The Intel Assabet (SA-1110 evaluation) board

Please see: http://developer.intel.com

Also some notes from John G Dorsey <jd5q@andrew.cmu.edu>: http://www.cs.cmu.edu/~wearable/software/assabet.html

Building the kernel

To build the kernel with current defaults:

```
make assabet_defconfig
make oldconfig
make zImage
```

The resulting kernel image should be available in linux/arch/arm/boot/zImage.

Installing a bootloader

A couple of bootloaders able to boot Linux on Assabet are available:

BLOB (http://www.lartmaker.nl/lartware/blob/)

> BLOB is a bootloader used within the LART project. Some contributed patches were merged into BLOB to add support for Assabet.

Compaq's Bootldr + John Dorsey's patch for Assabet support (http://www.handhelds.org/Compaq/bootldr.html) (http://www.wearablegroup.org/software/bootldr/)

> Bootldr is the bootloader developed by Compaq for the iPAQ Pocket PC. John Dorsey has produced add-on patches to add support for Assabet and the JFFS filesystem.

RedBoot (http://sources.redhat.com/redboot/)

> RedBoot is a bootloader developed by Red Hat based on the eCos RTOS hardware abstraction layer. It supports Assabet amongst many other hardware platforms.

RedBoot is currently the recommended choice since it's the only one to have networking support, and is the most actively maintained.

Brief examples on how to boot Linux with RedBoot are shown below. But first you need to have RedBoot installed in your flash memory. A known to work precompiled RedBoot binary is available from the following location:

- ftp://ftp.netwinder.org/users/n/nico/
- ftp://ftp.arm.linux.org.uk/pub/linux/arm/people/nico/
- ftp://ftp.handhelds.org/pub/linux/arm/sa-1100-patches/

Look for redboot-assabet*.tgz. Some installation infos are provided in redboot-assabet*.txt.

Initial RedBoot configuration

The commands used here are explained in The RedBoot User's Guide available on-line at http://sources.redhat.com/ecos/docs.html. Please refer to it for explanations.

If you have a CF network card (my Assabet kit contained a CF+ LP-E from Socket Communications Inc.), you should strongly consider using it for TFTP file transfers. You must insert it before RedBoot runs since it can't detect it dynamically.

To initialize the flash directory:

```
fis init -f
```

To initialize the non-volatile settings, like whether you want to use BOOTP or a static IP address, etc, use this command:

```
fconfig -i
```

Writing a kernel image into flash

First, the kernel image must be loaded into RAM. If you have the zImage file available on a TFTP server:

```
load zImage -r -b 0x100000
```

If you rather want to use Y-Modem upload over the serial port:

```
load -m ymodem -r -b 0x100000
```

To write it to flash:

```
fis create "Linux kernel" -b 0x100000 -l 0xc0000
```

Booting the kernel

The kernel still requires a filesystem to boot. A ramdisk image can be loaded as follows:

```
load ramdisk_image.gz -r -b 0x800000
```

Again, Y-Modem upload can be used instead of TFTP by replacing the file name by '-y ymodem'.

Now the kernel can be retrieved from flash like this:

```
fis load "Linux kernel"
```

or loaded as described previously. To boot the kernel:

```
exec -b 0x100000 -l 0xc0000
```

The ramdisk image could be stored into flash as well, but there are better solutions for on-flash filesystems as mentioned below.

Using JFFS2

Using JFFS2 (the Second Journalling Flash File System) is probably the most convenient way to store a writable filesystem into flash. JFFS2 is used in conjunction with the MTD layer which is responsible for low-level flash management. More information on the Linux MTD can be found on-line at: http://www.linux-mtd.infradead.org/. A JFFS howto with some infos about creating JFFS/JFFS2 images is available from the same site.

For instance, a sample JFFS2 image can be retrieved from the same FTP sites mentioned below for the precompiled RedBoot image.

To load this file:

```
load sample_img.jffs2 -r -b 0x100000
```

The result should look like:

```
RedBoot> load sample_img.jffs2 -r -b 0x100000
Raw file loaded 0x00100000-0x00377424
```

Now we must know the size of the unallocated flash:

```
fis free
```

Result:

```
RedBoot> fis free
  0x500E0000 .. 0x503C0000
```

The values above may be different depending on the size of the filesystem and the type of flash. See their usage below as an example and take care of substituting yours appropriately.

We must determine some values:

```
size of unallocated flash:      0x503c0000 - 0x500e0000 = 0x2e0000
size of the filesystem image:   0x00377424 - 0x00100000 = 0x277424
```

We want to fit the filesystem image of course, but we also want to give it all the remaining flash space as well. To write it:

```
fis unlock -f 0x500E0000 -l 0x2e0000
fis erase -f 0x500E0000 -l 0x2e0000
fis write -b 0x100000 -l 0x277424 -f 0x500E0000
fis create "JFFS2" -n -f 0x500E0000 -l 0x2e0000
```

Now the filesystem is associated to a MTD "partition" once Linux has discovered what they are in the boot process. From Redboot, the 'fis list' command displays them:

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00020000  0x00000000
RedBoot config    0x503C0000  0x503C0000  0x00020000  0x00000000
FIS directory     0x503E0000  0x503E0000  0x00020000  0x00000000
Linux kernel      0x50020000  0x00100000  0x000C0000  0x00000000
JFFS2             0x500E0000  0x500E0000  0x002E0000  0x00000000
```

However Linux should display something like:

```
SA1100 flash: probing 32-bit flash bus
SA1100 flash: Found 2 x16 devices at 0x0 in 32-bit mode
```

```
Using RedBoot partition definition
Creating 5 MTD partitions on "SA1100 flash":
0x00000000-0x00020000 : "RedBoot"
0x00020000-0x000e0000 : "Linux kernel"
0x000e0000-0x003c0000 : "JFFS2"
0x003c0000-0x003e0000 : "RedBoot config"
0x003e0000-0x00400000 : "FIS directory"
```

What's important here is the position of the partition we are interested in, which is the third one. Within Linux, this correspond to /dev/mtdblock2. Therefore to boot Linux with the kernel and its root filesystem in flash, we need this RedBoot command:

```
fis load "Linux kernel"
exec -b 0x100000 -l 0xc0000 -c "root=/dev/mtdblock2"
```

Of course other filesystems than JFFS might be used, like cramfs for example. You might want to boot with a root filesystem over NFS, etc. It is also possible, and sometimes more convenient, to flash a filesystem directly from within Linux while booted from a ramdisk or NFS. The Linux MTD repository has many tools to deal with flash memory as well, to erase it for example. JFFS2 can then be mounted directly on a freshly erased partition and files can be copied over directly. Etc...

RedBoot scripting

All the commands above aren't so useful if they have to be typed in every time the Assabet is rebooted. Therefore it's possible to automate the boot process using RedBoot's scripting capability.

For example, I use this to boot Linux with both the kernel and the ramdisk images retrieved from a TFTP server on the network:

```
RedBoot> fconfig
Run script at boot: false true
Boot script:
Enter script, terminate with empty line
>> load zImage -r -b 0x100000
>> load ramdisk_ks.gz -r -b 0x800000
>> exec -b 0x100000 -l 0xc0000
>>
Boot script timeout (1000ms resolution): 3
Use BOOTP for network configuration: true
GDB connection port: 9000
Network debug at boot time: false
Update RedBoot non-volatile configuration - are you sure (y/n)? y
```

Then, rebooting the Assabet is just a matter of waiting for the login prompt.

Nicolas Pitre nico@fluxnic.net

June 12, 2001

Status of peripherals in -rmk tree (updated 14/10/2001)

Assabet:

Serial ports:

Radio: TX, RX, CTS, DSR, DCD, RI

- PM: Not tested.
- COM: TX, RX, CTS, DSR, DCD, RTS, DTR, PM
- PM: Not tested.
- I2C: Implemented, not fully tested.
- L3: Fully tested, pass.
- PM: Not tested.

Video:

- LCD: Fully tested. PM

  (LCD doesn't like being blanked with neponset connected)

- Video out: Not fully

Audio:

UDA1341: - Playback: Fully tested, pass. - Record: Implemented, not tested. - PM: Not tested.

UCB1200: - Audio play: Implemented, not heavily tested. - Audio rec: Implemented, not heavily tested. - Telco audio play: Implemented, not heavily tested. - Telco audio rec: Implemented, not heavily tested. - POTS control: No - Touchscreen: Yes - PM: Not tested.

Other:

- PCMCIA:

- LPE: Fully tested, pass.
- USB: No
- IRDA:
- SIR: Fully tested, pass.
- FIR: Fully tested, pass.
- PM: Not tested.

Neponset:

Serial ports:

- COM1,2: TX, RX, CTS, DSR, DCD, RTS, DTR
- PM: Not tested.
- USB: Implemented, not heavily tested.
- PCMCIA: Implemented, not heavily tested.
- CF: Implemented, not heavily tested.
- PM: Not tested.

More stuff can be found in the -np (Nicolas Pitre's) tree.