Linux Kernel Driver Creation Guide for BeagleBone - Compiling and Downloading Kernel via UBoot

Tested under Ubuntu 20.04 with Target kernel 5.3.7

by Brian Fraser

Last update: Feb 27, 2022

This document guides the user through

1. Compiling the Linux kernel and downloading to the BeagleBone using UBoot.
2. Creating and compiling a new Linux driver.
3. Loading and unloading the driver.

Table of Contents

Formatting

1. Commands starting with `(host)$` are Linux console commands on the host PC:
   ```
   (host)$ echo "Hello PC world"
   ```
2. Commands starting with `(bbg)$` are Linux console commands on the target board:
   ```
   (bbg)$ echo "Hello Target world"
   ```
3. Commands starting with => are U-Boot console commands:
   ```
   => printenv
   ```

Document History

- Nov 2: Corrected typo in makefile step 4.1.6.
- Nov 13: Added `iptables` command to resolve `tftp` blocking issue.
- Mar 1, 2021: Updated to Ubuntu 20.04; clarified directions

- Mar 17, 2021: Updated language around kernel make file; corrected paths in examples.

1. Install TFTP Server

1. Install TFTP server on host:
   ```
   (host)$ sudo apt-get install tftpd-hpa
   ```
2. Configure the directory you wish to make public via TFTP[1]:
   ```
   (host)$ sudo gedit /etc/default/tftpd-hpa
   ```
   - Change the file to the following. Change *user_name* to your user name on your host system:
     ```
     TFTP_USERNAME="tftp"
     TFTP_ADDRESS="0.0.0.0:69"
     TFTP_OPTIONS="--create --listen --verbose /home/user_name/cmpt433/public"
     RUN_DAEMON="yes"
     ```
3. Check if the TFTP server is running:
   ```
   (host)$ netstat -a | grep tftp
   ```
   - It should display a line similar to:
     ```
     udp        0      0 0.0.0.0:tftp            0.0.0.0:*
     ```

     (May also include a line for udp6).
4. Restart the server for changes to take effect.
   ```
   (host)$ sudo service tftpd-hpa restart
   ```
5. Test the TFTP server is operating correctly:
   - Install the TFTP client:
     ```
     (host)$ sudo apt-get install tftp
     ```
   - Create a test file in the `~/cmpt433/public` folder which we'll download via TFTP:
     ```
     (host)$ cd
     (host)$ echo 'Coming via TFTP' > ~/cmpt433/public/test_tftp.txt
     ```
   - Download the file via TFTP (changing user_name to be your user name!):
     ```
     (host)$ tftp
     tftp> connect 127.0.0.1
     tftp> get /home/user_name/cmpt433/public/test_tftp.txt
     Received 17 bytes in 0.0 seconds
     tftp> quit
     (host)$ cat test_tftp.txt
     Coming via TFTP
     (host)$
     ```

[1] Using `gksudo` over `sudo` is preferred for running graphical applications as root. However, left as `sudo` here for simplicity.

6. Troubleshooting

   - While using TFTP to download the test file, if you get the error `‟Error code 2: Forbidden directory″` then:

     - Ensure you typed the command (path) correctly. Is the folder name correct?

     - Ensure that the `/etc/default/tftpd-hpa` file correctly lists the full path of the folder you are trying to access via TFTP.

     - Did you change “user_name” correctly in the config file and the TFTP command you typed?

   - If you get the error: `‟Error code 0: Permission denied.″`, then check that the file and the shared folder (`~/cmpt433/public/`) has read permissions for all users, and that all directories up to the one shared are readable and executable by all users.

     - Check the file permissions:

       ```
       (host)$ ls -la ~/cmpt433/public/test_tftp.txt
       -rw-rw-r-- 1 brian brian 16 Oct 11 14:58 test_tftp.txt
       ```

       - Must have the 'r' in 3 times in the permissions. Correct with:

         ```
         (host)$ chmod a+r ~/cmpt433/public/test_tftp.txt
         ```

     - Check the directory permissions:

       ```
       (host)$ ls -lad /home ~ ~/cmpt433 ~/cmpt433/public
       drwxr-xr-x  3 root  root  4096 May 13 23:57 /home
       drwxr-xr-x 39 brian brian 4096 Oct 27 19:55 /home/brian
       drwxr-xr-x  5 brian brian 4096 Oct 24 22:38 /home/brian/cmpt433
       drwxrwxr-x 15 brian brian 4096 Oct 27 20:01 /home/brian/cmpt433/public
       ```

       - Correct (making things at least read/write enough!) with:

         ```
         (host)$ chmod a+rx /home/ ~ ~/cmpt433/ ~/cmpt433/public/
         ```

     - If you get the error: “`tftp: test_tftp.txt: Permission denied`” it may mean you are trying to write the file (after downloading it) into a folder you don't have write permission for. Double check you have write permission in the current directory of your TFTP client.

     - An encrypted home directory might cause problems with the TFTP server.

   - There is a bug in Ubuntu 14.04 that makes the TFTP server not start at boot-up.

     - You can manually start the server each time with the above restart command.

     - Configure to start automatically by editing `/etc/init/tftpd-hpa.conf` and replacing the line:

       ```
       "start on runlevel [2345]"
       ```

       with:

       ```
       "start on (filesystem and net-device-up IFACE!=lo)"
       ```

       and now it starts on boot.

2. Building the Kernel

To build the kernel, we'll use scripts created by Robert Nelson (https://eewiki.net/display/linuxonarm/BeagleBone+Black) which target the BBB. First we get these from GitHub, then we execute them.

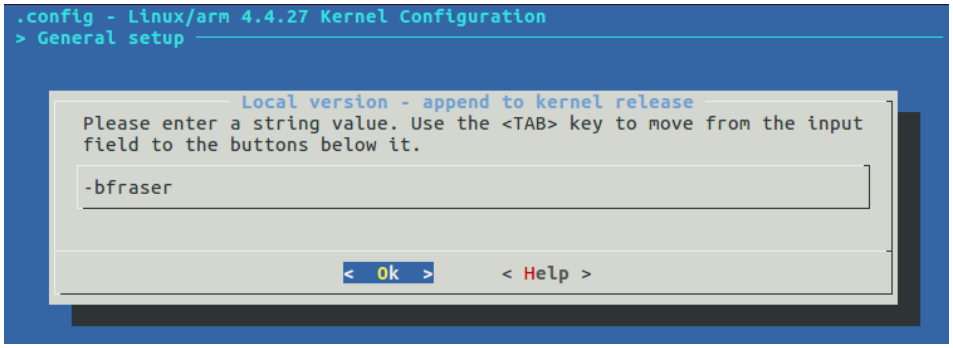
It will download ~1GB+, consume ~5 gigs of space on your host to download and build the kernel. It is best to do this in your normal Linux file system on the host; however, you can do all of this onto a USB memory stick of size ~8+ GB if your VM has not got the space. If you need to use a USB stick, first format your USB stick to be EXT4 (via the Disks program on Ubuntu) otherwise permissions and symbolic links will not work correctly. Took 3 hours on USB

1. Clone the BeagleBone build scripts from GitHub[2]:
   ```
   (host)$ cd ~/cmpt433/work
   (host)$ git clone https://github.com/RobertCNelson/bb-kernel.git
   ```
2. Checkout the scripts to download and build a specific kernel version:
   ```
   (host)$ cd bb-kernel/
   (host)$ git checkout tags/5.3.7-bone13
   ```
   - To get a more recent version, you could also use:
     ```
     (host)$ git checkout origin/am33x-v5.4 -b v5.4
     ```
     While typing "`origin/am`", if you hit TAB then GIT will list all matching branches. You can use this to select a different kernel version (putting it onto the appropriately named branch). Other kernel versions have not been tested for use in this course, but could very well work fine too.
   - If you are unable to checkout due to an error about uncommitted changes, try:
     ```
     (host)$ git stash
     ```
     This will "stash" the changed files for later retrieval (if desired).
   - OK to ignore a warning about "detached HEAD"
3. Some utilities are needed to build the kernel:
   ```
   (host)$ sudo apt-get install flex bison lzop lzma gettext
   (host)$ sudo apt-get install libmpc-dev u-boot-tools libncurses5-dev:amd64
   ```
   - If you are on a Mac M1 processor, which is ARM based, remove the ":amd64"
4. Run the build script to download the correct compiler, the kernel source code, patch it, and build it. This may take some time! (Took **22 minutes** on my home desktop VM; but requires input after about ~10 minutes -- see next step).
   ```
   (host)$ ./build_kernel.sh
   ```
   - This may fail and ask you to install additional packages. Install the packages, then retry the command.
   - This may fail with GIT asking for your email and name. You can use:
     ```
     (host)$ git config --global user.email "yourId@sfu.ca"
     (host)$ git config --global user.name "Your Name"
     ```
     - Change `yourId` and "`Your Name`" to the correct values.
   - If a failure occurs, correct the problem and re-run the script.

[2] If building off a USB drive, it will likely be mounted on your host in /media/<user-name>/ ...

5. When the blue kernel configuration menu appears, change the *Local Version* to be your user ID:
   - Select `General Setup -->` , and press Enter.
   - Select `Local version - append to kernel release` and press Enter.
   - Type in a dash and your SFU ID (your login) and press Enter, such as, and shown below in the screen shot:
     `-bfraser`

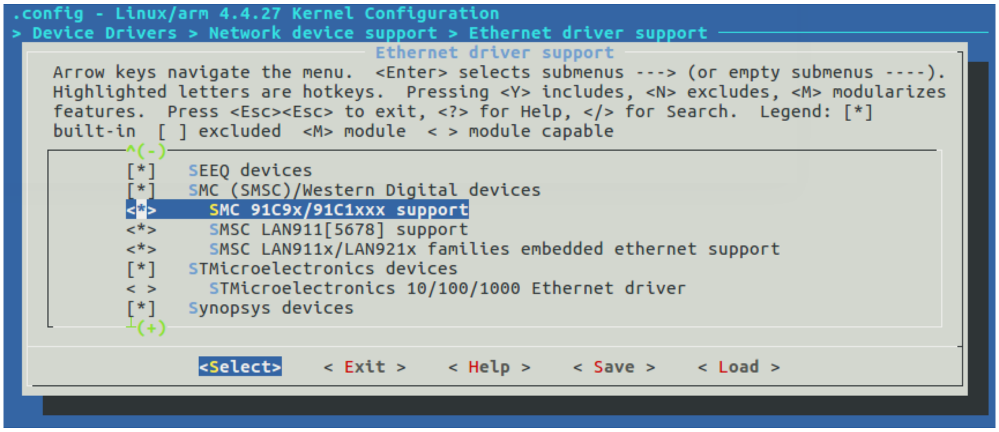
     Screen shot:



```
.config - Linux/arm 4.4.27 Kernel Configuration
> General setup
          Local version - append to kernel release
 Please enter a string value. Use the <TAB> key to move from the input
 field to the buttons below it.
 -bfraser
                   <  Ok  >        < Help >
```

   - Go back up to the top menu by pressing Right and select Exit

6. Also in the menu, build drivers for the BeagleBone's Ethernet chip into the kernel image
   - Select `Device Drivers --->`, and press Enter
   - Select `Network device support --->`, and press Enter
   - Select `Ethernet driver support --->`, and press Enter
   - Make the three SMC (or SMSC) Ethernet devices be built-in ("included") in the kernel image by selecting them and typing 'Y' to get the * beside them.
     - `SMC 91C9x/91C1xxx support`
     - `SMSC LAN911[5678] support`
     - `SMSC LAN911x/LAN921x families embedded ethernet support`
   - Screenshot



```
.config - Linux/arm 4.4.27 Kernel Configuration
> Device Drivers > Network device support > Ethernet driver support
                    Ethernet driver support
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable
          ^(-)
          [*]   SEEQ devices
          [*]   SMC (SMSC)/Western Digital devices
          <*>     SMC 91C9x/91C1xxx support
          <*>     SMSC LAN911[5678] support
          <*>     SMSC LAN911x/LAN921x families embedded ethernet support
          [*]   STMicroelectronics devices
          < >     STMicroelectronics 10/100/1000 Ethernet driver
          [*]   Synopsys devices
          (+)
       <Select>    < Exit >    < Help >    < Save >    < Load >
```

7. Save and exit the menu:
   - Save the file as `.config` (press Right to select `Save`, accept `.config` file name).

- Select `Exit` until you exit the menu (press Right to select Exit from menu).
- Note: if you rebuild the kernel again, this setting you set via the blue kernel configuration menu should persist correctly.
- **Once you exit, the kernel will continue to build. This may take a long time (3+ hours) on some systems. It may also seem to pause for a minute or two and then continue.**

  However, **rebuilding the system in the future will be quite a bit faster** because it does not need to download any tools, nor re-compile much of the work initially compiled.

8. Build Products:
   The custom build script you just ran will produce the following files:

   - Device tree file for the BeagleBone Green, loadable by UBoot to configure the peripherals on the board:
     `~/cmpt433/work/bb-kernel/KERNEL/arch/arm/boot/dts/am335x-bonegreen.dtb`

     Use `.../am335x-boneblack.dtb` if on a BeagleBone Black.
   - Linux kernel to booted on the target. The minor revision number may change as the scripts are updated (on GitHub) to refer to newer kernel versions, and your script will build a file with your user ID's name
     `~/cmpt433/work/bb-kernel/deploy/5.3.7-bfraser-bone13.zImage`

   Copy the build products into `~/cmpt433/public/` for access via TFTP on the target.

   ```
   (host)$ cd ~/cmpt433/work/bb-kernel/
   (host)$ cp KERNEL/arch/arm/boot/dts/am335x-bonegreen.dtb ~/cmpt433/public
   (host)$ cp deploy/5.3.7-bfraser-bone13.zImage ~/cmpt433/public
   ```

9. Troubleshooting

   - If you are running Ubuntu 14.04, the script may fail with an error stating you may need:
     - A new version of Git:

       ```
       (host)$ sudo apt-add-repository ppa:git-core/ppa
       (host)$ sudo apt-get update
       (host)$ sudo apt-get install git
       ```
     - Git clone fails with `gnutls_handshake()`:

       ```
       (host)$ sudo apt-get upgrade libcurl3-gnutls
       ```

2.1 Coping Modules to RFS

You can copy the kernel modules (drivers) which were build when you built your custom kernel to the target's root file system (RFS) so that these drivers can be loaded at run-time if a device is connected which requires a driver. If you do not copying these drivers over, when you boot to your custom kernel your target will not have many device drivers up and running (for example, support for USB devices).

1. On the **host**, copy the modules:
   ```
   (host)$ cd ~/cmpt433/work/bb-kernel/deploy/
   (host)$ cp 5.3.7-bfraser-bone13-modules.tar.gz  ~/cmpt433/public/
   ```
   - Change the user ID in the file name to match your files (i.e., it won't be "-bfraser")!
2. On the **target**, with NFS mounted to `/mnt/remote/`, do both:

   Change to the root directory (/):
   ```
   (bbg)$ cd /
   ```
   Extract the archive of modules:
   ```
   (bbg)$ sudo tar -zxvf /mnt/remote/5.3.7-bfraser-bone13-modules.tar.gz
   ```
   - You may get (many) warnings about time stamps being in the future; these can be ignored. (You can suppress these warnings by adding: `--warning=no-timestamp`
   - This will likely take around 180k on the eMMC
3. Check the modules installed:
   ```
   (bbg)$ ls /lib/modules/
   ```
   - Output:
     ```
     4.9.78-ti-rt-r94/  5.3.7-bfraser-bone13/
     ```
     Note that your initial folder name may vary depending on the kernel version your board initially ran.
   - The `tar` command also copies some files into `/lib/firmware`
4. You now have all the device drivers available for your newly compiled kernel! Connecting a device that requires a device driver to be loaded should now work.
   - When you next boot into your custom kernel, you'll see it can now load drivers!

3. Downloading the Kernel

3.1 Supported Connections

To perform these steps, you need *both*:

1. A **TTL serial** connection to the target (such as through the Zen Cape’s TTL over USB conection, or a separate USB to TTL cable).
2. A **Ethernet** connection between the host and the target.
   - Can be Ethernet-over-USB (passthrough).
   - Can be via a normal Ethernet connection to an existing LAN (for example, such as through a home router). In this case, both the host and target likely use DHCP to automatically get an IP address (which may change over time).
     - If your host is a VM, you will likely need to have a bridged network connection to the network instead of using Network Address Translation (NAT).
   - Can be via a direct Ethernet cable connection between the target and the host (such as a laptop). This likely uses a static IP address. No need to use a cross-over cable; the target hardware works with a standard Ethernet cable.
     - If running Linux natively (no virtual machine), then Linux should automatically detect the direct Ethernet connection to the target. You may need to set the IP address of the Linux host manually.
     - If using a virtual machine to run Linux on your host PC with a direct cable connection between the host PC and the target, then configure the VM to have two network connections: one as NAT connecting the guest OS to the host OS's normal internet connection (likely WiFi), and the second as a Bridged connection connecting the guest OS to the physical Ethernet port (and hence the target). See the quick-start guide for how to test this configuration.

If working in the SFU labs, you should be able to develop in CSIL (via a Virtual Machine) and use Ethernet over USB.

3.2 Enter UBoot

1. Connect to the target using a serial port connection (Zen cape's TTL, or a special USB to Serial cable) using `Screen`.
   - SSH will not work because we need access to UBoot. If having problems with your serial connection, see the Serial Connection Troubleshooting step at the end of this section.
2. Reboot the target into UBoot. You can either
   - Press the BBG’s reset button (outermost of the two beside the BBG’s Ethernet port)
   - Unplug and replug BBG’s power
   - If booted into Linux:
     ```
     (bbg)$ reboot
     ```

3. Press the space bar (or any key) when it starts to boot to enter the interactive UBoot prompt:

   ```
   U-Boot SPL 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29)
   Trying to boot from MMC2


   U-Boot 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29 -0600), Build:
   jenkins-github_Bootloader-Builder-32

   CPU  : AM335X-GP rev 2.1
   I2C:   ready
   DRAM:  512 MiB
   No match for driver 'omap_hsmmc'
   No match for driver 'omap_hsmmc'
   Some drivers were not found
   Reset Source: Global external warm reset has occurred.
   Reset Source: Power-on reset has occurred.
   MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
   Using default environment

   Board: BeagleBone Black
   <ethaddr> not set. Validating first E-fuse MAC
   BeagleBone Black:
   Model: SeeedStudio BeagleBone Green:
   debug: process_cape_part_number:[BB-BONE-ZEN-01]
   debug: process_cape_part_number:[42422D424F4E452D5A454E2D3031]
   BeagleBone: cape eeprom: i2c_probe: 0x54:
   BeagleBone: cape eeprom: i2c_probe: 0x55:
   BeagleBone: cape eeprom: i2c_probe: 0x56:
   BeagleBone: cape eeprom: i2c_probe: 0x57:
   Net:   eth0: MII MODE
   cpsw, usb_ether
   Press SPACE to abort autoboot in 2 seconds
   =>
   ```

4. The U-Boot prompt accepts commands, such as:
   - `help`: display a list of commands available.
   - `printenv`: display all environment variables currently set.
   - `setenv`: set an environment variable in U-Boot.
   - `boot`: continue booting as normal.

5. Note that U-Boot usually runs on boards which feature some EEPROM to which one can save the environment variables. Unfortunately, the BeagleBone does not have this. Therefore, each time you reboot all changes made to the environment variables in U-Boot will be lost.
   - The best way to work around this is to keep a copy of the necessary commands in a text-editor and copy-and-paste them into the U-Boot connection each boot as needed.

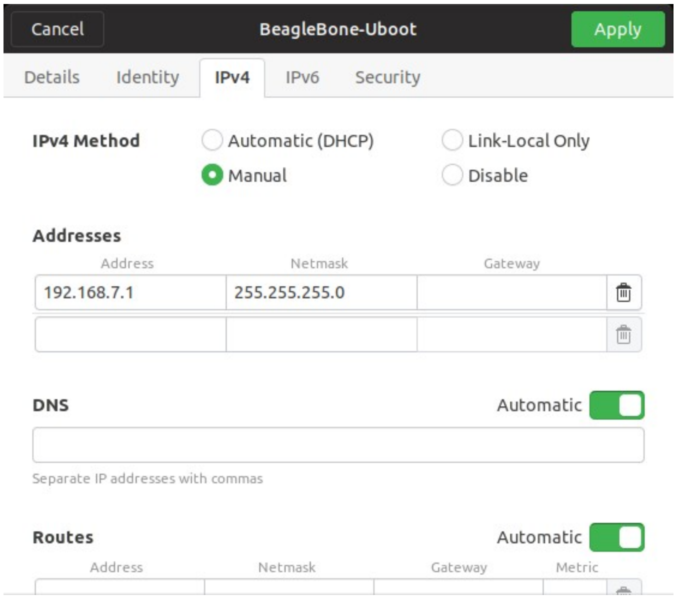
3.3 Acquire an IP address in Uboot

Pick just one of the following three options; see below sub-sections for steps:

1. Ethernet over USB
2. Physical Ethernet connection with DHCP
3. Physical Ethernet connection with static IP

If using Ethernet over USB for UBoot

1. Setup the host PC (just once) to have a new Ethernet profile with a fixed IP address. In Ubuntu:
   - Go to *Settings* → *Network*
   - Beside *Wired* (may be listed as the name of your Ethernet connection), click the +
   - On *Identity* tab: set name to “BeagleBone-UBoot”; leave MAC, cloned address, and MTU as they are.
   - On *IPv4* tab: change to **Manual**;
     Address **192.168.7.1**,
     Netmask **255.255.255.0**,
     Gateway empty, DNS empty, Routes empty, click Add / Apply.



IPv4 Method ○ Automatic (DHCP) ○ Link-Local Only ● Manual ○ Disable

Addresses

Address	Netmask	Gateway
192.168.7.1	255.255.255.0	

DNS Automatic

Separate IP addresses with commas

Routes Automatic

Address	Netmask	Gateway	Metric

2. Each time you reboot into UBoot, execute these commands:

   ```
   => setenv ethact usb_ether
   => setenv ipaddr 192.168.7.2
   => setenv serverip 192.168.7.1
   ```

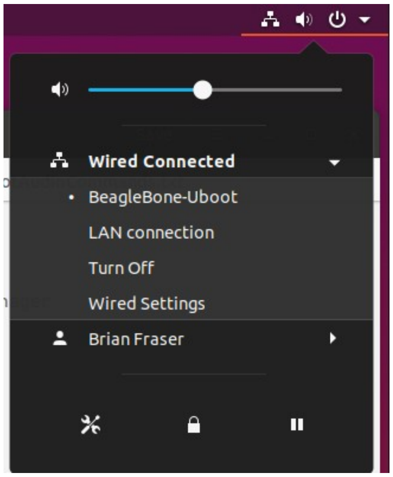
3. Try to ping target (UBoot) to host:

   ```
   => ping $serverip
   using musb-hdrc, OUT ep1out IN ep1in STATUS ep2in
   MAC b0:d5:cc:47:00:d5
   HOST MAC de:ad:be:af:00:00
   RNDIS ready
   musb-hdrc: peripheral reset irq lost!
   high speed config #2: 2 mA, Ethernet Gadget, using RNDIS
   USB RNDIS network up!
   Using usb_ether device
   host 192.168.7.1 is alive
   ```

4. Troubleshooting: If `ping` fails there are a number of things you may need to setup:
   - If using a VM, you must automatically map UBoot’s Ethernet over USB connection to the VM from the host OS. UBoot only keeps the network interface enabled while it’s trying to use it, so the connection will only show up while the ping is happening.
     - If running VM Ware, I have found that newly connected USB devices are automatically mapped to the VM, so as long as you are in the VM when running the commands, it is

    likely to automatically connect.

    - If running Virtual Box, you can create a rule to map UBoot’s Ethernet over USB to your VM. However, the device only shows up in the list while UBoot is doing the ping command. So, you’ll need to run the above UBoot ping command and quickly do the following (may take a few times to practice!):
      Go to: Devices → USB → USB Settings
      Click , select **Texas Instruments RNDIS/Ethernet Gadget**



- If the `ping` command seems to hang for a while and then fails with:
  ```
  ARP Retry count exceeded; starting again
  ping failed; host 192.168.7.1 is not alive
  ```
    - In Ubuntu’s top right corner, click the network drop-down. It will list all the existing network connections. Note what ones you see.
    - Re-run the ping command. While UBoot is trying to connect to your host, Ubuntu should show you another network device in the menu in the top right. On my computer, it is name “**Ethernet**”, whereas my normal network is shown as “PCI Ethernet”.
    - Click on the arrow icon for the new network connection and select BeagleBone-UBoot profile.
    - You may need to reboot the BeagleBone for this new network connection to show up.
- If, after playing around with network settings, your host is unable to connect to the internet, you may need to click on Ubuntu’s network button (top right) and for your connection to the internet try selecting a different Ethernet profile.

If using Physical Ethernet connection with DHCP for UBoot

Use this if an Ethernet cable is plugged into your BeagleBone which connects your BeagleBone to the same network as your host PC, and the network includes a router which supports DHCP.

1. Disable the auto-boot feature which tries to download a kernel image from the DHCP server, and then acquire an address:
   ```
   => setenv autoload no
   => dhcp
   ```
    - Expected output:
      ```
      link up on port 0, speed 100, full duplex
      BOOTP broadcast 1
      BOOTP broadcast 2
      DHCP client bound to address 192.168.0.113 (624 ms)
      ```
    - Note DHCP command can reset some environment variables in UBoot. Run the “dhcp” command *before* setting other variables to work around the issue.
2. Ping the server:

```
=> setenv serverip 192.168.86.155
=> ping ${serverip}
link up on port 0, speed 100, full duplex

Using cpsw device
host 192.168.2.1 is alive
```

If using Physical Ethernet connection with Static IP for UBoot

This is the situation where using a direct Ethernet cable connection to your PC (no DHCP).

1. Set UBoot's IP address and target address:
   ```
   => setenv ipaddr 192.168.2.2
   => setenv serverip 192.168.2.1
   ```

3. Ping the server:
   ```
   => ping ${serverip}
   link up on port 0, speed 100, full duplex
   Using cpsw device
   host 192.168.2.1 is alive
   ```

3.4 Kernel Downloading Steps

1. From the previous step, you will have set the `serverip` and UBoot will have an IP address.
2. Configure U-Boot environment variables for TFTP
   - Configure the full path of the root folder storing the TFTP files (change `user_name` to be your user name on your host pc!):
     ```
     => setenv tftproot /home/user_name/cmpt433/public/
     ```
   - Configure the file name of the Linux kernel image and the device tree file:
     ```
     => setenv bootfile ${tftproot}5.3.7-bfraser-bone13.zImage
     => setenv fdtfile ${tftproot}am335x-bonegreen.dtb
     ```
3. Test your target's ability to connect to the server using `ping`:
   ```
   => ping ${serverip}
   link up on port 0, speed 100, full duplex
   Using cpsw device
   host 192.168.7.1 is alive
   ```
   - If unable to ping (command stops after second line and seems to hang), it means that your network connection between the server and the target is incorrect. Double check the Supported Connections section (3.1) for details.
4. Download the device tree and kernel via TFTP. Should see '#'s printed to screen as it downloads.
   ```
   => tftp ${loadaddr} ${bootfile}
   => tftp ${fdtaddr} ${fdtfile}
   ```
5. Boot the image.
   - Set the kernel's "command line arguments" which are passed to the kernel when it boots[3]:

```
=> setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk1p1 ro rootfstype=ext4 rootwait
```

[3] *Note: For kernels 4.4.x and higher, it's mmcblk**1**p1; older kernels may use mmcblk**0**p1. See* [*https://groups.google.com/forum/#!topic/beagleboard/svbS36EDo4A*](https://groups.google.com/forum/#!topic/beagleboard/svbS36EDo4A)

- Boot the zImage you have downloaded. This boots the Linux kernel you compiled!
  ```
  => bootz ${loadaddr} - ${fdtaddr}
  ```

6. You can combine multiple U-Boot commands into one line by separating them with a semicolon (';'). Here is the full (one line!) commands that I use.

   - To use, copy text into a text document on your computer. Edit it to have correct kernel name (not -bfraser), and **put it all on one line**! Then copy-n-paste it into UBoot each time you reboot the board and want to boot your custom kernel.

   When using Ethernet over USB (server at 192.168.7.1):
   ```
   setenv ethact usb_ether;setenv ipaddr 192.168.7.2;setenv serverip
   192.168.7.1;setenv tftproot /home/brian/cmpt433/public/;setenv bootfile $
   {tftproot}5.3.7-bfraser-bone13.zImage;setenv fdtfile ${tftproot}am335x-
   bonegreen.dtb;tftp ${loadaddr} ${bootfile};sleep 1;tftp ${fdtaddr} $
   {fdtfile};setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk1p1 ro
   rootfstype=ext4 rootwait;bootz ${loadaddr} - ${fdtaddr};
   ```

   - Note this adds a sleep command between the two tftp commands to allow the network adapter to turn off, and then turn back on. If this command sequence fails, try increasing the sleep delay.

   When using DHCP (example has server at 192.168.0.133):
   ```
   setenv autoload no;dhcp;setenv serverip 192.168.0.133;setenv tftproot
   /home/brian/cmpt433/public/;setenv bootfile ${tftproot}5.3.7-bfraser-
   bone13.zImage;setenv fdtfile ${tftproot}am335x-bonegreen.dtb;tftp $
   {loadaddr} ${bootfile};tftp ${fdtaddr} ${fdtfile};setenv bootargs
   console=ttyO0,115200n8 root=/dev/mmcblk1p1 ro rootfstype=ext4
   rootwait;bootz ${loadaddr} - ${fdtaddr};
   ```

   When using Direct Cconnection (direct Ethernet cable):
   ```
   setenv ipaddr 192.168.2.2; setenv serverip 192.168.2.1;setenv tftproot
   /home/brian/cmpt433/public/;setenv bootfile ${tftproot}5.3.7-bfraser-
   bone13.zImage;setenv fdtfile ${tftproot}am335x-bonegreen.dtb;tftp $
   {loadaddr} ${bootfile};tftp ${fdtaddr} ${fdtfile};setenv bootargs
   console=ttyO0,115200n8 root=/dev/mmcblk1p1 ro rootfstype=ext4
   rootwait;bootz ${loadaddr} - ${fdtaddr};
   ```

7. Once the kernel boots, it should then load the root file system (RFS) installed in the target's eMMC. You should see your normal login prompt (customized in Assignment 1). Log-in as usual (likely user `debian`, password `temppwd`).

8. Limited Access

   - Unless you have already completed Section 2.1 (Coping Modules to RFS), many of the drivers will not have loaded, so don’t expect the board to function well until you install the drivers. For example, Ethernet over USB may not work without the drivers.
   - If you want to copy the drivers now, you can just reboot your board and it will automatically boot to the previous version of the kernel.

9. Full U-Boot interaction via Ethernet over USB; commands bold-underline:

```
=> setenv ethact usb_ether;setenv ipaddr 192.168.7.2;setenv serverip 192.168.7.1
=> setenv tftproot /home/brian/cmpt433/public/
=> setenv bootfile ${tftproot}5.3.7-bfraser-bone13.zImage
=> setenv fdtfile ${tftproot}am335x-bonegreen.dtb
=> tftp ${loadaddr} ${bootfile}
using musb-hdrc, OUT ep1out IN ep1in STATUS ep2in
MAC b0:d5:cc:47:00:d5
HOST MAC de:ad:be:af:00:00
RNDIS ready
musb-hdrc: peripheral reset irq lost!
high speed config #2: 2 mA, Ethernet Gadget, using RNDIS
USB RNDIS network up!
Using usb_ether device
TFTP from server 192.168.7.1; our IP address is 192.168.7.2
Filename '/home/brian/cmpt433/public/5.3.7-bfraser-bone13.zImage'.
Load address: 0x82000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ###############################################################
         340.8 KiB/s
done
Bytes transferred = 8548864 (827200 hex)
=> tftp ${fdtaddr} ${fdtfile}
using musb-hdrc, OUT ep1out IN ep1in STATUS ep2in
MAC b0:d5:cc:47:00:d5
HOST MAC de:ad:be:af:00:00
RNDIS ready
high speed config #2: 2 mA, Ethernet Gadget, using RNDIS
USB RNDIS network up!
Using usb_ether device
TFTP from server 192.168.7.1; our IP address is 192.168.7.2
Filename '/home/brian/cmpt433/public/am335x-bonegreen.dtb'.
Load address: 0x88000000
Loading: ######
         256.8 KiB/s
done
Bytes transferred = 82118 (140c6 hex)
=> setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk1p1 ro rootfstype=ext4 rootwait
=> bootz ${loadaddr} - ${fdtaddr}
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Device Tree to 8ffe8000, end 8ffff0c5 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0

<-- SNIP: Many lines omitted to fit screen... -->

beaglebone login: debian
Password:
debian@beaglebone:~$ uname -a
Linux beaglebone 5.3.7-bfraser-bone13 #1 PREEMPT Tue Oct 22 11:46:16 PDT 2019 armv7l
GNU/Linux
```

There may be up to a 60s pause with no output when your target is booting.

10. At the Linux command prompt, verify the correct kernel version:

```
(bbg)$ uname -r
5.3.7-bfraser-bone13
```

    - If you don't have the correct version, then you have likely booted the kernel stored on the board’s eMMC instead of downloading a new kernel. This may happen when you reboot the board and don't enter U-Boot. It’s OK if the last minor number of the kernel version and the user ID are different.
    - Note that these steps must be done each time you reboot the board in order to load the custom kernel.

11. Networking on your newly booted kernel:
    - You should now be able to mount `/mnt/remote/` via NFS as usual.
      If not, ensure you have installed the drivers for your new kernel. See section 2.1.
    - If using Ethernet over USB, and it’s not working, then try:
        - Click the Network icon in Ubuntu (top right) and change profile for Ethernet over USB connection.
        - Use `lsmod` to list loaded drivers. u_ether is the driver for Ethernet over USB.

```
(bbg)$ lsmod | grep u_ether
u_ether                24576  2 usb_f_ecm,usb_f_rndis
```

            - If it did not load, try:

```
(bbg)$ sudo modprobe g_ether
```

            - If this fails, the drivers extracted in section 2.1 may have extracted incorrectly. Check the size of the files:

```
(bbg)$ ls -laR /lib/modules/ | grep u_ether
```

              It should show two `u_ether.ko*` files. Ensure each has a non-zero size. If they 0 bytes in size, repeat extracting the driver archive (section 2.1).
        - If needed, use ifconfig to assign an IP address on the host and target.

```
(bbg)$ ifconfig usb0 192.168.7.2 netmask 255.255.255.0
(host)$ ifconfig enx1cba8ca2ed6c 192.168.7.1 netmask 255.255.255.0
```

          Note: device on host may be a different name; check `dmesg` for renaming.

12. Serial Connection Troubleshooting
    - Ensure that you are using the correct command to launch `screen`. Make sure you include the speed as it may work only intermittently if you do not:

```
(host)$ sudo screen /dev/ttyUSB0 115200
```

    - If nothing comes up in the terminal, reboot the board.
    - If your serial port stops working in the VM, disconnect it from the VM (in software) and reconnect. If it is unable to reconnect then plug it into a different USB. If all else fails, power down VM and VM software (launcher) and kill `vboxsvc.exe` (or restart host OS).
    - Or, use a serial terminal in Windows such as MobaXterm, Putty or Tera Term (linked on course website).

13. Troubleshooting
    - If you are unable to start a download, double check that the target has a correct IP address.

You may need to re-run DHCP each time the board re-boots if you are going to download an image via TFTP and using DHCP.

- Test your connection to the server using the U-Boot `ping` command (see earlier directions). If using a direct Ethernet cable connection, see the Networking guide for setting the static IP address as needed.
  - Note that UBoot does not respond to pings: You will not be able to ping the target from host while it's in UBoot.
- While downloading, if you see T's, it means that the target is unable to reach the server.

```
=> tftp ${loadaddr} ${bootfile}
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.0.123; our IP address is 192.168.0.138
Filename '/home/brian/cmpt433/public/4.9.0-rc1-bone0.zImage'.
Load address: 0x82000000
Loading: T T T T T T T T T T T T T T T T T T T T
Retry count exceeded; starting again
using musb-hdrc, OUT ep1out IN ep1in STATUS ep2in
MAC c8:a0:30:aa:dd:a2
HOST MAC de:ad:be:af:00:00
RNDIS ready
ERROR: The remote end did not respond in time.
at drivers/usb/gadget/ether.c:2388/usb_eth_init()
```

  - Double check you have the `serverip` set correctly (via ping)
  - Check the host is running a TFTP server (earlier section in this guide). May need to restart TFTP server on host:

    ```
    (host)$ sudo service tftpd-hpa restart
    ```

  - If executing a full sequence of commands, try running them one at a time. If that works but the combined command does not, then add a `sleep` statement between critical commands (like `tftp`).
  - Some students have had issues with `iptables` on the host disallowing TFTP connections. Resolve with:

    ```
    (host)$ sudo iptables -A INPUT -p udp --dport 69 -j ACCEPT
    ```

- If you get the error that the file is not found during TFTP download, double check that the file is named correctly, that it is in your public directory, and that you have correctly set the `tftproot` in the UBoot statements (changing "brian" and "bfraser" correctly from the examples).
- Do not encrypt your home folder which is being accessed by TFTP / NFS.
- Ensure you are not running any VPN software such as Hamachi. These can cause connectivity problems.
- When running the `bootz` command, if you see the error:

  ```
  Error: unrecognized/unsupported machine ID (r1 = 0x00000e05)
  ```

  Then you did not have a valid device tree file loaded. Ensure you are downloading the correct `.dtb` file, and passing its address as an argument to the `bootz` command.
- Unless you have completed Section 2.1 (Coping Modules to RFS), many of the drivers will not have loaded, so don't expect the board to function well at the moment. For example,

Ethernet over USB may not work without the drivers.

- If boot hangs with “`Waiting for root device /dev/mmcblk1p1`” near the bottom, then it’s likely there’s a problem with your device tree. Capture the output of your download attempt to a log file (Ctrl-a, H) and look what error messages UBoot generated. Is the DTB file name correct? Path correct?
- When working with the networking via Uboot, if the serial connection drops when you execute some command, it is likely a problem with the host PC and its USB. Try:
    - plug the BBG in via a powered USB hub (the connector on the BBG; not on the Zen).
    - plug the BBG into a USB power adapter (such as used to charge your phone); it won’t be able to communicate with the host via Ethernet over USB, but it should have enough power.
    - Unplug other USB devices from your PC which may be using too much power.

4. Creating a Test Driver

4.1 Cross Compiling a Driver

This section will create the driver in the `~/cmpt433/work/driver_demo/` directory of the host, cross-compile it and deploy it to the target.

1. Create a directory for the compiled drivers:
   ```
   (host)$ cd ~/cmpt433/public
   (host)$ mkdir drivers
   ```
2. Create a directory for the driver source code:
   ```
   (host)$ cd ~/cmpt433/work/
   (host)$ mkdir driver_demo/
   (host)$ cd driver_demo/
   ```
3. Create `testdriver.c` in `~/cmpt433/work/driver_demo/` directory with the following contents:
   ```
   // Example test driver:
   #include <linux/module.h>

   static int __init testdriver_init(void)
   {
         printk(KERN_INFO "----> My test driver init()\n");
         return 0;
   }

   static void __exit testdriver_exit(void)
   {
         printk(KERN_INFO "<---- My test driver exit().\n");
   }

   // Link our init/exit functions into the kernel's code.
   module_init(testdriver_init);
   module_exit(testdriver_exit);

   // Information about this module:
   MODULE_AUTHOR("Your Name Here");
   MODULE_DESCRIPTION("A simple test driver");
   MODULE_LICENSE("GPL");        // Important to leave as GPL.
   ```
4. Change the module information to include your name (the `MODULE_AUTHOR()`).
5. **After the `#include`, add the following line:**
   ```
   #error Are we building this file?
   ```
   - When compiled, this will generate an error which will prove you are building this file.
   - Later, once you are sure your file is being compiled correctly, you'll remove this line, but for the moment it will serve as our test that the process is working.

6. Create `Makefile` in `~/cmpt433/work/driver_demo/` with the following contents:

```
# Makefile for driver
# Derived from:
#   http://www.opensourceforu.com/2010/12/writing-your-first-linux-driver/
# with some settings from Robert Nelson's BBB kernel build script

# if KERNELRELEASE is defined, we've been invoked from the
# kernel build system and can use its language.
ifneq (${KERNELRELEASE},)
      obj-m := testdriver.o

# Otherwise we were called directly from the command line.
# Invoke the kernel build system.
else
      KERNEL_SOURCE := ${HOME}/cmpt433/work/bb-kernel/KERNEL/
      PWD := $(shell pwd)
      # Linux kernel 5.4 (one line)
      CC=${HOME}/cmpt433/work/bb-kernel/dl/gcc-arm-8.3-2019.03-x86_64-arm-
linux-gnueabihf/bin/arm-linux-gnueabihf-
      BUILD=bone13
      CORES=4
      image=zImage
      PUBLIC_DRIVER_PWD=~/cmpt433/public/drivers

default:
      # Trigger kernel build for this module
      ${MAKE} -C ${KERNEL_SOURCE} M=${PWD} -j${CORES} ARCH=arm \
            LOCALVERSION=-${BUILD} CROSS_COMPILE=${CC} ${address} \
            ${image} modules
      # copy result to public folder
      cp *.ko ${PUBLIC_DRIVER_PWD}

clean:
      ${MAKE} -C ${KERNEL_SOURCE} M=${PWD} clean
endif
```

   - **Important:**
     - The file name must be `Makefile` (case sensitive!); do *not* name it "makefile".
     - The “CC” statement is one line, no spaces or line-feeds.
   - Read the comments to understand what is going on. The basics are:
     - Make will evaluate the following statement as false:
       `ifneq (${KERNELRELEASE},)`
       so make will execute the “`else`” portion of the main “`ifneq`” statement. This sets up parameters for the main Linux kernel build system, and then invokes the Linux kernel build system asking it to build this folder.
     - The Linux kernel build then starts running (via Make) on this device driver's folder.
     - The following statement is evaluated and will be true:
       `ifneq (${KERNELRELEASE},)`
       so it adds our driver(s) to the `obj-m` variable, which is used by the rest of the kernel build system to build drivers.
   - Note that this `Makefile` alone is not sufficient to build your driver on its own; it leverages the general kernel build system.

7. Prove your `Makefile` works to build your code by having the `#error` directive break the build:

```
~/cmpt433/work/driver_demo$ make
# Trigger kernel build for this module
make -C ~/cmpt433/work/bb-kernel/KERNEL/
SUBDIRS=/home/brian/all-my-code/CMPT433-Code/12-TestDriver -j4 ARCH=arm
LOCALVERSION=-bone13
CROSS_COMPILE=/home/brian/cmpt433/work/bb-kernel/dl/gcc-arm-8.3-2019.03-
x86_64-arm-linux-gnueabihf/bin/arm-linux-gnueabihf-   modules
make[1]: Entering directory '/home/brian/cmpt433/work/bb-kernel/KERNEL'
  CC [M]  /home/brian/all-my-code/CMPT433-Code/12-TestDriver/testdriver.o
/home/brian/all-my-code/CMPT433-Code/12-TestDriver/testdriver.c:4:2: error:
#error Are we building this file?
 #error Are we building this file?
  ^~~~~
make[2]: *** [scripts/Makefile.build:281: /home/brian/all-my-code/CMPT433-
Code/12-TestDriver/testdriver.o] Error 1
make[1]: *** [Makefile:1626:
_module_/home/brian/all-my-code/CMPT433-Code/12-TestDriver] Error 2
make[1]: Leaving directory '/home/brian/cmpt433/work/bb-kernel/KERNEL'
make: *** [Makefile:31: default] Error 2
~/cmpt433/work/driver_demo$
```

   - You should see the "`Are we building this file?`" error. If so, great! You are building the file you expect!

     Comment out the `#error` statement in your `testdriver.c` and continue; otherwise, double check your `Makefile` and kernel build folder are correct.
   - You may get a warning about `SUBDIRS` in the Linux kernel makefile. It seems we can build fine even with this warning; work may be needed in the future to update the Makefile.

8. Build the driver by running `make`
   When successful, it will look like:

```
~/cmpt433/work/driver_demo$ make
# Trigger kernel build for this module
make -C /home/brian/cmpt433/work/bb-kernel/KERNEL/
M=/home/brian/cmpt433/work/driver_demo -j4 ARCH=arm \
      LOCALVERSION=-bone13 CROSS_COMPILE=/home/brian/cmpt433/work/bb-
kernel/dl/gcc-arm-8.3-2019.03-x86_64-arm-linux-gnueabihf/bin/arm-linux-
gnueabihf-  \
      zImage modules
make[1]: Entering directory '/home/brian/cmpt433/work/bb-kernel/KERNEL'
  Kernel: arch/arm/boot/Image is ready
  CC [M]  /home/brian/cmpt433/work/driver_demo/testdriver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/brian/cmpt433/work/driver_demo/testdriver.mod.o
  LD [M]  /home/brian/cmpt433/work/driver_demo/testdriver.ko
  Kernel: arch/arm/boot/zImage is ready
make[1]: Leaving directory '/home/brian/cmpt433/work/bb-kernel/KERNEL'
# copy result to public folder
cp *.ko ~/cmpt433/public/drivers
~/cmpt433/work/driver_demo$
```

9. Check that the .ko file was correctly copied to the `~/cmpt433/public/drivers/` folder:

```
(host)$ ls -l ~/cmpt433/public/drivers/
```

4.2 OPTIONAL: Natively Compiling Drivers on Target

Instead of cross-compiling the drivers from your PC, you can instead copy the `Makefile` and your `.c` source files to your target and build on the BeagleBone directly. Use this approach if you are having problems getting the custom kernel running on the board in some settings.

These directions will compile your driver **under the pre-installed kernel, for the pre-installed kernel.**

1. Ensure your board has an internet connection:
   ```
   (bbg)$ ping google.ca
   ```
2. Your board will need the kernel headers for your version of the kernel. Install them:
   ```
   (bbg)$ sudo apt-get install linux-headers-`uname -r`
   ```
   - This will create the `/lib/modules/<kernel-version>/build/` folder.
3. Have your `.c` driver source code in a directory accessible from the target, such as `/mnt/remote/mydriver/`
4. In the folder with your `.c` code, create a `Makefile` with the following contents:
   ```
   # Makefile for driver: native-compile

   obj-m := testdriver.o

   all:
   	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules

   clean:
   	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
   ```
5. Build the driver
   ```
   (bbg)$ cd /mnt/remote/mydriver/
   (bbg)$ make
   make -C /lib/modules/4.9.78-ti-r94/build/ M=/mnt/remote/mydriver modules
   make[1]: Entering directory '/usr/src/linux-headers-4.9.78-ti-r94'
     CC [M]  /mnt/remote/mydriver/testdriver.o
     Building modules, stage 2.
     MODPOST 1 modules
     CC      /mnt/remote/mydriver/testdriver.mod.o
     LD [M]  /mnt/remote/mydriver/testdriver.ko
   make[1]: Leaving directory '/usr/src/linux-headers-4.9.78-ti-r94'
   ```
6. Driver should be ready to go:
   ```
   (bbg)$ modinfo ./testdriver.ko
   (bbg)$ sudo insmod ./testdriver.ko
   ```
7. Discussion of Folders
   - You could do all your development on the target; however, it makes a lot of sense to use a powerful development environment on the host and share the files to the target via NFS.

     To do so, you must manage directory and file permissions so that the target can create new files in the folder. You'll need the NFS folder containing your .c files to be writable by the target, such as:
     ```
     (host)$ chmod a+rwx  <native-compile-folder-via-NFS>
     ```
     Be very careful to setup a development process where you can efficiently edit your code

(say on your host through an editor) and then build on the target. Be sure that you don't lose any of your files or changes due to having multiple copies and getting confused about which one to edit or check into Git.

- You may want to have your source code in some Git folder on your PC, and then use a script to copy the .c and Makefile to the NFS folder for the target to access. Or, better yet, work out how to do it with file system links (`man ln`)

8. Troubleshooting:
   - If you get any file permission errors, try copying the .c and `Makefile` to a folder on the target like `~/mydriver/` and re-run make in that folder.
   - If `make` fails to find the `/lib/modules/<kernel-version>/build/ folder`, then you likely need to install the kernel headers for your current specific version of the kernel. Find the version by running `uname -r`

5. Working with Drivers

To use the driver, it must be available on the target board at runtime.

1. Either mount via NFS a shared folder containing the .ko device driver, or copy it onto the target.
   - If you are now working across the Ethernet (instead of not Ethernet-over-USB) you may need to update the NFS share settings on your host, and the NFS mount command used on the target. See NFS guide for details.
2. On the target, change to the directory containing the .ko file. If mounted via NFS:
   ```
   (bbg)$ cd /mnt/remote/drivers/
   ```
3. List existing modules loaded on the target:
   ```
   (bbg)$ lsmod
   ```
   - You will *not* see the `testdriver` listed.
4. Find information about your compiled module using:
   ```
   (bbg)$ modinfo ./testdriver.ko
   filename:      /mnt/remote/drivers/./testdriver.ko
   license:       GPL
   description:   A simple test driver
   author:        Brian Fraser
   depends:
   vermagic:      5.3.7-bfraser-bone13 mod_unload modversions ARMv7 thumb2 p2v8
   ```
   - The “`vermagic`” field shows the kernel version your module is targeting.
   - You can run `modinfo` on any machine to see the information about a module, even if that module targets a different architecture.
5. Ensure your booted kernel version matches the vermagic field of the driver:
   ```
   (bbg)$ uname -r
   5.3.7-bfraser-bone13
   ```
   - Any difference between this string and the starting word in the vermagic will cause the driver to fail to load.
6. Load the driver:
   ```
   (bbg)$ sudo insmod testdriver.ko
   ----> My test driver init()
   ```
   - You may not see any output to the screen as the driver only prints to the kernel log.
     To see this, you may need to execute `dmesg`:
     ```
     (bbg)$ dmesg | tail -1
     [  938.788651] ----> My test driver init()
     ```
   - If you see "`insmod: cannot insert 'testdriver.ko': invalid module format`", it likely means that your target's current kernel was built with a different version string than your host is currently building. Either rebuild and download the kernel, or change the host to build the same version as the target (found by running `uname -r`).
   - If you see a message “loading out-of-tree module taints kernel”, you may ignore this.
7. View loaded modules on target:
   ```
   (bbg)$ lsmod
   ```
   - You should now see the `testdriver` loaded.
8. Remove on target (output may appear only in `dmesg`);
   ```
   (bbg)$ sudo rmmod testdriver
   <---- My test driver exit().
   ```

9. Troubleshooting:
   - If commands like `insmod` and `rmmod` fail with “Operation not permitted”, then ensure you are running the command as root (i.e., using `sudo`).
   - If you are unable to mount via NFS, double check:
     - Your mount script is targeting the correct IP address. If you are running on actual Ethernet, you may be at address 192.168.2.2 (instead of ...7.2)! Target the correct server IP.
     - If not using Ethernet-over-USB, then ensure your NFS settings on the server have been correctly updated to permit connections from the 192.168.2.0 subnet.
     - Ensure your network adapters on the host have the correct Ethernet profiles assigned to them. Click the network icon (top right) and for the Ethernet connection in question, force it to a different profile (the non-UBoot one).
     - Try assigning a static IP to the correct network adapter:
       ```
       (host)$ sudo ifconfig enx1cba8ca2ed6c 192.168.7.1
       ```
   - If you get an error that the versions are incomparable (or an invalid file format) consult Section 6.
   - If the Ethernet works in UBoot to boot the custom kernel, but does not stay working under Linux, then try:
     - Reboot to the stock kernel version and mount NFS.
     - Install the drivers for your custom kernel by following the directions section 2.1.
     - Reboot and re-download and run your custom kernel. Now when it finishes booting it should load the Ethernet over USB driver and get the IP address 192.168.7.2 as before.
   - If unable to load Ethernet support at all, you can still write and test a driver using the “Natively Compiling Drivers” directions in section 4.2.
   - If your driver will not correctly load and register (likely as a misc driver) then you may need to reboot your board and try again.

6. Version Incompatibilities

Linux is very strict about what kernel modules (.ko files) it will load. **Specifically, it enforces that the module has the identical version string as the kernel.** The version string is also called version-magic. This is necessary because a driver is linked against a specific kernel version's headers. Any change to these headers can make the drivers perform incorrectly. This section guides you through identifying the versions of a .ko file, and of the Linux kernel, and presents some strategies to get things working.

6.1 Understanding the Problem

1. On the **target**, identify the kernel version you are running. Below is shown the output for a version of the BeagleBone kernel (version installed on your board may be different):
   ```
   (bbg)$ uname -r
   4.1.15-ti-rt-r43
   ```
2. On either the host or the target, find the version of the kernel module you are building:
   - In the folder containing your .ko file, run:
     ```
     (bbg)$ modinfo testdriver.ko
     filename:       /mnt/remote/drivers/testdriver.ko
     license:        GPL
     description:    A simple test driver
     author:         Dr. Evil
     depends:
     vermagic:       5.3.7-bfraser-bone13 mod_unload modversions ARMv7 thumb2 p2v8
     ```
   - Here we see that the "vermagic" is `5.3.7-bfraser-bone13`
   - Sometimes the target will not have the `modinfo` tool and so the host's tool must be used.
3. In the case shown above the version of the kernel does not match the version of the module. The error when attempting to load this module on the incompatibility kernel is:
   ```
   (bbg)$ insmod testdriver.ko
   Error: could not insert module testdriver.ko: Invalid module format
   ```
   - Running `dmesg` shows additional information:
     ```
     (bbg)$ dmesg | tail -1
     [ 57.674358] testdriver: disagrees about version of symbol module_layout
     ```
4. Trouble shooting:
   - If you are working on more than one computer (such as in the lab, or with group members), ensure that all your (or group) build setups are building to the same kernel version, with the same version string. This will reduce the problem of incompatible versions.

6.2 Resolving the Problem

There are a number of options to resolve the above incompatibility:

1. **Rebuild Kernel and Module:**
   - Rebuild both the kernel and the modules (device drivers).
     - If using the scripts described here, run:
       ```
       (host)$ ~/cmpt433/work/bb-kernel/build_kernel.sh
       ```
       and run `make` on your device driver folder.
     - If using the kernel build scripts, run:
       ```
       (host)$ make kernel
       (host)$ make modules
       ```
   - Use U-Boot to re-download the kernel to the target.
   - On the host, copy the newly-built kernel module (.ko) to your NFS directory and then extract them into the `/lib/modules/` directory on the target.
   - Now both the kernel and the module should be the latest version which you are building.
   - If you have any difficulties, re-check that the kernel version and the module versions match. If they don't check out which one did not update correctly. Use "`make menuconfig`" on the Linux kernel to check what version you are trying to build.
   - This is the preferred option because it gets the target fully in synch with the build setup on your host and means that other modules will build and load correctly..
2. **Change the version of the kernel configuration you are building:**
   - Download the exact same version of the kernel that is executing on the target.
   - Use "`make menuconfig`" on the kernel to change the kernel version to match the version of the kernel currently installed on your target.
   - Then, rebuild the module (`make` on your device driver folder) and reload the newly rebuilt .ko module.
   - Since you changed the version to match the kernel you have installed, it should allow the module to load.
3. **Find files that match:**
   - Find a version of the .ko module that matches the version of the kernel that you are running on the target.
   - You might want to look in the /lib directory on the target, and use `modinfo` to check the version magic number.