EE5414 Laboratory - Part II Sample Report

Android Source Code compilation and Android Booting Procedure

Booting Android on any TI platform requires following software components

- Bootstrapping (x-load.bin.ift for NAND or MLO for MMC)

Describe and record how you can make the Kernel image, bootloader, MLO and File System.

How you compile the xloader?

```
#!/bin/bash
make CROSS_COMPILE=arm-eabi- distclean
make CROSS_COMPILE=arm-eabi- omap3beagle_config
make CROSS_COMPILE=arm-eabi-
../tools/signGP ./x-load.bin
mv x-load.bin.ift ../images/MLO
```

Results

Check your results and Is there any error message?

For example : Grep your error message by using ./xloader.sh 2> err.txt

The list below is the end of the compilation, explain what you got.

```
make[1]: Leaving directory `/home/ee5414/android/xloader/drivers'

UNDEF_SYM=`arm-eabi-objdump -x board/omap3beagle/libomap3beagle.a
cpu/omap3/libomap3.a lib/libarm.a fs/fat/libfat.a disk/libdisk.a drivers/libdrivers.a |sed -n -e
's/.*\(__u_boot_cmd_.*\)/-u\1/p'|sort|uniq`;\

                        arm-eabi-ld -Bstatic -T /home/ee5414/android/xloader/board/omap3beagle/x-
load.lds -Ttext 0x40200800 $UNDEF_SYM cpu/omap3/start.o \

                                    --start-group board/omap3beagle/libomap3beagle.a
cpu/omap3/libomap3.a lib/libarm.a fs/fat/libfat.a disk/libdisk.a drivers/libdrivers.a --end-group
-L /home/ee5414/android/froyo/prebuilt/linux-x86/toolchain/arm-eabi-
4.4.0/bin/../lib/gcc/arm-eabi/4.4.0 -lgcc \

                                    -Map x-load.map -o x-load

arm-eabi-objcopy --gap-fill=0xff -O binary x-load x-load.bin
```

What is the version of toolchain or gcc you are using?

Is the compilation successfully?

Is yes, where do you save your MLO?

- Bootloader (u-boot.bin)

```
#!/bin/bash
make CROSS_COMPILE=arm-eabi- distclean
make CROSS_COMPILE=arm-eabi- omap3_beagle_config
make CROSS_COMPILE=arm-eabi-
cp u-boot.bin ../images/
```

Results

Check your results and Is there any error message?

For example : Grep your error message by using ./uboot.sh 2> uberr.txt and ./uboot.sh > ub.txt

The list below is the end of the compilation, explain what you got.

```
make[1]: Entering directory
`/home/ee5414/android/uboot/arch/arm/cpu/arm_cortexa8'
make[1]: Nothing to be done for `u-boot.lds'.
make[1]: Leaving directory
`/home/ee5414/android/uboot/arch/arm/cpu/arm_cortexa8'
arm-eabi-gcc -E -g  -Os   -fno-common -ffixed-r8 -msoft-float   -
D__KERNEL__ -DTEXT_BASE=0x80e80000 -I/home/ee5414/android/uboot/include
-fno-builtin -ffreestanding -nostdinc -isystem
/home/ee5414/android/froyo/prebuilt/linux-x86/toolchain/arm-eabi-
4.4.0/bin/../lib/gcc/arm-eabi/4.4.0/include -pipe  -DCONFIG_ARM -
D__ARM__ -marm  -mabi=aapcs-linux -mno-thumb-interwork -march=armv5 -
include /home/ee5414/android/uboot/include/u-boot/u-boot.lds.h  -ansi -
D__ASSEMBLY__ -P -
</home/ee5414/android/uboot/arch/arm/cpu/arm_cortexa8/u-boot.lds >u-
boot.lds
UNDEF_SYM=`arm-eabi-objdump -x board/ti/beagle/libbeagle.a
lib/libgeneric.a lib/lzma/liblzma.a lib/lzo/liblzo.a
arch/arm/cpu/arm_cortexa8/libarm_cortexa8.a
arch/arm/cpu/arm_cortexa8/omap3/libomap3.a arch/arm/lib/libarm.a
fs/cramfs/libcramfs.a fs/fat/libfat.a fs/fdos/libfdos.a
fs/jffs2/libjffs2.a fs/reiserfs/libreiserfs.a fs/ext2/libext2fs.a
fs/yaffs2/libyaffs2.a fs/ubifs/libubifs.a net/libnet.a disk/libdisk.a
drivers/bios_emulator/libatibiosemu.a drivers/block/libblock.a
drivers/dma/libdma.a drivers/fpga/libfpga.a drivers/gpio/libgpio.a
drivers/hwmon/libhwmon.a drivers/i2c/libi2c.a drivers/input/libinput.a
drivers/misc/libmisc.a drivers/mmc/libmmc.a drivers/mtd/libmtd.a
drivers/mtd/nand/libnand.a drivers/mtd/onenand/libonenand.a
drivers/mtd/ubi/libubi.a drivers/mtd/spi/libspi_flash.a
drivers/net/libnet.a drivers/net/phy/libphy.a drivers/pci/libpci.a
drivers/pcmcia/libpcmcia.a drivers/power/libpower.a
drivers/spi/libspi.a drivers/rtc/librtc.a drivers/serial/libserial.a
drivers/twserial/libtws.a drivers/usb/gadget/libusb_gadget.a
drivers/usb/host/libusb_host.a drivers/usb/musb/libusb_musb.a
drivers/usb/phy/libusb_phy.a drivers/video/libvideo.a
```

Explain some of the resuts you have observed.

why is static library and not dynamic library and so on?

```
drivers/watchdog/libwatchdog.a common/libcommon.a lib/libfdt/libfdt.a
api/libapi.a post/libpost.a | sed  -n -e 's/.*\(__u_boot_cmd_.*\)/-
u\1/p'|sort|uniq`; cd /home/ee5414/android/uboot && arm-eabi-ld -
Bstatic -T u-boot.lds  -Ttext 0x80e80000 $UNDEF_SYM
arch/arm/cpu/arm_cortexa8/start.o --start-group lib/libgeneric.a
lib/lzma/liblzma.a lib/lzo/liblzo.a
arch/arm/cpu/arm_cortexa8/libarm_cortexa8.a
arch/arm/cpu/arm_cortexa8/omap3/libomap3.a arch/arm/lib/libarm.a
fs/cramfs/libcramfs.a fs/fat/libfat.a fs/fdos/libfdos.a
fs/jffs2/libjffs2.a fs/reiserfs/libreiserfs.a fs/ext2/libext2fs.a
fs/yaffs2/libyaffs2.a fs/ubifs/libubifs.a net/libnet.a disk/libdisk.a
drivers/bios_emulator/libatibiosemu.a drivers/block/libblock.a
drivers/dma/libdma.a drivers/fpga/libfpga.a drivers/gpio/libgpio.a
drivers/hwmon/libhwmon.a drivers/i2c/libi2c.a drivers/input/libinput.a
drivers/misc/libmisc.a drivers/mmc/libmmc.a drivers/mtd/libmtd.a
drivers/mtd/nand/libnand.a drivers/mtd/onenand/libonenand.a
drivers/mtd/ubi/libubi.a drivers/mtd/spi/libspi_flash.a
drivers/net/libnet.a drivers/net/phy/libphy.a drivers/pci/libpci.a
drivers/pcmcia/libpcmcia.a drivers/power/libpower.a
drivers/spi/libspi.a drivers/rtc/librtc.a drivers/serial/libserial.a
drivers/twserial/libtws.a drivers/usb/gadget/libusb_gadget.a
drivers/usb/host/libusb_host.a drivers/usb/musb/libusb_musb.a
drivers/usb/phy/libusb_phy.a drivers/video/libvideo.a
drivers/watchdog/libwatchdog.a common/libcommon.a lib/libfdt/libfdt.a
api/libapi.a post/libpost.a board/ti/beagle/libbeagle.a --end-group
/home/ee5414/android/uboot/arch/arm/lib/eabi_compat.o -L
/home/ee5414/android/froyo/prebuilt/linux-x86/toolchain/arm-eabi-
4.4.0/bin/../lib/gcc/arm-eabi/4.4.0 -lgcc -Map u-boot.map -o u-boot
arm-eabi-objcopy -O srec u-boot u-boot.srec
arm-eabi-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
```

Explain this? Also why use objcopy?

- Kernel Image (uImage)
- Filesystem (rootfs)

Should be similar to the xloader and uboot.

Besides, explain how you make boot.src by using mkbootsrc and why you need to do so.

Boot arguments

```
   #> mmc init
   #> fatload mmc 0 0x82000000 boot.scr
   #> source 0x82000000
```

```
  Beagleboard-xm:
  setenv bootargs 'console=ttyS2,115200n8 androidboot.console=ttyS2
mem=256M root=/dev/mmcblk0p2 rw rootfstype=ext3 rootdelay=1 init=/init
ip=off mpurate=1000 omap_vout.vid1_static_vrfb_alloc=y'
```

Explain this and why you need this?

Demonstrate how you can successfully install adb both on the target and host sides.

Install and uninstall applications

The USB network gadget g_ether is named usb0 (instead of eth0 or other network interface names). The normal set of Ethernet configuration tools should work, such as ifconfig, netstat, and route.

For example, the following commands will assign the network address 192.168.194.2 to the target. Run this on the target:

```
$> ifconfig usb0 192.168.194.2 netmask 255.255.255.224 up
```

On Host machine, run the following commands to establish the connection to the target:

```
$> sudo ifconfig eth0 192.168.194.1 netmask 255.255.255.224 up
$> sudo route add 192.168.194.2 dev eth0
```

The target and the host machine should be connected, run ping command to test the same:

```
$ ping -c 3 192.168.194.2
PING 192.168.194.2 (192.168.194.2) 56(84) bytes of data.
64 bytes from 192.168.194.2: icmp_seq=1 ttl=64 time=6.08 ms
64 bytes from 192.168.194.2: icmp_seq=2 ttl=64 time=0.511 ms
64 bytes from 192.168.194.2: icmp_seq=3 ttl=64 time=0.485 ms
--- 192.168.194.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.485/2.361/6.089/2.636 ms
```

- Configure the ADB Daemon to use an ethernet connection using setprop as shown below.

```
target #> setprop service.adb.tcp.port 5555
```

- If network is configured successfully (above steps) then Restart service adbd on the target,

```
target #> stop adbd
target #> start adbd
```

- Establish ADB connection

On the host machine execute following commands to establish adb connection

```
$ export ADBHOST= 192.168.194.2
$ adb kill-server
$ adb start-server
```

Verify the connection by executing

```
$ adb devices
```

If connected, device name should be listed as a "emulator"

```
$ adb devices
List of devices attached
emulator-5554     device
$ adb shell
```

adb over Ethernet

- Make sure Ethernet port on board and host machine are connected to the network

Running Applications

The root File System provided in this DevKit releases contains only standard Android components and applications. User might be interested to download & run android applications (.apk) available in the market. The below procedure gives the steps to be followed to download any .apk file to the board and run it on the platform.

Installing (.apk files) application on Target Platform

- From the host: You can use adb tool for package installation.

```
    $> adb install <package>.apk.
```

NOTE: Use -s option with the adb tool, to install the package on external storage.

On successful installation adb tool will report SUCCESS on host terminal, and the application would be listed on the android main menu.

Un-installing applications (.apk) using adb

- To un-install non-default components (that were installed later)
  - Method 1: On the host machine execute the following

```
    $> adb uninstall <package>.apk
```

  - Method 2: On target:

```
   Main menu -> Menu -> Settings -> Applications -> Manage
applications -> Find the package
   Tap on it -> Uninstall -> OK -> OK
```

  - On successful removal, the application would have been removed from the android main menu. All the short-cuts to the application also removed.
- To un-install default components, use the following commands from abd on host machine

```
    $ adb shell
    #rm /system/app/app.apk
```

On successful removal, the application would have been removed from the android main menu.

If you are successfully done the above works, then you can start to do the following exercise.

Assignment for this laboratory:

1. Based on the experience of this exercise, try to install Angstrom Linux to Beagle board.

Similar records for the Angstrom also.

Optional:

2. Write an led driver for the beagleboard and make the led lits on and off continuously.

Very high marks will be given.

For the kernel compilation, try to explain the names marked.

```
CLEAN   include/asm-arm/mach-types.h
  CLEAN   include/config

include/linux/utsrelease.h include/linux/bounds.h include/asm-arm/asm-
include/asm/asm-offsets.h
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/kxgettext.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/lex.zconf.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
scripts/kconfig/conf -s arch/arm/Kconfig
  CHK     include/linux/version.h
  UPD     include/linux/version.h
  SYMLINK include/asm -> include/asm-arm
  Generating include/asm-arm/mach-types.h
  CHK     include/linux/utsrelease.h
  UPD     include/linux/utsrelease.h
  HOSTCC  scripts/genksyms/genksyms.o
  HOSTCC  scripts/kallsyms
```

```
  UPD     include/linux/compile.h
  CC      init/version.o
  CC      arch/arm/kernel/compat.o
  CC      arch/arm/kernel/elf.o
  AS      arch/arm/kernel/entry-armv.o
  GEN     usr/initramfs_data.cpio
  AS      arch/arm/kernel/entry-common.o
  CC      arch/arm/kernel/irq.o
  AS      usr/initramfs_data.o
```

What they are doing in the gcc compiler?