Embedded Linux Project

FIRST DRAFT

Kevin Wu

June 24, 2008

Contents

1 About This Document

I am writing this document, not as a part of an assignment or official project, but for record keeping purposes. As such, you may find the tone to be somewhat colloquial, and the formatting to be unconventional.

1.1 Audience

This document is intended for anybody.

Depending on who you are, though, you might find this content to be trivial. I can only hope that the circumstances that I describe in section 2, Background, give you some perspective into where this is all coming from.

1.2 Status Log Quotes

Throughout this document, I will often copy-paste bits and pieces from a status log that I kept while I experimented and made progress.

> *here is an example of what some text from my status log will look like in this document. i don't really bother with syntax or grammar, and it appears like i am just chatting with someone online.*

These quotes show my thought process as I worked. With things happening so fast, I felt it would behoove me to keep a log that I could reference later if I needed to remember something. For each part that I describe in this document, I will try to include a piece from my log that gives more color to the situation, instead of paraphrasing everything.

1.3 Code and Shell Snippets

```
Code, commands, and standard output all look like this.
```

2 Background

In the spring of 2008, I took a class called "Embedded Linux." It was a 10-week course at UCSD Extension, and my first experience with embedded Linux. We met once a week for about 2 hours. Including myself, there were 10 people enrolled.

Our textbook was Building Embedded Linux Systems by Karim Yaghmour. The teacher attempted to cover one chapter per class, using Powerpoint presentations to summarize each chapter.

Prior to this class, I'd had casual experience with Linux from trying out various distributions on my desktop computer. I've settled with using Ubuntu for both my desktop and laptop, along with Windows, so I used it as my host system for the class.

Ubuntu supposedly makes things easy. I say "supposedly" because I can't claim that it's something I can fully appreciate, since I grew up using Windows. But from what I have read, and from my tinkerings with Linux, it seems that making Linux easy and accessible is an impossibly enormous undertaking. After taking this class, I am even more impressed.

Since embedded Linux was a new subject for me, I spent a lot of time reading the textbook. I started from the very beginning, and by the end of the class I had covered through chapter 6, and had also skimmed chapter 7.

In this class, we had the option of either doing weekly quizzes, or to do something interesting of our choosing with a single-board computer (SBC) that could run Linux. I chose the second option because I felt it would be more enlightening than simply reading the textbook and doing written quizzes.

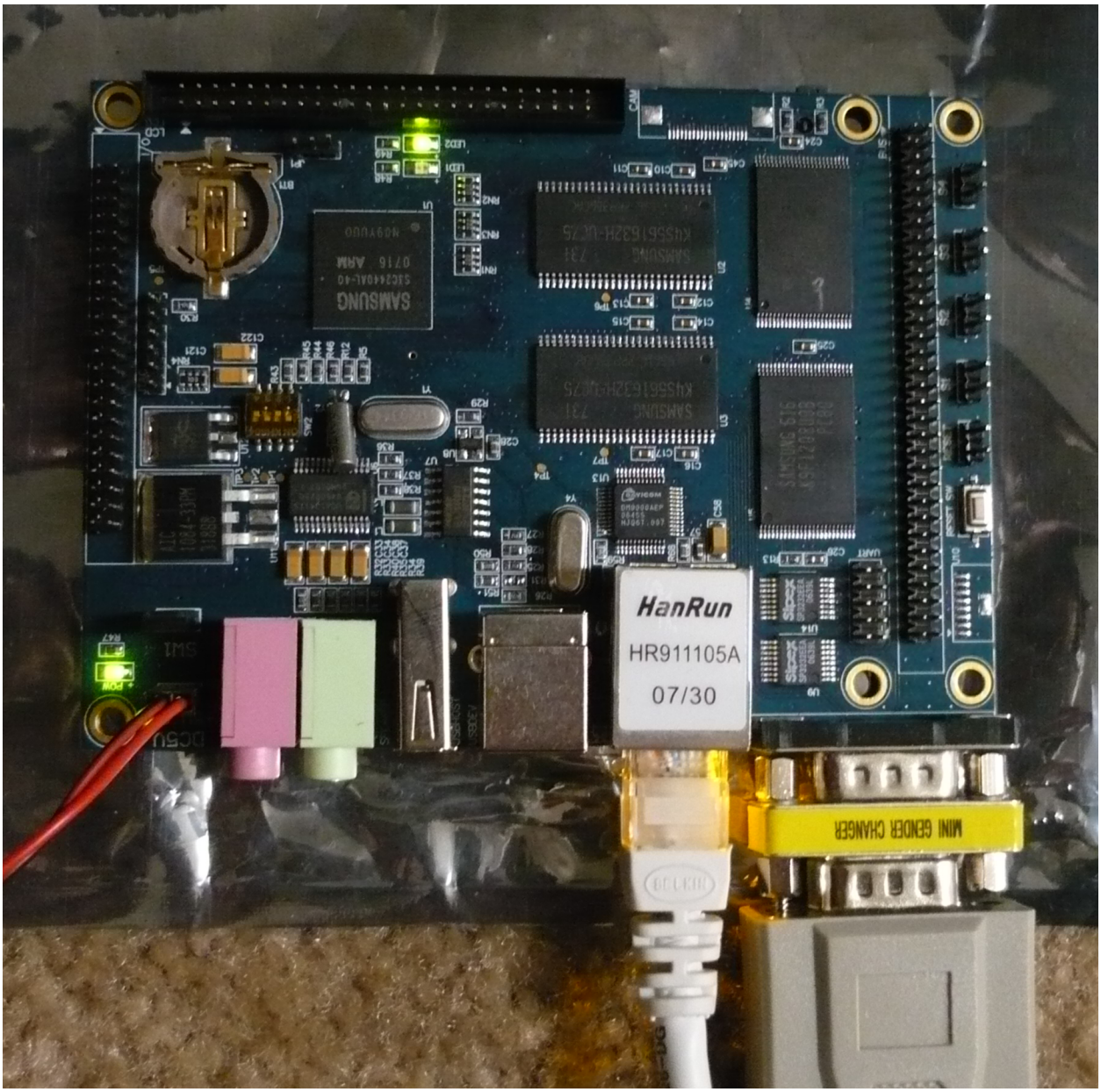
For those of us who chose this path, the teacher recommended that we purchase an SBC that already had a Linux distribution installed, and directed us to Gumstix and Glomation. I bought the Glomation GESBC-2440 because it seemed relatively new and had plenty of built-in features, the most important of which was the ethernet interface. Even though I wasn't sure what "interesting thing" I would do with the board, I wanted the board to allow me enough options.

3 SBC Project Ideas

Since the project was left open-ended, it was up to me to decide what to do with my SBC. It was difficult to come up with a concrete project idea because I did not fully know what my board could do, much less how to work with an embedded Linux system. Some people wanted to "roll" their own Linux distribution, which sounded interesting, but maybe too challenging given the time constraints of the class. At the same time, even though I bought an SBC with Linux pre-installed and cross-development toolchain already built, I didn't want everything to be simply handed to me as such. I wanted to know what building an embedded Linux system entailed, to do at least a part of it, and then to develop an application for it.

With this vague goal in mind, I began to think about some possible programs I could develop that would eventually run on my SBC. I wanted my SBC to do something practical, to do something that I would actually use. I soon discovered that learning about and working with my SBC and embedded Linux systems would severely limit the scope of any application I hoped to develop.

4 Glomation GESBC-2440



4.1 Bootloader

The GESBC-2440 uses U-Boot as its bootloader.

```
U-Boot 1.1.2 (Nov 15 2006 - 01:41:13)

U-Boot code: 33C00000 -> 33C25CC8  BSS: -> 33C5F610
RAM Configuration:
Bank #0: 30000000 64 MB
Get flash bank 0 size @ 0x0
Total Flash bank's sizes: 0x200000
protect monitor 25cc8 bytes @ address 0
Flash:  2 MB
```

```
NAND:Flash chip found:
 Manufacturer ID: 0xEC, Chip ID: 0x76 (Samsung unknown 64Mb)
1 flash chips found. Total nand_chip size: 64 MB
64 MB
In:    serial
Out:   serial
Err:   serial
Found DM9000 ID:90000a46 at address 18000000, ethaddr = 08:00:3e:26:0a:5b!
DM9000 work in 16 bus width
Not link of ethernet
Hit any key to stop autoboot:  1  0
SBC2440A#
```

4.2 Pre-installed Kernel

A patched version of Linux 2.6.18.2 came pre-installed on the GESBC-2440.

```
NAND read: device 0 offset 0, size 1293736 ...  1293736 bytes read: OK
## Booting image at 33000000 ...
   Image Name:   Linux-2.6.18.2
   Created:      2007-10-28   3:50:20 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1293672 Bytes =  1.2 MB
   Load Address: 30008000
   Entry Point:  30008000
   Verifying Checksum ... OK
OK

Starting kernel ...

Uncompressing Linux.....................................................
..... done, booting the kernel.
Linux version 2.6.18.2 (root@GESBC) (gcc version 4.1.2 20061115 (prerelease)
(Debian 4.1.1-21)) #586
Sun Oct 28 03:48:37 UTC 2007
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: SBZ2440
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C244X: core 399.651 MHz, memory 133.217 MHz, peripheral 66.608 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (2.116 MHz), fast, MPLL on, UPLL on
...
```

4.3 init and BusyBox

The GESBC-2440 uses BusyBox as its login and shell. After the kernel is loaded and the root filesystem is mounted, the system enters runlevel 5 and runs one script, /etc/rcS, which mounts filesystems, starts Opie, and starts BusyBox.

```
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 112K
  Vendor: Chipsbnk  Model: Flash Disk        Rev: 5.00
  Type:   Direct-Access                      ANSI SCSI revision: 02
SCSI device sda: 2061824 512-byte hdwr sectors (1056 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
s3c2410-sdi s3c2410-sdi: powered down.
```

```
SCSI device sda: 2061824 512-byte hdwr sectors (1056 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: Attached scsi removable disk sda
mount /etc as ramfs
mount /dev as tmpfs
mounting all
Scanning hardware
You seem to already have a /home/root/Applications directory.
Assuming it is the Opie Applications directory. Exiting.
Starting Opie....
ODevice() - found 'Hardware: SBZ2440'
ODevice() - unknown hardware - using default.
OGlobal::creating global configuration instance.
OConfig::OConfig()
<unknown>: ODevice reports transformation to be 0
<unknown>: setting QWS_DISPLAY to 'Transformed:Rot0:0'
qt_init() - starting in daemon mode...

Please press Enter to activate this console.


BusyBox v1.4.1 (2007-03-25 03:24:25 CST) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

[root@(none) /]# uname -m
uname -m
armv4tl
```

While BusyBox is full-featured, it was difficult to use vi, or to do tab-completion, or to edit shell commands in place, because all control and navigation characters appeared on the screen when they are typed. Perhaps these issues have been fixed in later versions. I thought it might have been my terminal emulator, so I tried various emulators and settings, and found PuTTY to work the best, but control characters still appeared.

```
...
[root@(none) /]# ^[[A^[[B^[^[^[
...
"rcS" 17L, 343C0^[^[^[^[^[^[lllllllllllllllllllllllllllllllllllllllllllllllllll
lllllllllllllllla
I rcS [modified] 19/19 100%udhcpc -i eth0 -b -p /var/run/udhcpc.eth0 >/dev/null 2>&1^[:wq
~- rcS 1/19 5%^[jjjjjjjjjjjjjjjjjjjjj
^[kk
- rcS 19/19 100%- rcS 18/19 94%- rcS 17/19 89%- rcS 18/19 94%^[kk
- rcS 18/19 94%- rcS 17/19 89%- rcS 16/19 84%- rcS 17/19 89%x^[:wq
...
```

5 Cross-development Toolchain

I learned the hard way that building a cross-development toolchain depends highly on the version of gcc that is used to do the build. gnuarm.com groups packages for binutils, gcc, and newlib according to the version of gcc installed on your host; but before I realized this, I tried building my own versions and ran into countless compilation errors. Unfortunately, even with the nicely grouped packages from gnuarm.com, I could not get past building the bootstrap gcc. From my project log:

> *ok, binutils build a success. gcc-4.2 is ok so far. now for building bootstrap gcc. configure went without a hitch. huh. i even used –without-headers. not sure if that is because it's been fixed, or whether i'm just building the "correct" version for the version of gcc i have installed.*
>
> *uh oh..*
>
> `../../gcc-4.1.1/gcc/gthr-posix.h:43:21: error: pthread.h: No such file or directory`
> `../../gcc-4.1.1/gcc/gthr-posix.h:44:20: error: unistd.h: No such file or directory`
>
> *so far everyone seems to be saying that i need a c library before i try to compile the bootstrap compiler. so why doesn't the book say that? ARGH.*
>
> *trying gcc-4.1 to build bootstrap gcc. same problem*
>
> *tired. moving onto trying to build glomation kernel using glomation toolchain. i think i still need to compile the toolchain though. argh.*

It seemed like picking the correct versions of the different components would take a lot of guess-and-check work, which I didn't have time to do.

5.1 crosstool

I almost gave up and used the toolchain provided by Glomation, but decided to try something that I found while googling compilation errors, called crosstool. It is a script that basically does all the work of trying different version combinations. In fact, there is a matrix, created from the author's own buildlogs created by the script, that shows which combinations work and don't work.

> *i looked for one that had the glibc installed on the board (2.3.2), and a gcc_core_dir that matched one i could just apt-get from the ubuntu repos (3.3.6). i edited the corresponding combination's .dat file and explicity included the gcc_core_dir variable, explained here: crosstool-vars.html and i ln -s -f /usr/bin/gcc-3.3 /usr/bin/gcc, just in case. so, here goes, before i end up using glomation's toolchain.*
>
> *ok, turns out some automatically generated files weren't being generated properly..*
>
> `crbuild-glibc/csu/version-info.h missing terminating '' character.`
>
> *did some googling (googled for crosstool build-glibc/csu/version-info.h missing terminating character) and it's because of the freakin shell. i thought i was using bash.. but it turns out, ubuntu uses dash. so, i did the symbolic link to bash.. and even used bash to execute the script.. and it worked!!!!!!!!!!*
>
> ```
> crosstool: final gcc built ok
> Cross-toolchain build complete. Result in
> /opt/crosstool/gcc-3.3.6-glibc-2.3.2/arm-unknown-linux-gnu.
> testhello: C compiler can in fact build a trivial program.
> Done.
> ```

I successfully compiled the obligatory "hello world" program and ran it on my board, as well as the "file" utility:

```
[root@(none) gesbc-2440_binaries]# ./file -m magic helloworld
./file -m magic helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1, for GNU/Linux 2.4.18,
dynamically linked (uses shared libs), not stripped
```

I later had to recreate a toolchain using crosstool because compiling my kernel required gcc v3.4.3 or higher.

6 Linux Kernel

For the GESBC-2440, Glomation provides kernel version 2.6.18.2. Still, I wanted my own kernel:

> *i tried getting the latest stable kernel from kernel.org, then getting the latest patch from arm.linux.org.uk. i patched the kernel source fine.. but when i did make menuconfig, i got an error from mconf.c. did some search, and it turns out it's because my gcc (the ones from the fiesty repos.. for my c/c++ class) is too new.. v4.1 or whatever. i need to use gcc v3.4 or something instead. i found a way to get around it, though.. use this patch:*
>
> ```
> Index: package/config/mconf.c
> ===================================================
> --- package/config/mconf.c (revision 10888)
> +++ package/config/mconf.c (working copy)
> @@ -101,7 +101,7 @@
> static int indent = 0;
> static struct termios ios_org;
> static int rows, cols;
> -static struct menu *current_menu;
> +struct menu *current_menu;
> static int child_count;
> static int single_menu_mode;
> ```
>
> *so, that worked.. but then i got a whole different slew of errors, similar to the ones i got when i tried to make menuconfig with the glomation kernel source. i picked a line and did a google search.. found this: stack_chk_fail. basically i needed to install ncurses-dev. craziness.*
>
> *so anyway, now i'm in the menuconfig. i'm unsure about almost everything. currently stuck on config_embedded..*

I configured the kernel based on the help blurbs for each option, but I soon noticed that my kernel didn't have options related to the S3C2440 or S3C2410. I was confused as to how each architecture gets added to the kernel.

> *tried to finish configuring the kernel, even though i think it won't work because i don't have the config options for the s3c2440. tried doing some google searches for s3c2440 linux kernel version, stuff like that. at this point i'm pretty tired.. not sure how it all comes together. how does the specific information for certain chips get added to kernel code? if that specific info isn't there, does linux not work on that chip? or does it just run a very generic version of linux that is specific to the generic architecture? i mean, aren't there different versions of x86? i guess different x86 chips have different features.. but is the kernel code mature enough such that all those different chips are included?*
>
> *i don't feel like compiling the toolchain now. maybe i will just use the glomation ones. i'm starting to understand why don recommends linux already on the board. it would definitely be easier. that would mean i just need to configure anjuta to use the glomation toolchain, instead of the toolchain already on my ubuntu, and i would just need to read the s3c2440 spec sheet to do some programming to manipulate the pins. but then i'm not really learning as much. and i've already sort of gotten myself this deep into it. i just don't understand how to get the right kernel, how to find out whether the s3c2440 is more or less supported in specific versions of the kernel. that is the big mystery now. is it more supported in debian? i looked at the debian etch glomation package and it's just the filesystem layout, no kernel source code or whatever.*

I eventually figured out that I didn't need to be concerned about it.

> *i went back to the main page of arm.linux.co.uk, went to developers, then i clicked on kernel compilation, kerncomp.php. very useful info, ESPECIALLY THIS:* ***(note: kernels later than 2.6.0-test2 do not require a -rmk or -vrs patch to be applied since ARM architecture support is already merged.)*** *AMAZING! i think that's where my confusion was. i thought from the start that i needed an arm kernel patch, so i looked for the latest one at kernel-patches/v2.6/. the latest one was for a 2.6.0 kernel, so i downloaded that kernel. it turns out arm support was merged into kernels after that, so i can just download the latest one and it should have arm, as well as my s3c2440.. which, i have just confirmed, it does! sweeeeeettt...*

Ultimately, I used the kernel package provided by Glomation. I learned that even though I had architecture support, my peripherals wouldn't necessarily work, and that device drivers would need to be written for them. I thought maybe I could do a diff between the Glomation-provided kernel and the kernel of the same version from kernel.org, and determine what device driver patches were created:

> *i've been looking at the files that are different, and i have no idea what is going on.. it's all c, but, i don't understand it. eh. i wonder whether i can merge the differences into the latest kernel. but i think i want to understand why there are differences and how whoever made the changes knew what changes to make.*
>
> . . .
>
> *how can i upgrade from glomation to the latest? would it be possible to patch the glomation kernel? that might not work, because patching depends on certain lines being in there. if stuff was changed around, then the patch might not get implemented. the long hard way would be to figure out WHY things were patched. go through each changed file, and figure out why it was changed. then try to implement the same changes, manually, in the latest kernel.*

7 S3C2440 ADC Project

After getting a better grasp of my board, the toolchain, and the kernel, I decided that my project would be to use the ADC to report voltage from a solar cell (photovoltaic cell). Even though I felt that it would be a fairly simple project (we had programmed the ADC on our Atmel AtTiny44 boards in my ECP3 class), I didn't know where to begin, or what this endeavor would entail. I began by purchasing some PV cells from Edmund Scientific, and reading up on the ADC in the S3C2440 datasheet.

I wanted to program the ADC in the same way I programmed it for the AtTiny44, by simply writing values to a register name and checking the results stored in another register. I looked up the register names in the kernel header files and found that "ADCCON" is the configuration register, while the first 10 bits of "ADCDAT0" would hold conversion results. I wrote a program to simply read from the ADCCON register, but got segmentation fault. I learned that "segmentation fault" meant I was trying to access a memory address that I wasn't allowed to access.

I contacted Glomation about whether they had any sample code for talking to the ADC. This is where things became complicated.

> *Dear Kevin Wu,*
>
> *We currently do not have any sample code to access the ADC on the S3C2440 chip. There are 2 ways to access the on-chip ADC of the S3C2440, a Linux device driver or a simple user space program. The device driver approach requires some in-depth knowledge of the Linux device drivers and is a little more complicated than the user space program but it provides system level access to the on-chip ADC. The user space program is simple to program and provide simple access of the on-chip ADC. Here is a link to a user space program that can access any memory location of the Linux system, devmem2.c. Since the on-chip ADC is memory mapped, you can actually access the ADC just using the devmem2 program in a sequence of writes and reads of the ADC registers. Please refer to the S3C2440 data sheet on how to program the ADC.*
>
> *Please be noted that the default installed kernel include a touch screen driver for the GESBC-2440 SBC which uses the on-chip ADC as touch screen interface. You will need to disable the touch screen driver from the kernel in order to use the on-chip ADC.*
>
> *Regards,*
>
> *Glomation Customer Support*

I compiled and ran devmem2 on my target. I read the value at the ADCCON address (0x5800000) to check whether it matched the reset value given in the datasheet, but it didn't. So I recompiled the kernel without touchscreen support, and created a uImage, which is the kernel image format that U-Boot accepts.

8 Loading Kernel Onto GESBC-2440

Loading the kernel is all done in U-Boot, my target's bootloader.

In order to load the kernel image onto the NAND flash on my target, I had to erase the NAND by issuing the command:

```
nand erase clean
```

Then I load the uImage onto the SDRAM chip via a TFTP server running on my host (I used this one), with the uImage file located in the TFTP server's shared folder:

```
set ipaddr 192.168.xxx.xxx
set serverip 192.168.xxx.xxx
tftp 33000000 uImage
```

This sets the target and server (host) IP addresses, and begins the TFTP transfer of uImage. Once the image is in SDRAM, I transfer it to the NAND flash:

```
nand write.jffs2 33000000 0 $(filesize)
```

write.jffs2 is used because the NAND is formatted with JFFS2 filesystem. It reads/writes 'filesize' bytes, starting at offset 0, to/from memory address 33000000. The filesize variable holds the size, in hex, of the image read from the last tftp command.

Now I have to tell U-Boot where the kernel image is:

```
set bootcmd 'nand read.jffs2 33000000 0 $(filesize); bootm 33000000'
```

This simply tells U-Boot that the bootable image is located at address 33000000, offset 0, and image size 'filesize'.

Loading the kernel was successful. My target booted, and I saw that it was indeed the kernel I had compiled.

```
NAND read: device 0 offset 0, size 1307664 ...  1307664 bytes read: OK
## Booting image at 33000000 ...
   Image Name:   Linux-2.6.18.2
   Created:      2008-06-18   3:11:52 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1307600 Bytes =  1.2 MB
   Load Address: 30008000
   Entry Point:  30008000
   Verifying Checksum ... OK
OK

Starting kernel ...

Uncompressing Linux.....................................................
..... done, booting the kernel.
Linux version 2.6.18.2 (wuziq@wuziq-laptop-linux)
(gcc version 3.4.5) #2
Tue Jun 17 20:11:48 PDT 2008
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: SBZ2440
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C244X: core 399.651 MHz, memory 133.217 MHz, peripheral 66.608 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (2.116 MHz), fast, MPLL on, UPLL on
...
```

Unfortunately, I didn't realize that my root filesystem was also located on the NAND flash, so I had accidentally erased it when I issued the "nand erase clean" command.

9 Mounting Root Filesystem over NFS

With time running out, I needed a new root filesystem as soon as possible. Although I read the chapter on how to set up the root filesystem, I hadn't actually done the work. I looked at the Glomation support site and saw that they provided a Debian-based file system for use on a USB drive. I tried downloading it, but discovered it was over 200 MB, so it wouldn't fit on my target by any means. I attempted to unpack it onto my USB stick, but there were permissions issues, which I think were resulting from the fact that my USB stick was formatted as fat16 instead of ext2.

Then I noticed instructions for mounting the root file system over NFS. They were for a Glomation board different from mine, but I used what I could and figured out the rest.

> *i'm noticing on USB-as-root-file-system.txt that it's just a matter of specifying where the rootfs in the bootloader. i think i could just follow the user manual instructions for "set up uboot boot environment" and change the bootargs to 'root=dev/sda1/rootfsfolder rw rootfstype=vfat noinitrd init=/linuxrc console=ttySAC0,115200'. i think i'll try it. the rootfs package just finished downloading. and luckily my thumbstick is 1gb.*
>
> *ok, there are some permissions issues in the rootfs package. i could sudo untar it to my host, but not to the thumbstick. hmmmm...*
>
> *ok what about nfs? looking at Ubuntu:Gutsy#NFS_Server and nfs-rootfs-setup-linux.txt. seeeeeems doable. but then so did usb.*

```
sudo apt-get install nfs-kernel-server nfs-common portmap
sudo dpkg-reconfigure portmap
sudo /etc/init.d/portmap restart
sudo /etc/init.d/nfs-kernel-server restart
sudo exportfs -a
wuziq@wuziq-laptop-linux:~/gesbc-2440/rootfs_nfs$ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100024    1   udp  36513  status
    100024    1   tcp  38860  status
    100000    2   udp    111  portmapper
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100021    1   udp  37927  nlockmgr
    100021    3   udp  37927  nlockmgr
    100021    4   udp  37927  nlockmgr
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100021    1   tcp  60456  nlockmgr
    100021    3   tcp  60456  nlockmgr
    100021    4   tcp  60456  nlockmgr
    100005    1   udp  32974  mountd
    100005    1   tcp  46968  mountd
    100005    2   udp  32974  mountd
    100005    2   tcp  46968  mountd
    100005    3   udp  32974  mountd
    100005    3   tcp  46968  mountd
```

> *seems to look ok.*
>
> *ok, now to try booting.*
>
> *dude i think it worked!*

```
s3c2410-sdi s3c2410-sdi: powered down.
Sending DHCP requests ., OK
IP-Config: Got DHCP answer from 192.168.2.1, my address is 192.168.2.109
```

```
IP-Config: Complete:
      device=eth0, addr=192.168.2.109, mask=255.255.255.0, gw=192.168.2.1,
     host=192.168.2.109, domain=, nis-domain=(none),
     bootserver=192.168.2.1, rootserver=192.168.2.101, rootpath=
Looking up port of RPC 100003/2 on 192.168.2.101
Looking up port of RPC 100005/1 on 192.168.2.101
VFS: Mounted root (nfs filesystem).
Freeing init memory: 112K
SBZ: try run /sbin/init
INIT: version 2.86 booting
devfsd: No devfs on /dev, not starting.

The device node /dev/sda1 for the root filesystem is missing,
incorrect, or there is no entry for the root filesystem
listed in /etc/fstab.

The system is also unable to create a temporary node in
/dev/shm to use as a work-around.

This means you have to fix this manually.

CONTROL-D will exit from this shell and REBOOT the system.

Press enter for maintenance
(or type Control-D to continue):
```

ok.. i don't know what those errors are about.. but i'm poking around and i think it worked!!!!!!!

```
root@192:/# cd ..
root@192:/# ls
bin/   dev/  home/    lib/         mnt/      opt/   root/  sys/  usr/
boot/  etc/  initrd/  lost+found/  nfsroot/  proc/  sbin/  tmp/  var/
root@192:/# uname
Linux
root@192:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 80:E2:66:60:00:01
          inet addr:192.168.2.109  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2878 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1538 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:51 Base address:0xe000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

The first thing I noticed is that I no longer had the vi, control character, or tab-completion problems that I had with BusyBox (see subsection 4.3). I was essentially using a "standard" Linux shell. Also, this root filesystem already came loaded with many more applications, including apt-get.

```
root@192:etc# apt-get update
Get:1 http://http.us.debian.org stable/main Packages [5449kB]
```

```
Get:2 http://http.us.debian.org stable/main Release [94B]
Get:3 http://http.us.debian.org stable/contrib Packages [53.6kB]
Get:4 http://http.us.debian.org stable/contrib Release [97B]
Get:5 http://http.us.debian.org stable/non-free Packages [71.0kB]
Get:6 http://http.us.debian.org stable/non-free Release [98B]
Fetched 5574kB in 49s (114kB/s)
Reading Package Lists... Done
```

9.1 U-Boot bootargs for Mounting Root Filesystem Over NFS

Here is how I set the boot arguments in U-Boot:

```
set bootargs 'nfsroot=192.168.xxx.xxx:/home/wuziq/gesbc-2440/rootfs_usb rw
console=ttySAC0,115200 ip=dhcp
```

"nfsroot" tells U-Boot to look at address 192.168.xxx.xxx and directory /home/wuziq/gesbc-2440/rootfs_usb for the root filesystem. "console" configures the console, and "ip=dhcp" tells U-Boot to use DHCP to obtain an IP address.

Saving these U-Boot environment variables is done with:

```
saveenv
```

9.2 fstab and inittab

The fstab file that came with the root filesystem included a line to mount /dev/sda1 as the root filesystem. Since the root filesystem is already mounted over NFS, I simply removed that line. This fixed the "no entry for the root filesystem listed in /etc/fstab" error, and allowed inittab to execute during bootup.

inittab, however, was not configured properly, so after executing the scripts in /etc/rc2.d, I got:

```
INIT:  no more processes in this runlevel
```

I figured out that it was because the terminal configuration at the end of the file was for a different board with different terminal settings:

```
T0:23:respawn:/sbin/getty -L ttyAM0 57600 vt100
T1:23:respawn:/sbin/getty -L ttyAM1 57600 vt100
```

I replaced those two lines with my terminal settings:

```
T0:23:respawn:/sbin/getty 115200 console
```

and it works. w0000000ttt.

```
VFS: Mounted root (nfs filesystem).
Freeing init memory: 112K
INIT: version 2.86 booting
devfsd: No devfs on /dev, not starting.
Cannot access the Hardware Clock via any known method.
Use the --debug option to see the details of our search for an access method.
Cannot access the Hardware Clock via any known method.
Use the --debug option to see the details of our search for an access method.
Cleaning up ifupdown...done.
Checking all file systems...
fsck 1.37 (21-Mar-2005)
... done.
Setting up networking...done.
Starting hotplug subsystem: usb isapnp ide scsi.
Setting MAC address to 00:11:22:33:44:55
```

```
SIOCSIFHWADDR: Device or resource busy
Setting up IP spoofing protection: rp_filter.
Enabling packet forwarding...done.
Configuring network interfaces...done.
Starting portmap daemon: portmap.
Cannot access the Hardware Clock via any known method.
Use the --debug option to see the details of our search for an access method.
Running ntpdate to synchronize clock.
Recovering nvi editor sessions... done.
INIT: Entering runlevel: 2
Starting internet superserver: inetd.
Starting OpenBSD Secure Shell server: sshd.
Starting NFS common utilities: statd.
Starting NFS servers: nfsd mountd.
Starting periodic command scheduler: cron.
Starting web server: apache-ssl.

Debian GNU/Linux 3.1 GESBC-93xx console

GESBC-93xx login: root
Last login: Fri Sep  1 08:00:52 2006 on ttyAM0
Linux GESBC-93xx #31 Thu Mar 23 13:01:48 MST 2006 armv4l GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@GESBC-93xx:root# ls
adc_test*   adc_test_new*   devmem2_test*   word.log
adc_test.c  adc_test_new.c  halfword.log    word_part2.log
adc_test.o  adc_test_new.o  memory_log.txt
```

10 "Hello, World" Kernel Module

Gauging from the progress of my classmates from their weekly updates, I felt that mounting a root filesystem over NFS was presentable material for my final project. Nevertheless, I had one more day until the last class, where we would be presenting our projects, and I still wanted to do what I had originally intended with the ADC.

I tried devmem2 again, but I was getting the same results as before. Were the kernel reconfiguration and recompilation for nothing? I tried writing a value to the ADCCON register address, then reading the value back, but the value remained the same as before the write:

```
root@192:root# devmem2 0x5800000 h 0x5901
/dev/mem opened.
Memory mapped at address 0x40017000.
Value at address 0x5800000 (0x40017000): 0x12
Written 0x5901; readback 0x12
```

I felt that I had done all I could do. But then I talked to a couple coworkers about accessing memory locations. They reminded me that the OS would prevent me from directly accessing memory, and that a device driver would probably be needed.

(10:58:34 AM) coworker: so
(10:58:39 AM) coworker: write a little c program
(10:59:12 AM) coworker: that sets uint16 ptr = 0x5800000 and writes to it*
(10:59:25 AM) coworker: and then trys 0x40017000 to write to it
(10:59:28 AM) coworker: see if either works
(10:59:59 AM) coworker: if that doesn't work, modify an existing drive to do it
(10:59:59 AM) me: i tried the first line, i get segmentation fault
(11:00:20 AM) coworker: what about second line
(11:00:26 AM) me: devmem2 does the second line
(11:00:47 AM) coworker: u dont need to write a whole driver
(11:00:58 AM) coworker: just insert some code into an existing driver
(11:01:01 AM) coworker: to do it
(11:01:07 AM) coworker: to see if it works
(11:01:07 AM) me: HMMMMMMMMMMMMMM
(11:01:14 AM) me: very interesting
(11:01:23 AM) coworker: thats what i would do cuz it's easy
(11:01:26 AM) me: a driver that the kernel is already using
(11:01:30 AM) coworker: ya
(11:01:36 AM) me: very interesting

This new information set me on a path that never crossed my mind for this class: writing a Linux device driver.

There was a book at my workplace, called Linux Device Drivers, 2nd Edition ("LDD2"). I flipped through the pages, not understanding anything, except for the "Hello, World" program at the beginning. I figured I would at least try writing and loading it, and then go on from there. This decision led me to a few hours of trying to debug compilation errors. I couldn't even get the module to compile on my host.

I finally realized that LDD2 was aimed at kernel version 2.4, and, from what I had read while debugging compilation errors, kernel version 2.6 handles modules in a very different way. I checked for a newer version of LDD, and found Linux Device Drivers, 3rd Edition at lwn.net. I also downloaded the driver examples from the author's FTP site.

I tried executing make, but it was looking for kernel headers whose version matched my kernel version (2.6.18.2). I had installed the kernel source tree on my target, but my target also had slightly newer kernel headers loaded onto it as well, along with "-s3c2410" appended to the source tree name. After some sloppy renaming, the make worked, but upon insmod-ing, I got:

```
root@GESBC-93xx:misc-modules# insmod ./hello.ko
hello: version magic '2.6.18-6-s3c2410 mod_unload ARMv4 gcc-4.1'
should be '2.6.18.2 mod_unload ARMv4 gcc-3.4'
insmod: error inserting './hello.ko': -1 Invalid module format
```

It turns out that make is using some information it finds in the kernel source tree to figure out version stuff. This is actually explained in LDD3. I found out that you can point make to your kernel source tree by setting the KERNELDIR variable.

After setting KERNELDIR, and creating arm-linux-gcc and arm-linux-ld symbolic links to my crosstool-created toolchain binaries, the make ran successfully, and modinfo showed matching magic versions:

```
wuziq@wuziq-laptop-linux:~/...$ modinfo hello.ko
filename:       hello.ko
license:        Dual BSD/GPL
depends:
vermagic:       2.6.18.2 mod_unload ARMv4 gcc-3.4
```

Finally, I successfully loaded and unloaded the driver:

```
root@GESBC-93xx:misc-modules# lsmod
Module                  Size  Used by
root@GESBC-93xx:misc-modules# insmod ./hello.ko
Hello, world
root@GESBC-93xx:misc-modules# lsmod
Module                  Size  Used by
hello                   1120  0
root@GESBC-93xx:misc-modules# rmmod hello
Goodbye, cruel world
root@GESBC-93xx:misc-modules# lsmod
Module                  Size  Used by
root@GESBC-93xx:misc-modules#
```

11 Miscellaneous

Here are some little tips that helped me along the way.

- To show the U-Boot environment variables, use "printenv":

```
SBC2440A# printenv

bootdelay=1
baudrate=115200
ethaddr=08:00:3e:26:0a:5b
filesize=1f44000
fileaddr=30000000
gatewayip=192.168.0.1
netmask=255.255.255.0
serverip=192.168.0.102
ipaddr=ip=192.168.0.8:192.168.0.102:192.168.0.102:255.255.255.0:GESBC:eth0:off
bootcmd=nand read.jffs2 33000000 0 13f410; bootm 33000000
bootargs=nfsroot=192.168.0.102:/home/wuziq/gesbc-2440/rootfs_usb rw
console=ttySAC0,115200 ip=dhcp
stdin=serial
stdout=serial
stderr=serial

Environment size: 458/65532 bytes
```

- To configure ethernet manually, if there's no DHCP client:

```
ifconfig eth0 192.168.2.103
route add default gw 192.168.2.1 eth0
ifconfig lo 127.0.0.1
echo "nameserver 192.168.2.1" > /etc/resolv.conf
```

- To see all possible options and variables that are configurable for a configure script, simply use:

```
./configure --help
```

- To quickly transfer files from host to target in an ethernet-connected setup, use scp:

```
wuziq@wuziq-laptop-linux:~/...$ scp hello.ko root@192.168.0.104:/root/.../
hello.ko                                  100% 1965     1.9KB/s   00:00
```

  I was able to do this with my NFS-mounted root filesystem. Tab-completion also worked for completing paths on the target, which I thought was pretty cool.

12 Conclusions, Future Work

Coming soon...