Build & Boot the Custom Linux Kernel Assignment

Conduct the following steps to accomplish various tasks in completing this assignment.

1. Informational Only

A kernel build generates two files in the top level directory: vmlinux and System.map.

The first, vmlinux, is the kernel as an ELF binary.

$ arm-cortex_a8-linux-gnueabihf-size vmlinux

This build the kernel, but show abbreviated command creating zimage

$ make -j 4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- zImage

This build kernel, but show the actual command

$ make -j 4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- V=1 zImage

<u>Compiling device trees</u>

To build the device tree, or trees if you have a multi-platform build. The dtbs target builds device trees according to the rules in arch/$ARCH/boot/dts/Makefile, using the device tree source files in that directory.

Following is a snippet from building the dtbs target for multi_v7_defconfig:

$ make ARCH=arm dtbs

<u>Compiling modules</u>

To build modules, you can build them separately using the modules target:

$ make -j 4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- modules

The compiled modules have a .ko suffix and are generated in the same directory as the source code, meaning that they are scattered all around the kernel source tree.

To install them into the staging area of your root filesystem (we will talk about root filesystems in the next chapter), provide the path using INSTALL_MOD_PATH:

$ make -j4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- \

INSTALL_MOD_PATH=$HOME/rootfs modules_install

Kernel modules are put in directory /lib/modules/[kernel version], relative to the root filesystem.

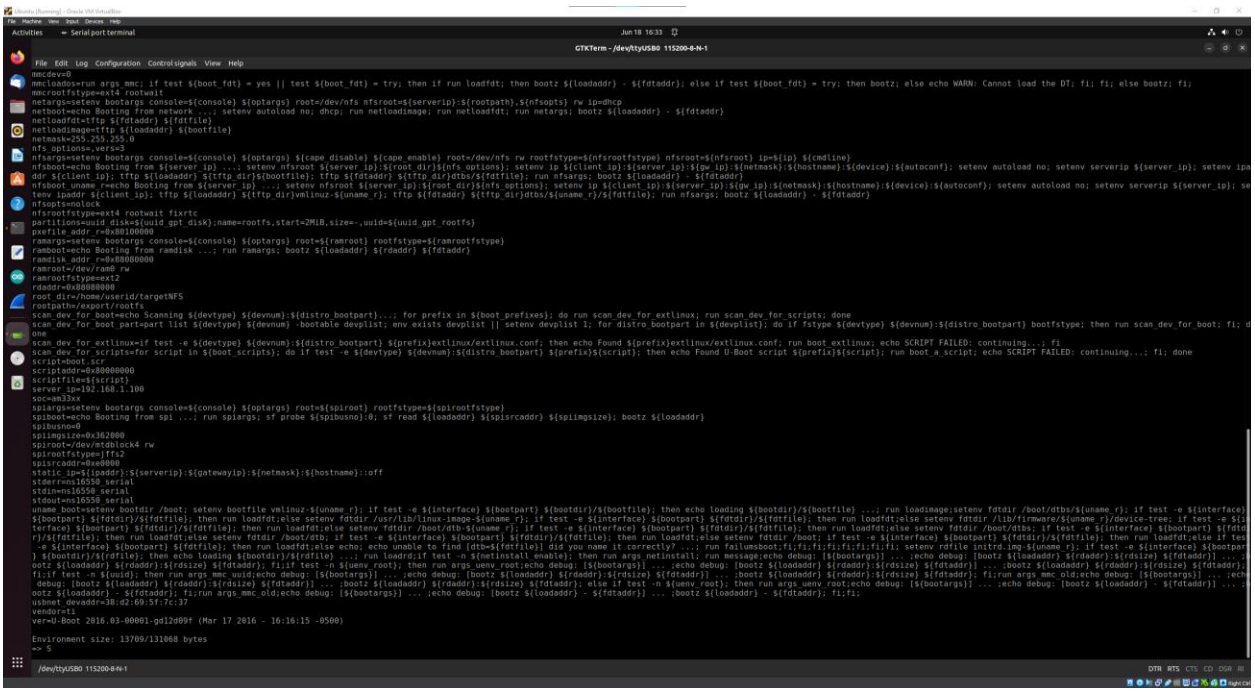
2. Follow these Assignment Step:

Boot BBB again

And once you see the U-boot logo, press the space bar to have it stop its search of Linux, then enter the u-boot command pr at the prompt, to pr a list of configuration options.

(take screenshot of the values)

As Below:



Linux Kernel

There are two options that you can follow, download a beagleboneblack linux version from github or download generic Linux from github, we will work with generic Linux, but both options are show.

Option 1:

On the Linux VM, clone the Beaglebone Linux kernel:

```
git clone git://github.com/beagleboard/Linux.git
```

Option 2:

Use this command to clone the stable kernel and check out the current master:

$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git

$ cd linux-stable

$ git checkout master

$ make ARCH=arm kernelversion

(to display the current version, what does it show)

As Below:

```
sebastian@Ubuntu:~/linux-stable$ git checkout master
Already on 'master'
Your branch is up to date with 'origin/master'.
sebastian@Ubuntu:~/linux-stable$ make ARCH=arm kernelversion
6.4.0-rc6
sebastian@Ubuntu:~/linux-stable$
```

Must redo the toolchain and set the option as follows:

"Target options --> Use specific FPU" and type "vfpv3" which is the floating point unit specification for the Arm Cortex A8. (make sure all setting are as indicated before, but with this added requirement)

Building a kernel for the BeagleBone Black (conduct these steps)

Complete sequence of commands to build a kernel, the modules, and a device tree for the BeagleBone Black, using the Crosstool-NG ARM Cortex A8 cross compiler:

$ sudo apt-get install libgmp3-dev

$ sudo apt-get install libmpc-dev

(must be done everytime to establish the PATH to the toolchain and env variables)

$ PATH=${HOME}/x-tools/arm-cortex_a8-linux-gnueabihf/bin/:$PATH

$ export CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf-

$ export ARCH=arm

$ export SYSROOT=$(arm-cortex_a8-linux-gnueabihf-gcc -print-sysroot)

$ cd linux-stable

(explain the difference between these two approaches to prepare, use the second one)

$ make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- mrproper

$ make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- distclean

$ make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- multi_v7_defconfig

$ make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- menuconfig

$ make -j4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- zImage

$ make -j4 ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- modules

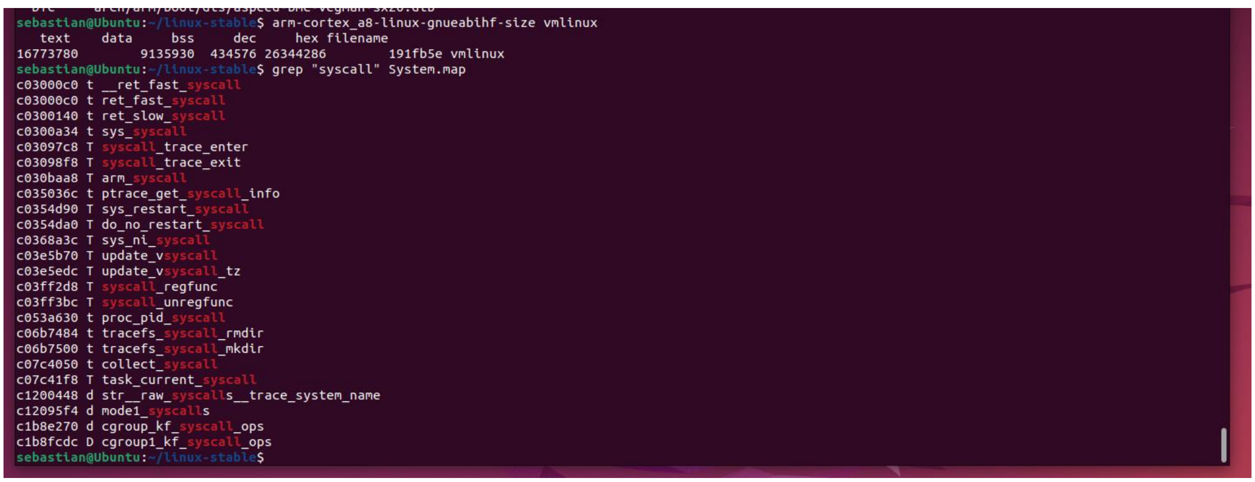
$ make ARCH=arm CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- dtbs

Now run these command and explain what they are outputting:

$ arm-cortex_a8-linux-gnueabihf-size vmlinux

$ grep "syscall" System.map

As Below:



```
sebastian@Ubuntu:~/linux-stable$ arm-cortex_a8-linux-gnueabihf-size vmlinux
   text    data     bss     dec     hex filename
16773780        9135930  434576 26344286        191fb5e vmlinux
sebastian@Ubuntu:~/linux-stable$ grep "syscall" System.map
c03000c0 t __ret_fast_syscall
c03000c0 t ret_fast_syscall
c0300140 t ret_slow_syscall
c0300a34 t sys_syscall
c03097c8 T syscall_trace_enter
c03098f8 T syscall_trace_exit
c030baa8 T arm_syscall
c035036c t ptrace_get_syscall_info
c0354d90 T sys_restart_syscall
c0354da0 T do_no_restart_syscall
c0368a3c T sys_ni_syscall
c03e5b70 T update_vsyscall
c03e5edc T update_vsyscall_tz
c03ff2d8 T syscall_regfunc
c03ff3bc T syscall_unregfunc
c053a630 t proc_pid_syscall
c06b7484 t tracefs_syscall_rmdir
c06b7500 t tracefs_syscall_mkdir
c07c4050 t collect_syscall
c07c41f8 T task_current_syscall
c1200448 d str__raw_syscalls__trace_system_name
c12095f4 d mode1_syscalls
c1b8e270 d cgroup_kf_syscall_ops
c1b8fcdc D cgroup1_kf_syscall_ops
sebastian@Ubuntu:~/linux-stable$
```

Booting the BeagleBone Black

We need a microSD card with U-Boot installed. Plug the microSD card into your card reader and from the linux-stable directory, copy the files

arch/arm/boot/zImage and arch/arm/boot/dts/am335xboneblack.dtb to the boot partition.

Unmount the card and plug it into the BeagleBone Black. Start a terminal emulator, such as gtkterm, and be prepared to press the space bar as soon as you see the U-Boot messages appear. Next, power

on the BeagleBone Black and press the space bar. You should get a U-Boot prompt, . Now enter the following commands to load Linux and the device tree binary: (only the ones in bold)

U-Boot# fatload mmc 0:1 0x80200000 zImage
reading zImage
7062472 bytes read in 447 ms (15.1 MiB/s)
U-Boot# fatload mmc 0:1 0x80f00000 am335x-boneblack.dtb
reading am335x-boneblack.dtb
34184 bytes read in 10 ms (3.3 MiB/s)
U-Boot# setenv bootargs console=ttyO0
U-Boot# bootz 0x80200000 - 0x80f00000
Flattened Device Tree blob at 80f00000
Booting using the fdt blob at 0x80f00000
Loading Device Tree to 8fff4000, end 8ffff587 ... OK
Starting kernel ...
[0.000000] Booting Linux on physical CPU 0x0
[...]
Note that we set the kernel command line to console=ttyO0. That tells Linux which device to use for console output, which in this case is the first UART on the board, device ttyO0. Without this, we would not see any messages after Starting the kernel..., and therefore would not know if it was working or not. The sequence will end in a kernel panic, for reasons I will explain later on.

Kernel panic

Once it starts the kernel and as it begins to look for a login shell, this happens:

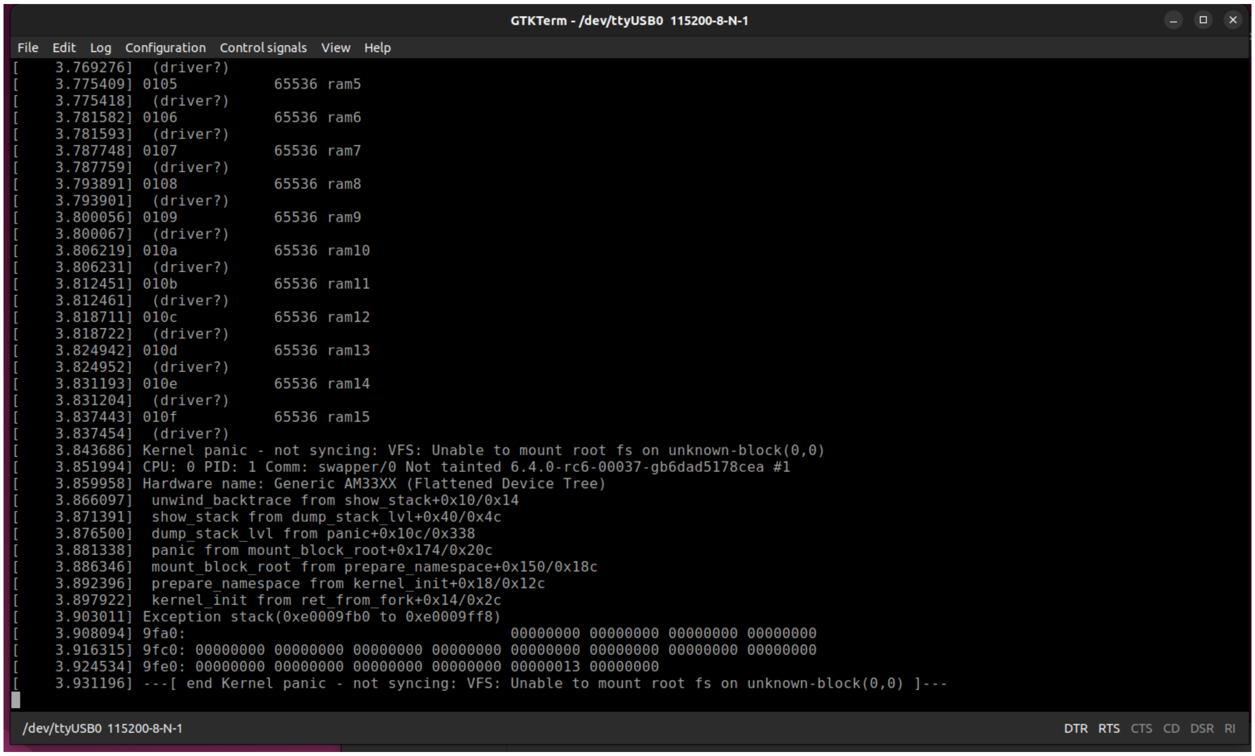
[1.886379] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)

[1.895105] ---[end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0, 0)

This is a good example of a kernel panic. A panic occurs when the kernel encounters an unrecoverable error. By default, it will print out a message to the console and then halt. You can set the panic command-line parameter to allow a few seconds before reboots following a panic. In this case, the unrecoverable error is no root filesystem, illustrating that a kernel is useless without a user space to control it. You can supply a user space by providing a root filesystem, either as a ramdisk or on a mountable mass storage device.

(Provide screenshot of system in this state)

As Below:



Now add to the boot partition in the SD card the script u-boot was looking for to start the Linux system automatically, without requiring to enter the command manually.

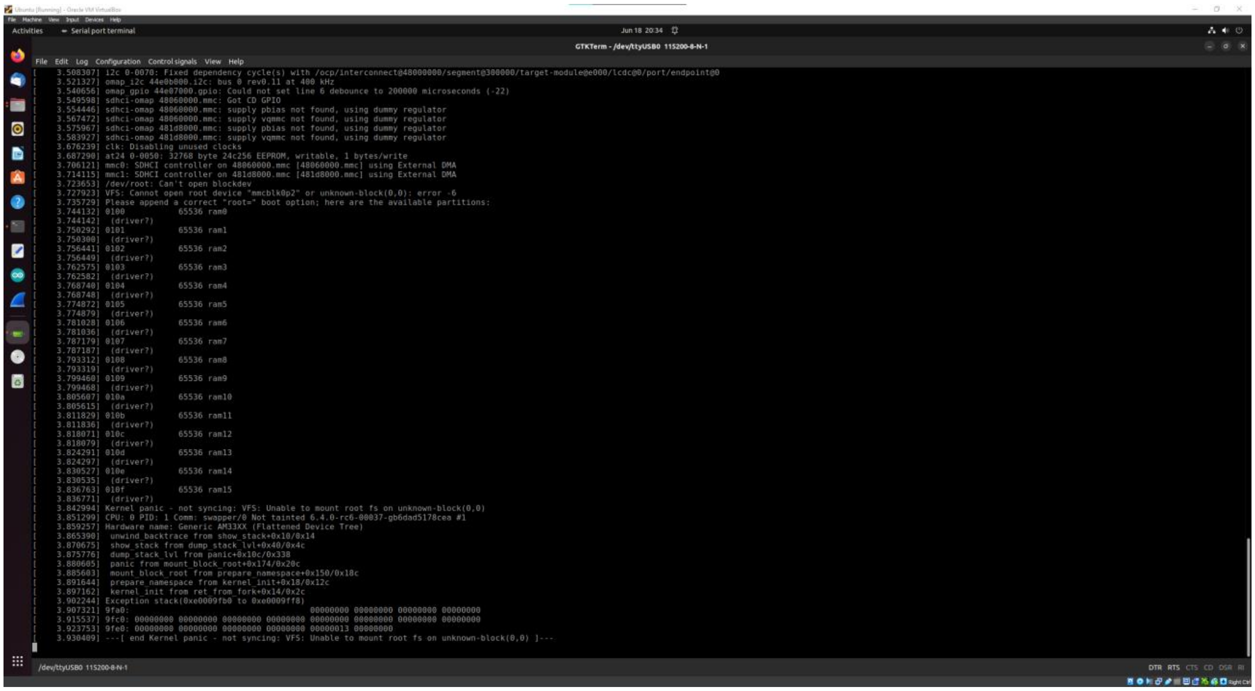
This is the basic uEnv.txt that your boards needs in order to boot your kernel. Create and put this file uEnv.txt in your boot partition.

bootdir=
bootfile=zImage
fdtfile=am335x-boneblack.dtb
loadaddr= 0x80200000
fdtaddr= 0x80f00000
loadzimage=fatload mmc 0:1 ${loadaddr} ${bootfile}
loadfdt=fatload mmc 0:1 ${fdtaddr} ${fdtfile}
uenvcmd=mmc rescan; run loadzimage; run loadfdt; setenv bootargs console=ttyS0,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext4 rootwait;
bootz ${loadaddr} - ${fdtaddr}

Now boot BBB from the SD card and you should be taken to the Kernel Panic message

(provide screenshot)

As Below:



Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Serial port terminal
Jun 18 20:34
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
[3.508307] i2c 0-0070: Fixed dependency cycle(s) with /ocp/interconnect@48000000/segment@300000/target-module@e000/lcdc@0/port/endpoint@0
[3.521327] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
[3.540656] omap_gpio 44e07000.gpio: Could not set line 6 debounce to 200000 microseconds (-22)
[3.549598] sdhci-omap 48060000.mmc: Got CD GPIO
[3.554446] sdhci-omap 48060000.mmc: supply pbias not found, using dummy regulator
[3.567472] sdhci-omap 48060000.mmc: supply vqmmc not found, using dummy regulator
[3.575967] sdhci-omap 481d8000.mmc: supply pbias not found, using dummy regulator
[3.583927] sdhci-omap 481d8000.mmc: supply vqmmc not found, using dummy regulator
[3.676239] clk: Disabling unused clocks
[3.687290] at24 0-0050: 32768 byte 24c256 EEPROM, writable, 1 bytes/write
[3.706121] mmc0: SDHCI controller on 48060000.mmc [48060000.mmc] using External DMA
[3.714115] mmc1: SDHCI controller on 481d8000.mmc [481d8000.mmc] using External DMA
[3.723653] /dev/root: Can't open blockdev
[3.727923] VFS: Cannot open root device "mmcblk0p2" or unknown-block(0,0): error -6
[3.735729] Please append a correct "root=" boot option; here are the available partitions:
[3.744132] 0100   65536 ram0
[3.744142] (driver?)
[3.750292] 0101   65536 ram1
[3.750300] (driver?)
[3.756441] 0102   65536 ram2
[3.756449] (driver?)
[3.762575] 0103   65536 ram3
[3.762582] (driver?)
[3.768740] 0104   65536 ram4
[3.768748] (driver?)
[3.774872] 0105   65536 ram5
[3.774879] (driver?)
[3.781028] 0106   65536 ram6
[3.781036] (driver?)
[3.787179] 0107   65536 ram7
[3.787187] (driver?)
[3.793312] 0108   65536 ram8
[3.793319] (driver?)
[3.799460] 0109   65536 ram9
[3.799468] (driver?)
[3.805607] 010a   65536 ram10
[3.805615] (driver?)
[3.811829] 010b   65536 ram11
[3.811836] (driver?)
[3.818071] 010c   65536 ram12
[3.818079] (driver?)
[3.824291] 010d   65536 ram13
[3.824297] (driver?)
[3.830527] 010e   65536 ram14
[3.830535] (driver?)
[3.836763] 010f   65536 ram15
[3.836771] (driver?)
[3.842994] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[3.851299] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 6.4.0-rc6-00037-gb6dad5178cea #1
[3.859257] Hardware name: Generic AM33XX (Flattened Device Tree)
[3.865390] unwind_backtrace from show_stack+0x10/0x14
[3.870675] show_stack from dump_stack_lvl+0x40/0x4c
[3.875776] dump_stack_lvl from panic+0x10c/0x338
[3.880605] panic from mount_block_root+0x174/0x20c
[3.885603] mount_block_root from prepare_namespace+0x150/0x18c
[3.891644] prepare_namespace from kernel_init+0x18/0x12c
[3.897162] kernel_init from ret_from_fork+0x14/0x2c
[3.902244] Exception stack(0xe0009fb0 to 0xe0009ff8)
[3.907321] 9fa0:   00000000 00000000 00000000 00000000
[3.915537] 9fc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[3.923753] 9fe0: 00000000 00000000 00000000 00000000 00000013 00000000
[3.930409] ---[end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)]---
/dev/ttyUSB0 115200-8-N-1
DTR RTS CTS CD DSR RI