TS ARM Linux Developer's Manual

Technologic SYSTEMS

Technologic SYSTEMS

Technologic Systems, Incorporated

16610 East Laser Drive, Suite 10
Fountain Hills, AZ 85268

480-837-5200
FAX 837-5300

info@embeddedArm.com

http://www.embeddedARM.com/

This revision of the manual is dated

September 15, 2004

All modifications from previous versions are listed in the appendix.

Table Of Contents

Introduction

This manual is a brief introduction to the use of the Linux on a Technologic Systems' ARM-based Single Board Computer (SBC). Many technical questions are answered within this document This manual is not meant as a tutorial to Linux (or Linux Development). Technologic Systems' ARM products offer a different set of tools when working with them as compared to a Technologic Systems' x86 products. This manual documents the basic knowledge needed to work with the ARM products. Currently, this document addresses the TS-7200 ARM board.

If there is a need for a quicker time to market solution, Technologic Systems does offer software engineering services. If one needs modified hardware, Technologic Systems may be able to accommodate your needs. Please contact Technologic Systems for more information.

Startup

Booting

Upon power up, the board executes proprietary Technologic Systems boot-code, then immediately executes RedBoot. RedBoot is a feature rich boot-ROM monitor, that allows manipulation of the on-board flash, JFFS2 images, loading and execution of a kernel or executable from either tftp (trivial ftp) or flash, and gdb debugging stubs. From RedBoot, one can load and execute any standalone binary. Most commonly, a Linux kernel or a Windows CE binary is used. One can also write applications within the Ecos environment and load them with RedBoot.

RedBoot

By default, a pre-existing RedBoot script is executed if not interrupted by the user within one second. A Linux kernel is loaded into memory from flash, booting into the pre-existing JFFS2 file-system. One can view the RedBoot defaults for the board, as well as the default script, by entering at the RedBoot command prompt:

```
fconfig -l
```

The defaults can be changed by simply entering

```
fconfig
```

at the RedBoot prompt and answering the prompts. A final chance to write or discard the changes to the board will be given by RedBoot.

The default script instructs RedBoot to load the Linux kernel from the flash, and instruct the Linux kernel to use the JFFS2 image on the flash chip for its root file-system. The Linux **kernel must be loaded into memory address 0x00218000**. Loading the kernel from flash is done automatically by RedBoot in the default script with the following command:

```
fis load zimage
```

After loading the kernel, the default script then executes the kernel with the following command:

```
 exec -c "console=ttyAM0,115200 root=/dev/mtdblock1”
```

Loading and executing Kernels from RedBoot

RedBoot can load a kernel or executable via the serial console, a tftp server, or directly from flash. The default boot-script loads the kernel from flash into memory by executing

```
 fis load zimage -b 0x00218000
```

One can see where in flash the kernel is and where in memory it will be loaded to by typing

```
 fis list
```

which shows the various areas of flash RedBoot is aware of. The following is a typical output of *fis list*

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
(reserved)        0x60000000  0x60000000  0x00620000  0x00000000
RedBoot           0x60620000  0x60620000  0x00040000  0x00000000
RedBoot config    0x607C0000  0x607C0000  0x00001000  0x00000000
FIS directory     0x607E0000  0x607E0000  0x00020000  0x00000000
zimage            0x60660000  0x00218000  0x000C0000  0x00218000
```

From the above example, the kernel executable labeled zimage is stored at flash address 0x60660000, and will be loaded into memory address 0x00218000.

To load a kernel from a tftp server, the following command is needed

```
 load -r -b 0x00218000 -h <tftp server IP>  <kernel
name>
```

At this point, it is possible to write the kernel now in memory to flash, thereby overwriting the pre-existing kernel stored in flash. First, one must delete the existing zImage file from flash, then write the new one to disk. The following commands accomplish this:

```
 fis delete zimage
 fis create zimage
```

Note that the *delete* command specified “zimage”, which refers to the name of the fis entry to delete. The *create* command specified “zimage”, which is the name of the executable loaded into memory. If the name of the downloaded executable is different, please specify that name when doing a *create* command.

Now that a kernel has been loaded into memory, it can be executed. This is accomplished with the following command:

```
 exec -c “console=ttyAM0,115200 ip=dhcp
root=/dev/mtdblock1”
```

the *exec* command executes the loaded kernel image, passing to the kernel the arguments specified via the *-c* switch. In the previous example, kernel messages are sent out on the first serial port (note that ttyAM0 is used instead of the familiar ttyS0) at 115200 baud and the root file-system is on the first mtdblock of the flash chip.

If one was to load the root-filesystem from an nfs server, the following command would suffice:

```
  exec -c “console=ttyAM0,115200 ip=dhcp nfsroot=<ip of
nfs server>:/path/to/nfsroot
```

To load the root file-system from the Compact Flash card, the following command should be used instead:

```
  exec -c “console=ttyAM0,115200 ip=dhcp root=/dev/hda1”
```

Updating Root File-Systems

It may please you to know that updating the Compact Flash file-system is Debian. Adding new packages, removing undesired ones, is done all through Debian's package management. *apt*, *dpkg*, all behave as expected. A full in-depth discussion on Debian is outside the scope of this document. Please visit Debian's web-site (http://www.debian.org) for more documentation on using apt, dpkg, or other debian related questions not answered in this document.

The on-board flash contains a custom-made JFFS2 image. JFFS2 is a compressed, **J**ournaling **F**lash **F**ile System.

Setting up the Network (Debian on CF)

To configure the network interfaces when booting into Debian on the Compact Flash card, edit the file /etc/network/interfaces. A typical *interfaces* file would contain the following:

```
 # /etc/network/interfaces -- configuration file for ifup(8),
ifdown(8)

 # The loopback interface
 auto lo eth0
 iface lo inet loopback

 # The first network card
 auto eth0
 #iface eth0 inet dhcp
 iface eth0 inet static
 address 192.168.1.1
 netmask 255.255.255.0
 gateway 192.168.1.2
```

Those lines starting with a # symbol are comments. The line *auto lo eth0* means both the loopback interface and the first ethernet interface will be started automatically by the Debian networking scripts. The above example shows that eth0 would be assigned the static address of 192.168.1, using 192.168.1.2 as the default gateway. If one was to comment out those lines, and then uncomment the line *iface eth0 inet dhcp*, then eth0 would use a dhcp client to obtain it's IP and other relevant network information.

Setting up the Network (Flash File-System)

To configure the network when booting to the JFFS2 image on the flash chip, the files in /etc/sysconfig/ must be edited. Network interfaces are configured on a file per interface basis. The first Ethernet device, eth0, is controlled by the file /etc/sysconfig/ifcfg-eth0. An example of ifcfg-eth0 is shown below

```
$ cat ifcfg-eth0
DEVICE=eth0
IPADDR=192.168.0.50
```

```
NETMASK=255.255.255.0
NETWORK=192.168.0.0
BROADCAST=192.168.0.255
#BOOTPROTO=dhcp
BOOTPROTO=static
ENABLE=yes
```

Those lines starting with a # symbol are comments. As the above example shows, eth0 is given the static address of 192.168.0.50. If one wishes eth0 to obtain its IP from a DHCP server, then change the line

```
BOOTPROTO=static
```

to

```
 BOOTPROTO=dhcp
```

Updating the JFFS2 Image

The JFFS2 image can be created on your host computer, with binaries created via a cross compiler, then placed in a directory structure on the host computer. A new JFFS2 image can be constructed from that directory structure with the following command:

```
mkfs.jffs2 -p6291456 -e131072 -o /tmp/my_jff2.img
```

Pre-made JFFS2 images can be found in the Developer's CD or on Technologic Systems' web-site

The JFFS2 image file can then be copied over to the Single Board Computer and then written to the flash chip by using the following command:

```
dd if=my_jffs2.img of=/dev/mtdblock/1
```

Mounting Other Devices

Occasionally, one may wish to boot into one device and access the other. This requires knowing where those other devices are. The JFFS image on the flash is stored on */dev/mtdblock/1*. To mount it, the following will suffice:

```
mount /dev/mtdblock/1 /mnt
```

The Compact Flash card is seen as the Primary Master IDE device, otherwise known as */dev/hda*. The following command demonstrates mounting the Compact Flash card to the currently running system:

```
mount /dedv/hda1 /mnt
```

Mounting nfs roots requires that the *portmap* daemon is running, before executing the mount command. The following example demonstrates mounting an NFS file-system hosted on a server at 192.168.1.3

```
portmap &
mount -t nfs 192.168.1.3:/path/to/nfsroot /mnt
```

Development Kit

The standard developer's kit includes a CD with a patched Linux source tree, documentation, and cross tool-chains. The developer's kit also includes a Compact Flash card pre-installed with Debian stable.

Compact Flash Card

The developer's kit comes with a CF card pre-installed with Debian. Installed on the Debian CF card are common utilities such as Samba, apache, perl, open-ssh, and native arm compiler tools, such as gcc.

With Debian, one can easily install and remove software packages. It is recommended that one visits the Debian home-page at http://www.debian.org for further information. For a quick demonstration of how easy it is to remove and install programs with Debian, try the following commands:

```
apt-get install hexedit
hexedit /etc/passwd
^C (hit CTRL+C to safely exit)
apt-get remove hexedit
```

apt-get install installs a package name, while apt-get remove removes the named package. For further support on Debian, please visit the Debian home-page.

To create a CF card from scratch, one must format the entire CF card as ext2, then unpack the Development file system for NFS root or Compact Flash. Latest versions of our pre-made Debian install can be found on our website at http://www.embeddedx86.com/linux/ARM.htm. This should all be done from the host PC running Linux. The walk-through below assumes the CF card is plugged into the USB dongle and the CF card has been assigned to /dev/sda:

```
fdisk /dev/ide/host0/bus0/target0/lun0/disc
```

1. d (to delete existing partitions. Repeat for all partitions)
2. n (for new partition)
3. p (for Primary partition)
4. 1 (make the new partition primary number 1)
5. hit the enter key for the default starting cylinder
6. hit the enter key again for the default last cylinder
7. p (to print out the partition table)
8. If the first partiton does not have a star in the *boot* field, then enter 'a' at the prompt and then '1' to make the first partition bootable.
9. To commit these changes to the disk, enter 'w' to write out the new partition table to the disk.

Now that the CF card has been partitioned, it must be formatted. The following command will format the first partition on the CF card as an ext2 file-system.

```
mkfs.ext2 /dev/sda1
```

All that is left is to mount the CF card and unpack the tar file of the Development Debian File System.

```
mount /dev/sda1 /mnt/misc
tar -C /mt/misc -xvjf  debian256-8-25-2004.tar.bz2
```

The filename may change as updates are made and posted on the CD or on the web-site.

A simple fsck will ensure file-system integrity.

```
fsck /dev/sda1
```

Cross tool chains

While the Debian Compact Flash card includes a suite of compiler tools, including a native arm gcc c compiler, one may wish to use their desktop PC for compiling and development. The recommended cross-compiler is Dan Kegel's CrossTool (http://kegel.com/crosstool/). Simply download the latest version, and run the demo-arm.sh script for an arm cross compiler that is suitable for your system. CrossTool 0.28 or greater now supports cygwin. Technologic Systems does provide pre-made versions of crosstool for both Linux and cygwin. They can be found on the Arm Development CD or on the Technologic Systems' web-site. To install the Linux binaries, unpack the tar file at the root of your system as the root user. To install the Cygwin binaries, unpack the tar file at the root of your Cygwin environment. Be sure to include in the PATH environment variable the path of the crosstool binaries. For example, the pre-made Linux crosstool binaries are located at */usr/local/opt/crosstool/arm-linux/gcc-3.3.2-glibc-2.3.2/bin/*

Linux Kernels

Technologic Systems offers kernel patches for those kernels that are shipped with the board. Please visit our website (http://www.embeddedarm.com) to download the appropriate patch.

Once downloaded, the patch can then be applied to a matching, unpacked Linux kernel. In order to compile the kernel from a non ARM host machine, a cross compiler is needed. Technologic Systems offers some pre-compiled cross compilers, such as Dan Keegel's CrossTool.

To compile the kernel, simply edit the top level Makefile and ensure that the CROSS_COMPILER variable is equal to your system's cross compiler prefix to gcc. Then type

```
make ts7200_defconfig
make oldconfig
make dep
make vmlinux
make modules
```

in order to build a kernel as shipped by Technologic Systems. The resulting vmlinux file can be placed onto the flash chip as described before.

Recommended Readings

For those who are new to Linux, it will be beneficial to own and work within a full-features desktop Linux OS. Commonly, those new to Linux tend to use distributions such as Redhat's Fedora, Mandrake Linux, or Suse. For those who are unwilling to install a Linux OS onto their system, Knoppix is recommended. Knoppix (www.knoppix.org) is a live-cd that boots into Linux, and while offering a feature rich Linux OS, it won't touch the hard-drive unless told to.

For those with are familiar with Linux but not familiar with programming in Linux, the following web-sites and books are excellent resources

www.ibm.com/developerworks
full of tutorials and articles for Linux programming. For example, they have an excellent set of tutorials on programming threads in Linux

www.kernelnewbies.org
Good starting point for those who are new to Linux kernel development.

www.montavista.com
Monta Vista is a global leader in embedded Linux development. They offer their own Operating System, as well as their set universal development tools (including IDEs). Monta Vista tools are platform independent, and work well for a Windows only environment.

Linux Device Drivers, 2nd Edition
By Alessandro Rubini & Jonathan Corbet
Highly recommended book for those who wish to learn the fundamentals to device driver development in Linux

Appendix A: Cygwin Hello World walkthrough

This is a quick demonstration on using Windows and Cygwin to create your own hello world application using Linux tools. It is assumed that either the Cygwin crosstool tar file has been unpacked to the root of the Cygwin environment or has been built from scratch. Naturally, it is assumed that Cygwin has been installed on your Windows PC.

1) Create hello-world.c

   This can be done with any editor, either using Vim from within Cygwin, or using notepad from within Windows

```
include <stdio.h>
int main (void)
{
        printf (“Hello World!\n”);
        return 0;
}
```

2) Ensure that the CrossTool binaries are in your systems path

```
export PATH=$PATH:/opt/crosstool/arm-unknown-
linux-gnu/gcc-3.3.2-glibc-2.3.2/bin
```

3) Create the hello executable

```
arm-unknown-linux-gcc -Wall -o hello helloworld.c
```

4) Verify that the resulting hello binary is an ARM binary file.

```
file hello
```

5) Run inetd on the SBC

```
inetd
```

6) Copy over the hello binary to the SBC via ftp. Log in with user root with no password

   ftp 192.168.0.50 (be sure to use the IP that matches your setup)
   Login
   enter “binary” to ensure binary transfers
   enter “put hello”

7) **On the SBC**, change the permissions of the hello binary to become executable

```
cd /root
chmod +x ./hello
```

8) Run the binary

```
./hello
```