Linux Kernel Training: Lecture 2

Booting the BeagleBone Black

Sam Protsenko

March 27, 2020

GlobalLogic

Agenda

1. U-Boot Basics
2. IDE Considerations
3. Workshop: Bringing up the BBB

U-Boot Basics

Embedded Board

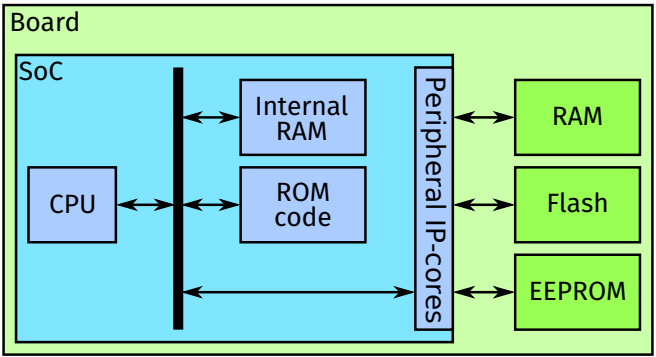


Figure 1: Simplified view of Embedded board

What is Bootloader?

ROM code has limitations:

- Doesn't know about RAM
- Doesn't know board name
- Not flexible enough

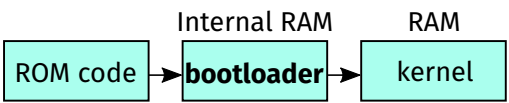
What is Bootloader?

ROM code has limitations:

- Doesn't know about RAM
- Doesn't know board name
- Not flexible enough

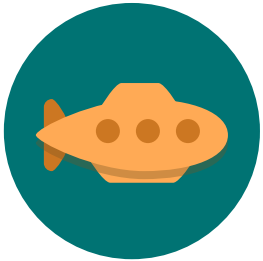
Bootloader to the rescue:

- Resides in flash (can be upgraded)
- Able to configure RAM
- Knows boot procedure
- Convenient features



Why U-Boot?

- Bootloader for Embedded boards
  - Popular for Android devices
  - Adoption in automotive
- GPLv2
- 13 architectures (consider ARM)
- ~300 boards
- Device drivers, lib routines
- Resembles Linux kernel a lot
- Scripting, extensive command set



U-Boot Features

- Boots from various sources
- Boots various OSs
- Monitor (U-Boot shell)
  - Commands
  - Environment
- 2 stage boot (SPL + U-Boot)
- “Falcon” mode (SPL only)

Two Stage Boot

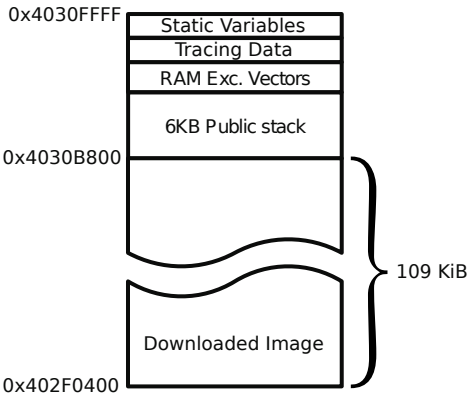


Figure 2: SRAM layout (from AM335x TRM)

Two Stage Boot

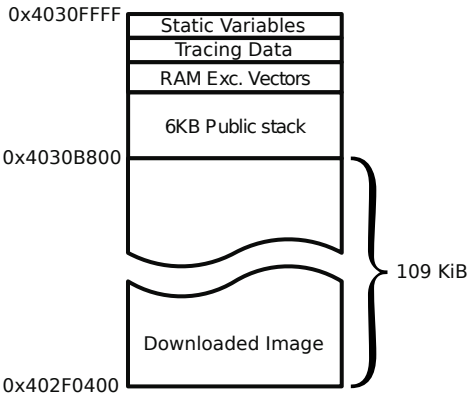


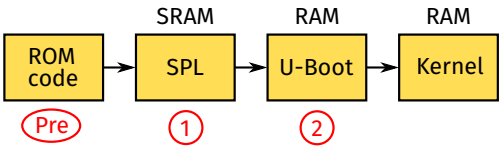
Figure 2: SRAM layout (from AM335x TRM)

But bootloader is bigger!

For BeagleBone Black, `u-boot.img` is 391 KiB.

Two Stage Boot (cont'd)

Let's add an intermediate stage (SPL):

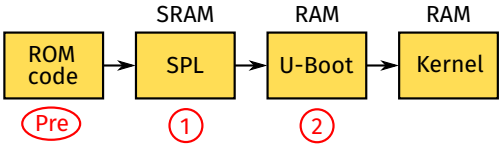


For **BeagleBone Black**:

Stage	Size
SPL	75 KiB
U-Boot	391 KiB

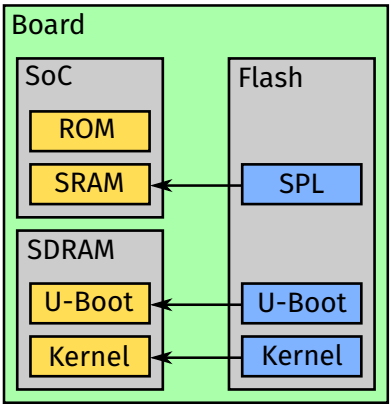
Two Stage Boot (cont'd)

Let's add an intermediate stage (SPL):



For BeagleBone Black:

Stage	Size
SPL	75 KiB
U-Boot	391 KiB



Boot Sources

ROM code boot source is usually selected:

- Via **SYSBOOT** DIP switch
- By hard-wiring SYSBOOT lines (pull-up/pull-down)
- Using some USER button
- By inserting SD card (“Card Detect” pin)

Boot Sources

ROM code boot source is usually selected:

- Via **SYSBOOT** DIP switch
- By hard-wiring SYSBOOT lines (pull-up/pull-down)
- Using some USER button
- By inserting SD card (“Card Detect” pin)

U-Boot can boot kernel from:

- Flash devices (eMMC, SD card)
- Network boot (TFTP, NFS)
- Peripheral boot (USB, serial console)

Flashing Methods

Most commonly used flashing methods:

- Via USB:
  - fastboot (Android)
  - DFU
- Via SD card

Flashing Methods

Most commonly used flashing methods:

- Via USB:
  - fastboot (Android)
  - DFU
- Via SD card

To unbrick the board (bad U-Boot on eMMC):

- Boot from SD card and re-flash
- YMODEM boot (via UART)
- USB peripheral boot
- Use JTAG

U-Boot Shell

- U-Boot has a “monitor” program
- Command line interface
- A set of commands is implemented
- Resembles Bash (has scripting capabilities)
- Press “Space” to get into U-Boot shell

U-Boot Shell

- U-Boot has a “monitor” program
- Command line interface
- A set of commands is implemented
- Resembles Bash (has scripting capabilities)
- Press “Space” to get into U-Boot shell

Most commonly used commands:

```
bdinfo, bootm, bootz, crc32, dfu, dhcp, env,
fastboot, fatload, fdt, gpt, i2c, md, mmc, mw,
part, ping, reset, run, tftpboot, version, ...
```

U-Boot Environment

- Keeps all U-Boot shell variables
- Just a set of strings in `key=value` format

U-Boot Environment

- Keeps all U-Boot shell variables
- Just a set of strings in `key=value` format

Set default environment:

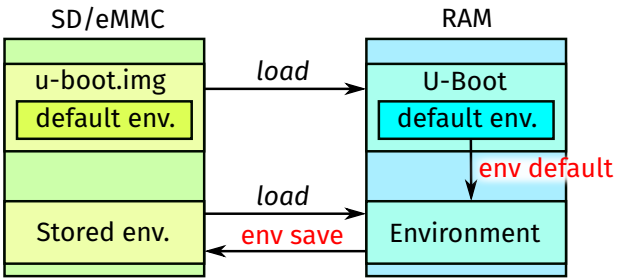
```
=> env default -f -a
=> env save
```

U-Boot Environment

- Keeps all U-Boot shell variables
- Just a set of strings in `key=value` format

Set default environment:

```
=> env default -f -a
=> env save
```



U-Boot Environment (cont'd)

```
=> env print

board_name=A335BNLT
board_rev=0A5A
board_serial=1813BBBK4642
bootargs=
bootcmd=if test ${boot_fit} -eq 1; then ...
fdtfile=am335x-boneblack.dtb
findfdt=if test $board_name = A335BONE; then ...
partitions=uuid_disk=${uuid_gpt_disk}; ...
```

Use-case: Format eMMC

Using `gpt` command:

```
=> gpt write mmc 1 $partitions
```

Use-case: Format eMMC

Using `gpt` command:

```
=> gpt write mmc 1 $partitions
```

or Android way:

```
=> fastboot 0
$ fastboot oem format
```

Use-case: Format eMMC

Using `gpt` command:

```
=> gpt write mmc 1 $partitions
```

or Android way:

```
=> fastboot 0
$ fastboot oem format
```

- Consider `$partitions` variable...
- Check with `part list mmc 1`

Use-case: Flashing eMMC

Via DFU:

```
=> setenv dfu_alt_info $dfu_alt_info_emmc
=> dfu 0 mmc 1
$ dfu-util -D MLO -a MLO.raw
```

Use-case: Flashing eMMC

Via DFU:

```
=> setenv dfu_alt_info $dfu_alt_info_emmc
=> dfu 0 mmc 1
$ dfu-util -D MLO -a MLO.raw
```

Via fastboot (Android way):

```
=> fastboot 0
$ fastboot flash rootfs rootfs.img
```

Use-case: Boot Linux

```
=> setenv mmcdev 1
=> setenv bootpart 1:2
=> run findfdt
=> run mmcboot
```

Use-case: Boot Linux

```
=> setenv mmcdev 1
=> setenv bootpart 1:2
=> run findfdt
=> run mmcboot


mmcboot=
  mmc dev ${mmcdev}; mmc rescan;
  load mmc ${bootpart} ${loadaddr} /boot/zImage
  load mmc ${bootpart} ${fdtaddr} /boot/${fdtfile}
  setenv bootargs console=ttyO0,115200n8 root=... rw ...
  bootz ${loadaddr} - ${fdtaddr}
```

Boot Sequence

`do_bootm_linux()` calls:

- 1:
  - `boot_prep_linux()`
  - `boot_setup_linux()`
  - `image_setup_libfdt()`
  - Populates `/chosen` node with `bootargs`, `initrd-start`, etc.
- 2:
  - `boot_jump_linux()`
  - `kernel_entry(0, machid, r2)`
  - FDT blob address is passed to kernel via `r2`

Boot Methods Table

Boot method	Description
SD card boot	- Unbrick the board - Doesn’t touch eMMC
eMMC boot	- Regular boot - Fastest and easiest for user
TFTP boot	- Network boot - Doesn’t touch eMMC - Volatile RootFS (RAM disk)
NFS boot	- Network boot - Transparent RootFS (from host) - Useful for kernel development

IDE Considerations

IDE for Kernel Development: Vim

Pros:

- Very fast
- Available everywhere
- A lot of kernel hackers using it

Cons:

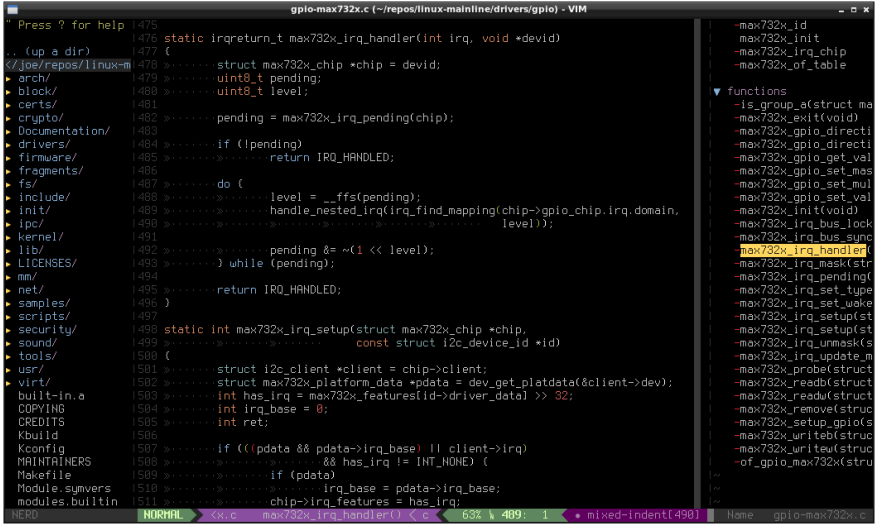
- Learning curve rather steep
- Some functionality is missing
- Requires some configuration

Indexing example (simplified):

```
make ARCH=arm SUBARCH=omap2 cscope tags
```

Details: https://stackoverflow.com/a/33682137/3866447

IDE for Kernel Development: Vim (cont'd)



IDE for Kernel Development: Eclipse

Pros:

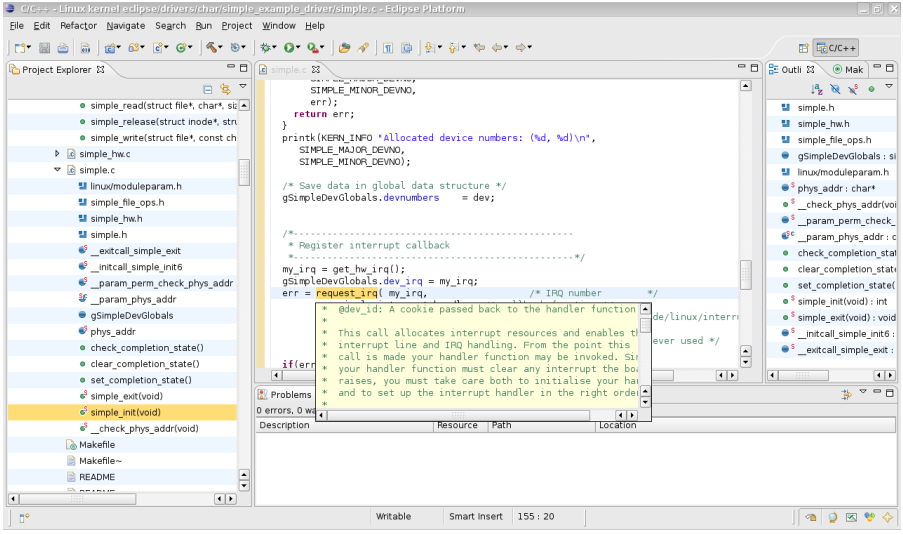
- Easy to use
- Nice tools (indexing, macro unwrapping, etc)

Cons:

- Very slow
- Uses a lot of resources
- Not available on servers, etc.

Be sure to configure CDT for kernel development. Details:
https://wiki.eclipse.org/HowTo_use_the_CDT_to_navigate_Linux_kernel_source

IDE for Kernel Development: Eclipse (cont'd)



IDE for Kernel Development: Other Choices

Other possible IDEs:

- Emacs
- KDevelop
- QtCreator

Kernel source web-browser (very useful):

- elixir.bootlin.com

Questions so far?

Take Five

Workshop: Bringing up the BBB

Short Quiz

1. Who didn’t manage to build all the SW?
2. Who doesn’t have a laptop along?

Training Centre Infrastructure

Training Centre PC info:

- Press F9 on boot (show boot menu)
- Select second drive (TS64GSSD370S, 64 GB)
- Login: Lin-Ker
- Password: 123

What does it have?

- Ubuntu 18.04, internet connection, prepared environment
- Toolchains (`/opt/*`)
- U-Boot, Linux kernel, BusyBox (see `~/repos/*`)
- **Missing**: TRM, datasheet, schematic, BBB instructions guide

Training Centre Infrastructure (cont'd)

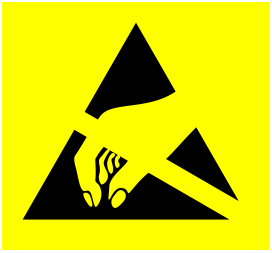
Notes:

- Better to use home laptop to keep the single environment
- We only have one card reader :(

Hardware Equipment

- **The board**: BeagleBone Black (Rev C)
- **Serial cable**: TTL-232R-3V3
- **OTG USB cable**: mini-USB to USB
- **micro-SD card + adapter**: 8 GB, class 10
- **Ethernet patch cord**: for network boot and networking tasks
- **Development kit**: a set of hardware, will be used later

ESD Safety Note



- Discharge before touching
- Use ESD mat and wrist strap
- Don’t wear ESD unsafe clothes
- Remove power plug before touching

Connectivity: Overview

Now we are going to:

- Connect mini-USB cable
  - Powering the board (low consumption use-case)
  - Flashing (`fastboot`, `dfu-util`)
- Connect serial console cable (white dot = black wire = GROUND)
  - Will be used via `minicom` tool
- Insert SD card to the slot (once it's flashed with software)

Connectivity: Serial Console

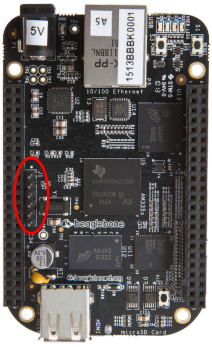


Figure 3: BeagleBone Black serial connection

Figure 4: TTL-232R-3V3 FTDI cable

Connectivity: Serial Console (cont'd)

- Board uses UART port (TTL levels, 3.3V) to communicate:
  - Rx line for receiving characters from host
  - Tx line for transceiving characters to host
- Chip in adapter cable converts UART <-> USB
  - FT232R (FTDI): reliable, recommended
  - PL2303: cheap alternative
- Chip driver exposes `/dev/ttyUSB` file (char device)
- `minicom` (or similar tool) allows user to interact with board via `/dev/ttyUSB`

Connectivity: Client USB (OTG)

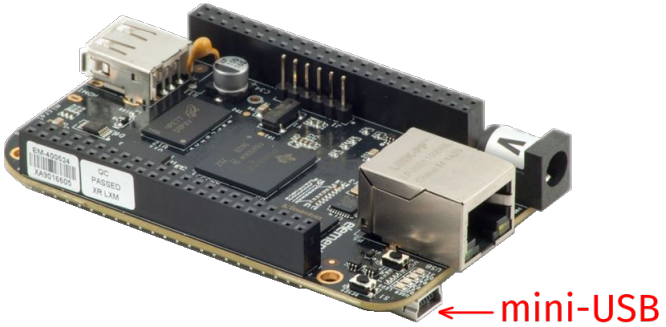


Figure 5: BeagleBone Black USB connection

Configure and Run `minicom`

- Run minicom configuration:

```
$ sudo minicom -s
```

- Select “Serial port setup” menu item and choose next settings:
  - Serial Device: `/dev/ttyUSB0`
  - Bps/Par/Bits: 115200 8N1
  - Hardware Flow Control: No
- Then select “Save setup as dfl” and “Exit from Minicom”
- Run minicom:

```
$ sudo minicom
```

Configure and Run `minicom` (cont'd)

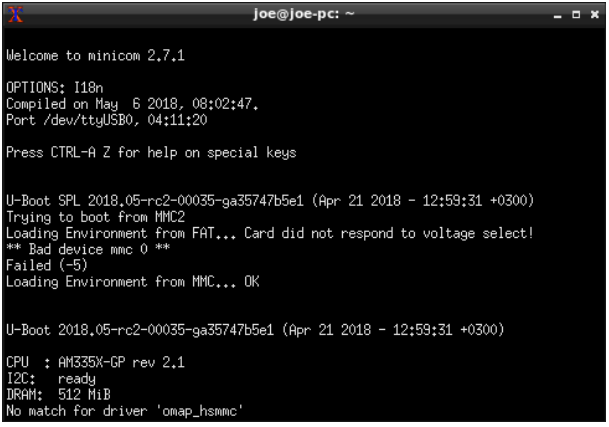


Figure 6: Terminal with minicom

Formatting SD Card (page 1)

- Insert your SD card in your laptop (use adapter)
- Locate SD card device file (should be `/dev/mmcblk0`):

```
$ sudo dmesg | tail
$ sudo fdisk -l
```

- Unmount SD card (mounted automatically by Ubuntu):

```
$ sudo umount /dev/mmcblk0p1
$ sudo umount /dev/mmcblk0p2
```

- Clear SD card MBR:

```
$ sudo dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=1
```

Formatting SD Card (page 2)

- Create new partition table:

```
$ sudo sfdisk /dev/mmcblk0 << EOF
2048,100M,0x0c,*
,,L,-
EOF
```

- Format both partitions:

```
$ sudo mkfs.vfat -F 32 -n "boot" /dev/mmcblk0p1
$ sudo mkfs.ext4 -F -L "rootfs"  /dev/mmcblk0p2
```

Formatting SD Card (page 3)

- Check new partitions:

```
$ sudo fdisk -l
```

- You should see something like this:

```
Device          Boot  Start  Size  Id  Type
/dev/mmcblk0p1  *      2048  100M   c  W95 FAT32 (LBA)
/dev/mmcblk0p2       206848  14.9G 83  Linux
```

Prepare Bootable SD Card (page 1)

- Unmount SD card (mounted automatically by Ubuntu):

```
$ sudo umount /dev/mmcblk0p1
$ sudo umount /dev/mmcblk0p2
```

- Mount partitions:

```
$ sudo mkdir /mnt/{boot,rootfs}
$ sudo mount /dev/mmcblk0p1 /mnt/boot
$ sudo mount /dev/mmcblk0p2 /mnt/rootfs
```

- Check that partitions mounted properly:

```
$ mount | grep mmcblk
```

Prepare Bootable SD Card (page 2)

- Copy U-Boot files to SD card (“boot” partition):

```
$ cd ~/repos/u-boot
$ sudo cp MLO u-boot.img /mnt/boot
```

- Copy rootfs files to SD card (“rootfs” partition):

```
$ cd ~/repos/busybox/_install
$ sudo cp -R . /mnt/rootfs
```

Prepare Bootable SD Card (page 3)

- Unmount SD card:

```
$ sudo umount /mnt/boot
$ sudo umount /mnt/rootfs
```

- Remove your SD card from your laptop

Booting from SD Card

- Unplug mini-USB cable from the board
- Insert SD card in board's slot
- Press and hold USER/BOOT button
- Plug power cable and mini-USB cable back to board
- Release USER/BOOT button
- BusyBox will be loaded from SD card

Assignments

Assignment #1

- Finish up the whole BBB guide:
  - Do all kinds of boot, using 3rd chapter (eMMC boot, TFPT boot, NFS boot)
  - Use previously built software (from lecture 1 assignment)
- Proof:
  - Send me screenshots that prove eMMC boot works for you
  - Send me screenshots that prove NFS boot works for you

Assignment #2

- Install and configure IDE of your choice (use links from slides)
- In your IDE (work dir must be the kernel source dir):
  - Open `init/main.c` file
  - Find `start_kernel()` function implementation
  - In `start_kernel()`, look for `setup_arch()` call
  - Jump to `setup_arch()` (using IDE capabilities)
  - Get back to `start_kernel()` (using IDE capabilities)
  - Find where `start_kernel()` is being called from (using IDE capabilities)
- Proof: send me screenshot that shows your configured IDE

Thank you!

Appendix: Abbreviations

Abbreviations

- BBB - BeagleBone Black
- DFU - Device Firmware Upgrade
- DTB - Device Tree Blob
- eMMC - Embedded MMC (MultiMedia Card)
- ESD - Electrostatic Discharge
- FDT - Flattened Device Tree (the same as DTB)
- FTDI - Future Technology Devices International (company name)
- GPIO - General Purpose Input Output
- GPT - GUID Partition Table
- IDE - Integrated Development Environment
- LED - Light-Emitting Diode

Abbreviations (cont'd)

- MBR - Master Boot Record
- MLO - MMC Loader (TI image format for SPL)
- NFS - Network File System
- OTG - USB On-The-Go
- PCB - Printed Circuit Board
- PHY - Physical layer chip
- PMIC - Power Management IC (Integrated Circuit)
- RootFS - Root File System
- SD card - Secure Digital card
- SoC - System on Chip
- SPL - Secondary Program Loader (first part of U-Boot)

Abbreviations (cont'd)

- TFTP - Trivial File Transport Protocol
- TRM - Technical Reference Manual
- TTL - Transistor-to-Transistor Logic
- UART - Universal Asynchronous Receiver-Transmitter