Series 2000

IM-DOC-021
Installation
Manual

Using the DiskOnChip® with Linux OS

Written by: Sasha Geller and Esther Spanjer

Apr-99
91-SR-005-10-7L REV. 1.1

M-Systems
Flash Disk Pioneers

Limited Warranty

(a) M-Systems warrants that the Licensed Software — **prior to modification and adaptation by Licensee —** will conform to the documentation provided by M-Systems. M-Systems does **not** warrant that the Licensed Software will meet the needs of the Licensee or of any particular customer of Licensee, nor does it make any representations whatsoever about Licensed Software that has been modified or adapted by Licensee. .

(b) Subsection (a) above sets forth Licensee's sole and exclusive remedies with regard to the Licensed Software.

M-SYSTEMS MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE LICENSED SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO OTHER WARRANTIES WITH RESPECT TO THE LICENSED SOFTWARE ARISING FROM ANY COURSE OF DEALING, USAGE OR TRADE OR OTHERWISE.

IN NO EVENT SHALL M-SYSTEMS BE LIABLE TO LICENSEE FOR LOST PROFITS OR OTHER INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, WHETHER UNDER THIS AGREEMENT, IN TORT OR OTHERWISE.

(c) Licensee shall not make any promise, representation, warranty or guaranty on behalf of M-Systems with respect to the Licensed Software except as expressly set forth herein.

Please note: The Licensed Software is not warranted to operate without failure. Accordingly, in any use of the Licensed Software in life support systems or other applications where failure could cause injury or loss of life, the Licensed Software should only be incorporated in systems designed with appropriate and sufficient redundancy or back-up features.

1. Introduction

M-Systems' DiskOnChip® is a new generation of single-chip flash disk. It contains built-in firmware that provides full hard disk emulation and allows the DiskOnChip to operate as a boot device.
When used under Linux, the DiskOnChip is managed by TrueFFS®, (True Flash File System) technology based device driver, attached to the standard Linux file system [`ext2`].

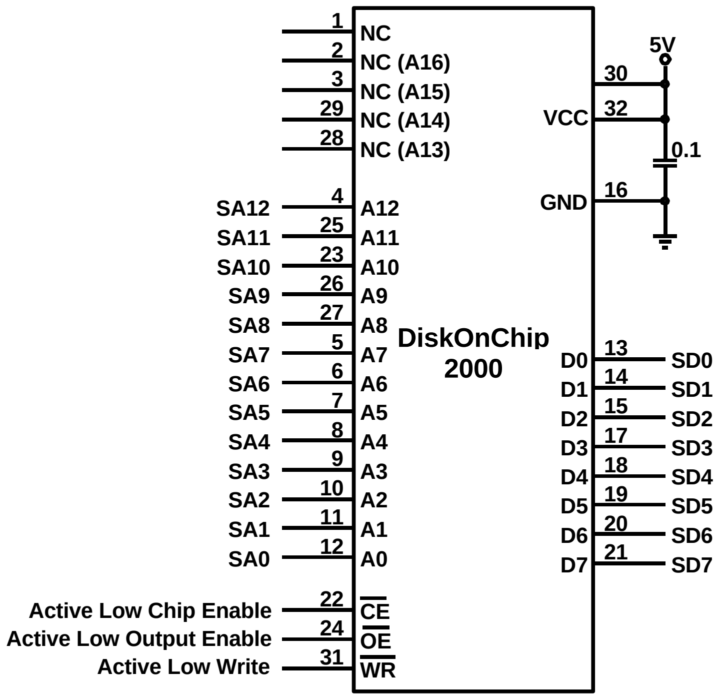
This application note is intended for system integrators designing with the DiskOnChip2000, DiskOnChip Millennium or DiskOnChip DIMM and describes how the DiskOnChip can be installed as an additional disk or as a boot device under Linux.

It is assumed that the reader is familiar with the Operating System in use.

The hardware requirements of the DiskOnChip will be briefly discussed, the main part of this installation manual is related to software installation. This will include basic driver installation and boot issues.

2. Hardware requirements for the DiskOnChip

Originally designed for PC environments, the DiskOnChip can also be used in different hardware environments. The minimum requirements are a twelve-bit address bus, an eight-bit data bus, and three active low control signals (CE, OE, WR). Following is a drawing of the DiskOnChip and its pins. For more detailed information of the DiskOnChip hardware environment, please refer to the DiskOnChip datasheet or to Application Note AP-DOC-10, "Designing with the DiskOnChip, AP-DOC-30 "Designing with the DiskOnChip® Millennium in a RISC Environment" or AP-DOC-31 "Designing with the DiskOnChip® Millennium in a PC Environment".



3. Requirements for installation of the DiskOnChip into Linux

In order to prepare the DiskOnChip to boot Linux, the following software programs and tools are required:

- Linux should be installed on your HDD. It is possible to check the kernel version by typing the command `uname -r`.
- The Linux kernel sources should be installed in Linux (if you don't have them, refer to kernel-HOWTO at http://sunsite.unc.edu/LDP/).
- A DOS boot diskette or a HDD that boots into DOS.
- M-Systems' DiskOnChip DOS utilities diskette.
- M-Systems' DiskOnChip driver for Linux.

Note: If you are about to install Linux, make sure you also install the kernel sources package and that it is possible to pass a full compilation of your sources.

3.1 Utilities diskette content

The DiskOnChip utilities diskette contains the following files:

`dformat`	DiskOnChip formatting utility
`dupdate`	Utility for updating DiskOnChip firmware
`docpmap`	Utility to retrieve information about the DiskOnChip
`doc121.exb`	DiskOnChip firmware image. "121" is the firmware version, the

	actual diskette might contain higher versions of the firmware, e.g. doc122.exb, doc123.exb, etc.
`doc2.fff`	Alternative firmware image for the DiskOnChip

3.2 Linux driver for DiskOnChip

The compressed file `driver.tgz` contains the following files:

/usr/src/linux/drivers/block/flash_doc/flobj.obj	DiskOnChip device driver core object file
/usr/src/linux/drivers/block/flash_doc/fldrvlnx.c	Source code for driver integration with kernel
/usr/src/linux/drivers/block/flash_doc/makefile	Driver's makefile
/tmp/doc-driver/doc-patch-2.0.##	Patch for kernel sources 2.0.##
/tmp/doc-driver/lilo-patch	Patch for Lilo to make pLilo
/tmp/doc-driver/plilo	Linux loader, patched to use with the DiskOnChip
/tmp/doc-driver/lilo.conf	Lilo sample configuration file
/tmp/doc-driver/boot.b	Boot loader, udpated to use with DiskOnChip
/tmp/doc-driver/copy2doc	Sample scripts, aid tool to create a root file system
/tmp/doc-driver/pam.d/other	File meant only for Redhat & Caldera, defines permission access
/tmp/doc-driver/samplefs.txt	Sample root file system listing

4. Installing the DiskOnChip as an additional drive

Before the DiskOnChip can be used as the boot disk for Linux (see Chapter 5), it first needs to be installed as an additional disk in the system (Linux is booted from a HDD). This chapter describes how to prepare the DiskOnChip and Linux to configure the DiskOnChip as an additional disk in the system. In order to achieve this, the firmware on the DiskOnChip needs to be updated (par. 4.1) and then the DiskOnChip device driver needs to be integrated into Linux (par. 4.2). This is done in the following order:

1. Linux needs to be configured with the required devices (par. 4.2.1 Preparing Linux for integration).
2. The DiskOnChip driver needs to be added to the kernel and the kernel is recompiled (par. 4.2.2. Adding the driver to the kernel).
3. The compiled kernel is booted from HDD (par. 4.2.3. Booting the compiled kernel from HDD).
4. A Linux partition needs to be created on the DiskOnChip (par. 4.2.4. Creating a

Linux partition on the DiskOnChip)

5. A native Linux file system needs to be created on the DiskOnChip (par. 4.2.5. Creating a native Linux file system on the DiskOnChip)

4.1 Updating the firmware

Before the DiskOnChip can be used as the boot disk or as an additional disk in Linux, it needs to be formatted with the alternative firmware image. Please perform the following steps:

1. Plug the DiskOnChip into its socket and boot your system into DOS
2. Insert the DiskOnChip utility diskette into your floppy drive and type the following command to format the DiskOnChip with alternative firmware:

```
dformat /win:d000 /s:doc2.fff /y
```

Note: If you receive the error: "No DiskOnChip 2000 (R) was found at D000:0", then run the DOS command `docpmap /i` to find out at which address the DiskOnChip is located.

4.2 Integrating the DiskOnChip driver into Linux

4.2.1 Preparing Linux for integration

In order to perform the following steps, you must be logged in as the superuser (root). To prepare Linux for integration of the driver, please perform the following steps:

1. The floppy drive must be active. To verify that this is actually the case, type:

   ```
   # mount
   ```

   If the `/dev/fd0` doesn't appear in the first column, type:

   ```
   # mount/dev/fd0 /mnt
   ```

   From this point it is assumed that the DiskOnChip utility diskette that contains the Linux driver is mounted at `/mnt`.
2. It is necessary that the kernel sources are installed. To check this, type:

   ```
   # ls /usr/src/linux
   ```

   If this directory exists, then the kernel sources are installed. If not, please refer to http://sunsite.unc.edu/LDP/ or type:

   ```
   zcat /usr/doc/HOWTO/Kernel-HOWTO.gz | more
   ```
3. In order to install the driver into Linux, unzip and untar the file `driver.tgz` as follows:

   ```
   # cd /tmp
   # mkdir temp
   # cd temp
   # tar -zxvf /mnt/driver.tgz
   # cp -rf . /
   # cd ..
   # rm -fr temp
   # cd /usr/src/linux/
   ```

```
# patch -p0 < /tmp/doc-driver/doc-patch-2.0.##
```

The last command will patch the current kernel sources in order to include the DiskOnChip driver for Linux (with ## = kernel version). Please notice that the patch for the kernel is working only on original kernel sources, and the patch is version specific. In case the utility `patch` is not available in your Linux environment, refer to chapter 6 “Troubleshooting”.

Note: If you have a kernel version other than 2.0.29-2.0.35, please don’t perform the last command (`# patch -p0 < /tmp/doc-driver/doc-patch-2.0.##`), but patch the sources manually (see Appendix III Applying a patch manually).

4. Configure the kernel as follows:

```
# cd /usr/src/linux
# make menuconfig
```

or

`# make config` if the last command doesn’t work.

Define your system by marking the correct devices. If you aren’t sure what the purpose of the device is, then leave it as it is. Make sure that you mark the M-Systems DiskOnChip as “Y”es under the ‘Floppy, IDE and other devices’. After you are finished, exit and the new configuration will be saved.

Note: For further details about compiling the kernel or how to apply a patch, refer to http://sunsite.unc.edu/LDP/ or type:

```
# zcat /usr/doc/HOWTO/Kernel-HOWTO.gz | more
```

5. After the configuration is finished, it is necessary to remove the old object files.

Type the following command:

```
# make clean
```

4.2.2 Adding the driver to the kernel

In order to compile the DiskOnChip driver for Linux type the following command:

```
# cd /usr/src/linux/drivers/block/flash_doc
# make
```

Check dependencies:

```
# cd/usr/src/linux

# make dep
```

And compile the kernel (this may take up to 15 minutes):

```
# make zImage
```

If the kernel is compiled successfully, a similar message should be shown:

```
Root device is (3, 3)
Boot sector 512 bytes
Setup is 4332 bytes
System is 374 kB
Sync
Make[1]:Leaving directory ‘/usr/src/linux-2.0.32/arch/i386/boot’
```

If there are any compilation errors, refer to chapter 6 “Troubleshooting”.

4.2.3 Booting the compiled kernel from HDD

Type the following commands:

```
# cp /usr/src/linux/arch/i386/boot/zImage  /doc2000
# vi /etc/lilo.conf
```

Please add the following lines at the bottom of the file (press ‘INS’ to edit the file and save and exit by typing <Esc>, ‘:’, ‘w’, ‘q’):

```
image = /doc2000
root = /dev/hda1
label = doc2000
read-only
```

Note: The device `/dev/hda1` points to HDD that Linux boots from. Hda1 is the *first* partition (1) on the *first* IDE hard disk (a), hdb2 is the *second* partition (2) on the *second* IDE hard disk (b), etc. In case it is not clear which device it is, look at the start of the file `lilo.conf` and search for the line `first root = ...`)

Make the inodes for the DiskOnChip:

```
# cd /dev
# mknod fla b 62 0
# mknod fla1 b 62 1
# mknod fla2 b 62 2
# mknod fla3 b 62 3
# mknod fla4 b 62 4
```

Number 62 stands for major device number. Since it is hard coded into the driver, any other number wouldn’t work.

Run Lilo (Linux Loader) to create the map for the kernel and make sure that “doc2000’ is listed (if not, please return to the beginning of paragraph 4.2.3):

```
# lilo
```

To load the updated kernel with the driver for the DiskOnChip, please perform the following steps:

1. Reboot the computer and load Linux.
2. When the Lilo prompt is displayed, press <Ctrl> or <Alt> or <Tab>. The screen will show :

   ```
   Lilo boot:
   ```

3. Type “doc2000” to load the recompiled kernel:

```
Lilo boot:doc2000
```

4.2.4 Creating a Linux partition on the DiskOnChip

In order to create a Linux partition on the DiskOnchip, all the DOS partitions on the DiskOnChip need to be removed and a native Linux File system [`ext2`] needs to be created. This step is vital in order to make the DiskOnChip bootable.

Run the `fdisk` utility to create a Linux partition that spans the entire DiskOnChip. Mark the partition as bootable and write the partition table to the DiskOnChip. Type the following command to run the `fdisk` utility:

```
# fdisk /dev/fla
```

Note: You may create more than one primary partition. Just make sure that the boot flag of the main partition boot is active.

In order to delete the existing partition on the DiskOnChip and to create a new one, perform the following steps within the `fdisk` utility:

1. To display the contents of the partition table:

```
Command(m for help):p
```

2. To delete all existing partitions. Enter each partition number for deletion:

```
Command(m for help):d
```

3. Create a new Linux native partition:

```
Command(m for help): n
Command action e extended
               p primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-XXX): 1
Last cylinder or +size or +sizeM or +sizeK ([1]-XXX):XXX
```

4. Change the type of the partition to `Linux native`:

```
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): 83
```

5. Make the partition bootable:

```
Command (m for help): a
Partition number (1-4): 1
```

6. Recheck the partition table:

```
Command (m for help): p

Disk /dev/fla: 16 heads, 9 sectors, 1002
Cylinder units = cylinders of 144 * 512 bytes
```

```
Device     boot  begin  Start  End   Blocks Id  System
/dev/fla1     *      1        1     1002   72139+ 83   Linux
native
```

7. Save the new partition table (disregard any `fdisk` warnings):

```
Command (m for help): w
```

8. Reboot the machine to let the new partition table load into memory and load Linux again with the new compiled kernel. This step is vital if there is more than one partition.

Note: For further information on the `fdisk` utility, refer to the *man* pages.

4.2.5 Creating a native Linux file system on the DiskOnChip

In order to initialize the file system, on the newly created Linux partition, on the DiskOnChip, type the following command:

```
# mke2fs /dev/fla1
```

Note: When a small capacity DiskOnChip is used (4MB or smaller), more space for inodes need to be allocated. Type the following command:

```
# mke2fs -i 2048 /dev/fla1
```

Mount the file system to a directory, in order to make it accessible:

```
# mkdir /diskonchip
# mount /dev/fla1 /diskonchip
```

From now onwards it is possible to use the DiskOnChip as an additional disk in your system.

Note: If you created more than one partition, please repeat this step for each partition (`dev/fla1, /dev/fla2, etc.`)

5. Booting Linux from the DiskOnChip

Being able to boot Linux from the DiskOnChip is of great importance. It makes it possible to use the DiskOnChip as the only disk in the system, holding the OS itself in addition to all other applications and files.
In order to make a block device bootable on Linux there are several steps that need to be taken. The kernel and the Lilo program should be copied to the block device, and a root file system needs to be created. Creating a root file system on Linux is necessary, as from this root file system the kernel is activating several programs, such as:

`Init`	Initialize all processes
`Swapon`	Activate swapping
`Mount`	Mount the root and proc filesystems
`Sh`	Shell

Note: For further details refer to http://sunsite.unc.edu/LDP/ or type the following command: `#zcat /usr/doc/HOWTO/Bootdisk-HOWTO.gz | more`

Please notice that the DiskOnChip firmware (i.e. `doc110.exb`) collides with Lilo. This means that it is not possible to load Linux from the HDD after the original firmware is restored (see par. 5.1.4). This doesn't mean that the HDD is non-functional, it only means that the alternate firmware (`doc2.fff`) needs to be reloaded (see par. 4 .1).
If it is required to boot Linux from both the HDD and the DiskOnChip, it is necessary to use both pLilo and `boot.b` (provided with the DiskOnChip driver).
For further details, refer to Appendix II.

5.1 Creating a root file system

5.1.1 Introduction

This section is based on Bootdisk-HOWTO and other experiments. Since each distribution has a different file locations and different installations, it is possible that you won't succeed in booting Linux the first time from the DiskOnChip. When you receive errors, follow each error and try to fix things, but reload Linux from the HDD and remount the DiskOnChip every time you do.
Note: There are several programs that create a root file system. It is not possible to say that any of them is complete right now, but they are worth trying. The following program is recommended:
Yard: Yard creates rescue disks (also called boot disks) for Linux. A rescue disk usually contains utilities for diagnosing and manipulating hard disks and file systems. It is used when it is not possible (or if it is not required) to boot from your HDD. This package contains mainly perl scripts. Refer to http://www.croftj.net/~fawcet/yard/
Note: For a sample root file system, please refer to the file `/tmp/doc-driver/samplefs.txt` which is part of the compressed file `driver.tgz`

5.1.2 Overview

A root file system must contain everything that is needed to support a full Linux system. To achieve this, the disk must include the minimum requirements for a Linux system:

- Basic file system structure
- Minimum set of directories: /dev, /proc, /bin, /etc, /lib, /usr, /tmp
- Basic set of commands: sh, ls, cp, mv, etc.
- Minimum set of config files: rc, inittab, fstab, etc.
- Devices: /dev/hd*, /dev/tty*, /dev/fd0, etc.
- Runtime libraries to provide basic functions used by utilities.

5.1.3 Populating the file system

From this point onwards it is assumed that the DiskOnChip is mounted to the `/diskonchip` directory.

The sample script `copy2doc` that is provided with the driver is located in directory `/tmp/doc-driver`. This sample script is created for RedHat 5, SuSE 5.2 and Caldera, and is customized for RedHat 5. It might be necessary to change the script according to your distribution. Files that are specific to a distribution can be found in the script with the distribution name written after it. To exclude a file, make sure the '#' mark is present at the beginning of the line. To include a file, make sure the '#' mark is removed.

The sample script only copies the basic files that are needed for booting. For any other operation it is necessary to add more files.

To create a root file system based on the sample script `copy2doc`, perform the following steps:

1. Go to the `/diskonchip` directory:

```
    # cd /diskonchip
```

2. If necessary, customize the sample script (for other distributions):

```
    # vi /tmp/doc-driver/copy2doc
```

3. Run the script:

```
    # sh /tmp/doc-driver/copy2doc
```

4. Create the list of files to be mounted:

```
    # vi /diskonchip/etc/fstab
```

5. Press <INS> to start editing and insert the following lines:

```
    /dev/fla1   /       ext2  defaults 1 1
    /proc       /proc   proc  defaults 0 0
```

   Press <ESC>, ':', 'w', 'q' to save the file. It is possible to add more devices here (for more information, refer to the *man* pages).

6. For RedHat and Caldera it is also necessary to copy the configuration file for the pam library (responsible for making authentic users):

```
   # cp /tmp/doc-driver/pam.d/other
         /diskonchip/etc/pam.d/other
```

Note: Appendix I explains how to create your own root file system manually. Although this is more complicated than using the above mentioned sample script, it is highly recommended.

All modules should be placed in `/lib/modules/`. It is necessary to at least include the programs `insmod`, `rmmod` and `lsmod`. If it is required to load the modules automatically, then also include `modprobe`, `depmod` and `swapout`. When using `kerneld`, include it along with `/etc/conf.modules`.

Some system programs, such as `login`, complain when the file `/var/run/utmp` and the directory `/var/log` do not exist. To solve this, type the following commands:

```
    # mkdir -p /diskonchip/var/{log,run}
    # touch /diskonchip/var/run/utmp
```

After all the needed libraries and programs are set up, run `ldconfig` to remake `/etc/ld.so.cache` on the root file system. The cache tells the loader where to find the libraries. To remake `ld.so.cache`, type the following command:

```
    # cd /diskonchip

    # chroot /diskonchip  /sbin/ldconfig
```

The command `chroot` is necessary, because `ldconfig` always remakes the cache for the root file system.

5.1.4 Copying the kernel, updating the boot sector and rebooting

To copy the kernel and to update the boot loader files, type the following commands:

```
# mkdir /diskonchip/boot
# cp /usr/src/linux/arch/i386/boot/zImage
    /diskonchip/boot/doc2000
# rdev /diskonchip/boot/doc2000 /dev/fla1
# cp /tmp/doc-driver/plilo /diskonchip/sbin
# cp /tmp/doc-driver/boot.b /diskonchip/boot
# cp /tmp/doc-driver/lilo.conf /diskonchip/etc
#  /diskonchip/sbin/plilo  -C  /diskonchip/etc/lilo.conf  -i
     /diskonchip/boot/boot.b -m /diskonchip/boot/map
```

Verify that after the last command the device `doc2000` is listed on the screen.

If it is required to load other partitions, then the file `/diskonchip/etc/lilo.conf` should be edited.

Note: pLilo is the patched Lilo, in order for the DiskOnChip firmware not to collide with the Linux bootloader.

The final steps in the process of making the DiskOnChip bootable for Linux are as follows:

1. Unmount the DiskOnchip

```
# cd /
# unmount  /dev/fla1
```

2. Reboot and load DOS, and reinstall the original firmware:

```
A:> dupdate /win:D000 /s:DOC110.EXB
```

3. Reboot the machine and disable the HDD in the BIOS setup or make the DiskOnChip the first boot device in the system by using the following command:

```
A:> dupdate /win:D000 /s:DOC110.EXB /FIRST
```

4. Linux will boot now from the DiskOnChip.

6. Troubleshooting

1. Adding more programs to Linux root file system
If the DiskOnChip boots Linux without a problem and it is required to add more programs to the Linux root file system, then mount the HDD and copy the needed files.

2. DiskOnChip does not boot Linux
There are several errors that you can encounter during boot:

2.1 If the DiskOnChip does not boot at all, please follow all the instructions from the start of this Installation manual, remember to also update the original firmware of the DiskOnChip (`doc110.exb` for example) with the alternate firmware (`doc2.fff`) in order to boot Linux from your HDD.

2.2 If the kernel boots, but it gets stuck on:

```
VFS: Unable to mount -.
```

Most likely you forgot to do:

```
# rdev /diskonchip/boot/doc2000  /dev/fla1
```

2.3 If the DiskOnChip boots and the kernel is loading, but it gets stuck after:

```
VFS: Mounted root (ext 2 filesystem) readonly.
```

Most likely the `init` program or some if its configuration files weren't copied.

3. Can't log in

If you can't login when booting Linux from the DiskOnChip, make sure that:

- your default shell is installed;

- the pam libraries were placed as explained in par. 5.1.3 (only for RedHat and Caldera);

Refer to Appendix II in order to solve this problem.

4. Kernel doesn't compile correctly or patch utility is not available

If the kernel doesn't compile correctly, and the problem is not caused by the DiskOnChip driver, please refer to http://sunsite.unc.edu/LDP/ or type:

```
 #zcat /usr/doc/HOWTO/Kernel-HOWTO.gz | more
```

5. Kernel is too big

When you run pLilo and you receive the error 'Kernel xxx is too big', please recompile the kernel with the `bzImage` option (In step 4.2.2, use `make bzImage` instead of `make zImage`). For further information, please refer to http://sunsite.unc.edu/LDP/ or type:

```
 #zcat /usr/doc/HOWTO/Kernel-HOWTO.gz | more
```

7. Additional information and Tools

Additional information about the DiskOnChip, including application notes, can be found at http://www.m-sys.com.
Additional tools and documents are listed in the following table:

AP-DOC-10 .... :	Designing with the DiskOnChip 2000
AP-DOC-15	Obtaining DiskOnChip 2000 information
AP-DOC-16 .... :	Using the DiskOnChip 2000 with QNX
AP-DOC-17 .... :	Using the DiskOnChip 2000 with Windows CE
AP-DOC-19 .... :	Using the DiskOnChip 2000 with Windows 95
AP-DOC-30	Designing with the DiskOnChip® Millennium in a RISC Environment
AP-DOC-31	Designing with the DiskOnChip® Millennium in a PC Environment
DiskOnChip 2000 Data Sheet........ :	DiskOnChip Data Sheet
DiskOnChip Millennium	DiskOnChip Millennium Data Sheet
DiskOnChip DIMM	DiskOnChip DIMM Data sheet
DiskOnChip 2000 Utilities .... :	DiskOnChip 2000 Utilities User Manual
DiskOnChip2000-EVB.................. :	DiskOnChip Evaluation Board
DiskOnChip2000-PIK................... :	DiskOnChip Programmer and Integrators Kit
DiskOnChip-GANG...................... :	8 Socket Gang Programmer

Appendix I: Making a root file system

In order to create your own root file system, please perform the following steps:

1. Make the following directories:

```
    # cd /diskonchip
    # mkdir bin dev etc lib mnt proc sbin tmp usr var
```

2. Create devices in the `/dev` directory. You can either do this manually or just copy the `/dev` directory from the HDD. If you wish to save space, it is possible to remove non-required devices, i.e. if you don't have a SCSI drive, then remove all the sd* devices.

```
    # cp -dpR  /dev  /diskonchip
```

   This commands copies many unnecessary inodes to the DiskOnChip. Removing them causes no problem, as long as you make sure that the ones listed in the sample file system are present.

3. Copy and configure the files in the `/etc` directory:

```
    # cp -dr  /etc/rc.d  /diskonchip/etc  (for RedHat)
```

   or

```
    # cp -dr /sbin/init.d  /diskonchip/sbin  (for SuSE)
    # cp -d /etc/inittab  /diskonchip/etc
```

4. Copy the password file and make sure that each user has it's default shell installed:

```
    # cp /etc/passwd  /diskonchip/etc
    # cp /etc/shadow /diskonchip/etc (possible that you don't have this)
    # cp /etc/group   /diskonchip/etc
```

5. Create the file `/etc/fsstab` that contains the list of files to be mounted:

```
    # vi /diskonchip/etc/fstab
```

   Press <INS> to start editing and insert the following lines:

```
    /dev/fla1 /         ext2      defaults 1 1
    /proc     /proc     proc      defaults 0 0
```

   Press <ESC>, ':', 'w' and 'q' to save the file.
   It is possible to add more devices here. For more information, refer to the *man* pages.

6. There are several programs that need to be copied in order to have a functional environment. Other programs are not as important, although it would be rather difficult to work without them. All other programs that are not listed below are considered optional. Copy these programs to the directories `/bin` or `/sbin` as follows:

```
    # cp /bin/{program_name}
         /diskonchip/bin/{program_name}
```

/bin directory:

cat	echo	mount
chmod	hostname	mv
chown	kill	ps
cp	ln	rm
cut	login	rmdir
dd	ls	sh
df	mkdir	su
dircolors	mke2fs	sync
du	mknod	umount
e2fsck	more	uname

/sbin directory:

mingetty	shutdown
halt	swapoff
init	swapon
ldconfig	telinit
mkswap	update
reboot	runlevel
rdev	

The filename of `mingetty` varies with the distribution, i.e. RedHat and SuSE use `mingetty`, Slackware uses `agetty`. To find out what the name of this file in your distribution is, perform a grep on "getty":

```
# grep getty /etc/inittab
```

7. The `/lib` directory contains all the shared libraries and loaders. Only the appropriate libraries need to be copied to the `/lib` directory. In order to check which libraries are needed, type the following command for each file in these 2 directories:

```
# ldd /sbin/{filename}
```

or

```
# ldd /bin/{filename}
```

For example:

```
# ldd /sbin/mke2fs
     libext2fs.so.2 → /lib/libext2fs.so.2
     libcom_err.so.2 → /lib/libcom_err.so.2
     libuuid.so.1 → lib/libuuid.so.1
     libc.so.5 → /lib/libc.so.5
```

This will show which libraries are needed for the program `mke2fs`. In this example, it would be necessary to copy the following 4 libraries.

```
# cp /lib/ext2fs.so.2  /diskonchip/lib
# cp /lib/libcom_err.so.2  /diskonchip/lib
# cp /lib/libuuid.so.1  /diskonchip/lib
# cp /lib/libc.so.5  /diskonchip/lib
```

In case you have a long list of files to be copied, it is also possible to run the following command:

```
# ldd /bin/* > lib_list
# more lib_list
```

Copy the library loaders as follows:

```
# cp lib/ld.so /diskonchip/lib           (a.out loader)
# cp /lib/ld_linux.so  /diskonchip/lib   (elf loader)
```

Note: It is possible to use objcopy to reduce the size of the libraries. For example:

```
# objcopy -strip-debug  /diskonchip/lib/lib.so.5
```

Appendix II: Booting from a HDD when DiskOnChip firmware is active

Since Lilo and the DiskOnChip firmware share the same memory area, the system will hang during boot when using an unpatched Lilo.

Updating the Lilo solves this problem. The existing boot loader needs to be updated as follows:

```
# /tmp/doc-driver/plilo -i  /tmp/doc-driver/boot.b
```

This command uses the patched Lilo supplied by the driver and patched `boot.b`, and your default `/etc/lilo.conf`. Make sure that you run this command in the Linux that booted from HDD.

pLilo is actually forcing you to use smaller kernels, because it allocates part of the physical memory to the driver. Therefore big kernels (around 470 KB) will have a problem with pLilo (pLilo will stop with an error 'Kernel xxx is too big'). There are two solutions to this problem:

- Compile the kernel with the bzImage option ('`make bzImage`'). Please refer to `/usr/doc/HOWTO/Kernel-HOWTO.gz` for more information.
- Commenting out all big kernels from `lilo.conf` (only during the installation procedure).

M-Systems provides a patch file to Lilo sources to make pLilo. In order to activate the patch, you must have the Lilo sources available on your system. After you verified that the sources are available, type the following command:

```
# patch -p0 < lilo-patch
```

You now have pLilo sources.

Appendix III: Applying a patch manually

There are two ways the DiskOnChip driver can be integrated into the kernel, manually or with a patch utility. This appendix explains how to patch the kernel manually.
Note: If you are using kernel version 2.0.29-2.0.35, you can use the patches that are available in directory `/usr/src/linux/drivers/block/flash_doc/fl.##.` Or use any of these patches as a base for you own patch.

Go to you local kernel sources:

```
# cd /usr/src/linux
```

There are 5 files that you will need to patch:

```
/usr/src/linux/include/linux/blk.h
/usr/src/linux/include/linux/major.h
/usr/src/linux/drivers/block/Makefile
/usr/src/linux/drivers/block/Config.in
/usr/src/linux/drivers/block/ll_rw_blk.c
```

Review the closest patch file provided in the file `driver.tgz.` For example: if you want to patch kernel 2.2.0, you will need to review the patch file for 2.0.35.
Each of these files look something like this:

```
--- include/linux/major.h Tue Aug 12 23:57:23 1997
+++ /tmp/package/usr/src/linux/include/linux/major.h   Mon  Aug  17
14:05:19 1998
@@ -69,6 +69,8 @@
 #define APBLOCK_MAJOR   60   /* AP1000 Block device */
 #define DDV_MAJOR       61   /* AP1000 DDV block device */

+#define IGEL_FLASH_MAJOR 62 /* tom@igel.de 2.6.98 */
+
 #define SPECIALIX_NORMAL_MAJOR 75
 #define SPECIALIX_CALLOUT_MAJOR 76
```

Each of the above mentioned files need to be edited in order to create the correct patch for your kernel.
In the above example the file `/usr/src/linux/include/linux/major.h` is patched. The two lines that start with the '+' mark are added to the current source code, lines that start with the '-' mark are removed from the source code. Please repeat this process for all of the 5 above-mentioned files.