root file system is easier and takes very little configuration vs. a NAND based root file system.
6. The final optimization is to observe how long the kernel took to print each line of output.
- Impact: About 1.5 seconds
  - Removable?: Yes. Using the quiet parameter
  - Difficulty: Easy. Simple adding the **quiet** parameter to the kernel bootargs will reduce the amount of kernel print messages.

#### NOTE

This is usually something you would do towards the end of your optimizations since you will also lose a lot of useful messages once this option is enabled. You can still see these messages using **dmesg** once the system is booted.



7. Overall we can save about 5.7 seconds on the boot time with the optimizations above. We will obviously not get back all of the time since it will take some amount of time to initialize other types of file systems as well.

## Finding File System Optimizations[[edit](#)]

### Description[[edit](#)]

Now that you have an output from the boot process we can go through that output by scrolling back in history and reviewing the pieces of the boot that took the most time. This section will highlight areas in the root file system that can be optimized as well as a estimate of the difficulty in reducing that time. We will also identify where we can inject our own steps to perform a simple splash screen display and touchscreen read in the boot process.

### Key Points[[edit](#)]

1. Identify steps in the root file system that consume a large amount of time
2. Build a list of the optimizations to make in later steps
3. Identify where to place our init script to start our application early in the boot process
  - If you have not already done the [Init Scripts Lab](#) you should do this to familiarize yourself with how init scripts work.
4. The root file system is the point at which we will add our splash screen application and be ready to receive touchscreen event.
  - The sooner we can add this in the boot process the sooner the system will be "booted"
5. You will see that we can make the system "boot" (become "available") quickly while continuing to load the rest of the system services in the background.

### Lab Steps[[edit](#)]

1. Scroll up in your output to identify the beginning of the root file system boot process. This should look like the following

```

33.812 2.980: [ 4.447113] PHY: 0:00 - Link is Up - 100/Full
35.068 1.256: [ 4.476531] Sending DHCP requests ., OK
35.076 0.008: [ 5.707000] IP-Config: Got DHCP answer from 0.0.0.0, my address
is 192.168.10.131
35.080 0.004: [ 5.714996] IP-Config: Complete:
35.088 0.008: [ 5.718383] device=eth0, addr=192.168.10.131, mask=255.255
.255.0, gw=192.168.10.1,
35.092 0.004: [ 5.726470] host=192.168.10.131, domain=, nis-domain=(none
),
35.101 0.009: [ 5.732849] bootserver=0.0.0.0, rootserver=192.168.10.130,
rootpath=
35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^MINIT: version 2.86 booting
35.524 0.052: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount

```

2. Scroll down in the output until you find the beginning of the end of the root file system boot. This should look like the following

```

36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount
37.588 0.108: ^MINIT: Entering runlevel: 5
37.756 0.168: Starting system message bus: dbus.
37.796 0.040: Starting telnet daemon.
37.848 0.052: Starting syslogd/klogd: done
37.928 0.080: Starting thttpd.
39.216 1.288:
39.224 0.008:
39.228 0.004: | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ |
39.232 0.004: | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ |
39.236 0.004: | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ | _ _ |
39.244 0.008:
39.244 0.000:
39.248 0.004: Arago Project http://arago-project.org am335x-evm tty00
39.248 0.000:
39.252 0.004: Arago 2011.09 am335x-evm tty00
39.252 0.000:

```



3. You have now identified the file system boot step and can begin to look for areas of optimization.
4. The first are taking a large amount of time is the init process

```

35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^MINIT: version 2.86 booting
35.524 0.032: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount
37.588 0.108: ^MINIT: Entering runlevel: 5
37.756 0.168: Starting system message bus: dbus.
37.796 0.040: Starting telnet daemon.
37.848 0.052: Starting syslogd/klogd: done
37.928 0.080: Starting thttpd.
39.216 1.288:
39.224 0.008:
39.228 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.232 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.236 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.244 0.008:

```

- Impact: About 0.256 seconds
  - Removable?: No. It is possible to modify the init process but usually not required or recommended. This would require a rebuild of the entire init process.
  - Difficulty: Hard. Requires modifying the base process for the entire Linux system.
5. The first process being run is S01psplash which is responsible for showing the TI logo and progress bar on the screen. Instead we can replace this with our own custom image and script to read touchscreen events. So we can **mark this as the location to add our application start script** and remove the psplash script.
  6. The following items are listed to give you an idea of other areas you can impact boot time but not necessarily items we are going to optimize

### IMPORTANT

The reason we are not optimizing these items is because it is OK for those items to run in the background since the system is already "booted". If you wanted to remove those items you would usually just remove the init script that calls the process

7. The next step taking time is the initialization of the udev process.

```
35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^MINIT: version 2.86 booting
35.524 0.032: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw, not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount
37.588 0.108: ^MINIT: Entering runlevel: 5
37.756 0.168: Starting system message bus: dbus.
37.796 0.040: Starting telnet daemon.
37.848 0.052: Starting syslogd/klogd: done
37.928 0.080: Starting thttpd.
39.216 1.288:
39.224 0.008:
39.228 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.232 0.004: | | | | | | | | | | | | | | | | | | | | | | | | | | | |
39.236 0.004: | | | | | | | | | | | | | | | | | | | | | | | | | | | |
39.244 0.008:
```

- Impact: About 1.172 seconds
  - Removable?: Maybe. You can remove udev or use a different manager like mdev if you are working with a static system. However, if you are wanting to be able to attach new devices to the system udev is recommended.
  - Difficulty: Medium. It is easy to not start udev but modifying your system to handle all the attached devices can be difficult.
8. Next we see that caching the udev nodes is taking a good chunk of time.

```
35.132 0.031: [ 5.765106] VFS: Mounted root (nfs filesystem) on device 0:13.
35.136 0.004: [ 5.771575] Freeing init memory: 248K
35.268 0.256: ^MINIT: version 2.86 booting
35.524 0.032: Please wait: booting...
35.604 0.080: Starting udev
36.776 1.172: [ 7.406311] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
36.844 0.068: Root filesystem already rw. not remounting
36.856 0.012: Caching udev devnodes
37.376 0.520: ALSA: Restoring mixer settings...
37.448 0.072: /usr/sbin/alsactl: load_state:1625: No soundcards found...
37.480 0.032: NOT configuring network interfaces: / is an NFS mount
37.588 0.108: ^MINIT: Entering runlevel: 5
37.756 0.168: Starting system message bus: dbus.
37.796 0.040: Starting telnet daemon.
37.848 0.052: Starting syslogd/klogd: done
37.928 0.080: Starting thttpd.
39.216 1.288:
39.224 0.008:
39.228 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.232 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.236 0.004: | _ _ _ _ _ | _ _ _ _ _ | _ _ _ _ _ |
39.244 0.008:
```

- Impact: About 0.520 seconds
  - Removable?: Maybe. If we are not using udev you can remove the caching of the udev nodes
  - Difficulty: Easy. Just remove the init script
9. One of the last big consumers of time is the creation of the shell.



## Key Points[\[edit\]](#)

1. Placement of the splash screen early in the boot process allows the system to be "booted" quickly.
2. The rest of the system can continue to boot in the background while waiting for the user to touch the screen.

## Lab Steps[\[edit\]](#)

1. Start minicom using the following command

**minicom -w**

### IMPORTANT

The above command launches minicom using the defaults and with line wrap enabled. If you do not see output during the following steps you will likely need to use **minicom -s -w** to allow reconfiguring the serial port to use

2. Press **Enter** to bring up the login prompt and login using:

Username: **root**

3. First we need to remove the existing *psplash* init script which is displaying the progress bar during boot. This can be done by simply removing the symlink **S01psplash** from **/etc/rcS.d**
  - **rm /etc/rcS.d/S01psplash**
4. We now need to create our own script for doing the splash operation and waiting for a touchscreen event
  - **vi /etc/init.d/mysplash.sh** and type

```
#!/bin/sh
cat /home/root/startup.rawr16 > /dev/fb0
head -c 1 /dev/input/event0 > /dev/null
cat /dev/zero > /dev/fb0
```

### NOTE

The above script does a raw write of a 16 bit per pixel to the frame buffer. It then waits to read 1 byte of input from the input device node to determine when the touchscreen is pressed. At that point it blanks the display.

### NOTE

The startup.rawr16 image will be copied later

5. Because we want to run the above script and allow the rest of the system to continue booting we will create a wrapper script that just calls this script and has it run in the background.

- **vi /etc/init.d/run-mysplash.sh** and type

```
#!/bin/sh
echo "Starting splash screen and waiting for touchscreen"
/etc/init.d/mysplash.sh &
```

#### NOTE

In the above script the echo line is used so that we can get a timestamp of when our splash operation started. The **&** is required to background the mysplash.sh process

6. Make the scripts executable
  - **chmod +x /etc/init.d/mysplash.sh**
  - **chmod +x /etc/init.d/run-mysplash.sh**
7. **On your Linux Host:** Open a terminal window and copy the **startup.rawr16** file to the /home/root directory of the NFS share
  - **sudo cp /home/sitara/sitara-training-helper-files/boot\_time\_optimization/startup.rawr16 /home/sitara/<SDK install dir>/targetNFS/home/root**
8. Now back in your **target console** link this script into the init process flow at the location where *psplash* used to be.

```
cd /etc/rcS.d/
ln -s ../init.d/run-mysplash.sh S01run-mysplash.sh
```

9. This should now make this one of the first processes to run on system boot. Reboot the board using the following command and observe where the splash screen is started
  - **init 6**
10. After the reboot you should see the splash screen up waiting for a touchscreen press and on the serial console you should see that they system has continued to boot in the background.

## Writing the File System to SD[[edit](#)]

### Description[[edit](#)]

Now that we have finished optimizing the root file system for this lab we can go ahead and write the file system to the SD card partition reserved for the file system (*/dev/mmcbk0p2*). In this case the beaglebone acts as our SD card reader rather than having to use our Linux host system.

## Key Points[\[edit\]](#)

- How to make a tarball of an existing NFS file system
- How to mount an SD card partition on the target board
- Extracting a file system to the SD card
- Modifying the bootargs to boot using the root file system on the SD card

## Lab Steps[\[edit\]](#)

1. The first step is to make a tarball of our existing NFS image. Shutdown the board using **init 0** and use the following commands on the **Linux Host** to make a file system tarball
  - **cd /home/sitara/<SDK install dir>/targetNFS**
  - **sudo tar czf ../base-file-system.tar.gz \***

### IMPORTANT

The above command will make a file system tarball that extracts into the current directory. The `../` is used so that the tarball being created is NOT picked up as part of the tar process

2. Now copy the **base-file-system.tar.gz** tarball back into the NFS file system so it will be available on the target
  - **sudo cp ../base-file-system.tar.gz .**
3. Reset the target board. Once the board has booted, on the target console login using **root**
4. Re-Format the SD card root file system partition. This will erase the SDK file system and make room for our file minimal file system instead.
  - **mkfs.ext3 -L rootfs /dev/mmcbk0p2**

### NOTE

The `"-L rootfs"` option was used to give the file system volume a human readable name

5. Mount the second partition of the SD card, which was reserved for the file system to the **/media/mmc1** mount point
  - **mount /dev/mmcbk0p2 /media/mmc1**
6. Now cd to the mounted file system and extract the root file system tarball into the blank file system.
  - **cd /media/mmc1**
  - **tar xzf /base-file-system.tar.gz**
7. Now you can unmount the file system and perform a sync operation to make sure everything is flushed to the SD card
  - **cd ..**
  - **umount mmc1**

- **sync**
8. Now we will mount the SD card boot partition and modify the uEnv.txt so that we can boot this new SD file system instead of the NFS.
- **mount /dev/mmc0blk0p1 /media/mmc1**
  - **vi /media/mmc1/uEnv.txt** and make the following changes
    - Delete the following line: (You can scroll to the line and press **d** + **d** to delete the line)

**tftp\_nfs\_boot=.....**

- Add the following line:

**tftp\_mmc\_boot=dhcp \${loadaddr} \${bootfile}; run mmc\_args; bootm  
\${loadaddr}**

- Change the uenvcmd to

**uenvcmd=tftp\_mmc\_boot**

9. The resulting file should look like:

```
serverip=xxx.xxx.xxx.xxx  
rootpath=/xxx/xxx/xxx/  
bootfile=uImage-am335x-evm.bin  
ip_method=dhcp  
uenvcmd=run tftp_mmc_boot  
tftp_mmc_boot=dhcp ${loadaddr} ${bootfile}; run mmc_args; bootm  
${loadaddr}
```

10. Unmount the MMC device and reboot the board

- **umount mmc1**
- **init 6**

11. You should now see the board boot using the file system on the SD card.

## Optimizing the Linux Kernel[\[edit\]](#)

### Description[\[edit\]](#)

This section will cover making optimizations to the Linux kernel that will save on boot time. This kernel is in some ways already reduced since many interfaces are not defined in the board port kernel we are using but configuration options could be optimized further depending on your target system. The optimizations done in this lab will mostly involve changing the settings passed to the kernel during boot.



## Key Points[\[edit\]](#)

- Many optimizations can be as simple as changing the kernel boot settings
- Setting lpj value
- Changing IP method
- Avoiding kernel message prints

## Lab Steps[\[edit\]](#)

1. You should at this point have your board booted to the SD file system flashed in the previous lab. Go ahead and login as **root**.
2. Mount the SD card boot partition so that we can modify the uEnv.txt file to affect the parameters passed to the Linux kernel
  - **cd /media**
  - **mount /dev/mmcblk0p1 mmc1**
3. Make a copy of your uEnv.txt file for backup purposes
  - **cp mmc1/uEnv.txt mmc1/uEnv.txt.orig**
4. Edit the **mmc1/uEnv.txt** file and make the following changes
  - **vi /media/mmc1/uEnv.txt**
    - Add the line

**optargs=quiet lpj=3590144**

- Delete following lines

**ip\_method=dhcp**  
**rootpath=xxxxxx**

- The resulting file should look like:

```
serverip=xxx.xxx.xxx.xxx
bootfile=uImage-am335x-evm.bin
uenvcmd=run tftp_mmc_boot
tftp_mmc_boot=dhcp ${loadaddr} ${bootfile}; run mmc_args; bootm
${loadaddr}
optargs=quiet lpj=3590144
```

### 5. NOTE

The above modifications set the quiet option to reduce kernel printing, pre-set the loops per jiffie value to avoid needing to calculate this again, and most importantly removed the NFS settings including the ip\_method setting. This means that the kernel will not request an IP while booting, but the file system can request the IP during start up AFTER our splash screen has started

6. Un-mount the SD card and reboot the board
  - **umount mmc1**
  - **init 6**

7. This time you should notice that there was very little output once the kernel was booting. You also did not have the delay associated with obtaining and IP address in the kernel.

## Writing the Linux Kernel to SD[[edit](#)]

### Description[[edit](#)]

This section will cover how to place the Linux kernel on the SD card and configure u-boot to read the kernel from the FAT partition on the SD card. This will allow us to stop downloading the kernel over the ethernet connection, which is a time consuming operation and instead load the kernel from the SD device which should be faster.

#### NOTE

The current u-boot implementation for NAND is not optimized and yields a slow kernel read (about 1MB/s). This is not what you would want for a fast boot so instead an SD card is used which has a high speed driver available. There are instructions for flashing the kernel to NAND at the end of this lab as an information item

### Key Points[[edit](#)]

- How to write the kernel to SD
- Changing u-boot settings to boot the kernel from SD

### Lab Steps[[edit](#)]

1. Assuming the system is booted login using **root**
2. Download the kernel image to your local file system. You can find the serverip to use by running the **ifconfig** command on your host.
  - **ftftp -g -r uImage-am335x-evm.bin <serverip>**

#### NOTE

The serverip used is the same as the one used in your uEnv.txt file

3. Mount the SD card
  - **mount /dev/mmcblk0p1 /media/mmc1**
4. Copy the downloaded **uImage-am335x-evm.bin** to the SD card as **uImage**
  - **cp uImage-am335x-evm.bin /media/mmc1/uImage**

#### IMPORTANT

The kernel image should be named **uImage** because this is the default name that u-boot will look to load from the SD card boot partition

- **vi /media/mmc1/uEnv.txt** and make the following changes
  - Remove these lines

**serverip=xxxx** This is not needed because we will not be booting from tftp  
**bootfile=xxxx** Again, we will not be booting from tftp so this is not required  
**tftp\_mmc\_boot** Since we are now going to boot the kernel from the SD card and use the file system from SD card we can now use the built in u-boot defaults

- Modify the uenvcmd line to look like

**uenvcmd=run mmc\_boot**

- The resulting file should look like:

```
uenvcmd=run mmc_boot
optargs=quiet lpj=3590144
```

### IMPORTANT

There is a subtle optimization here that will save you over a second. Using the built in defaults for **mmc\_boot** the board is now set to read the uImage from the SD card to the address **kloadaddr**. This address is configured such that the image is written into DDR at the proper kernel start location. This means that the kernel does not need to be relocated from one place in DDR to another which saves about 1.4 seconds as we identified in the labs above

5. Un-mount the SD card and reboot your board using the following commands
  - **umount /media/mmc1**
  - **init 6**
6. You should now see the device boot reading the kernel from the SD card

### NOTE

You should no longer see the #'s that marked the tftp of the kernel which is a good sign it is being read from elsewhere

## Optimizing the U-Boot/SPL [[edit](#)]

### Description [[edit](#)]

This section is going to cover making some of the optimizations that were identified previously. Some of these optimizations were already done by modifying where the kernel is loaded from and how it is booted. The rest of these optimization will be done my modifying the actual u-boot sources and configuration for the desired result.

## Key Points[\[edit\]](#)

- Disabling the NAND environment check
- Removing image verification
- Programming default U-boot environment
- Changing default boot command
- Reducing the boot delay
- Disabling ethernet initialization

## Lab Steps[\[edit\]](#)

1. The following steps will walk through modifying the u-boot sources to optimize your boot time. On your *Linux Host* go to the u-boot board-port tree
  - **cd /home/sitara/board-port/sitara-board-port-uboot**
2. This step will modify the **include/configs/am335x\_evm.h** to Disable the NAND environment check. This step was identified as taking about 0.764 seconds of boot time. Since we are going to modify the default boot arguments anyway there is no need for the NAND environment to be saved.
  - **gedit include/configs/am335x\_evm.h**
    - Search for the following lines (You can use **ctrl+F** to do a search in gedit):

```
/* NAND support */  
#ifdef CONFIG_NAND  
#define CONFIG_CMD_NAND
```

- Just above these lines add the following lines:

```
#undef CONFIG_NAND_ENV  
#define CONFIG_ENV_IS_NOWHERE
```

- Your file should now look like:

```
#undef CONFIG_NAND_ENV  
#define CONFIG_ENV_IS_NOWHERE
```

```
/* NAND support */  
#ifdef CONFIG_NAND  
#define CONFIG_CMD_NAND
```

### 3. IMPORTANT

Adding this optimization will currently break reading Linux kernel images from NAND in u-boot. The system needs to do the configuration of the NAND for the environment to be ready to read the images such as the Linux kernel. This will be addressed in the future.

#### 4. IMPORTANT

The lines added were placed in the file near the NAND support structure for convenience. The effect is to unset the U-boot environment being saved in NAND. It is also important to tell u-boot that since the environment is not in NAND that it is located NOWHERE

5. This next modification will disable the network driver initialization in u-boot. Since we are no transferring the kernel over tftp this is the remainder of the networking boot impact left to be removed.

#### NOTE

This does not prevent the kernel or file system from starting the network later

- Search for the following line

```
#define CONFIG_DRIVER_TI_CPSW
```

- Remove that line from the configuration file
6. Next we are going to turn off the image verification. As mentioned in the steps identifying the optimizations, the image verify takes about a second and since we don't do anything with a failed image we will not bother to verify it.

- Search for the following lines:

```
#define CONFIG_EXTRA_ENV_SETTINGS \  
"bootfile=uImage\0" \  
"loadaddr=0x82000000\0" \  
"
```

- Add the following line after bootfile=uImage

```
"verify=n\0" \  
"
```

#### NOTE

This line can be added anywhere in the CONFIG\_EXTRA\_ENV\_SETTINGS variable, the location picked here was arbitrary.

7. The next modification to the default U-boot environment will allow us to no longer depend on the uEnv.txt file for setting parameters. This means we will be able to stop doing the read of that file from the SD card.

- Search for the following line:

```
"optargs=\0" \  
"
```

- Modify this line to instead set:

```
"optargs=quiet lpj=3590144\0" \  
"
```

8. Now we will configure `BOOTDELAY` to remove the 3 seconds we are seeing during boot
  - Search for the following lines:

```
#ifndef CONFIG_RESTORE_FLASH  
/* set to negative value for no autoboot */  
#define CONFIG_BOOTDELAY 3
```

- Change the `CONFIG_BOOTDELAY` line to:

```
#define CONFIG_BOOTDELAY 1
```

## 9. IMPORTANT

There are two definitions of `CONFIG_BOOTDELAY` in the default configuration file. This is because for flashing/restore operations we want no boot delay and to use a different boot command. This is why the **`#ifndef CONFIG_RESTORE_FLASH`** flag is used to wrap the `CONFIG_BOOTDELAY` and `CONFIG_BOOTCOMMAND` variables

## 10. IMPORTANT

You may be asking why we set the `CONFIG_BOOTDELAY` to 1 instead of 0. This is because if we make any mistakes we still have a chance to interrupt u-boot and interact with the system. So for this lab we will end up taking the boot time and subtracting 1 second. In your production environment once you are happy with the system you would likely change this to a 0 to remove that one second boot time impact

11. Lastly we will define the custom boot command so that we do not need to use the `uEnv.txt` file anymore
  - Search for the following lines

```
#define CONFIG_BOOTCOMMAND \  
"if mmc rescan; then " \  
"echo SD/MMC found on device ${mmc_dev};" \
```

- This whole indented block of code is one long string for the `CONFIG_BOOTCOMMAND` variable. Delete this entire string and replace it with the following lines:

```
#define CONFIG_BOOTCOMMAND \  
"mmc rescan; run mmc_boot"
```

## IMPORTANT

We had to add the **`mmc rescan`** to the boot command here because the rescan had not happened before. In the `uEnv.txt` file the rescan had already happened

because of the default boot command and we did not need to perform the rescan a second time

- You final code should look like:

```
/* set to negative value for no autoboot */
#define CONFIG_BOOTDELAY 1

#define CONFIG_BOOTCOMMAND \
"mmc rescan; run mmc_boot"
```

12. You can now **save** your file and exit gedit

13. Use the following commands to configure and compile a new **u-boot.img** and **MLO** image

- **make ARCH=arm CROSS\_COMPILE=/home/sitara/ti-sdk-am335x-evm-<sdk version>/linux-devkit/bin/arm-arago-linux-gnueabi-am335x\_evm\_config**

### IMPORTANT

There is a space between arm-arago-linux-gnueabi- and am335x\_evm\_config

- **make ARCH=arm CROSS\_COMPILE=/home/sitara/ti-sdk-am335x-evm-<sdk version>/linux-devkit/bin/arm-arago-linux-gnueabi-**

14. When the build finishes you should now have a new **MLO** and **u-boot.img** file in the **/home/sitara/board-port/sitara-board-port-uboot/** directory.

15. For the next steps you should go ahead and copy these files to the **/tftpboot** directory so they can be downloaded to the target device.

- **cp MLO /tftpboot/MLO-optimized**
- **cp u-boot.img /tftpboot/u-boot-optimized**

## Writing the U-Boot/SPL to SD[[edit](#)]

### Description[[edit](#)]

This section will cover transferring the new MLO and u-boot.img that you created to the SD card

### Key Points[[edit](#)]

- How to tftp the images to the target file system
- Using the target file system to access the SD card

## Lab Steps[\[edit\]](#)

1. Your target board should be booted and you should login as **root**
2. TFTP you optimized boot loader files. Remember that you can get the TFTP server address by using the **ifconfig** command on your Linux host.
  - **tftp -g -r MLO-optimized <server ip>**
  - **tftp -g -r u-boot-optimized <server ip>**
3. Mount the boot partition of the SD card
  - **mount /dev/mmcblk0p1 /media/mmc1**
4. Copy the new boot loader files to the boot partition of the SD card.
  - **cp MLO-optimized /media/mmc1/MLO**
  - **cp u-boot-optimized /media/mmc1/u-boot.img**
5. Unmount the boot partition
  - **umount /media/mmc1**
6. Reboot the board
  - **init 6**
7. On this boot you should have noticed that the boot delay went down to only 1 second, that the network was not initialized, that the uEnv.txt file was not being read from the SD card, and that the kernel image verify was disabled.

## Measuring the Final Boot Time[\[edit\]](#)

### Description[\[edit\]](#)

In this lab we will re-run the tstamp program and observe the change in boot time to get to an initial splash screen

### Key Points[\[edit\]](#)

- Measuring the changed boot time
- Observing the touch event being processed

## Lab Steps[\[edit\]](#)

1. Assuming the target board is booted login as **root**
2. Shutdown the target board so that the root file system is cleanly un-mounted and the target is ready for a reset
  - **init 0**
3. Once you see the following message your board is shutdown
  - **[ xxx.xxxxxx] System halted.'**
4. Exit your minicom terminal using the following key commands
  - **Ctrl+A**
  - **z**



- x
  - **Select Yes to exit minicom**
5. Make sure your serial port is properly configured using the stty command
    - **stty -F /dev/ttyUSB1 115200**
  6. You can now use the tstamp program to measure boot times. To do this use the command
    - **cat /dev/ttyUSB1 | tstamp**
  7. Press the **Reset** button on the top-right corner of the LCD cape and let the system boot
  8. Look for the message we printed when we started the splash screen. You should see output like:

```

2.248 0.000:   Load Address: 80008000
2.252 0.004:   Entry Point:  80008000
2.260 0.008:   XIP Kernel Image ... OK
2.264 0.004: OK
2.264 0.000:
2.264 0.000: Starting kernel ...
2.264 0.000:
2.828 0.564: Uncompressing Linux... done, booting the kernel.
3.028 0.200: [   0.139038] mtdoops: mtd device (mtddev=name/number) must be supplied
INIT: version 2.06 booting
3.736 0.156: Starting splash screen and waiting for touchscreen
3.740 0.012: Please wait. booting...
3.828 0.080: Starting udev
4.872 1.044: Remounting root file system...
4.936 0.064: Caching udev devnodes
5.276 0.340: ALSA: Restoring mixer settings...
5.344 0.068: /usr/sbin/alsactl: load_state:1625: No soundcards found...
5.428 0.084: Configuring network interfaces... [   2.543853] PHY 0:01 not found
5.440 0.012: udhcpc (v1.13.2) started
5.492 0.052: Sending discover...
8.496 3.004: Sending discover...
11.500 3.004: Sending discover...
12.096 0.596: Sending select for 192.168.10.133...

```

### IMPORTANT

Why is the boot time 3.736 seconds? Weren't we trying to get under 3 seconds? Remember that we left a second delay in u-boot to allow us to get to a u-boot prompt in case something went wrong. if you scroll back in the boot history you will see that second delay is still there. So if you delete 1 second from the boot time the total time is **2.736 Seconds**, so we actually beat our goal.

9. Go ahead and touch the touchscreen. You should see the panel go blank.
10. Congratulations, you have managed to create a system that can boot in under 3 seconds.
11. In su

## Further Notes[[edit](#)]

- In this lab we still left the OneNAND driver in the Linux kernel to enable NAND support. If you are booting completely from SD card you could also remove this driver from the Linux kernel configuration.

```
30.080 0.008: [ 0.713348] i2c-core: driver [tsl2550] using legacy resume method
30.088 0.008: [ 0.721282] mtdoops: mtd device (mtddev=name/number) must be supplied
30.092 0.004: [ 0.728637] omap2-nand driver initializing
30.096 0.004: [ 0.733245] ONFI flash detected
30.104 0.008: [ 0.736846] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xcc (Micron NAND 512MiB 3,3V 16-bit)
30.112 0.008: [ 0.746063] Creating 8 MTD partitions on "omap2-nand.0":
30.116 0.004: [ 0.751647] 0x000000000000-0x000000020000 : "SPL"
30.124 0.008: [ 0.758026] 0x000000020000-0x000000040000 : "SPL.backup1"
30.132 0.008: [ 0.764923] 0x000000040000-0x000000060000 : "SPL.backup2"
30.136 0.004: [ 0.771850] 0x000000060000-0x000000080000 : "SPL.backup3"
30.144 0.008: [ 0.778747] 0x000000080000-0x000000260000 : "U-Boot"
30.152 0.008: [ 0.785949] 0x000000260000-0x000000280000 : "U-Boot Env"
30.156 0.004: [ 0.792785] 0x000000280000-0x000000780000 : "Kernel"
30.168 0.012: [ 0.801300] 0x000000780000-0x000020000000 : "File System"
30.380 0.212: [ 1.018005] OneNAND driver initializing
30.388 0.008: [ 1.023400] CAN device driver interface
30.392 0.004: [ 1.027465] CAN bus driver for Bosch D_CAN controller 1.0
30.444 0.052: [ 1.075897] davinci_mdio davinci_mdio.0: davinci mdio revision 1.6
```

- Impact: About 0.212 seconds
- Removable?: Yes, if you do not plan on using NAND in Linux both as a root file system or for additional persistent storage.
- Difficulty: Medium. You would need to re-configure to disable the **Device Drivers -> Memory Technology Device (MTD) support -> OneNAND Device Support** option and re-compile the kernel.
- There is currently work going on in the U-Boot community to allow the SPL image to directly boot the Linux kernel without loading U-Boot first. This support is not available today but in the future may enable saving another ~1.2 seconds during boot.

## Additional Methods[[edit](#)]

These methods can be used in place of the SD card methods above to flash the components to the NAND rather than to the SD card. In this lab the SD card was chosen because it has good throughput, particularly in u-boot in the which the NAND

performance is not currently optimized. It is also worth noting that it takes about .5 seconds to scan and mount a 32MB UBI/UBIFS NAND volume. UBI is faster than JFFS2 but it scales linearly (see [http://www.linux-mtd.infradead.org/doc/ubi.html#L\\_scalability](http://www.linux-mtd.infradead.org/doc/ubi.html#L_scalability)). This is in part because unlike in an MMC/SD card we are the controller and are responsible for scanning the NAND, etc rather than relying on another controller to do this in parallel like what you see with a MMC/SD card which is a Flash Translation Layer device.

Still, for completeness these steps were added here to provide a reference for how to perform this lab with different media.

#### NOTE

The performance of SD cards can vary depending on the card. You should test multiple cards and find the one with the best performance. This lab used the card that shipped with the EVM

## Flashing the File System to NAND with UBIFS[[edit](#)]

### Description[[edit](#)]

Now that we have finished optimizing the root file system for this lab we can go ahead and write the file system to the NAND partition reserved for the file system.

### Key Points[[edit](#)]

- How to make a tarball of an existing NFS file system
- How to mount a NAND partition as a UBIFS file system
- Copying file to UBIFS
- Modifying the bootargs to boot using the root file system in UBIFS
- Additional information for creating UBIFS images and other methods for doing so can be found at:
  - [http://processors.wiki.ti.com/index.php/UBIFS\\_Support](http://processors.wiki.ti.com/index.php/UBIFS_Support)
  - <http://www.sakoman.com/OMAP/how-to-write-an-ubifs-rootfs-image-to-nand.html>

### Lab Steps[[edit](#)]

1. The first step is to make a tarball of our existing NFS image. Shutdown the board using **init 0** and use the following commands on the **Linux Host** to make a file system tarball
  - **cd /home/sitara/<SDK install dir>/targetNFS**
  - **sudo tar czf ../base-file-system.tar.gz \***

## IMPORTANT

The above command will make a file system tarball that extracts into the current directory. The `../` is used so that the tarball being created is NOT picked up as part of the tar process

2. Now copy the **base-file-system.tar.gz** tarball back into the NFS file system so it will be available on the target
  - **sudo cp ../base-file-system.tar.gz .**
3. Reset the target board. Once the board has booted, on the target console login using **root**
4. If you scroll back in the kernel boot messages you will see the partition definitions the kernel is using for the NAND. In this case the output should look like the output below which shows that the File System partition is the last partition. Since the numbering is zero based this would make this partition **7**. Remember this for the next steps.

```
[ x.xxxxxx] 0x000000000000-0x000000020000 : "SPL"  
[ x.xxxxxx] 0x000000020000-0x000000040000 : "SPL.backup1"  
[ x.xxxxxx] 0x000000040000-0x000000060000 : "SPL.backup2"  
[ x.xxxxxx] 0x000000060000-0x000000080000 : "SPL.backup3"  
[ x.xxxxxx] 0x000000080000-0x000000260000 : "U-Boot"  
[ x.xxxxxx] 0x000000260000-0x000000280000 : "U-Boot Env"  
[ x.xxxxxx] 0x000000280000-0x000000780000 : "Kernel"  
[ x.xxxxxx] 0x000000780000-0x000002780000 : "File System"  
[ x.xxxxxx] 0x000002780000-0x000002960000 : "U-Boot.backup"  
[ x.xxxxxx] 0x000002960000-0x000002e60000 : "Kernel.backup"  
[ x.xxxxxx] 0x000002e60000-0x000020000000 : "Extended File System"
```

5. First we will erase the flash partition
  - **flash\_erase /dev/mtd7 0 0**

## NOTE

You may see some bad block messages or messages saying some blocks could not be erased. This is OK

6. Next we will format the UBI mtd device for UBIFS with a sub page size of 512
  - **ubiformat /dev/mtd7 -s 512 -O 2048**

## IMPORTANT

Since we specify an sub page size of 512 which is used for writing erase blocks we need to specify the VID header offset (using the `-O` option) as 2048 to make sure it is in the next erase block and now in the first erase block

7. Next we tell UBI that we are going to be working with mtd device 7 using **ubiattach**
  - **ubiattach /dev/ubi\_ctrl -m 7 -O 2048**

#### NOTE

Specifying /dev/ubi\_ctrl is not strictly necessary as this is the default but is used here for safety sake.

#### IMPORTANT

The UBI device number is assigned automatically and will be used in the next step. This is assigned starting from 0 and incrementing by 1. So for the first UBI device this will be /dev/ubi0 and the second will be /dev/ubi1, and so forth. You can find which ubi device this has been attached to in the first line of output from the **ubiattach** command

#### IMPORTANT

We use the -O 2048 option to match the location of the VID header given when we formatted the UBIFS

8. You may need to press **Enter** to return to a command prompt
9. Now we will create a UBIFS volume using **ubimkvol**
  - **ubimkvol /dev/ubi0 -N rootfs -m'**

#### NOTE

The above command makes a UBI volume on UBI device ubi0 with a label of **rootfs** and taking up the full size of the device

#### IMPORTANT

If you use a different label than **rootfs** you will also need to change the default u-boot environment to use this new label since it uses rootfs by default.

10. Now you will create a mount point for the UBIFS NAND file system and mount the file system
  - **mkdir /mnt/nand**
  - **mount -t ubifs ubi0:rootfs /mnt/nand**

#### NOTE

You may see some read errors during the mount operation. These can be ignored

11. You may need to press **Enter** to return to a command prompt
12. Now cd to the mounted file system and extract the rootfs tarball into the UBIFS mount.
  - **cd /mnt/nand**

- **tar xzf /base-file-system.tar.gz**
13. Now you can unmount the file system and perform a sync operation to make sure everything is flushed to NAND
- **cd ..**
  - **umount nand**
  - **sync**
14. Now we will mount the SD card and modify the uEnv.txt so that we can boot this new NAND file system.
- **mkdir /mnt/mmc**
  - **mount /dev/mmcblk0p1 /mnt/mmc**
  - **vi /mnt/mmc** and make the following changes
    - Add the following line:

```
tftp_nand_boot=dhcp ${loadaddr} ${bootfile}; run nand_args; bootm  
${loadaddr}
```

- Change the uenvcmd to

```
uenvcmd=tftp_nand_boot
```

15. Unmount the MMC device and reboot the board
- **umount mmc**
  - **init 6**
16. You should now see the board boot using the file system in the NAND flash. You can verify this using the **mount** command and looking for a line like:

```
ubi0:rootfs on / type ubifs (rw,relatime)
```

## Flashing the Linux Kernel to NAND[\[edit\]](#)

### Description[\[edit\]](#)

This section will cover how to place the Linux kernel into the NAND flash. This will allow us to stop downloading the kernel over the ethernet connection, which is a time consuming operation and instead load the kernel from the NAND device which should be faster.

### Key Points[\[edit\]](#)

- How to flash the kernel to NAND
- Changing u-boot settings to reflect the actual image size
- Copying and booting the kernel from NAND

## Lab Steps[[edit](#)]

### From Linux[[edit](#)]

1. Assuming the system is booted login using **root**
2. Determine the **Kernel** flash partition.
  - **cat /proc/mtd**

You should expect output like:  
mtd6: 00500000 00020000 "Kernel"

#### NOTE

You will see the full list of the mtd partition when using doing **cat /proc/mtd** and can use the output to find the /dev/mtd device node corresponding to each partition

3. Erase the **Kernel** flash partition.
  - **flash\_erase /dev/mtd6 0 0**

#### IMPORTANT

The device node to erase was based on the mtd device found for the "Kernel" partition in the /proc/mtd output

4. Download the kernel image to your local file system. You can find the serverip to use by running the **ifconfig** command on your host.
  - **tftp -g -r uImage-am335x-evm.bin <serverip>**

#### NOTE

The serverip used is the same as the one used in your uEnv.txt file

5. Write the kernel image using the following command
  - **nandwrite -p /dev/mtd6 uImage-am335x-evm.bin**

#### IMPORTANT

You need to use the -p option to pad the kernel image to a full page size or else you will see write errors

6. The result of the write operation is that the last line will tell you the offset of the last page written. i.e.:

*Writing data to block 24 at offset 0x300000*

7. If you take this last offset and add one page size (0x20000) you will have the size of the kernel image that needs to be copied from NAND. In this case **0x320000**. Remember this number for the following steps.
8. In u-boot the default size of the kernel to load is 0x500000. We do not need to actually copy this much data from the NAND to the DDR so any extra data copy is actually a waste of time. So now you will update the uEnv.txt file to overwrite the default value with the **0x320000** value we found above.
  - **mount /dev/mmcbk0p1 /media/mmc1**
  - **vi /media/mmc1/uEnv.txt** and make the following changes
    - Remove these lines

**serverip=xxxx** This is not needed because we will not be booting from tftp  
**bootfile=xxxx** Again, we will not be booting from tftp so this is not required  
**tftp\_nand\_boot** Since we are now going to boot the kernel from NAND and use the file system from NAND we can now use the built in u-boot defaults

- Modify the uenvcmd line to look like

**uenvcmd=run nand\_boot**

- Add the following line to change the size of the kernel loaded

**nand\_img\_siz=0x320000**

9. Un-mount the SD card and reboot your board using the following commands
  - **umount /media/mmc1**
  - **init 6**
10. You should now see the device boot reading the kernel from NAND

#### NOTE

You should no longer see the #'s that marked the tftp of the kernel which is a good sign it is being read from elsewhere

#### From U-Boot[\[edit\]](#)

1. Assuming the system is booted login using **root**
2. Determine the **Kernel** flash partition.
  - **cat /proc/mtd**

You should expect output like:

*mtd5: 00020000 00020000 "U-Boot Env"*

**mtd6: 00500000 00020000 "Kernel"**

*mtd7: 02000000 00020000 "File System"*

#### NOTE



You will see the full list of the mtd partition when using doing **cat /proc/mtd** and can use the output to find the /dev/mtd device node corresponding to each partition

3. Make note of the offset and size for the kernel partition. In this case that is:
  - **OFFSET: 0x280000**
  - **SIZE: 0x500000**

### IMPORTANT

The OFFSET can be found by either adding up the size of all the mtd devices before the kernel, or if you scroll back in your kernel boot history (You can use the **dmesg** command to see the kernel boot log even with the **quiet** option set) you will see a line like

**0x000000280000-0x000000780000 : "Kernel"**

This line is in the format <start of partition (offset)> - <end of partition> which means you could use this to get both the partition, and with some simple math the size as well.

4. Reboot the board and stop the auto boot by pressing any key when prompted
  - **init 6**
5. Set the NAND ECC mode to match the Linux kernel which is BCH8
  - **nandeccl hw 2**
6. Erase the **Kernel** flash partition.
  - **nand erase 0x280000 0x500000**
7. Download the kernel image to DDR. You can find the serverip to use by running the **ifconfig** command on your host.
  - **mw.b \${loadaddr} 0xFF 0x500000**

### NOTE

This command will write all 1's to the memory address where we are going to transfer the kernel image. This is a nice to do not a strict requirement

- **setenv serverip <your server ip>**
- **setenv autoload n**
- **dhcp**
- **tftp \${loadaddr} uImage-am335x-evm.bin**

### IMPORTANT

The kernel image is downloaded to the address stored in **loadaddr**. In this case that is 0x82000000 but you can always look for the **Load address** in the tftp output

## NOTE

The serverip used is the same as the one used in your uEnv.txt file

8. Make a note of the actual kernel image size shown as the last line out output. i.e.

**Bytes transferred = 3154760 (302348 hex)**

9. Write the kernel image using the following command
  - **nand write \${loadaddr} 0x280000 0x500000**

## IMPORTANT

For help with the various nand command you can just type **nand** on the u-boot prompt and you will be given the help output for the nand system

10. Using the tftp transfer size from above you can find the size of the kernel image that needs to be copied from NAND by rounding it to the next erase size (0x20000). In this case **0x320000**. Remember this number for the following steps.

## NOTE

You can find the erase size using the **nand info** command and looking for the **sector size**

11. In u-boot the default size of the kernel to load is 0x500000. We do not need to actually copy this much data from the NAND to the DDR so any extra data copy is actually a waste of time. So now you will update the uEnv.txt file to overwrite the default value with the **0x320000** value we found above. To do this boot your board using the following commands and modify the uEnv.txt (remember that we are using the target board as our SD card reader as well for this lab)
  - **boot**
  - Once the target boots login as **root**
  - **mount /dev/mmcbk0p1 /media/mmc1**
  - **vi /media/mmc1/uEnv.txt** and make the following changes
    - Remove these lines

**serverip=xxxx** This is not needed because we will not be booting from tftp  
**bootfile=xxxx** Again, we will not be booting from tftp so this is not required  
**tftp\_nand\_boot** Since we are now going to boot the kernel from NAND and use the file system from NAND we can now use the built in u-boot defaults

- Modify the uenvcmd line to look like

**uenvcmd=run nand\_boot**

- Add the following line to change the size of the kernel loaded

**nand\_img\_siz=0x320000** This is the value you found above

12. Un-mount the SD card and reboot your board using the following commands
  - **umount /media/mmc1**
  - **init 6**
13. You should now see the device boot reading the kernel from NAND

#### NOTE

You should no longer see the #'s that marked the tftp of the kernel which is a good sign it is being read from elsewhere

## Flashing the U-Boot/SPL to NAND[[edit](#)]

### Description[[edit](#)]

This section will cover how to place the boot loaders into the NAND flash. This will allow us get rid of the SD card altogether.

### Key Points[[edit](#)]

- How to flash the boot loaders to NAND

### Lab Steps[[edit](#)]

#### From Linux[[edit](#)]

1. Assuming the system is booted login using **root**
2. As with the Linux kernel you will need to determine the boot loader flash partition. In this case there are two, the **SPL** and **U-Boot** partitions
  - **cat /proc/mtd**

#### NOTE

You may wonder why there are multiple partitions for the SPL. This is because the ROM boot loader will try looking into the first 4 sectors on the flash to find a valid SPL image in case the first sector is bad

You should expect output like:

```
mtd0: 00020000 00020000 "SPL"  
mtd0: 00020000 00020000 "SPL.backup1"  
mtd0: 00020000 00020000 "SPL.backup2"  
mtd0: 00020000 00020000 "SPL.backup3"  
mtd0: 001e0000 00020000 "U-Boot"
```

## NOTE

You will see the full list of the mtd partition when using doing **cat /proc/mtd** and can use the output to find the /dev/mtd device node corresponding to each partition

3. Erase the **SPL** and **U-Boot** flash partitions.
  - **flash\_erase /dev/mtd0 0 0**
  - **flash\_erase /dev/mtd4 0 0**

## IMPORTANT

The device node to erase was based on the mtd device found for the "SPL" and "U-Boot" partitions in the /proc/mtd output

4. Download the SPL and U-Boot images to your local file system. You can find the serverip to use by running the **ifconfig** command on your host.
  - **tftp -g -r MLO-optimized <serverip>**
  - **tftp -g -r u-boot-optimized <serverip>**

## NOTE

The serverip used is the same as the one used in your uEnv.txt file

5. Write the SPL and U-Boot images using the following commands
  - **nandwrite -p /dev/mtd0 MLO-optimized**
  - **nandwrite -p /dev/mtd4 u-boot-optimized**

## IMPORTANT

You need to use the -p option to pad the kernel image to a full page size or else you will see write errors

6. The beaglebone with the 16bit NAND cape is already configured to try to boot from NAND first. So all you need to do now is reboot your board using:
  - **init 6**
7. You should now see the device boot reading the u-boot.img from NAND instead of SD. You should notice that the MMC initialization that was happening before the u-boot.img read is no longer being done because you are now booted from NAND.

## From U-Boot[\[edit\]](#)

1. Assuming the system is booted login using **root**
2. Determine the **SPL** and **U-Boot** flash partitions.
  - **cat /proc/mtd**

You should expect output like:

```
mtd0: 00020000 00020000 "SPL"  
mtd0: 00020000 00020000 "SPL.backup1"  
mtd0: 00020000 00020000 "SPL.backup2"  
mtd0: 00020000 00020000 "SPL.backup3"  
mtd0: 001e0000 00020000 "U-Boot"
```

#### NOTE

You will see the full list of the mtd partition when using doing **cat /proc/mtd** and can use the output to find the /dev/mtd device node corresponding to each partition

3. Make note of the offset and size for the SPL and U-Boot partitions. In this case they are:
  - SPL:
    - **OFFSET: 0x0**
    - **SIZE: 0x20000**
  - U-Boot:
    - **OFFSET: 0x80000**
    - **SIZE: 0x1e0000**

#### IMPORTANT

The OFFSET can be found by either adding up the size of all the mtd devices before the partition you are interested in, or if you scroll back in your kernel boot history (You can use the **dmesg** command to see the kernel boot log even with the **quiet** option set) you will see lines like

```
0x000000000000-0x000000020000 : "SPL"  
0x000000080000-0x000000260000 : "U-Boot"
```

This line is in the format <start of partition (offset)> - <end of partition> which means you could use this to get both the partition, and with some simple math the size as well.

4. Reboot the board and stop the auto boot by pressing any key when prompted
  - **init 6**
5. Set the NAND ECC mode to match the Linux kernel which is BCH8
  - **nandecclw 2**
6. Erase the **SPL** and **U-Boot** flash partitions.
  - **nand erase 0x0 0x20000**
  - **nand erase 0x80000 0x1e0000**
7. Download the SPL image to DDR. You can find the serverip to use by running the **ifconfig** command on your host.
  - **mw.b \${loadaddr} 0xFF 0x20000**

## NOTE

This command will write all 1's to the memory address where we are going to transfer the kernel image. This is a nice to do not a strict requirement

- **setenv serverip <your server ip>**
- **setenv autoload n**
- **dhcp**
- **tftp \${loadaddr} MLO-optimized**

## IMPORTANT

The SPL image is downloaded to the address stored in **loadaddr**. In this case that is 0x82000000 but you can always look for the **Load address** in the tftp output

## NOTE

The serverip used is the same as the one used in your uEnv.txt file

8. Make a note of the actual image size shown as the last line out output. i.e.

**Bytes transferred = 79953 (13851)**

9. Write the SPL image using the following command
  - **nand write \${loadaddr} 0x0 0x20000**
10. Download the U-Boot image to DDR.
  - **mw.b \${loadaddr} 0xFF 0x1e0000**

## NOTE

This command will write all 1's to the memory address where we are going to transfer the kernel image. This is a nice to do not a strict requirement

- **tftp \${loadaddr} u-boot-optimized**

## IMPORTANT

The U-Boot image is downloaded to the address stored in **loadaddr**. In this case that is 0x82000000 but you can always look for the **Load address** in the tftp output

11. Make a note of the actual kernel image size shown as the last line out output. i.e.

**Bytes transferred = 228144 (37b30 hex)**

12. Write the kernel image using the following command


- **nand write \${loadaddr} 0x80000 0x1e0000**


### IMPORTANT

For help with the various nand command you can just type **nand** on the u-boot prompt and you will be given the help output for the nand system

13. You can now type the following command at the u-boot prompt to see if you have flashed your images to NAND successfully.

- **reset**

	<p>Keystone=</p> <p>1. switchcategory:MultiCore=</p> <ul style="list-style-type: none"> <li>▪ For technical support on MultiCore devices, please post your questions in the <a href="#">C6000 MultiCore Forum</a></li> <li>▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the <a href="#">BIOS Forum</a></li> </ul> <p>Please post only comments related to the article <b>Sitara Linux Training: Boot Time Reduction</b> here.</p>	<p>For technical support on MultiCore devices, please post your questions in the <a href="#">C6000 MultiCore Forum</a></p> <p>For questions related to the BIOS MultiCore SDK (MCSDK), please use the <a href="#">BIOS Forum</a></p> <p>Please post only comments related to the article <b>Sitara Linux Training: Boot Time Reduction</b> here.</p>
<h2>Links</h2>		



<a href="#">Amplifiers &amp; Linear</a>	<a href="#">DLP &amp; MEMS</a>	<a href="#">Processors</a>	<a href="#">Switches &amp; Multiplexers</a>
<a href="#">Audio</a>	<a href="#">High-Reliability</a>	▪ <a href="#">ARM Processors</a>	<a href="#">Temperature</a>
<a href="#">Broadband</a>	<a href="#">Interface</a>	▪ <a href="#">Digital Signal Processors (DSP)</a>	<a href="#">Sensors &amp; Control ICs</a>
<a href="#">RF/IF &amp; Digital</a>	<a href="#">Logic</a>	▪ <a href="#">Microcontrollers (MCU)</a>	<a href="#">Wireless</a>
<a href="#">Radio</a>	<a href="#">Power</a>	▪ <a href="#">OMAP Applications Processors</a>	<a href="#">Connectivity</a>
<a href="#">Clocks &amp; Timers</a>	<a href="#">Management</a>		
<a href="#">Data Converters</a>			

Retrieved from

"[https://processors.wiki.ti.com/index.php?title=Sitara Linux Training: Boot Time Red](https://processors.wiki.ti.com/index.php?title=Sitara_Linux_Training:_Boot_Time_Reduction&oldid=157984)  
[uction&oldid=157984](https://processors.wiki.ti.com/index.php?title=Sitara_Linux_Training:_Boot_Time_Reduction&oldid=157984)"

Categories:

- [AM437x](#)
- [AM335x](#)
- [AM35x](#)
- [AM37x](#)
- [AM1x](#)
- [AM18x](#)

## Navigation menu

### Personal tools

- [Log in](#)
- [Request account](#)

### Namespaces

- [Page](#)
- [Discussion](#)



### Variants

### Views

- [Read](#)
- [View source](#)



- [View history](#)



## More

## Search

<input type="text" value="Search"/>	<input type="submit" value="Go"/>
-------------------------------------	-----------------------------------

## Navigation

- [Main Page](#)
- [All pages](#)
- [All categories](#)
- [Recent changes](#)
- [Random page](#)
- [Help](#)

## Toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)
- [Page information](#)

- 
- This page was last edited on 31 July 2013, at 16:57.
  - Content is available under [Creative Commons Attribution-ShareAlike](#) unless otherwise noted.

- [Privacy policy](#)
- [About Texas Instruments Wiki](#)
- [Disclaimers](#)
- [Terms of Use](#)

