Session Exercises

Setup requirements

- Beagle Bone Black Kit (BBB+USB Cable)
- USB2TTL Cable
- 4GB or more uSD
- uSD Reader
- Linux Machine (Preferred) or VM with Ubuntu 18.04 or higher

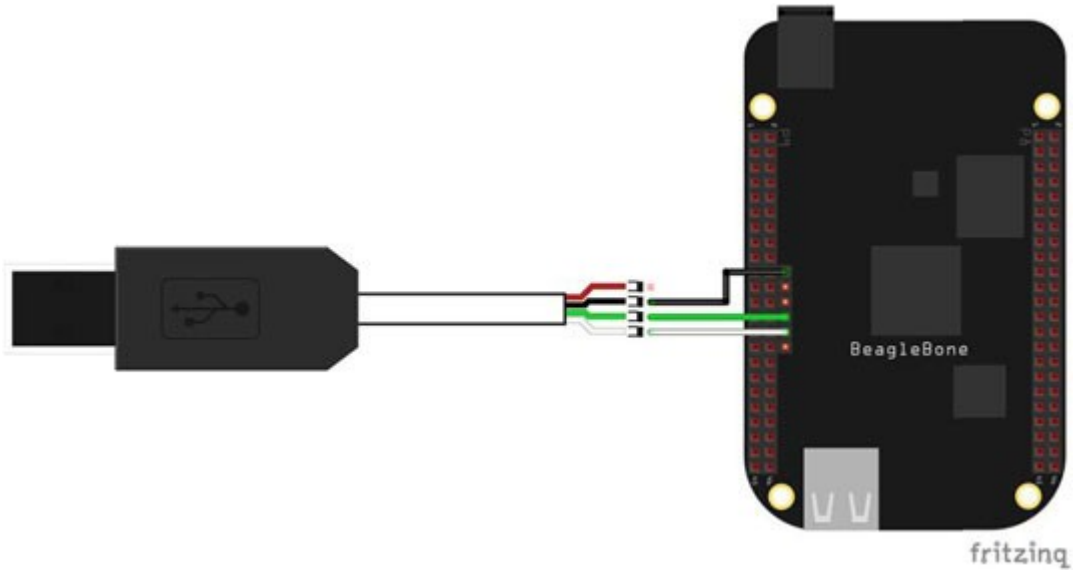
Package Installation

- Install the packages with following command:
  $ sudo apt install sed make binutils gcc g++ bash patch gzip bzip2 perl tar cpio python unzip rsync wget libncurses-dev git minicom libssl-dev flex bison

Default Bootup Setup (One time)

Connect USB2TTL cable

- Connect USB2TTL cable to BBB and system as per below steps. Do not power up BBB
- Connect the USB side of the TTL cable to your computer
- Connect the wires to the J1 header on your BeagleBone Black as shown:
  - Black wire to pin 1
  - Green wire to pin 4
  - White wire to pin 5



Getting the serial console messages

On Linux system (install the minicom package if not already there):

- sudo minicom -s and do the setup for baud 115200 bits 8n1 hw & sw flow control off
- sudo minicom -o
- Power on BBB to boot into Linux
- Login into BBB as root
- uname -r # Verify your original kernel version
    - df -h /boot/uboot | tail -1 | awk '{print $6}' # Determine / or /boot/uboot
    - poweroff

PS If you don't see any boot up messages or login prompt on the serial console, try swapping the green & white wire

Customizing the build-root for Beaglebone Black

Follow the below steps:

1) Create the required directories
   $ cd
   $ mkdir Exercises
   $ git clone https://github.com/buildroot/buildroot
   $ cd buildroot
   $ git checkout -b br-2-2025 2025.02.6

   OR

   If both of above doesn't work, download the tar ball from https://buildroot.org/downloads/buildroot-2025.02.6.tar.gz and untar with

   $ tar -xvf buildroot-2025.02.6.tar.gz
   $ cd buildroot

2) Configure the buildroot by using 'menuconfig' utility
   $ make menuconfig

3) Select Target options menu. Under that
   - Select "Target architecture" as ARM (little endian)
   - Select "Target Architecture Variant" as cortex-A8
   - Select "EABIHF" as Target ABI

4) Select Toolchain Menu
   - Select Toolchain Type as External Toolchain
   - Select Toolchain as ARM14.2 rel1

5) Select the System Configuration Menu. Under that
   Put some custom values for System hostname, System banner and Root password
6) Select the Kernel Menu
   - Select Custom version as "Kernel Version" and enter 6.12.54 in "Kernel Version" text field
   - For kernel configuration, enter 'omap2plus' in the 'defconfig name' option
   - Select 'Build a Device Tree Blob' option and type
   - ti/omap/am335x-boneblack as the 'In-tree Device Tree Source file names'
   - Select "Needs host OpenSSL"

-

7) Leave Target Packages and Filesystem Images as is.
8) Select “Bootloaders” menu and under that
   - Select ‘U-Boot’
   - Select Kconfig as “Build system”
   - Select ‘Custom Version’ for ‘U-Boot Version’ and below that select 2024.04 as custom version
   - Use ‘am335x_evm’ as ‘Board defconfig’
   - Select u-boot.img as the ‘U-Boot binary format’
   - Select ‘Install U-Boot SPL binary image’ and Use ‘MLO’ as ‘U-Boot SPL/TPL binary image name(s)’
   - DEVICE_TREE=am335x-boneblack as “Custom make options”

9) Exit & save the configuration
10) Build the images using
    $ make 2>&1 | tee build.log

Preparing the SD Card

1. Connect the SD Card to the host machine and figure out the corresponding device file (usually /dev/sdb or /dev/mmcblk0). Use dmesg to figure out the device file for the SD card.
2. Unmount all partitions of your SD card (they are generally automatically mounted by Ubuntu).
   $ umount /dev/sdb* (Considering the sd card device file is /dev/sdb)
3. Erase the beginning of the SD card to ensure that the existing partitions are not going to be mistakenly detected:
   $ sudo dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=16

   In general, if you use the internal SD card reader of a laptop, it will be mmcblk0, while if you use an external USB SD card reader, it will be sdX (i.e. sdb, sdc, etc.). Be careful /dev/sda and /dev/sdb are typically the harddisks as well

4. SD Card needs to be split in 2 partitions:
   - First Partition of type FAT32 would hold Bootloaders (MLO and u-boot.img), kernel image (zImage) and Device Tree (am335x-boneblack.dtb)
   - Second partition for the root filesystem

5. Start the cfdisk tool for that:
   $ sudo cfdisk <device file for SD card (eg. /dev/mmcblk0)
6. Choose the dos partition table
7. Create the first partition of size 256MB, primary, with type e (W95 FAT16) and mark it bootable
8. Create a second partition, also primary, of size 2GB, with type 83 (Linux)
9. Create a third partition, also primary, with the rest of the available space, with type 83 (Linux)
10. Write the partition table
11. Exit cfdisk
12. Format the first partition
    $ sudo mkfs.vfat -F 32 -n boot /dev/mmcblk0p1 **(Would be /dev/sdb1 if device file is /dev/sdb)**

13. Format the Second partition
    $ sudo mkfs.ext4 -L FirstRootfs /dev/mmcblk0p2
14. Format the Third partition
    $ sudo mkfs.ext4 -L SecondRootfs /dev/mmcblk0p3
15. Remove the SD card and insert it again, the two partitions should be mounted automatically, in /media/$USER/boot and /media/$USER/FirstRootfs
16. Copy MLO, u-boot.img, am335x-boneblack.dtb and zImage to the boot partition of the SD card. The images can be found under <buildroot>/output/images directory
17. Extract the rootfs.tar file to the Second partition
    $ cd <Path to Buildroot>/output
    $ sudo tar -C /media/$USER/FirstRootfs/ -xf images/rootfs.tar (Need to make sure where the mountpoint is)
18. Next create the directory by name extlinux in the boot (first) partition and copy the extlinux.conf under it (extlinux.conf could be found under git repo or shared point. Typically under Images directory)
19. Unmount the paritions, remove the SD Card, put it into the card and power up the board.
20. If board doesn't boot and shows 'CCCC', then raw dump the bootloaders using the following steps.
21. umount the SD Card using 'umount /dev/sdb*'
22. Next, dummp MLO using disk dump utility as below:
    $ cd <path to buildroot>/output/images
    $ sudo dd if=MLO of=<device file for sd card> seek=256 bs=512 conv=notrunc
    $ sudo dd if=u-boot.img of=<device file for sd card> seek=768 bs=512 conv=notrunc

Adding an init script to the rootfs

The idea over here is to add the script which automatically gets executed during the boot up. The script would set up the network over usb

1. Get into the buidroot directory
   $ cd Exercises/buildroot
   Create the directory with name rootfs-overlay
   $ mkdir -p board/test/bbb/rootfs-overlay/
2. Since the file needs to be created under /etc/init.d, let's create the similar directory structure and add the script
   $ mkdir -p board/test/bbb/rootfs-overlay/etc/init.d/
   $ cp <path to repo>/Images/S30usbgadget board/test/bbb/rootfs-overlay/etc/init.d/ (S30usbgadget is available in repo (under Scripts) or sharepoint provided during the training)
3. Add executable permission for S30usbgadget
   $ chmod a+x board/test/bbb/rootfs-overlay/etc/init.d/S30usbgadget
4. Configure the path for rootfs-overlay in buildroot
   $ cd board/test/bbb/rootfs-overlay
   $ pwd (copy the path)
   $ cd -
   $ make menuconfig
   Locate Root filesystem overlay directories (ROOTFS_OVERLAY) under System configuration and paste the path for the overlay directory

5. Next thing is to add the script for configuring the network
   $ mkdir -p board/test/bbb/rootfs-overlay/etc/network/

Copy the interfaces script (From the Scripts folder of the repo or sharepoint)
$ cp <path to interfaces> board/test/bbb/rootfs-overlay/etc/network/

6. $ make
7. Plug the SD Card into Laptop
8. Untar the rootfs.tar to the SD card
   $ cd <path to buildroot>
   $ sudo rm -rf /media/$USER/FirstRootfs/* (Make sure that its mounted)
   $ sudo tar -C /media/$USER/FirstRootfs/ -xvf output/images/rootfs.tar
   (Rootfs for SD card would be under /media/$USER/FirstRootfs/)
9. Unmount the SD card, plug it into the board and boot up the board
10. Configure the host network address using:
    nmcli con add type ethernet ifname enxf8dc7a000001 ip4 192.168.7.3/24
    (To be executed On PC/Laptop)
11. Verify if you are able to ping the system using 'ping 192.168.7.2'

Adding dropbear as an ssh server

The idea over here is to add the ssh and scp capability into the Root Filesystem:

1. Get into the buidroot directory
   $ cd Exercises/buildroot
1. $make menuconfig
2. Search for dropbear by pressing '/' and then enter DROPBEAR
   It will give you a list of results, and each result is
   associated with a number between parenthesis, like (1). Then simply press 1, and menuconfig will jump to the right option.
3. Select the dropbear, exit the menuconfig and initiate the build
   $ make
4. Insert the SD card in PC & update the rootfs in SD Card
   $ cd <path to buildroot>/
   $ sudo rm -rf /media/$USER/FirstRootfs/* (Make sure that the FirstRootfs is mounted)
   $ sudo tar -C /media/$USER/FirstRootfs/ -xvf output/images/rootfs.tar
5. Unmount the SD card partitions and test it on the board
6. Try to connect with the board using
   $ ssh root@192.168.7.2
   if prompted for password, set the password at beaglebone black with 'passwd'. Try to set the small password to avoid typing it all the time

U-Boot

Adding the command in u-boot

The idea over here is to add the custom command in the uboot

1. Get into the u-boot directory
   $ cd <path to buildroot>/output/build/uboot-2024.04
2. Make a copy of already existing command say led.c under cmd
   $ cd cmd
   $ cp led.c myprint.c
3. Modify the myprint.c as follows:
   - Remove everything except do_led and U_BOOT_CMD
   - Change the name of the function do_led to do_myprint
   - Deleting everything in do_led, except the 'return 0'
   - Include the 'printf' statement above 'return 0'
4. Modify the U_BOOT_CMD as below:
   U_BOOT_CMD(

```
        myprint, 1, 1, do_myprint,
        "My printf",
        "My Test command"
);
```

5. Next thing is to modify the Kconfig to give out the menu option for our command. For this, modify the Kconfig to add the menu option as below:
   ```
   config CMD_MYPRINT
        bool "MY Test Print"
        help
          Enable the Test printf
   ```
6. Next thing is to modify the Makefile to add the below line:
   obj-$(CONFIG_CMD_MYPRINT) += myprint.o
7. Get into the buildroot directory and execute the following
   $ make uboot-menuconfig
   Search for MYPRINT and select this option
8. Rebuild the uboot
   $ make uboot-rebuild
9. Check if myprint.o is generated under buildroot/output/build/uboot-2021.04/cmd/
10. Update the u-boot in the SD Card's first partition (applicable only if MLO & u-boot.img are not raw-dumped. If its raw-dumped, skip to the next step)
    This can be done in 2 ways.
    Using the scp command:
    $ mount /dev/mmcblk1p0 /mnt (on the board)
    $ scp buildroot/output/images/u-boot.img root@192.168.7.2:/mnt/ (On host)
    $ sudo umount /mnt (on board)
    $ reboot & stop at the uboot prompt by pressing 'Enter' during boot up
    Another way is to Transfer by inserting the SD card in PC
    $ cp buildroot/output/images/u-boot.img /media/$USER/boot/
    $ Unmount the SD card
11. if u-boot is raw-dumped, then follow the below steps.
    This can be done in 2 ways.
    (i) Using the scp command:
    $ scp buildroot/output/images/u-boot.img root@192.168.7.2:(On host)
    $ dd if=u-boot.img of=<device file for sd card> seek=768 bs=512 conv=notrunc
    $ reboot & stop at the uboot prompt by pressing 'Enter' during boot up
    Or (ii) Transfer by inserting the SD card in PC:
    $ sudo dd if=u-boot.img of=<device file for sd card> seek=768 bs=512 conv=notrunc
    $ Unmount the SD card and plug it into the board
12. While booting up, press 'Enter' or 'Space' key to get the uboot prompt
13. Test if myprint command is available in the prompt & execute the same

Creating & Using the uboot patch

The idea over here is to make the uboot changes persistent across the build. This can be achieved by creating the patches and saving it in appropriate board directory

1. Get into the uboot build directory
   $ cd <path to buildroot>/output/build/u-boot-2024.04
2. Temporarily mv the changes to some other place and Intialize the git for uboot and commit the initial changes
   $ git init

$ git add .
$ git commit (Add the commit message & exit)

3. Copy back the relevant files (myprint.c, Makefile and Kconfi) to respective directory (i.e at uboot-2024.04/cmd)
   $ git add .
   $ git commit (Add the comment & exit)
   $ git add cmd/Kconfig
   $ git commit (Add the comment & exit)
4. Create the patches
   $ git format-patch -2
   This would create 2 patches
5. Next step is to create the patches directory in buildroot and copy the patches
   $ cd buildroot
   $ mkdir board/test/patches/uboot/
   $ cp ../u-boot-2024.04/*.patch board/test/patches/uboot/
6. Next, update patches path in menuconfig
   $ make menuconfig
   Navigate to Bootloaders and update the path in 'Custom U-boot Patches' to point to above patch directory
7. Next step is to create the defconfig for uboot
   $ make uboot-menuconfig
   Search for MYPRINT, select the same & exit the menuconfig
8. Generate the defconfig for these changes
   $ make uboot-savedefconfig
   This would generate the defconfig at u-boot build directory. Verify if CONFIG_MYPRINT is set. Copy the above configuration file at board/test/bbb/uboot.config
9. Run the menuconfig for the buildroot
   $ make menuconfig
   In uboot, instead of using a defconfig, chose using a custom config file
   In the configuration file path, enter the path board/test/bbb/uboot.config
   Exit the menuconfig
10. For testing, let's first dir-clean the uboot and perform menuconfig again
    $ make uboot-dirclean
    $ make uboot-menuconfig
    Check if MYPRINT option is already set and also the patch shoulod been applied as well
11. $ make uboot-rebuild
    Make sure that myprint.o is generated. This means the patch is applied correctly and the uboot is compiled with the same

Kernel

Using custom defconfig for the kernel

The idea is to modify the kernel configuration as per the requirement, test it and make it permanent

1 Start the Kernel menuconfig tool
  $ make linux-menuconfig
  Update the Local Version option in General Setup Menu

2. Exit the menuconfig and this would generate the updated .config file under output/build/linux-<version>
   This is the temporary change and would be deleted at the next make clean. So, the idea is to make this change persistent
3. Run buildroot menuconfig
   $ make menuconfig
   In the Kernel menu, instead of Using a defconfig, chose Using a custom config file. This will allow us to use our own custom kernel configuration file, instead of a pre-defined defconfig that comes with the kernel sources.
   In the Configuration file path, enter board/test/bbb/linux.config
4. Exit menuconfig
5. Run ‘make linux-update-defconfig’
   This will generate the configuration file in board/test/bbb/linux.config. It will be a minimal configuration file.
6. Now, you may restart the build of the kernel

Code Changes & Patching the Kernel

The Idea over here is to modify the kernel code, test the changes, create the patches and apply the same

1. Get into <buildroot>/output/build/linux-<version>/ and modify drivers/char/misc.c by adding the printk statement
2. Rebuild the kernel
   make linux-rebuild
   This would generate the updated image. Test the same
3. Next step is to generate the patch for the above changes and place it under the directory – board/test/bbb/patches/linux
4. The buildroot needs to be configured to apply these patches before building the kernel. To do so, run menuconfig, go the to the Build options menu, and adjust the Global patch directories option to board/test/bbb/patches/
   $ make linux-menuconfig
5. Clean up the linux package completely so that its sources will be re-extracted and our patches get applied the next time
6. $ make linux-dirclean
   In output/build/, the linux-<version> directory will have disappeared
7. Next, run ‘make linux-menuconfig’
   This will Extract the Linux kernel sources, apply the patch, load the kenrel config and starts the kernel menuconfig tool.
8. Build the kernel with linux-rebuild