Porting the Linux Kernel to Beagle Bone Black

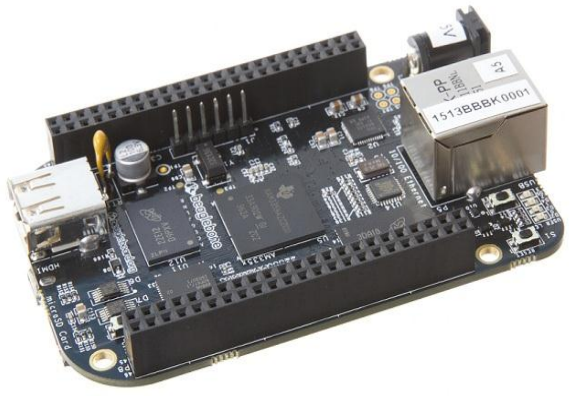
Jaydevsinh Jadeja[1] Chintan Kapadiya[2]
[1,2] M. E. Student
[1, 2] E & C Department
[1, 2]B. H. Gardi College of Engg & Tech, Rajkot

Abstract---ARM development boards are the ideal platform for accelerating the development and reducing the risk of new SoC designs. The combination of ASIC and FPGA technology in ARM boards delivers an optimal solution in terms of speed, accuracy, flexibility and cost. The Embedded modules, based on ARM, can become very complex machines since these are meant to support varied tasks such as memory management, process management and peripheral interfaces.For seamless integration of these functional modules an OS has to be ported on these ARM based CPUs .Traditionally this OS porting is often the specialized work of third party vendors having expertise in this domain. For every new CPU architecture, the OS has to be customized, compiled and burnt into the core .With the coming of age of Linux as an embedded OS all this has changed quite significantly. Being in Open Source domain, Linux kernel can be freely downloaded and compiled for any system architecture and this includes ARM based systems also. This enables the developers to port the OS themselves. This paper describes the details of porting of Linux kernel to Beagle bone black.Building linux kernel; u-boot and x-loader are described in detail. SD card configuration is also described. [1]

Keywords: Porting the Linux Kernel, Beagle Bone Black

I. INTRODUCTION

Pre-built OS images are available like Angstrom, Ubuntu, Debian of various version, however to build your own OS for some reason, than you need to get the source code, patches for ARM7 and build using GCC cross compiler configuration to have it available to build whatever modules you would like. And all the process is fully describe in step vise.



A. *Step 0: Get build environment ready*
Step 0.0: install Debian/Ubuntu on a machine or VM Environment and update/patch it up I prefer 64-bit Linux
Step 0.1:"sudo apt-get update"
Step 0.2: "sudo apt-get upgrade"
Step 0.3:"sudo apt-get installs <stuff>" like Vim Editor, minicom etc…

B. *Step 1: Get communication to Beagle Bone Black into a "known working state"*
Step 1.0: beagle bone black has prebuilt angstrom Distribution
Step 1.1: otherwise download latest image for angstrom Distribution and write using disk-32 image Writer.https://launchpad.net/win32-image-writer/ + download
Step 1.2: now open terminal and type
Cd /dev
ls
Step 1.3: now connect beagle bone black by using usb cable.
Step 1.4: again type ls in terminal and carefully check which Parameter is newly added.
In my board it is ttyACM0.
Step 1.5: sudo minicom –s
Go to serial port setup and change the first line as /dev/ttyACM0 save as default and exit.
Step 1.6: press enter if beagle bone log in is not displayed.
Step 1.7: log in as root. Your beagle bone black is working.

C. *Step 2: Setup cross-compilation build environment Adapted from:*
http://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-BasicRequirements
Step 2.0: install 32-bit versions of key components sudo apt-Get update && sudo apt-get upgrade && sudo apt-get install libc6:i386 listed++6:i386 libncurses 5:i386 zlib1g:i386
Step 2.1: download / extract cross-compiler
wget-chttps://launchpad.net/linaro-toolchain-binaries/trunk/2013.07/+download/gcc-linaro-Arm-linuxgnueabihf-4.8-2013.07-1_linux.tar.xz
Tar xJf gcc-linaro-arm-linux-gnueabihf-4.8-2013.07-1_linux.tar.x
Export CC=`pwd`/gcc-linaro-arm-linux-gnueabihf-4.8-2013.07-1_linux/bin/arm-linux-gnueabihf
Step 2.2: Test set up
${CC} gcc –version
arm-linux-gnueabihf-gcc (crosstool-NG linaro-1.13.1-4.8-2013.07-1 - Linaro GCC 2013.07) 4.8.2 20130624(prerelease)
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
Warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

D. *Step 3: Build U-Boot*
Step 3.0: Download U-Boot

git clone git://git.denx.de/u-boot.git
Cd u-boot/
Git checkout v2013.07 -b tmp

Step 3.1: Configure and Build U-Boot
Wget https://raw.github.com/eewiki/u-boot-patches/master/v2013.07/0001-am335x_evm-uEnv.txt-bootzn-fixes.patch
Patch -p1 < 0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch
make ARCH=arm CROSS_COMPILE=${CC} distclean
make ARCH=arm CROSS_COMPILE=${CC} am335x_evm_config
Make ARCH=arm CROSS_COMPILE=${CC}

Step 3.2: during distclean if it generates any file than clean its
Again.

E. *Step 4: Upgrade device tree compiler*

Step 4.0: Download and upgrade dtc
Cd...
Wget –c
https://raw.github.com/RobertCNelson/tools/master/pkgs/dtc.sh
Chmod +x dtc.sh
./dtc.sh

F. *Step 5: Build Kernel and modules*

Step 5.0: Download kernel
Git clone git://github.com/RobertCNelson/linux-dev.git
Cd linux-dev/
Git checkout origin/am33x-v3.8 -b tmp

Step 5.1: Build kernel
./build_kernel.sh
Cd...

G. *Step 6: Get rootfs setup on flash card*

Step 6.0: Download, verify and extract rootfs
Debian:
Wget -c https://rcn-ee.net/deb/minfs/wheezy/debian-7 .1-minimal-armhf-2013-08-25.tar.xz
Md5sum debian-7.1-minimal-armhf-2013-08-25.tar.xz
Tar xJf debian-7.1-minimal-armhf-2013-08-25.tar.xz
Ubuntu:
Wget -c https://rcn-ee.net/deb/minfs/raring/ubuntu-13.04-minimal-armhf-2013-08-25.tar.xz
Md5sum ubuntu-13.04-minimal-armhf-2013-08-25.tar.xz
Tar xJf ubuntu-13.04-minimal-armhf-2013-08-25.tar.xz

Step 6.1: figure out what device your MMC appears as in
Build machine
Ls /dev
Insert an MMC or a USB<->MMC adapter
Ls /dev

Look for changes which have appeared. In my case, with a USB flash adapter, MMC cards appears as /dev/sdb

Step 6.2: setup a temp export for ease of use, wipe disk and
Setup partition table
Export DISK=/dev/sdb
Sudo dd if=/dev/zero of=${DISK} BS=1M Count=16
Sudo fdisk /dev/sdb
P (print)
D (delete any existing partitions)
W (writes and exit)
Sudo partprobe
Sudo fdisk /dev/sdb
P (print partition table)
N (new partition)
P (select primary)
1 (partition number)
<Enter> (accept start at 2048)
+100M
T (change partition type)
1 (partition 1)
B (change to FAT32)
P (print partition table)
A (toggle boot flag)
1 (partition 1)
P (print partition table)
N (new partition)
2 (partition #2)
<Enter> (accept default start at XXXXX)
<Enter> (accept default, last sector on The flash card)
P (print)
W (writes and exit)
Sudo partprobe if it fails than you can simply plug in your usb again.

Step 6.3: format partitions
Sudo mkfs.vfat -F 32 ${DISK} 1 -n boot
Sudo mkfs.ext4 ${DISK} 2 -L rootfs

Step 6.4: mount partitions
Sudo mount ${DISK} 1 /media/boot/
Sudo mount ${DISK} 2 /media/rootfs/

Step 6.5: install U-Boot on flash card
Sudo cp -v ./u-boot/MLO /media/boot/
Sudo cp -v ./u-boot/u-boot.img /media/boot/

Step 6.6: save uEnv.txt into the build directory
VI uEnv.txt (for edit new file)
I (for insert mode) and copy paste following by pressing ctrl+shift+v

Step 6.7: copy the uEnv.txt file into /media/boot
Sudo cp -v ./uEnv.txt /media/boot/

Step 6.8: install rootfs
Look in linux-dev/deploy
Ls... Check version ie: 3.8.13-bone35.2
Export environment variable for kernel version
export kernel_version=3.8.13-bone35.2
Copy rootfs
Sudo tar xfvp ./ubuntu-13.04-minimal-armhf-2013-08-25/armhf-rootfs-ubuntu-raring.tar -C /media/rootfs

H. *Step 7: Get kernel setup on flash card*

Step 7.0: copy kernel files, device tree files and modules

```
Sudo cp -v ./linux-dev/deploy/${kernel_version}.zImage /media/boot/zImage
Sudo mkdir -p /media/boot/dtbs/
Sudo tar xfov ./linux-dev/deploy/${kernel_version}-dtbs.tar.gz -C /media/boot/dtbs/
Sudo tar xfv ./linux-dev/deploy/${kernel_version}-firmware.tar.gz -C /media/rootfs/lib/firmware
Sudo tar xfv ./linux-dev/deploy/${kernel_version}-modules.tar.gz -C /media/rootfs/
```

```
#u-boot eMMC specific overrides; Angstrom Distribution (BeagleBone Black) 2013-06-20
kernel_file=zImage

initrd_file=uInitrd

loadzimage=load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${kernel_file}

loadinitrd=load mmc ${mmcdev}:${mmcpart} 0x81000000 ${initrd_file}; setenv initrd_size ${filesize}

loadfdt=load mmc ${mmcdev}:${mmcpart} ${fdtaddr} /dtbs/${fdtfile}

#

console=ttyO0,115200n8

mmcroot=/dev/mmcblk0p2 ro

Mmcrootfstype=ext4 rootwait fixrtc

##To disable HDMI/eMMC...

#optargs=capemgr.disable_partno=BB-BONELT-HDMI, BB-BONELT-HDMIN, BB-BONE-EMMC-2G

##3.1MP Camera Cape

#optargs=capemgr.disable_partno=BB-BONE-EMMC-2G

mmcargs=setenv bootargs console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} ${optargs}

#zImage:

uenvcmd=run loadzimage; run loadfdt; run mmcargs; bootz ${loadaddr} - ${fdtaddr}

#zImage + uInitrd: where uInitrd has to be generated on the running system.

#boot_fdt=run loadzimage; run loadinitrd; run loadfdt

#uenvcmd=run boot_fdt; run mmcargs; bootz ${loadaddr} 0x81000000:${initrd_size} ${fdtaddr}
```

I. *Step 8: Edit configuration files on flash card*

Step 8.0: edit /etc/fstab in the MMC

```
Sudo nano /media/rootfs/etc/fstab
```

```
/dev/mmcblk0p2 / auto errors=remount-ro 0 1
/dev/mmcblk0p1 /boot/uboot auto defaults 0 2
```

Step 8.1: edit /etc/network/interfaces

```
Sudo nano /media/rootfs/etc/network/interfaces
```

```
Auto lo
Iface lo inet loopback
Auto eth0
Iface eth0 inet dhcp
```

Step 8.2: edit /etc/udev/rules.d/70-persistent-net.rules

```
Sudo nano /media/rootfs/ /etc/udev/rules.d/70-persistent-net.rules
```

```
# BeagleBone: net device ()
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR {dev_id} =="0x0", ATTR {type} =="1", KERNEL=="eth*",
NAME="eth0"
```

J. *Step 9: Configure a Serial console for debugging access*

Step 9.0: debian: edit /etc/inittab in the MMC

```
Sudo nano /media/rootfs/etc/inittab
Add:
T0:23: respawn: /sbin/getty -L ttyO0 115200 vt102
Ubuntu: create serial.conf
Sudo nano /media/rootfs/etc/init/serial.conf
Add:
Start on stopped RC RUNLEVEL= [2345]
Stop on runlevel [! 2345]
Respawn
Exec /sbin/Getty 115200 ttyO0
```

K. *Step 10: Sync MMC and remove sudo sync*

Then plug in SD card into the beagle bone black and connect using usb cable. This time you cannot get serial console using ttyACM0, you required FTDI cable, or RS-232 to TTL 3.3v converter (refer my another paper how to get a serial console in beagle bone black). You can login as user: ubuntu and password: temppwd.

II. Conclusion

The paper discussed a generic process for the build Linux for arm core family Beagle bone black, it has limited functionality, it just enables the kernel to boot and send debug message through the configured serial port.

Acknowledgement

We thank to our HOD, Prof. C.D.Parmar, for providing necessary facilities towards carrying out this work. We acknowledge the diligent efforts of our Internal Guide Prof.S.N.Pandya and External Guide Mr.Bhargav Shah in assisting us towards implementation of this idea.

References

[1] Pratyusha.Gandham, Ramesh N.V.K / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 Vol. 2, Issue 2,Mar-Apr 2012, pp.1614-1618
[2] http://the8thlayerof.net/2013/10/16/beagle-bone-black-notes-how-to-build-a-kernel-for-a-beagle-bone-black-from-scratch-part-1/
[3] http://www.kernel.org.
[4] http://elinux.org/BeagleBone
[5] http://beagleboard.org/Products/BeagleBone%20Black
[6] http://www.armhf.com/

[7] HU Jie, ZHANG Gen-bao, "Research transplanting method of embedded linux kernel based on ARM platform", 2010 International Conference of Information Science and Management Engineering, E-ISBN : 978-1-4244-7670-1, 8 Aug. 2010

