



DE NAYER Instituut
Onderzoeksgroep Digitale Technieken
J. De Nayerlaan 5
B-2860 Sint-Katelijne-Waver
Tel. (015) 31 69 44
Fax. (015) 31 74 53
Email: ppe@denayer.wenk.be
gvd@denayer.wenk.be
bvd@denayer.wenk.be
Web: <http://emsys.denayer.wenk.be>

*Application Note: Installing Embedded Linux on the
Motorola MPC5200 Lite Evaluation Board*

Version 1.2 rev. 27/04/2004

**HOBU-fund
research project 030106**

Title : Multifunctional embedded digital camera system
Project manager : ing. Patrick Pelgrims
Project engineers : ing. Gerrit Van de Velde
ing. Björn Van de Vondel

Contents:

- 1. Introduction**
 - a. MPC5200
 - b. Icecube
 - c. Scope and Background Information
 - d. Credits
 - e. Short License Info
 - f. Full License Text
- 2. Host setup (Workstation)**
 - a. ELDK
 - b. TFTP
 - c. NFS
 - d. Minicom
- 3. U-Boot**
 - a. Running U-Boot From RAM
 - b. Running U-Boot From Flash
- 4. Linux Kernel**
 - a. Default
 - b. Modified Flash Partitioning
- 5. Linux From Scratch**
 - a. Basic Root File System
 - b. Libraries
 - c. Device Files
 - d. System Applications
 - e. System Initialization
- 6. Installing a Root Flash File System**
- 7. References**

1. Introduction

1.a MPC5200

- MPC603e series processor core that implements the PowerPC™ architecture
- Double precision FPU
- Instruction & Data MMU
- 16K instruction and data caches
- DMA
- SDR and DDR memory interface
- PCI v2.2
- ATA/IDE interface
- USB 1.1 Host
- I2C, I2S, CAN, GPIO, 8 timers, SPI, ...

1.b “Icecube”

Motorola offers two different evaluation and development packages for the MPC5200 processor. The Lite5200 Evaluation Board (EVB), nicknamed “icecube” is a classic, stand-alone evaluation board, and the Total5200 Development Platform is a complete, self-contained system, which includes graphics and audio support and is suitable for system prototyping with little or no additional hardware.

This document focuses on the LITE5200 version 2.0 evaluation board. With this document you’ll be able to populate the onboard 16Mb flash chip with a boot loader a Linux kernel and a root file system on NFS or flash. All this can be achieved without the use of expensive third party programs. The only things that you need are a workstation with ethernet and a serial communications port running Linux.

1.c Scope and Background Information

Originally, the sole purpose of this document was to keep track of how Linux could be installed on the embedded platform. Later on, this document was adapted and improved in order to make it available to the public and the members of our research project.

To understand this document completely, you should already have at least some experience in the following fields:

- Embedded Systems
- Linux
- C programming
- GNU software development tools

Also, reading the following documents can prove to be very helpful. In fact, the rest of the text is based on them.

- “Building Embedded Linux Systems” - Karim Yaghmour (O’Reilly)
- “Denx U-Boot and Linux Guide” - Wolfgang Denk (<http://www.denx.de>)
- “The Linux MTD, JFFS HOWTO” - Vipin Malik (vipin@embeddedlinuxworks.com)

This document will NOT provide any information about bringing up custom boards and NOT about supporting immature hardware or adding support for unsupported hardware.

1.d Credits

This document could not be written without the help of the U-Boot and Embedded PowerPC community and mailing lists. Thanks to anybody who contributed in any way.

1.e Short License Info

Copyright (c) 2004 by Gerrit Van de Velde, Björn Van de Vondel and Patrick Pelgrims. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>). Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.

1.f Full License Text

Open Publication License

v1.0, 8 June 1999

I. REQUIREMENTS ON BOTH UNMODIFIED AND MODIFIED VERSIONS

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

Copyright (c) <year> by <author's name or designee>. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, vX.Y or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

The reference must be immediately followed with any options elected by the author(s) and/or publisher of the document (see section VI).

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

II. COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

III. SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

IV. REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements: The modified version must be labeled as such.

The person making the modifications must be identified and the modifications dated. Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.

The location of the original unmodified document must be identified.

The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

V. GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.

All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.

Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

VI. LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

A. To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

To accomplish this, add the phrase 'Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

B. To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

2. Host setup (Workstation)

In order to be able to follow the instructions in this document, you must have a Linux workstation and have root access to it. In our project we used a Redhat Linux 9.0 (Shrike) distribution but any other distribution should work too. When installing the Linux, be sure to add the NFS and TFTP packages or install them separately to be able to use a network filesystem. Using NFS will help to shorten your development stage.

Denx has released an embedded Linux development kit (ELDK) which holds a set of predefined toolchains to compile a kernel and user programs. It supports PowerPC, ARM and MIPS architectures for your embedded targets and PC/Linux or SPARC/Solaris host environments. We'll use this ELDK as a starting point for this documentation since it was also the starting point for us during the project.

2.a ELDK

To install the ELDK, follow the instructions explained in Denx U-Boot and Linux Guide (DULG):

Make a directory to which you will install the ELDK. I will use <ELDK-dir> to point to that directory from this point in the document.

```
$ mkdir <ELDK-dir>
$ ./<ELDK-installmedia>/install -d <ELDK-dir>
(This will install all possible targets and their toolchains.)

$ su -l root
<enter the super user password>
$ cd /<ELDK-dir>/ppc_82xx/dev
$ /<ELDK-installmedia>/ELDK_MAKEDEV
(This will make all device entries in the root filesystems of the targets)

$ cd /<ELDK-dir>/
$ /mnt/cdrom/ELDK_FIXOWNER
(This will fix the settings of certain files in the directory structure)

$ export CROSS_COMPILE=ppc_82xx-
$ PATH=$PATH:/<ELDK-dir>/usr/bin:/<ELDK-dir>/bin
(these settings can be copied to /home/<user>/.bashrc and saved in this file to avoid typing this all over again after each reboot - after this you can compile your own files with "${CROSS_COMPILE}gcc -o helloworld helloworld.c")
```

2.b TFTP

Download the TFTP server binaries and install them. You can check whether your workstation has them installed by typing

```
$ rpm -q tftp
```

After installation, edit the settings of the TFTP server:

```
$ vi /etc/xinetd.d/tftp
```

Find the line with “disable = yes” and put it into comment with a # character at the beginning of the line. Save the file. This will enable the server by default.

Then make a directory which will hold all TFTP files we'll use on our target platform.

```
$ mkdir /tftpboot/  
$ mkdir /tftpboot/MPC5200  
$ chmod 555 /tftpboot/  
(To make it world readable)
```

2.c NFS

Check whether the NFS packages are already installed with the command:

```
$ rpm -q nfs
```

If they're not already installed, install them and afterwards, edit the settings needed to work with networked filesystems as root.

```
$ su root  
<enter the super user password>  
$ vi /etc/exports
```

Add the following entry:

```
/dir_naar_eldk/ppc_82xx subnet.of.the.target/255.0.0.0(rw,no_root_squash)
```

Save the file and exit vi.

Also edit the hosts.allow file otherwise your workstation will block all traffic with the embedded platform.

```
$ vi /etc/hosts.allow
```

Add the following line:

```
ALL: ip.address.of.target
```

Save and exit vi.

Restart the NFS service after this. (Still as root)

```
$ /sbin/service nfs restart
```

2.d Minicom

As terminal application on the workstation, we'll be using minicom. To start with the Icecube board out of the box, set the configuration of the terminal with a baudrate of 9600, 8N1 databits and hardware flow control on.

At this point in the document, you should be able to communicate with the Icecube board when it comes right out of the box. Attach the power cable and UART serial connector and open a minicom terminal.

```
$ minicom /dev/ttyS0
```

Press reset on the board and you should see some output of the Motorola standard monitor program, dBUG.

Adjust some settings:

```
DEBUG> set baud 115200
```

Now you'll have to change the minicom settings in order to be able to store the new setting. Set baudrate to 115200 instead of the previously configured 9600 and reinitiate communications.

```
DBUG> store
```

3. U-boot

DBUG is not powerful enough to help us building a fully self hosting embedded system. Therefore we'll install a more flexible and open source boot loader, called U-Boot. Universal Boot loader has support for different CPU architectures (MPC5xx, 82xx, 7xx, 74xx, 4xx; ARM7, 9, StrongARM and Xscale; MIPS 4Kc, 5Kc; x86; ..) and is released under GPL. This boot loader can be programmed into flash in another place than the original Motorola monitor program. In this way we can preserve the working monitor in case something goes wrong. Motorola provides a jumper on the Icecube board (J73-74-75) to switch from one configuration to another. They're called High Boot and Low Boot, according to the address it refers to. High boot (dBUG) starts from flash address 0xFFFF0000 and lowboot starts from address 0xFF000000. Don't switch from one configuration to another without powering down the board first.

U-Boot is a very active project at the time of writing, so we suggest to subscribe to the mailing list if you are planning to make use of this boot loader. Also, get the latest code via CVS since a lot of (important) patches are included every now and then.

Get the latest U-Boot source code first:

```
$ cd <ELDK-dir>/usr/src
$ cvs -d:pserver:anonymous@cvs.u-boot.sourceforge.net:/cvsroot/u-boot login
<press enter when promoted for a password>
$ cvs -z6 -d:pserver:anonymous@cvs.u-boot.sourceforge.net:/cvsroot/u-boot \
  co -c
(To see which modules are available)
$ cvs -z6 -d:pserver:anonymous@cvs.u-boot.sourceforge.net:/cvsroot/u-boot \
  co -P uboot
$ cd uboot
```

3.a Running U-Boot From RAM

If you do not want to burn U-Boot into flash right from the start, you can always run the U-Boot from RAM. You must download a S-Record file of U-Boot into RAM and start it from there. Place the jumper on the board to high boot configuration.

```
$ make distclean
$ vi <ELDK-dir>/usr/src/uboot/Makefile
(find address 0xFF000000 in the LOWBOOT section of MPC5200 and change them into 0x00020000)
$ make MPC5200LITE_LOWBOOT_config
$ make
$ cp u-boot.srec /tftpboot/MPC5200/
```

```
dBUG> dn -s MPC5200/uboot.srec
dBUG> go 20100
```

This will start U-Boot from RAM location 0x00020100

3.b Running U-Boot From Flash

If you instead want to burn the U-Boot boot loader into flash after you verified whether it's working on your board, you can follow these steps:

```
$ make distclean
$ make MPC5200LITE_LOWBOOT_config
(with the default address of 0xFF000000)
$ make
$ cp u-boot.bin /tftpboot/MPC5200/u-boot-lowboot.bin
```

Fire up a minicom terminal window and power up the Icecube board and start in high boot configuration (dBUG Monitor).

```
dBUG> set mac 00:01:AF:52:01:xx
(replace xx with the right code, see Icecube manual)
dBUG> set client ip.address.of.board
dBUG> set server ip.address.of.workstation
dBUG> set netmask 255.0.0.0
dBUG> dn -i -o 0x100000 /tftpboot/MPC5200/u-boot-lowboot.bin
dBUG> fe 0xFF000000 0xFF050000
dBUG> fp 0xFF000000 0xFF040000 0x100000
(this will program the lowboot address space of the flash chip with the U-Boot loader)
```

Now power off the board and change the appropriate jumper setting to low boot instead of high boot. Power up the board again and you should see U-Boot booting in the minicom terminal.

Now we're going to set the U-Boot environment variables to the right values so we can load the kernel. We're going to define as less variables as possible and still be able to boot in four different configurations.

- kernel via TFTP, root file system over NFS (most flexible, used in the early development stages)
- kernel in flash, root file system over NFS (when the kernel is proven to be stable)
- kernel via TFTP, root file system in flash (irrelevant and useless)
- kernel in flash, root file system in flash (end stage)

Unset all existing variables:

```
=> setenv bootcmd
=> setenv bootdelay
=> setenv preboot
=> setenv netdev
=> setenv nfsargs
=> setenv ramargs
=> setenv addip
=> setenv flash_nfs
=> setenv flash_self
=> setenv net_nfs
=> setenv rootpath
=> setenv bootfile
```

Set our own variables and save them:

```
=> setenv bootcmd tftp\; bootm
(to start the kernel from tftp)
=> setenv preboot echo\;echo Autostarting. Press any key to abort..\;echo
=> setenv serverip ip.address.of.workstation
```

```

=> setenv ipaddr ip.address.of.target
=> setenv netmask 255.0.0.0
=> setenv ethaddr 00:01:AF:52:01:xx
(replace xx with the right mac address - see MPC5200 manual to find out how to find it)
=> setenv bootdelay 5
=> setenv hostname icecube
=> setenv nfsroot root=/dev/nfs rw
nfsroot=192.168.0.10:/home/gvd/denx/rootfs
=> setenv ip
ip=$(ipaddr):$(serverip):$(serverip):$(netmask):$(hostname)::off
=> setenv flashroot root=/dev/mtdblock2 rw
=> setenv rootfs $(nfsroot)
=> setenv bootfile MPC5200/uImage
=> setenv bootargs $(rootfs) $(ip)
=> saveenv

```

Now we've set up the boot loader. We can start building the linux kernel now and try to boot it.

4. Linux Kernel

We'll use U-Boots built in capability to transport files with TFTP. We'll build the Linux kernel with the ELDK environment on our workstation.

Download latest source tree:

```

$ cd <ELDK-dir>/ppc_82xx/usr/src/
$ cvs -d :pserver:anonymous@www.denx.de:/cvsroot login
<press return when prompted for a password>
$ cvs -z6 -d :pserver:anonymous@www.denx.de:/cvsroot co -P
linuxppc_2_4_devel

```

4.a Default

```

$ cd <ELDKDIR>/ppc_82xx/usr/src/linuxppc_2_4_devel
$ make mrproper
$ make menuconfig
<Load config> : arch/ppc/configs/icecube_5200_defconfig
exit and save
$ make uImage
$ cp arch/ppc/images/boot/uImage /tftboot/MPC5200/uImage
$ mkimage -n 'Linux PPC MPC5200 2.4' -A ppc -O linux -T kernel -C gzip -a
00000000 -e 00000000 -d arch/ppc/boot/images/vmlinux.gz
/tftboot/MPC5200/vmlinux.img

```

The uImage will be used when we start the kernel over TFTP. Otherwise, if we want to use a kernel image that is located in the flash, an image must be created.

The last command is needed to use the compiled kernel image in flash. U-Boot requires headers to start the kernel instead of raw data in the flash. Start and entry addresses are pointing to 0x00000000, this is not an error and is required although the image will be stored in another physical address space (0xFF.....).

To start the kernel, it is enough to fire up the board, have it started in U-Boot with our stored environment variables and let it autoboot. When the countdown reaches zero, the "bootcmd" will be run, which is

```

=> tftp
(This will download the bootfile - MPC5200/uImage to the default ram address 10000)

```

```
=> bootm
(This will boot an image, located at default address 100000)
```

If you tried this and know that the kernel which was compiled just now is working well, you can download the image into flash. If changes to the kernel have to be made, it is required to erase that flash part and rewrite it with a new kernel image.

To download the image of the kernel into flash, issue the following commands in U-Boot.

```
=> tftpboot 100000 MPC5200/vmlinux.img
=> erase FF100000 FF1FFFFFFF
=> cp.b 100000 FF100000 $(filesize)
=> bootm FF100000
```

You can change the U-Boot environment variables as you like, to:

```
=> setenv bootcmd bootm FF100000
=> saveenv
=> reset
```

This will reset the board and automatically start the kernel from flash. The bootargs environment variable are arguments that will be passed to the kernel command line. If the root part points to a valid NFS directory on the host, and the host is configured to accept network accesses to that directory, the Linux will start on the development board with a root file system that is physically located on the workstation. This way, you are able to change the need files, scripts, user programs etc. on the fly. When you feel the root file system is ready to move to the onboard flash RAM, it can be packed in a JFFS, CRAMFS (not supported by the default MPC5200Lite config) or JFFS2 image. The image can then be stored into flash with the Linux command dd or cat. More on this topic is to be found later on in this document.

4.b Modified Flash Partitioning

By default, the flash partitioning of the detected flash chips is a bit confusing. Certainly if you are using U-Boot from LOWBOOT config in flash. The default flash memory map defines user "spare" space which isn't spare at all because U-Boot is residing at that location. To edit the flash partitioning, issue the next commands:

```
$ cd <ELDK-dir>/ppc_82xx/usr/src/linuxppc_2_4_mpc5200/drivers/mtd/maps/
$ vi icecube.c
(find static struct mtd_partition icecube_partitions_16M[] and alter it for example to...)
static struct mtd_partition icecube_partitions_16M[] = {
    {
        name:    "Das U-Boot",
                /* U-Boot Bootloader      */
        offset: 0,
        size:    1 << 20,
                /* 1 Megabyte             */
    },
    {
        name:    "linuxppc_2_4_mpc5200 kernel", /* Linux kernel image */
        offset: 1 << 20,
        size:    1 << 20,
                /* 1 Megabyte             */
    },
    {
        name:    "Root Filesystem (JFFS)",
                /* Root File System      */
        offset: 2 << 20,
        size:    13 << 20,
                /* 13 Megabyte            */
    },
    {
```

```

        name:    "dBUG Motorola Firmware",        /* dBUG Motorola        */
        offset: 15 << 20,
        size:    1 << 20,                        /* 1 Megabyte          */
    }
};
(save and exit vi)

```

After this modification you can compile the kernel as you would do normally. The resulting operating system can access the different flash partitions as we expect it and will not overwrite our own boot loader if we write the image with the root file system to flash.

5. Linux from scratch

Most of the techniques described in this part are taken from Karim Yaghmour's "Building Embedded Linux Systems". Only the most important parts that we needed to develop our own system are taken into account here.

5.a Basic Root File System

The main directory structure of the root file system is standardised in the FHS (Filesystem Hierarchy Standard). We'll follow this standard as close as possible. Except from the fact that we don't need multi user environment on an embedded system, so no /home/user directories are included in the structure.

Make the directories:

```

$ cd <ELDK-dir>
$ mkdir rootfs
$ cd mkdir
$ mkdir bin dev etc lib proc sbin tmp usr var
$ chmod 1777 tmp
(sticky bit is set to guarantee that only the owners of files in this directory can delete the files)
$ cd usr
$ mkdir bin lib sbin
$ cd ../var
$ mkdir lib lock log run tmp
$ chmod 1777 tmp
$ cd ..

```

5.b Libraries

Libraries need to be installed on the embedded target to develop applications. Basically, you have a choice between some library packages. The standard glibc library the dietlibc library which is optimised for size and the uClibc library. We'll use the first set because that's the most complete library and size is not really an issue since the board has 16Mb flash. That's more than enough to hold a full featured library set.

We'll copy the libraries in the <ELDK-dir>/ppc_82xx/lib to our own file system:

```

$ cd <ELDK-dir>/ppc_82xx/lib
$ for file in libc libcrypt libdl libm libpthread libresolv libutil
> do
> cp $file-*.so <ELDK-dir>/rootfs/lib
> cp -d $file.so.[*0-9] <ELDK-dir>/rootfs/lib
> done
$ cp -d ld*.so* <ELDK-dir>/rootfs/lib

```

```
$ ppc_82xx-strip <ELDK-dir>/rootfs/lib*.so
```

The last command will strip the libraries to further reduce size of the installed libraries.

5.c Device Files

All device nodes should be created in the file system. We can use ELDK's MAKEDEV script to do so.

```
$ cd <ELDK-dir>/rootfs/  
$ /<ELDK-installmedia>/ELDK_MAKEDEV
```

Alternatively, we can also make our own device nodes and leave out the ones that are not needed in our embedded system. We can issue a shell script that checks for the existence of the required devices at startup and if they don't exist, we can tell Linux to create them with the right mknod commands. The MAKEDEV is easier to use since you don't need to look up the right major and minor numbers of each device. Here's an example of the manual solution:

```
# !/bin/sh  
  
echo "Making Device Entries";  
if test -f /dev/mtd0  
then mknod /dev/mtd0 c 90 0  
fi  
if test -f /dev/mtd1  
then mknod /dev/mtd1 c 90 2  
fi  
chmod 664 /dev/mtd*  
  
<snip>
```

5.d System Applications

In order to have some of the most important commands that come with workstation Linux environments on the embedded system, one can choose to cross compile all different programs like ls, mount, etc. This would take an awful lot of time and a better solution is at hand. Some people saw the need of a package that combines many of these commands into one binary. In this project we used Busybox (<http://www.busybox.net>) and Tinylogin (<http://tinylogin.busybox.net>).

The first one combines most needed commands. The second one has functionality to make users, add groups, change passwords etc.

First we build Busybox. Download latest stable busybox version off the internet and unpack it.

```
$ cd <ELDK-dir>/sysapps/busybox-(version)/  
$ vi Config.h  
(comment out all features you want to include in the busybox binary. Some important features like ifconfig, ping, insmod are disabled by default)  
$ make TARGET_ARCH=ppc CROSS=ppc_82xx- PREFIX=<ELDK-dir>/rootfs all install
```

After this we build Tinylogin. Download the latest stable version and unpack it.

```
$ cd <ELDK-dir>/sysapps/tinylogin-(version)/  
$ vi Config.h
```

```
(comment out all features you want to include in the tinylogin binary. You can choose to use
shadow passwords here - which is more secure)
$ make CROSS=ppc_82xx- PREFIX=<ELDK-dir>/rootfs all
$ su -m
<enter super user password>
$ make PREFIX=<ELDK-dir>/rootfs install
$ exit
```

Tinylogin and Busybox are now ready to be used. Only two binaries are created in the /usr/bin directory. All commands are referenced through symbolic links to one of those two binaries. In order to login to the embedded system, some files should be generated or copied from the workstation file system to the target file system.

```
$ cp /etc/group <ELDK-dir>/rootfs/etc
$ cp /etc/passwd <ELDK-dir>/rootfs/etc
$ cp /etc/shadow <ELDK-dir>/rootfs/etc
```

Some files might need some editing, since you don't want to use your own password for the workstation on the embedded system as well. And you have to change the initial shell to sh. Busybox has a Ash shell, not a Bash like on most workstations. The simplest method to create a new user and password for your embedded system is to create a dummy user on your workstation and copy the password strings it generated in /etc/passwd and /etc/shadow to the ones on your root file system of the target platform.

Additionally, one could also install the embutils package (<http://www.fefe.de/embutils/>) but we did not do this (until now?). Remark: this requires a diet libc library on your host system to build it. You can also include your own applications at this stage. Just cross compile them and place them in an appropriate directory in the root file system of the target embedded platform. Some useful applications on about any networked embedded system are for instance a SSH daemon, a webserver (Boa), an FTP server, NTP to synchronize the time with remote systems, and so on.

5.e System Initialization

On workstations, a full featured System V init is included but on embedded targets that's often overkill for what we need. Therefore, we used the Busybox init system. The init system requires some files and settings during startup.

```
$ cd <ELDK-dir>/rootfs/etc

$ vi inittab
(inittab content:)
::sysinit:/etc/init.d/rcS
::respawn:/sbin/getty 115200 ttyS0
::restart:/sbin/init
::shutdown:/bin/umount -a -r
(save and exit vi)

$ mkdir init.d
$ vi init.d/rcS
(rcS content:)
mount -a

$ vi fstab
(fstab content:)
# /etc/fstab
```

```
# device          directory  type  options
#
none             /proc     proc  defaults
(save and exit vi)
```

6. Installing a Root Flash File System

In order to make a flash file system, we must have a working set of MTD (Memory Technology Device) utilities on our workstation. To download the latest source code with CVS, go to the directory where you want the code and issue the next commands:

```
$ cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs login
(password: anoncvs)
$ cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs co mtd
```

After download go to the utils directory and compile the set of utilities.

```
$ cd mtd/utils
$ make
```

After compilation, you'll find a set of binaries. Particularly the mkfs.jffs and mkfs.jffs2 are important to us.

To make valid image files of the file system we want to use in our flash, issue next commands as root:

```
$ <path to>/mkfs.jffs -d rootfs/ -b -o images/rootfs-jffs-bigendian.img
$ <path to>/mkfs.jffs2 -d rootfs -a b -o images/rootfs-jffs2-bigendian.img
```

We've created two different file system type images with big endian storage. If you accidentally use the wrong bit ordering, you'll notice a bunch of "magic number" or "bit mask" errors while mounting the image from flash if you're using the JFFS2 image. For the JFFS image, you can mount the flash partition, but you won't find any files on the mount... If you encounter this, don't panic as it's probably only the ordering that must be changed. Depending on which file system type you want to use, move one of the two images into your root file system under rootfs/tmp and boot up your development board using NFS. When started up, you can burn the image into the flash device's right partition. Earlier we mentioned how you change the partition table of your flash and in this case we'll write the image into the mtd2 char device:

```
$ cat /tmp/rootfs-jffs2-bigendian.img > /dev/mtd2
(or)
$ dd if=/tmp/rootfs-jffs2-bigendian.img of=/dev/mtd2 bs=64k
(choose the right sector size)
```

To verify the image, we'll mount the block device:

```
$ mkdir /mnt/flashdisk
$ mount -t jffs2 /dev/mtdblock2 /mnt/flashdisk
$ ls /mnt/flashdisk
(verify directory structure)
```

```
$ mount
```

```
rootfs on / type rootfs (rw)
/dev/root on / type nfs \
(rw,v2,rsize=4096,wsiz=4096,hard,udp,nolock,addr=192.168.0.150)
```

```
none on /proc type proc (rw)
/dev/mtdblock2 on /mnt type jffs2 (rw)
```

```
$ df
```

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
rootfs	18042160	7165904	9959752	42%	/
/dev/root	18042160	7165904	9959752	42%	/
/dev/mtdblock2	13312	3920	9392	29%	/mnt

```
$ umount /mnt/flashdisk
$ rm -rf /mnt/flashdisk
```

Now that we're certain that the partition on our flash is working well, we can move on to starting the Linux operating system entirely from flash. To do that, we can append some arguments to the kernel that inform it that a root file system exists in flash and not via NFS like we did until now. Change the bootargs environment of U-Boot to:

```
=> setenv bootargs $(flashroot) $(ip) init=init
=> saveenv
=> reset
```

Now the board will reset and the freshly stored boot arguments will be passed to the kernel. The result of this operation is that the entire system will boot from flash.

7. References

- Wolfgang Denk: “DULG: Denx U-Boot and Linux Guide” (<http://www.denx.de>)
- Karim Yaghmour: “Building Embedded Linux Systems” (O’Reilly)
- Vipin Malik: “The Linux MTD, JFFS HOWTO”
- Motorola Website (<http://e-www.motorola.com>)
- Mailinglists:
 - o <http://lists.linuxppc.org> (embedded)
 - o <https://lists.sourceforge.net/lists/listinfo/u-boot-users>
 - o <http://lists.infradead.org/mailman/listinfo/linux-mtd>