M-Systems
Flash Disk Pioneers

DiskOnChip Software Utilities for TrueFFS 6.x

User Manual

February 2004
91-SR-001-86-7L Rev. 1.3

M-Systems
Flash Disk Pioneers

DOCUMENT CONTROL INFORMATION

DCO No.: N/A

	Title	Name	Date
Issued by:	DiskOnChip Product Manager	Amir Fridman	February 5, 2004
Approved by:	VP Operations and Product Management	Rami Koren	February 5, 2004

TABLE OF CONTENTS

1. Introduction 4
1.1. DiskOnChip Products Supported by TrueFFS 4
1.2. Terms and Abbreviations 5
2. DFORMAT Utility 7
2.1. DFORMAT Syntax 7
2.2. Using DFORMAT Flags 8
2.2.1. Device-Specific Flags (INFTL/SAFTL-Formatted DiskOnChip Devices) 9
2.2.2. Flags for Advanced Operations 10
2.2.3. DFORMAT Usage Examples 11
3. DINFO Utility 12
3.1. DINFO Syntax 12
3.2. Using DINFO Flags 12
4. DIMAGE Utility 14
4.1. DIMAGE Syntax 14
4.2. Creating the Source DiskOnChip 15
4.3. Copying the Source DiskOnChip to an Image File 15
4.4. Copying the Image File to Target DiskOnChip Devices 16
4.5. DIMAGE Error Messages 17
5. Utilities Package for Windows 2000 and Windows XP 18
5.1. The Utility DLL Package 18
5.1.1. Purpose 18
5.1.2. Package Description 18
5.1.3. Functionality 20
5.2. Customization Routine API 21
5.2.1. User-Defined Access Routines 21
5.2.2. User-Defined OS Routines 22
6. Known Limitations 24
7. Additional Documents and Tools 25
How to Contact Us 26

1. INTRODUCTION

This manual describes the following DiskOnChip utilities for Windows 2000/XP and DOS that may be used with M-Systems DiskOnChip products:

- DFORMAT
- DINFO
- DIMAGE

Note: TrueFFS 6.2.1 supports only SAFTL-formatted DiskOnChip devices. The utilities (DINFO, DFORMAT, and DIMAGE) provided with TrueFFS 6.2.1 support only SAFTL-formatted DiskOnChip devices.

A description of the utility flags is provided, including specific examples and basic instructions to assist you in easy and quick installation of DiskOnChip on your target platform.

This manual is intended for system integrators who are familiar with the PC environment and the operating system in use. It is also recommended to read the relevant DiskOnChip data sheets and installation instructions for your specific operating system. Refer to Section 7 for additional documents and tools available.

The latest versions of the DiskOnChip utilities can be downloaded from M-Systems' website at www.m-systems.com.

1.1. DiskOnChip Products Supported by TrueFFS

M-Systems' DiskOnChip is a family of high-performance flash disks. The DiskOnChip series includes the following products:

Devices supported by TrueFFS 6.2.1:

- DiskOnChip G3
- DiskOnChip P3
- DiskOnChip 2000 G3

Devices supported by TrueFFS 5.x:

- DiskOnChip Millennium Plus
- Mobile DiskOnChip
- DiskOnChip 2000 TSOP
- DiskOnChip Millennium
- DiskOnChip 2000
- DiskOnChip DIMM 2000

A TrueFFS driver is required to work with all DiskOnChip products. TrueFFS is natively supported by every major OS, such as Windows CE, VxWorks, Symbian, Palm and Linux. The latest TrueFFS drivers can be obtained from M-Systems for these operating systems and others,

such as XP Embedded, and QNX, at http://www.m-systems.com/download. For other environments (including OS-less) the TrueFFS Software Development Kit (SDK) can be obtained. When using DiskOnChip as the boot device in a non-x86 environment, M-Systems' Boot Software Development Kit (BDK) package is required. Contact M-Systems regarding availability for both packages.

The following sections describe the DiskOnChip utilities and how they are used:

- Section 2 describes the DFORMAT utility
- Section 3 describes the DINFO utility
- Section 4 describes the DIMAGE utility

1.2. Terms and Abbreviations

Table 1: Glossary of Terms and Abbreviations

Interface Term	Definition
Socket	The physical location where a DiskOnChip device can reside. The maximum number of sockets is defined by the **SOCKETS** definition in **FLCUSTOM.H**.
Physical Drive / Physical Device	DiskOnChip device inserted in a socket. The number of physical drives depends on the number of sockets registered by the socket components in the TrueFFS application. Physical drives are numbered from zero, and serve as the four least significant bits (LSB 0-3) of the drive handle (see below). See also *TrueFFS Sockets and Drives*.
Partition / Volume	A partition is part of a physical drive handled as an independent unit. A partition can be either a disk partition (logical drive partition) or a binary partition (see page 5). A physical device can contain up to four partitions of any type, provided at least one of them is a disk partition.
Disk Partition (Logical Drive / BDTL Partition / BDTL Volume) **Note**: The term disk partition / BDTL volume / BDTL partition are used in parallel for the same independent area on the flash. Binary partition / BDK partition / binary volume are also interchangeable.	A disk or a BDTL (Block Device Translation Layers) partition is a partition formatted and supported by one of the TrueFFS Translation Layers (NFTL, INFTL or SAFTL), making it capable of supporting a block device driver and file system. Most operations on the disk partition require a mount operation on the partition, using one of the API functions provided for that purpose. This enables the TrueFFS driver to acknowledge the presence of the partition, check the validity of its BDTL and/or FAT data formats, and initialize its supporting data structures in memory. The disk partitions are numbered starting from zero and serve as bit 4-7 of the drive handle. The maximum number of logical drives is defined by the **VOLUMES** definition in **FLCUSTOM.H**.

Interface Term	Definition
Binary Partition / Binary Volume	A binary partition is not supported by the TrueFFS block device translation layer (BDTL), but can be used as a direct-access storage area. The binary partitions are numbered starting from zero, independently of the disk partitions co-existing on the same physical drive. This number serves as bit 4-7 of the drive handle used in the functions accessing the binary partitions. The maximum number of binary partitions is defined by the parameter **BINARY_PARTITIONS** defined in **FLCUSTOM.H**.
Binary Sub-Partition	All of the binary partition blocks are marked with a unique signature. Since TrueFFS-SDK (OSAK) 4.1, a dedicated routine enables changing this signature for a contiguous subset of the partition's blocks, creating several separated areas within the binary partition. Each area is called a binary sub-partition. When first formatted, the binary partition contains a single sub-partition.
SPL Partition	A simplified binary partition that does not use the matching algorithm, making it ideal for storing the SPL code. The simpler logic simplifies the code required for reading, and allows it to fit on the DiskOnChip XIP block.
Firmware Space \ EXB Space	DiskOnChip devices can initialize a segment of their 8KB memory window with a small XIP piece of code. Usually a dedicated area on the flash is loaded to DiskOnChip's internal static RAM. This code is called the IPL (Initial Program Loader) and can be used for system initialization. For systems that need a larger initialization code, the IPL can be used to load a Secondary Program Loader (SPL) from the flash to the system RAM. The SPL usually comprises a few simple calls to the M-Systems BDK, allowing you to load any size of code (while skipping bad blocks) provided the device is large enough. M-Systems provides a firmware file for all of its products that contain an IPL, SPL and a driver for x86 BIOS platforms. This file has an EXB extension.
INFTL Formatted Device	Inverse NAND Flash Translation Layer (INFTL) is the flash management algorithm used by TrueFFS to manage DiskOnChip 2000, Mobile DiskOnChip, and DiskOnChip Millennium Plus products.
NFTL Formatted Device	NAND Flash Translation Layer (NFTL) is the flash management algorithm used by TrueFFS to manage DiskOnChip 2000 and DiskOnChip Millennium products.
SAFTL Formatted Device	Sequential Access Flash Translation Layer (SAFTL) is the flash management algorithm used by TrueFFS to manage DiskOnChip G3/P3 and DiskOnChip 2000 G3 products.

2. DFORMAT UTILITY

Before the TrueFFS driver can access DiskOnChip, the device must be formatted (similar to a floppy disk). Formatting initializes the flash media and writes a new and empty DOS FAT file system to DiskOnChip. When formatting is complete, DiskOnChip contains only a root directory.

DiskOnChip can be formatted more then once, however, all stored data on the device is erased during the formatting process.

Note: When DiskOnChip is reformatted, the boot image (i.e. firmware file **DOCxx.EXB**) is retained by default.

Identical versions of the DFORMAT utility and the firmware file are required for formatting. If the versions are different, the formatting procedure stops and DFORMAT returns an error message. The following sections describe how to use the DFORMAT utility and provide a description of its flags.

2.1. DFORMAT Syntax

The DFORMAT syntax is:

```
DFORMAT [Drive-letter]
  OR
[/WIN:address] [/Flag:n:parameter:size suffix]
```

Where:

`Drive-letter`	Drive letter of the DiskOnChip drive
`WIN`	Memory address where DiskOnChip resides
`Flag`	See full flag list and their description below
`N:`	Partition number (n = 0 for devices using NFTL; n = 0 - 3 for devices using INFTL or SAFTL). Default value is 0.
`Size`	Partition size (size units explained below)
`suffix`	Size units. Can be either `M` for *0x100000, or `K` for *0x400

Example:

```
DFORMAT /win:D0000 /BDKL0:1M
```

Formats the DiskOnChip device located at memory address D0000 with a 1MB binary partition, while the rest is formatted by default as a disk partition.

2.2. Using DFORMAT Flags

The following tables define the various flags used with the DFORMAT utility.

Table 2: Most Commonly Used DFORMAT Flags

DFORMAT Option	Description
/WIN:address	Memory address at which DiskOnChip is located. Use either this parameter or the drive letter. The address should be specified in hex (for example, /WIN:D0000).
/NOFORMAT	Use this flag to update the DiskOnChip firmware or the IPL without reformatting the entire device. It can also be used to read the Bad-Block Table, using the **/LOG** option, without reformatting the device. **/NOFORMAT** can be used with the following flags: **/LOG**, **/IPL**
/Y	Instructs the system not to pause for confirmation before beginning to format.
/FLOOR:n	Specifies the floor number to be formatted. This flag is used in conjunction with the **/FLOORS** flag.
/FLOORS:n	Specifies the total number of floors. Note: Only one floor is formatted. This flag is used in conjunction with the **/FLOOR** flag.
/? & /H	Shows the full Help screen.

Note: By default, DiskOnChip is shipped from the factory configured as the last drive in the system. When other hard drives are installed, DiskOnChip is installed as the last drive. However, if no hard drives are installed, DiskOnChip is still installed as drive [C:]. When configured as the first drive, (using the **/FIRST** option), DiskOnChip is always installed as drive [C:].

Table 3: Binary Partition Flags

DFORMAT Option	Description
/BDKF[n]: Boot Image File	Places the boot image file in the binary partition [n]. Up to three binary partitions can be defined in DiskOnChip devices managed by INFTL or SAFTL. DiskOnChip devices managed by NFTL support only one binary partition. This flag can also be used to load a custom IPL onto pages 0 and 1 of DiskOnChip Millennium 8MB (MD2810), provided that the data is padded to the required size. Loading a customer IPL onto DiskOnChip Millennium Plus or Mobile DiskOnChip requires the **/IPL** flag (refer to Section 2.2.1).

DFORMAT Option	Description
/BDKN[n]: BDK Partition Signature [Default value = BIPO]	Binary partition [n] signature (4 characters).
/BDKL[n]: BDK Partition Size	Defines the size of the binary partition [n]. Useful for reserving space in the binary partition (over the size of the binary program placed using the **/BDKF** flag), for later upgrades.
/LEAVE:k	Leaves the content of the first [k] binary partitions.

Table 4: DOS/FAT File System Flags

DFORMAT Option	Description
/LABEL[n]: Label [Default value = NULL]	The string used as the DOS volume label of the formatted partition [n].
/NODOS[n]	Use this flag to prevent creating a DOS FAT file system while formatting partition [n]. Only low-level formatting is performed. Useful for systems with file systems other then FAT.
/DOSID[n]: ID	FAT partition [n] identification number (ID).
/FAT[n]: Number	Number of FAT copies on the partition [n]. The default is 2.
/OLD_FORMAT[n]	Format disk partition [n] with one sector per cluster.

2.2.1. Device-Specific Flags (INFTL/SAFTL-Formatted DiskOnChip Devices)

This section describes the DFORMAT flags that can be used only with INFTL- and SAFTL-formatted DiskOnChip devices.

Table 5: Device-Specific DFORMAT Flag Options

DFORMAT Option	Description
/BDTLL[n]:Partition Size	Sets the size of the disk partition [n]. The size of the last disk partition does not have to be defined. For example, /BDTL0:1MB creates two partitions, the first 1MB (n = 0) in size, and the second (n = 1) the remaining size of the flash disk. Up to four partitions (including binary partitions) can be defined on DiskOnChip Millennium Plus and Mobile DiskOnChip. This flag is not applicable for older DiskOnChip devices, as they support only one disk partition.
/IPL:File | !	**file** writes a custom IPL (up to 2048 bytes in DiskOnChip G3 64MB (512Mb), DiskOnChip P3, and up to 4096 in DiskOnChip G3 128MB(1Gb)). Useful in non-x86 systems where the standard IPL is not relevant. **!** erases the IPL (the first 1024 bytes).
/BDKP[n]:RWCL:Password [Default value = no protection]	Sets the protection type (**Read**/**Write**/**Change**/**Lock**) and the protection key (password) of the binary partition [n] (n = 0-2): **R**ead: Read-only mode **W**rite: Write-only mode **C**hange: Enable changing the protection type (**R**/**W** or both). **L**ock: Defines whether the LOCK# signal overrides the password.

DFORMAT Option	Description
/BDTLP[n]:RWCL: Password [Default value = no protection]	Sets the protection type (**Read**/**Write**/**Change**/**Lock**) and the protection key (password) of the disk partition [n] (n = 0-3).
/BDKSPL[n]	Makes BDK partition number [n] an SPL partition.
/BDKZ[n]: Password	Inserts the key of the binary partition number [n] (n = 0-2, default is 0). Useful when formatting devices with previously defined protected partitions.
/BDTLZ[n]: Password	Inserts the key of the disk partition number [n] (n=0-3, default is 0) Useful when formatting devices with previously defined protected partitions.

Note: Up to two partitions may be protected. Only one of them may be set to Change**.**

2.2.2. Flags for Advanced Operations

Table 6 defines the DFORMAT flag options for advanced operations.

Table 6: Advanced Operation Flag Options

DFORMAT Option	Description
/LOG:file	Copies the Bad-Block Table (BBT), stored on DiskOnChip, to a file. Required when a test that destroys the BBT is performed on DiskOnChip. When this flag is used, DiskOnChip is also reformatted. If you want to preserve data on DiskOnChip and only read the BBT, use the **/NOFORMAT** flag. Note: The BBT returned by this flag contains several blocks that are for TrueFFS internal use.
/USE:k	Percent of usable space to be formatted. Lower capacity increases write performance. The default value is 98.
/SPARE[n]:k	Number of spare units of disk partition [n]. Starting from TrueFFS 5.1.x, the default is 2.
/UNFORMAT	Removes any existing DiskOnChip formatting and restores DiskOnChip to its virgin state. This flag is used, for example, when you have two cascaded Mobile DiskOnChip devices and want to add another device. Regular DFORMAT on the extra device will not be effective in building the shared BBT. The **/UNFORMAT** flag must be run to integrate the three devices as one unit.
/WINL:Address	Determines the lower memory limit of the window in which the DiskOnChip device will be searched. If the /WIN option was used, this option is ignored.
/WINH:Address	Determines the higher memory limit of the window in which the DiskOnChip device will be searched. If the /WIN option was used, this option is ignored.
/FAST	Causes TrueFFS to use the multiple byte/word read and write routines in the access layer.

2.2.3. DFORMAT Usage Examples

Example 1

```
DFORMAT C:
```

This formats DiskOnChip, assuming that DiskOnChip is set as drive [C:].

Example 2

```
DFORMAT /WIN:D0000 /BDKF0:CEIMAGE.bin /BDKL1:1M /BDKF1:RegistryFile
```

This formats DiskOnChip with two binary partitions and one disk partition. One binary partition is used to load and run the Windows CE image, and the other is used to store up to 1MB of registry information. The disk partition occupies the reminder of the media.

Example 3

```
DFORMAT /BDKL0:1M /BDKSPL0 /BDKL1:2M
```

This formats DiskOnChip with two binary partitions and one disk partition. One binary partition, 1MB in size, is an SPL partition for the SPL code, and the other is a normal binary partition 2MB in size. The disk partition occupies the reminder of the media.

Example 4

```
DFORMAT /WIN:D0000 /FAT:1
```

This formats the DiskOnChip located at memory address D000. It also places a single FAT copy on DiskOnChip (Many OSs do not use the second FAT copy).

Example 5

```
DFORMAT /WIN:D0000 /BDKLZ2:xxxxxxxx
```

This reformats DiskOnChip with a protected disk partition. If the password provided in the command line is not correct, the format operation fails.

Example 6

```
DFORMAT /WIN:D0000 /BDTLL0:12M /BDTLP1:WC:xxxxxxxx
```

This formats DiskOnChip with two disk partitions. The second partition can be write protected and changed (the protection can be switched later, to either read protection or read/write protection).

Example 7

```
DFORMAT /WIN:D0000 /LOG:BBT.TXT /NOFORMAT
```

This reads the DiskOnChip BBT to the BBT.TXT file without reformatting the device.

3. DINFO UTILITY

The DINFO utility displays information of the DiskOnChip, such as physical size and partition number. Available options include printing/displaying specific information, not just general information. If no options are selected, DINFO displays general information.

3.1. DINFO Syntax

The DINFO syntax is:

```
Dinfo [/WIN:Address] [/FLAG:parameter]
```

Where:

`/WIN`: Memory window address. (If not input, DINFO searches for all connected DiskOnChip devices and displays their information.)

`/FLAG`: See the flag list in Table 7 for details.

3.2. Using DINFO Flags

Table 7 defines the various flags used with the DINFO utility.

Table 7: Typical DINFO Flags

DINFO Option	Description
`/WIN:Address`	Memory address where DiskOnChip is located. The address should be specified in hex (for example, `/WIN:D0000`). If the `/WIN` option is not specified, the utility searches for DiskOnChip devices between default addresses. For example, when running the utilities with M-Systems EVBs for x86-based platforms, the default addresses are 0xC8000 – 0xE0000. All devices that are located will be related.
`/WINL:Address`	Determines the lower memory limit of the window where DINFO will search for the DiskOnChip device. If the `/WIN` option was used, this option is ignored.
`/WINH:Address`	Determines the higher memory limit of the window, where DINFO will search for the DiskOnChip device. If the `/WIN` option was used, this option is ignored.
`/BBT`	Displays Bad Block Table (BBT) information: Number of bad units, list of bad blocks, and the percentage of bad blocks. Note: The bad blocks reported by this option include blocks used for internal TrueFFS usage.
`/OTP`	Displays OTP and device ID information: • OTP Size field: Refers to the maximum capacity available • OTP Used Size: Actual size of the customer OTP • Lock Status: Indicates whether the customer OTP is locked • Unique ID: ID taken from the FOTP.

DINFO Option	Description
/BDTL	Displays the disk partition information for the selected DiskOnChip, including the number of disk partitions on the media, capacity data, format type and protection status for each individual partition: Also provides the number of sectors/heads/cylinders.
/BDK	Displays the binary partition information for the selected DiskOnChip, including the number of partitions on the media, start unit, end unit, capacity data, signature, and protection for each individual partition. Information about sub-partitions is also provided.
/FAST	Causes TrueFFS to use the multiple byte/word read and write routines in the access layer.
/IPL	Provides information about the DiskOnChip IPL: • Type: RAM or ROM • Size: in bytes
/H	Displays the usage screen.
/Log: filename	Saves the information to a log file. DINFO prints the selected information to the screen, and saves the information to a specified file.

Example: Getting DiskOnChip Configuration Information

```
C:\> DINFO   /WIN:d0000
```

Finds DiskOnChip at address 0xd0000 and displays its configuration information. The output is shown in Figure 1.

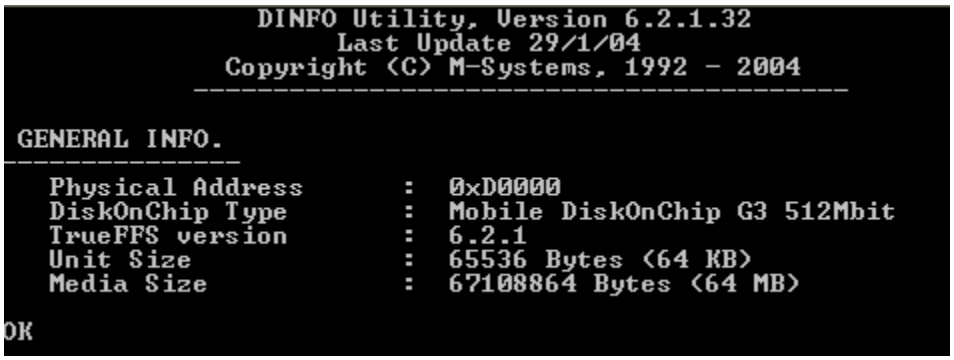

Figure 1: DiskOnChip Configuration Information Displayed Using the DINFO Utility

4. DIMAGE UTILITY

The DIMAGE utility is used for duplicating DiskOnChip, writing an image to a target DiskOnChip, and reading the DiskOnChip image. Duplicating DiskOnChip contents to another device using TrueFFS 6.2.1 can be done directly by copying the contents of a source DiskOnChip to the target DiskOnChip. As a result, all target DiskOnChip contents are identical to the source DiskOnChip, meaning that they have the same functionality when inserted into the target platform.

***Warning**: All target DiskOnChip devices must have the identical part number and capacity as the source DiskOnChip. For example, if the source DiskOnChip is DiskOnChip G3 512Mbit, then the target DiskOnChip must also be DiskOnChip G3 512Mbit.*

The duplication process includes the following two stages:

- Preparing the source DiskOnChip
- Copying the contents of the source DiskOnChip to the target DiskOnChip using the DIMAGE utility

Note: To mass-duplicate DiskOnChip efficiently, it is recommended to use one of the programmer solutions recommended by M-Systems. The availability list of all programmer solution that support DiskOnChip is available on the M-Systems website www.m-systems.com.

The DIMAGE utility can copy:

- From a source DiskOnChip device to a target DiskOnChip device
- From an image file to a target DiskOnChip device
- From a source DiskOnChip device to an image file

4.1. DIMAGE Syntax

The DIMAGE syntax is:

```
DIMAGE { /WINSRC:address or /FILESRC:file } { /WINTRG:address or
/FILETRG:file}  [/SRCBDK#:password] [/SRCBDTL#:password]
[/TRGBDK#:password] [/TRGBDTL#:password]
[/CHBDTL#:password][/CHBDK#:password]
```

Table 8 describes the DIMAGE flag options.

Table 8: DIMAGE Flag Options

DIMAGE Option	Description
`/FILESRC:file`	Name of the source image.
`/FILETRG:file`	Name of the target image.
`/WINSRC:address`	Memory address where the source DiskOnChip is located.

DIMAGE Option	Description
/WINTRG:address	Memory address where the target DiskOnChip is located.
/SRCBDK#:password	Protection key of the protected binary partition on the source device (the default is 0). Inserting this flag disables the read/write protection of the specific binary partition.
/SRCBDTL#:password	Protection key of the protected disk partition on the source device (the default is 0). Inserting this flag disables the read/write protection of the specific disk partition.
/TRGBDK#:password	Sets the protection of the binary partition on the target DiskOnChip device, using the specified password.
/TRGBDTL#:password	Sets the protection of the disk partition on the target DiskOnChip device, using the specified password.
/CHBDK#:password	Changes the password for the binary partition while getting or placing an image.
/CHBDTL#:password	Changes the password for the disk partition while getting or placing an image.
/SINGLE:floor	Copies data from one floor in the image file to the target DiskOnChip device.
/FAST	Causes TrueFFS to use the multiple byte/word read and write routines in the access layer.
/VERIFY	Verifies the programming operation.

4.2. Creating the Source DiskOnChip

The source DiskOnChip includes all target application files and is usually bootable. The preparation process for the source DiskOnChip comprises the following steps:

- Formatting DiskOnChip, using the DFORMAT utility, on the target platform
- Copying all target application files to DiskOnChip
- If required, making DiskOnChip bootable (refer to Section 1)

After the source DiskOnChip device has been properly prepared, follow the guidelines described in the following sections to duplicate it (as many times as required).

4.3. Copying the Source DiskOnChip to an Image File

At this stage, the source DiskOnChip includes all target application files, and it is ready to be duplicated. Use the DIMAGE utility to copy the source DiskOnChip contents to an image file on your hard disk.

To copy the source DiskOnChip to an image file:

1. Power off the system.
2. Insert the source DiskOnChip into the appropriate socket.
3. Power on the system.

4. Run the following command line:

```
DIMAGE  /WINSRC:Address   /FILETRG:image_file_name.IMG.
```

Note: When duplicating a DiskOnChip with active hardware protection, you must use DIMAGE with the **/SRCBDK#** or **/SRCBDTL#** flags in order to disable the protection. The password will be included in the virtual image for later use by DIMAGE.

Example

```
DIMAGE /WINSRC:D0000 /FILETRG:MYDOC.IMG /SRCBDK0:12345678
```

Copies the contents of a source DiskOnChip with a protected disk partition into the file `MYDOC.IMG` on your hard disk.

4.4. Copying the Image File to Target DiskOnChip Devices

At this stage, the contents of the source DiskOnChip are stored in an image file on the hard disk. Copying this image file to the target DiskOnChip results in an identical DiskOnChip target device. The DIMAGE utility is used for this purpose.

To copy the image file to the target DiskOnChip:

1. Power off the system.
2. Insert a target DiskOnChip device, with the same part number and capacity as the source DiskOnChip device, into the appropriate socket.
3. Power on the system.
4. Run the following command line:

```
DIMAGE /FILESRC:image_file_name  /WINTRG:Address
```

   When the duplication process is complete, the target DiskOnChip has the identical contents and functionality as the source DiskOnChip. If the source DiskOnChip is protected, then the target DiskOnChip is also protected with the same attributes and using the same passwords.

5. Repeat steps 1 through 4 as many times as required to duplicate additional target DiskOnChip devices.

Example 1

```
DIMAGE /FILESRC:MYDOC.IMG /WINTRG:D0000
```

Copy the contents of the `MYDOC.IMG` file to the target DiskOnChip.

Example 2

```
DIMAGE /FILESRC:MYDOC.IMG /WINTRG:D0000 /TRGBDK1=newpassw
```

The target image has the second binary partition protected with a password. The password for the second binary partition, **newpassw**, is inserted**.** The image of the source DiskOnChip is then copied to a target DiskOnChip.

4.5. DIMAGE Error Messages

Table 9 describes the DIMAGE error messages.

Table 9: DIMAGE Error Messages

Error Message	Description
Not enough memory	There is not enough system memory to complete the operation.
Can't open file	The system cannot open the image file. This may occur during write or read cycles (for example, if DIMAGE is trying to write the image to DiskOnChip but cannot locate the image file).
Adapter not found	The system cannot find a DiskOnChip adapter.
Can't write to file	The system cannot write to the image file. This may occur in DIMAGE when the media to which the system is writing suddenly becomes unavailable (Disk Full).
Source file format error	DIMAGE detected that the image file being written to the target device is not a valid image file.
Hardware protection	The system is trying to write to a target device with protected partitions. You must first remove the protection (DFORMAT using the `BDTLZ[n]` or the `BDKZ[n]` flag).
Too many bad blocks	The number of bad blocks on the target device is outside the acceptable range. Not enough space is available for writing the image. Note: This type of error does not normally occur unless DiskOnChip was misused.
Wrong parameters passed	There was an error in the parameters given in the command line.

5. UTILITIES PACKAGE FOR WINDOWS 2000 AND WINDOWS XP

The DFORMAT, DINFO and DIMAGE utilities can be obtained in different packages for running on different OSs. For example, there are separate utility packages for DOS and Windows 2000/XP. All general information described in the previous sections, including the utility command line syntax, applies for all utility packages. This section specifically discusses the Windows 2000/XP package, describing the special installation requirements and customization features.

In order to operate one of the utilities under Windows 2000/XP, you must install the virtual-to-physical memory driver (using the installation wizard), open a command prompt window, and execute the utility as described in Sections 2 to 4.

5.1. The Utility DLL Package

5.1.1. Purpose

The DLL provides flexibility for the user, enabling adjusting the core utility to the target platform without re-compiling the utility itself. You can install your own implementation of low-level read/write routines, or customize the progress bar and print functions used by the utility.

The JTAG is an example of user implementation of the low-level read and write routine. Installing JTAG read and write routines enables formatting, identifying and even programming any DiskOnChip device on any type of platform/CPU with a JTAG connection, without changing the utility.

The print routine may require customization when the utility is integrated with an existing tool. The utility inputs and outputs can be redirected to the existing tool user interface.

5.1.2. Package Description

The utility DLL package is comprised of the following folders:

- Install shield
- Binary
- Customization

Install Shield

Applications that run on Windows at the user level cannot access physical memory directly. Instead, they can access virtual memory pointing to that physical memory location. M-Systems supplies a dedicated driver called Mapmemory.dll, which allows the utility .EXE file to map the DiskOnChip memory range into virtual memory. The driver can be easily installed using a dedicated Install Shield installation wizard.

This folder contains the files required to run the wizard that installs Mapmemory.dll. You must install Mapmemory.dll before running the utilities included in the Binary folder.

To install Mapmemory:

1. Launch the installation wizard using the setup.exe program.
2. A Welcome screen is displayed. Click **Next**.
3. The wizard displays a list of the files that are going to be installed. When the subsequent screen is displayed, click **Next**.
4. The driver files are installed in the specified location, and the wizard informs the user that the installation process is complete.
5. When the appropriate screen is displayed, click **Finish**.
6. The wizard prompts you to restart your computer before using the M-Systems utility.

Binary

Each utility (DINFO, DFORMAT, and DIMAGE) is a DLL file based on the TrueFFS SDK. The file is referred to as Utility_dll.dll, where <Utility> can be any of the three component utilities. The Binary folder includes three Utility_dll.dll files: Dinfo_dll.dll, Dformat_dll.dll and Dimage_dll.dll.

In addition, the folder contains a set of Utility.exe files (one for each DLL) that is used for loading and running the DLLs.

To run a utility:

1. Install Mapmemory using the installation wizard.
2. Open a command prompt window.
3. Execute the required utility from the Binary folder.

If you use the standard Windows 2000/XP OS with an M-Systems EVB, the utilities do not require customization, and there is no need to access the Customization folder.

Customization

This folder contains the necessary tools for adapting the utilities to meet specific customer requirements. It contains the following folders:

- **Binary release**: This is an empty folder. Any project in the customization folder will automatically copy its output to this folder.
- **Access_wrapper**: This folder contains a Visual Studios 6 project that produces a second DLL containing the user implementation for some or all of the DiskOnChip access routines. This file may contain functions such as write-byte, read-byte and write-block functions. (See Table 10 for a complete list of user-defined access routines.) The file created is called user_access.dll, and is automatically copied to the Binary release folder.
- **Os_wrapper**: This folder contains a Visual Studios 6 project that produces a third DLL containing the user implementation for system-dependent services, such as the print

function. (See Table 11 for a complete list of user-defined OS routines.) The file created is called user_os.dll, and is automatically copied to the Binary release folder.

- **Dformat**: This folder contains a sample application for the DFORMAT utility. The example is given as a Visual Studios project that:
  - Loads the dformat_dll.dll file.
  - Sets the names of the user_access dll and user_os dll files that will be used.
  - Calls the dformat_dll.dll main entry point with the given arguments.

  The .EXE file is automatically copied to the Binary release folder.
- **Dimage**: This folder contains a sample application for the DIMAGE utility. The example is given as a Visual Studios project that:
  - Loads the dimage_dll.dll file.
  - Sets user_access.dll and user_os.dll as the default user implementation DLLs.
  - Calls the Dimage_dll.dll main entry point with the given arguments.

  The .EXE file is automatically copied to the Binary release folder.
- **Dinfo**: This folder contains a sample application for the DINFO utility. The example is given as a Visual Studios project that:
  - Loads the dinfo_dll.dll file.
  - Sets user_access.dll and user_os.dll as the default user implementation DLLs.
  - Call the Dinfo_dll.dll main entry point with the given arguments.

  The .EXE file is automatically copied to the Binary release folder.

5.1.3. Functionality

For a specific utility, all the components described above (Utility_dll.dll, Utility.exe, user_access.dll and user_os.dll) must be in the same folder.

When Utility.exe is launched, the following occurs:

1. Utility.exe designates user_access.dll and os_access.dll as the default user-implemented DLLs.
2. Utility.exe loads Utility_dll.dll and calls the main entry point (`ExMainDll`) with the user arguments.
3. Utility_dll.dll loads user_access.dll and user_os.dll and looks for the customizable routines. If a specific routine is not found, or if the DLL itself is not present, Utility_dll.dll uses its own default implementation.
4. Utility.exe executes the utility engine using the installed resources.

5.2. Customization Routine API

5.2.1. User-Defined Access Routines

The routines in this section should be implemented in the user_access.dll project. To insert your implementation, uncomment the relevant function and add your code. It is not necessary to customize all of the routines in the file. Routines that are not customized simply use the default implementation.

If you have customized the block access routines, you must use the `/FAST` flag.

Table 10: Functions That May Be Customized in user_access.dll

Function	Description
`void EXAPI Write_Byte(FLByte val,volatile void* address);`	Writes a byte to the address location.
`void EXAPI Write_Word(FLWord val,volatile void* address);`	Writes a word to the address location.
`void EXAPI Write_Dword(FLDword val,volatile void* address);`	Writes a double word to the address location.
`FLByte EXAPI Read_Byte(volatile void* address);`	Reads a byte from the address location.
`FLWord EXAPI Read_Word(volatile void* address);`	Reads a word from the address location.
`FLDword EXAPI Read_Dword(volatile void* address);`	Reads a double word from the address location.
`void memcpy_from_io_8bit(EXBYTE EXFAR *dest, volatile EXBYTE EXFAR * src, EXWORD count)`	Copies a block of bytes from the flash to a destination address.
`void memcpy_to_io_8bit( volatile EXBYTE EXFAR *dest,EXBYTE EXFAR *src, EXWORD count)`	Copies a block of bytes from a memory address to the flash address.
`void memset_to_io_8bit( volatile EXBYTE EXFAR * dest,EXBYTE val, EXWORD count)`	Sets a block of memory to a specified value.
`void memcpy_from_io_16bit(EXBYTE EXFAR *dest, volatile EXBYTE EXFAR * src, EXWORD count)`	Copies a block of words from the flash to a destination address.
`void memcpy_to_io_16bit( volatile EXBYTE EXFAR *dest,EXBYTE EXFAR *src, EXWORD count)`	Copies a block of words from a memory address to the flash address.
`void memset_to_io_16bit( volatile EXBYTE EXFAR * dest,EXBYTE val, EXWORD count)`	Sets a block of memory to a specific value.
`void Get_Search_Range(EXDWORD*low_range,EXDWORD *high_range)`	Gets the search range for DiskOnChip.
`void *Map_Memory(unsigned long dwAddress,unsigned long dwLen)`	Maps a block of physical memory to virtual memory.

5.2.2. User-Defined OS Routines

The routines in this section should be implemented in the user_os.dll project. To insert your implementation, uncomment the relevant function and add your code. It is not necessary to customize all of the routines in the file. Routines that are not customized simply use the default implementation.

Table 11: Functions That May Be Customized in user_os.dll

Function	Description
`ExStatus EXAPI ExOsOpenCmdLineDevice( void** data );`	Opens a command line device.
`ExStatus EXAPI ExOsCloseCmdLineDevice( void* data );`	Closes a command line device.
`ExStatus EXAPI ExOsReadCmdLineDevice( void*data, EXWORD argc, void** retCommand );`	Reads a command from a command line device.
`ExStatus EXAPI ExOsOpenMsgDevice( void** data );`	Opens a message device.
`ExStatus EXAPI ExOsCloseMsgDevice( void* data );`	Closes a message line device.
`ExStatus EXAPI ExOsWriteMsgDevice( void* data, EXCHAR* pStr );`	Writes a string to a message device.
`ExStatus EXAPI ExOsOpenErrorMsgDevice( void** data );`	Opens an error message device.
`ExStatus EXAPI ExOsCloseErrorMsgDevice( void* data );`	Closes an error message line device.
`ExStatus EXAPI ExOsWriteErrorMsgDevice( void* data, EXCHAR* pStr );`	Writes string to error message device
`ExStatus EXAPI ExOsOpenPrgBarDevice( void** data, EXCHAR* pPrgBarName, EXDWORD dwPrgBarSize );`	Opens a Progress Bar device.
`ExStatus EXAPI ExOsClosePrgBarDevice( void* data );`	Closes the Progress Bar line device.
`ExStatus EXAPI ExOsWritePrgBarDevice( void* data, EXDWORD dwDoneSoFar );`	Writes a string to the Progress Bar device.
`ExStatus ExOsOpenFile( void** data, EXCHAR* name, EXWORD wMode );`	Opens a file device.
`ExStatus EXAPI ExOsCloseFile( void* data, EXWORD wMode );`	Closes a file.
`ExStatus EXAPI ExOsWriteToFile( void* data, EXDWORD size, void EXFAR* buff );`	Writes a buffer to a file.
`ExStatus EXAPI ExOsReadFromFile( void* data, EXDWORD size, void EXFAR* buff, EXDWORD* byteRead );`	Reads a buffer from a file.
`ExStatus EXAPI ExOsGetLengthOfSerialFileDevice( void* data, EXDWORD* retLen );`	Gets the length of a file.

Function	Description
`ExStatus EXAPI ExOsOpenUserInputDevice( void** data );`	Opens a user input device
`ExStatus EXAPI ExOsCloseUserInputDevice( void* data );`	Closes a user input device
`ExStatus EXAPI ExOsWaitForCharUserInputDevice( void*data, ExWaitForCharChoices* waitForCharChoices );`	Prints user choices and waits for char from the user input device.
`void* EXAPI ExMemAlloc( EXDWORD size );`	Allocates a memory block.
`void EXAPI ExMemFree( void* memBlock );`	Frees a memory block.
`void* EXAPI ExMemSet( void* dest, EXWORD ch, EXDWORD size );`	Sets a buffer to a specified character.
`EXWORD EXAPI ExMemCmp( void* buf1, void* buf2, EXDWORD count );`	Compares characters between two buffers.
`void* EXAPI ExMemCpy( void* dest, void* src, EXDWORD count );`	Copies a character between buffers.
`EXSWORD EXAPI ExRand( void );`	Gets a random number.
`void EXAPI ExSrand( EXWORD seed );`	Sets the starting point for generating a series of pseudorandom integers.
`EXDWORD EXAPI ExTime( EXSDWORD* timer );`	Gets the system time.
`EXCHAR* EXAPI ExStrTime( EXCHAR* timestr );`	Copies the system time to a buffer.
`EXCHAR* EXAPI ExStrDate( EXCHAR* datestr );`	Copies the date to a buffer, in the format mm/dd/yy.
`EXSDWORD EXAPI ExStrtol( const EXCHAR* nptr, EXCHAR** endptr, EXWORD radix );`	Converts strings to a long-integer value.
`EXDWORD EXAPI ExStrtoul( const EXCHAR* nptr, EXCHAR** endptr, EXWORD radix );`	Converts strings to an unsigned long-integer value.
`EXSWORD EXAPI ExAtoi( const EXCHAR* string );`	Converts strings to an integer value.
`EXCHAR* EXAPI ExItoa( EXSWORD value, EXCHAR* string, EXSWORD radix );`	Converts an integer to a string.
`void EXAPI ExClearScreen( void );`	Clears the screen.
`EXWORD EXAPI ExStrLen( const EXCHAR* string );`	Gets the length of a string.
`EXCHAR* EXAPI ExStrCat( EXCHAR* strDestination, const EXCHAR* strSource );`	Appends a string.
`EXWORD EXAPI ExCreateDirectory( const EXCHAR* dirName );`	Creates a new directory.
`EXWORD EXAPI ExGetDrive( void );`	Gets the current disk drive.
`EXWORD EXAPI ExChangeDrive( EXWORD drive );`	Changes the current working drive.
`EXWORD EXAPI ExGetDiskFree( EXWORD drive, ExDiskInfoRecord* diskInfo );`	Gets information regarding the number of available sectors, their size, number of clusters, and sector per cluster.
`EXCHAR EXAPI ExToupper( EXCHAR ch );`	Converts lower case to upper case.

6. KNOWN LIMITATIONS

TrueFFS 6.2.1 has the following known limitation:

- The DFORMAT firmware flags are not yet supported by the utilities provided with TrueFFS 6.2.1. The DFORMAT firmware flags will be supported in the next version of the DiskOnChip software utilities package.

It is recommended to review the enclosed readme.txt file for additional known limitations.

7. ADDITIONAL DOCUMENTS AND TOOLS

A variety of application notes, user manuals, data sheets and tools are available from M-Systems for use with DiskOnChip products under various OSs and environments.

These documents are available through M-Systems distributors, directly from M-Systems offices worldwide, and on the M-Systems website (www.m-systems.com).

Document/Tool	Description
Various Data Sheets	DiskOnChip products
Installation Manual	Using DiskOnChip with Windows CE
Developer Guide	DiskOnChip Boot Software Development Kit 6.x (BDK)
Developer Guide	DiskOnChip Driver Extended Functions Based on TrueFFS 6.x

M-Systems
Flash Disk Pioneers

HOW TO CONTACT US

USA

M-Systems Inc.
8371 Central Ave, Suite A
Newark CA 94560
Phone: +1-510-494-2090
Fax: +1-510-494-5545

China

M-Systems China Ltd.
Room 121-122
Bldg. 2, International Commerce & Exhibition Ctr.
Hong Hua Rd.
Futian Free Trade Zone
Shenzhen, China
Phone: +86-755-8348-5218
Fax: +86-755-8348-5418

Japan

M-Systems Japan Inc.
Asahi Seimei Gotanda Bldg., 3F
5-25-16 Higashi-Gotanda
Shinagawa-ku Tokyo, 141-0022
Phone: +81-3-5423-8101
Fax: +81-3-5423-8102

Europe

M-Systems Ltd.
7 Atir Yeda St.
Kfar Saba 44425, Israel
Tel: +972-9-764-5000
Fax: +972-3-548-8666

Taiwan

M-Systems Asia Ltd.
Room B, 13 F, No. 133 Sec. 3
Min Sheng East Road
Taipei, Taiwan
R.O.C.
Tel: +886-2-8770-6226
Fax: +886-2-8770-6295

Internet

www.m-systems.com

General Information

info@m-sys.com

Sales and Technical Information

techsupport@m-sys.com