Compiling and booting custom Debian Kernel Image for Cubieboard

Abhinav Gupta

April 25, 2013

1 Introduction

CubieBoards are ARM Cortex A10 based development boards like Raspberry Pi and BeagleBoards. The major difference between the Cubieboard and other development boards is the computation bandwidth provided by the chipset involved. Another advantage of this board is the presence of multiple interface options available on the board giving it an edge over the current revisions of Beagle and Pi.

One major reason for this computation advantage of CubieBoard is that the pinouts are heavily multiplexed amongst multiple thus concentrating the computational power only on the pins that are absolutely required by the application. The hardware pin modes required by the application are specified in the kernel image that is uploaded on the board. Thus instead of having a generic kernel, you need to make your own image depending on your Hardware kernel and subsequently make a bootable SD card with appropriate partitions.

2 Linux-Sunxi

Sunxi represents the family of ARM SoC (System on Chip) designed for embedded systems, and made by Allwinner Tech. in Zhuhai (Guangdong, China). The most popular sunxi SoC model is the Allwinner A10 and the Allwinner A13. Their predecessor was an ARM9 named Boxchip F20 (sun3i) and their successors are A20 and Allwinner A31.

There is a git branch maintained by the open source community maintaining the linux branches that can be uploaded on the Allwinner boards. The one in question for us being the Cubieboard. The required source codes are maintained on github. We shall here build the linaro toolchain and the kernel for version 3.0. You definitely need another Linux machine to go ahead in this tutorial, preferably something with a package manager. Assuming you are on a Linux Machine, go ahead and install the following packages on your machine

- **arm-linux-gnueabihf-** : Tools for Cross Compiling source codes for the ARM architecture
- **sfdisk** : An easier tool for disk partitioning. Required for SD card partitioning
- **mkimage** : Tool for creating image (scr) files from cmd files
- **axel** : Easier tool for downloading packages
- **unp** : Easier tool for unpacking tar files

3 Building the source files and setting up the SD card

Once you have the tools you are ready to clone all the kernel and bootloader source codes. Pick up an appropriate directory which is preferably empty an ensure that you have more than 1 GB for space empty. After than run the following commands in the terminal:

```
git clone git://github.com/linux-sunxi/u-boot-sunxi.git
git clone git://github.com/linux-sunxi/linux-sunxi.git
git clone git://github.com/linux-sunxi/sunxi-tools.git
git clone git://github.com/linux-sunxi/sunxi-boards.git
```

3.1 Bootlaoder

Now that we have all the necessary source files, we start building the source files one by one, first up will be the bootloaders. To do that, we first go into the u-boot-sunxi directory and use the arm-linux-gnueabhihf- tool we got before to cross compile for the ARM architecture. As mentioned before, the linux-sunxi source files is for the collective AllWinner boards. So we need to compile for the specific board we are targeting to. In this case it is the Cubieboard_512. Other targets can be found in board/allwinner. So we run the following commands. It is advantageous to be in the superuser mode from this point onwards. To do that run *sudo su* in command line and enter your login password:

```
cd u-boot-sunxi
make cubieboard_512 CROSS_COMPILE=arm-linux-gnueabihf-
```

This make command will generate a u-boot.bin in the main u-boot-sunxi directory and a sunxi-spl.bin in the spl folder. The sunxi-spl.bin is an Initial Program Loader (IPL) which is points to the u-boot-sunxi.bin which is the Secondary Program Loader (SPL). The IPL is the first file to come after the Master Boot Record and performs the Power On Self Test (POST) checking if all the

hardware interfaced are working properly or no at which point it passes on to the SPL which initializes all the boot parameters and passes on to the kernel image.

3.2 The .fex file

A FEX file defines various aspects of how the SoC works. It configures the GPIO pins and sets up DRAM, Display, etc parameters.

Each line consists of a key = value pair combination under a [sectionheader]. All three, [sectionheader], key and value are case-sensitive. For comments a semi-colon (;) is used and everything following a semi-colon is ignored. The chip does not parse a textual version of a fex file, it gets cleaned and compiled by a fex-compiler. A reverse engineered open source version exists in the [sunxi-tools]. Also a de-compiler which takes a binary script.bin and creates a textual script.fex. Usually, script.bin can be found on the nand-a boot partition on A10 devices.

Now to find the appropriate fex file, go in the sunxi-boards/sys_config/a10. In this folder you will see the fex files of different target boards made by AllWinner. Pick out the cubieboard_512.fex file and make the changes that you indend. In this case we are trying to access certain GPIO pins. Thus we append the following to the fex file:

```
[gpio_para]
gpio_used = 1
gpio_num = 4
gpio_pin_1 = port:PG00<1><default><default><default>
gpio_pin_2 = port:PB19<1><default><default><default>
gpio_pin_3 = port:PG02<1><default><default><default>
gpio_pin_4 = port:PG04<1><default><default><default>
```

This configures 4 GPIO pins as input. You can further learn about fex files from this link `http://linux-sunxi.org/Fex_Guide`
Now we need to compile this fex file. To do this the source files contains a tool that parses this text file to a script.bin file that the hardware can understand. To do that enter the sunxi-tools and run the command 'make fex2bin'.

Now move back to the folder which has the edited fex file and run the following command:

```
../../../sunxi-tools/fex2bin cubieboard_512.fex script.bin
```

Now you have the proper bin file in place. Now we need to compile the kernel image and we can begin with the SD card partition.

3.3 Building the Kernel

Descend into your linux tree, and check out the correct branch:

```
cd linux-sunxi
git checkout origin/sunxi-3.0
```

Then run:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sun4i_defconfig
```

And now:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j5 uImage modules
```

Once this stops building you can run:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=output modules_install
```

After all the files are made, it will spit out a uImage in the /linux-sunxi/arch/arm/boot/ folder. This is the final kernel image which will go on the SD card. Now we move on to the setting up and partitioning of the SD card.

3.4 Setting up the SD card

Now we set up the SD card. The first MB of the SD card contains the partition table along with a SPL and IPL binaries for booting. The next partition contains the Kernel Images and the boot.scr files. The biggest partition contains the rootfs which takes around more than 2GB space. Thus it is best to use a 4GB SD card.

Now assuming that you have installed the sfdisk tools and your card is connected to the PC, we begin with checking the partition table and the physical characteristics of the SD card, such as bytes per sector and all that. Also it is assumed that the card is named sdd in the /dev/ folder. To know the name of your sdcard, run the command 'sudo ls -la /dev/sd'. This will list all the disk type devices connected to your machine and you can read the name from there. After running 'fdisk -l' this is the typical 4GB SD card partition that you should see:

```
Disk /dev/sdd: 3965 MB, 3965190144 bytes
122 heads, 62 sectors/track, 1023 cylinders, total 7744512 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xb4e92fdc

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1            2048       34815       16384   83  Linux
/dev/sdd2           34816     7744511     3854848   83  Linux
```

Notice how sdd1 starts at 2048 sector. As the byte per sectors is 512, 2048 sector means that the partition sdd1 starts at 1MB. This is extremely important here as the first MB will have the MBR, Partition Table, SPL and IPL binaries. So we need to make the partitions keeping this in mind, else your board won't boot. So now as the first step we will clean up the SD card. To do that, we need to delete everything in the first write uBoot -¿ 32k on (uBoot is an SPL - secondary program loader)MB as it contains the MBR, FAT table and the partition table. Thus in a way writing zeroes in the first MB will in a way format the card. To do this run the following command:

```
dd if=/dev/zero of=/dev/sdd bs=1M count=1
```

This writes zeroes for the byte size (bs) of 1MB, starting from the first sector itself. Thus rewriting over the entire column and subsequently formatting the entire card. If you run 'fdisk -l' now, it will show no partition table exists. Now run the following bunch of codes, if you are good with writing shell scripts, you can make one and run yourself:

```
sfdisk -R /dev/sdd
```

Now we use sfdisk to create a sector wise partition. Assuming that the hardware characteristics of SD card are the same as above (i.e 512 bytes per sector) and the fact that the first partition *has* to start at 1MB, our partition one starts from sector 2048 and ends at 131072(64MB) and the second partition starts at 131073 and runs till the end. Thus we use sfdisk commanf with the -uS option, which means, partition by sectors and partition accordingly.

```
sfdisk -uS /dev/sdd
2048,131072,c #Here it tells you to define sdd1
131073,,L #Here it tells you to define sdd2. Being the last partition,
 just press enter twice to exit
```

Now we format these partition appropriately by running the following commands:

```
mkfs.vfat /dev/sdd1
mkfs.ext4 /dev/sdd2
sync
```

4 Burning the files and rootfs on the SD card

We should now have 2 partitions. One 64M in vfat format, and another in ext4. We also need to make this bootable, so lets blat in the IPL and SPL to the SD at the appropriate places. Files are the ones in the u-boot-sunxi folder we cloned from git.

Write spl ->8k and then write uBoot ->32k on the SD card.

```
dd if=sunxi-spl.bin of=/dev/Sdd bs=1024 seek=8
dd if=u-boot.bin of=/dev/sdd bs=1024 seek=32
```

Now create a folder in your own root and mount both the partitions there. Do the following assuming that you are the superuser (sudo su):

```
mkdir /mnt
mkdir /mnt/boot
mkdir /mnt/rootfs

mount -t auto /dev/sdd1 /mnt/boot
mount -t auto /dev/sdd /mnt/rootfs

cd /mnt/boot
```

Now, create a file in the /mnt/boot directory named boot.cmd and write the following in it:

```
setenv bootargs console=ttyS0,115200 console=tty0 noinitrd init=/init root=/dev/sdd2
rootfstype=ext4 rootwait panic=10 ${extra}
fatload mmc 0 0x43000000 script.bin
fatload mmc 0 0x48000000 kernel.img
bootm 0x48000000
```

The first line calls the kernel, and passes parameters onto it. You'll note that I setup 2 consoles, one on tty0 and another on ttyS0. I like to have both a serial and a display console on kernel bootup.

Now we need to compile it, to do that, run the following:

```
mkimage -C none -A arm -T script -d boot.cmd boot.scr
```

Next step is to copy the kernel image and the script file. Assuming you are in the folder where you cloned all the source files earlier. (i.e the directory which has all the linux-sunxi and sunxi-boards folders), do the following:

```
cp ./linux-sunxi/arch/arm/boot/uImage /mnt/boot
cp ./sunxi-boards/sys_config/a10/script.bin /mnt/boot
```

Now we have all the pre boot binaries, bootloaders, kernel image and the script.bin files all loaded on to the SD card. Now we come to the last part, uploading the file system. I prefer the newest version of the Debain filesystem as it more flexible and light in nature. Now you need to download the tar file of the filesystem and unpack it at the place where your second partition is mounted (i.e /mnt/rootfs). So now using the axel and unp packages that we installed earlier, we run the following:

```
cd /tmp
axel http://dl.cubieforums.com/loz/rootfs/debian-unstable-armhf.tar.bz2
cd /mnt/rootfs
unp /tmp/debian-unstable-armhf.tar.bz2
```

This will unpack the Debain filesystem on the second partition. After this, you need to copy the compiled drivers, firmware etc to the appropriate place in the rootfs. This is an absolutely necessary step as any changes in the driver files wont take effects if they are not copied to the appropriate places in the file system. Here is an example snippet of this (Assuming again that the user is in the linux-sunxi folder):

```
mkdir -p /rootfs/lib/modules
rm -rf /rootfs/lib/modules/
cp -r ./output/* /rootfs/
```

Now just unmount the the SD card:

```
umount /mnt/boot
umount /mnt/rootfs
```

In case it says, it is busy, open a new instance of the terminal and run. It will surely work. After this, just boot up your cubieboard with this SD card, hopefully it should bootup and ask for login and password. For this debian distro, login is root, password is password.