AP-DOC-020
Application
Note

DiskOnChip
Boot Developers Kit

Dmitry Shmidt & Raz Dan

April-99
---------- Rev. 1.21

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

Booting a system from the DiskOnChip is of utmost importance for engineers that design the DiskOnChip into their target platform. It allows them to use only a very small boot ROM and store the Operating System itself on the DiskOnChip.

This application note is intended for system integrators designing with the DiskOnChip2000, DiskOnChip Millennium or DiskOnChip DIMM. It will explain how to take advantage of a special DiskOnChip feature that allows booting virtually any system from it. The application note will briefly discuss the hardware requirements of the DiskOnChip. Furthermore, it will explain the principal of operation and the practical aspects of the boot solution. Finally, it will discuss the utilities required for formatting and updating the DiskOnChip.

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

2. Hardware Requirements for the DiskOnChip

The DiskOnChip can be easily connected to any CPU bus. The minimum requirements are a 13-bit address bus, an eight-bit data bus, and three control signals. These control signals are identical to the SRAM or EEPROM signals for reading, writing and chip enable. They are typically found on every hardware platform and can be easily interfaced with.
The following figure shows a simplified structure of a typical hardware system, focusing on the DiskOnChip connections. For a detailed discussion of the DiskOnChip hardware environment, please refer to the DiskOnChip data-sheet 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*".

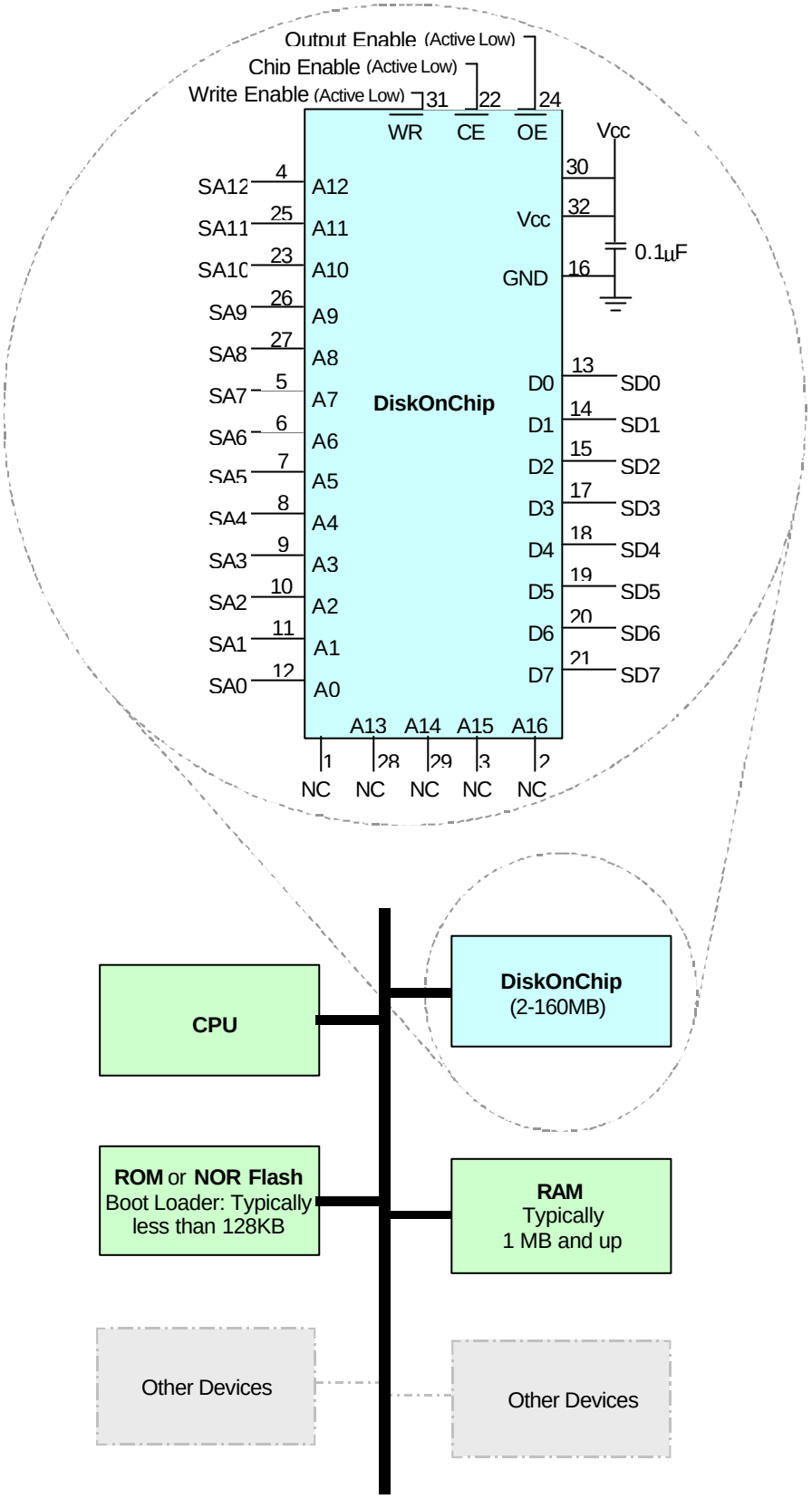

Figure 1 – Block Diagram of a Typical System with a DiskOnChip

3. Booting Up the System from the DiskOnChip

The generic boot solution presented in this application note is based on the following components:

- The DiskOnChip. Its size must be large enough to contain the Operating System image and the required file storage area. The DiskOnChip is partitioned into two sections:
  - The first partition holds the OS image. This partition will be called "BDK Partition" in the remainder of this application note
  - The second partition holds the file storage area, which is accessed through a filesystem
- A very small ROM or NOR Flash, containing the following:
  - The code required for minimal system initialization. This code depends on the OS and the platform. This code should, at the least, initialize the chip select unit such that the ROM, RAM and the DiskOnChip can be accessed. Other components, such as display, keyboard and mouse do not have to be initialized at this time. Since this code depends heavily on the OS and the platform, this subject won't be discussed in detail in this application note.
  - The code for copying the OS image from the DiskOnChip to RAM is referred to as "Boot Loader". This code is supplied along with this application note, and will be thoroughly discussed in the next sections.
  - The code for executing the Operating System, once it is in RAM. This code is OS dependent, and might include switching processor modes or other special system initialization. This code is beyond the scope of this application note.

Upon power-up, the code stored in ROM will be executed first. It will do the necessary initializations, copy the Operating System image from the "BDK Partition" of the DiskOnChip into RAM, and execute it. The following drawing outlines the flow of control. Note that the file storage area of the DiskOnChip is not involved in the boot process.

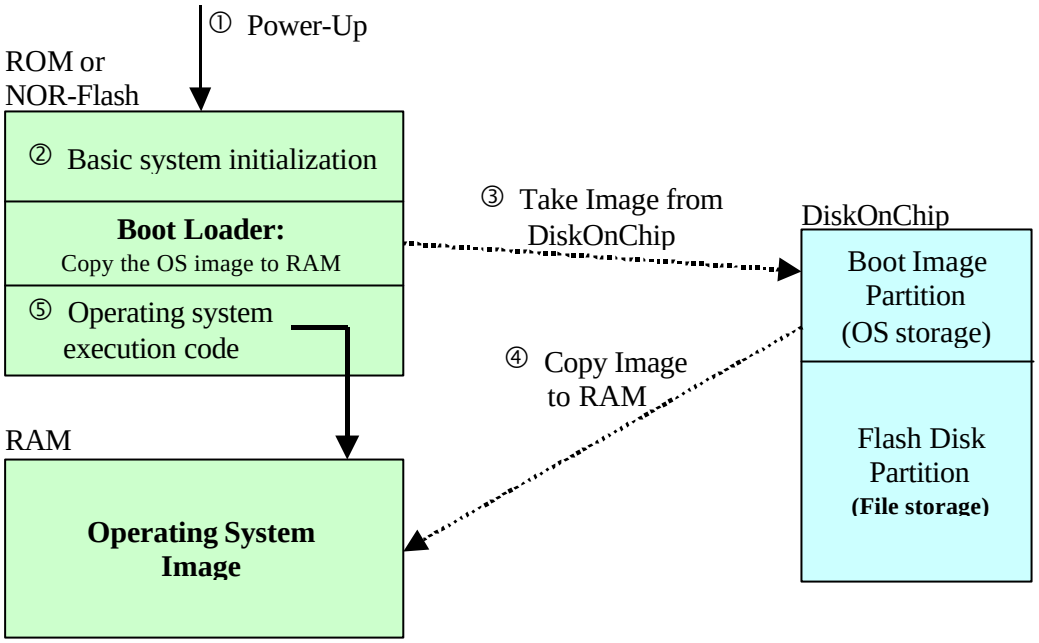

Figure 2 – Boot Process

4. The Boot Loader

4.1 The Operating System Image Storage Format

The BDK Partition on the DiskOnChip is composed of units. The unit size depends on the size of the Erase Block within the flash chip. Different flash chips have different Erase Block sizes. You can use the function **getBootPartitionInfo**, described below, to retrieve the unit size. Typically the units are 8 KB long. Each unit is marked with a signature. Bad units, if any, are not used, and have no signature. The signature is made up of 4 distinct characters, followed by a 4-digit hexadecimal number. This number starts at 0 for the first unit and is increased by one for each consecutive unit. The last unit is always marked by the hex number FFFF, independent of it' s ordinal number.
The 4-character prefix is determined while creating the Operating System image part of the DiskOnChip. See section 5 for further details.

Note: *When using DFORMAT utility version 1.20 or up, the signature offset is by default 8. Previous versions have a default value of 0. Make sure that the #DEFINE SIGN_OFFSET in the file DOC_BDK.H is set to the same value (see Figure-3).*

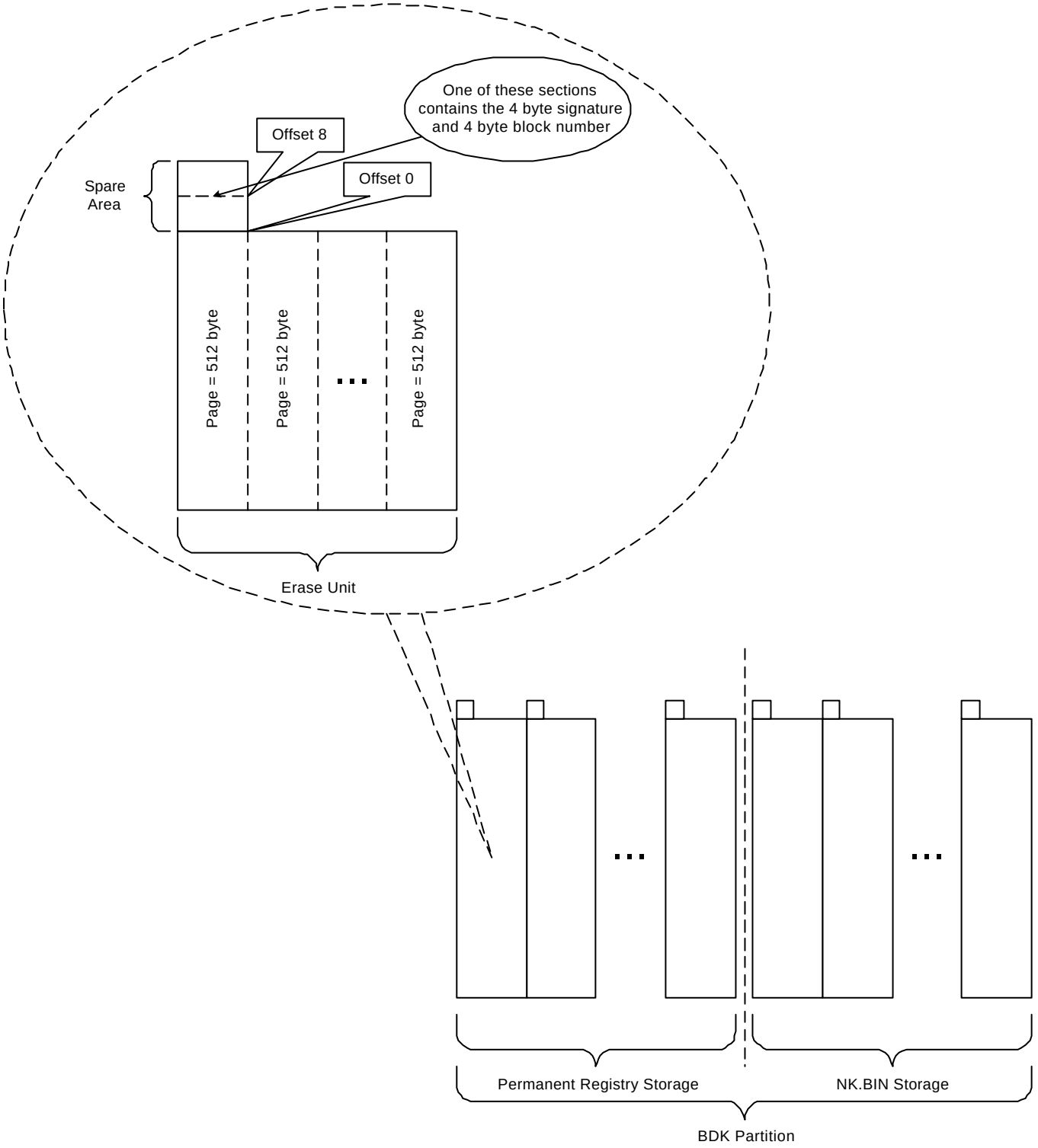

Figure 3 – BDK Partition Structure

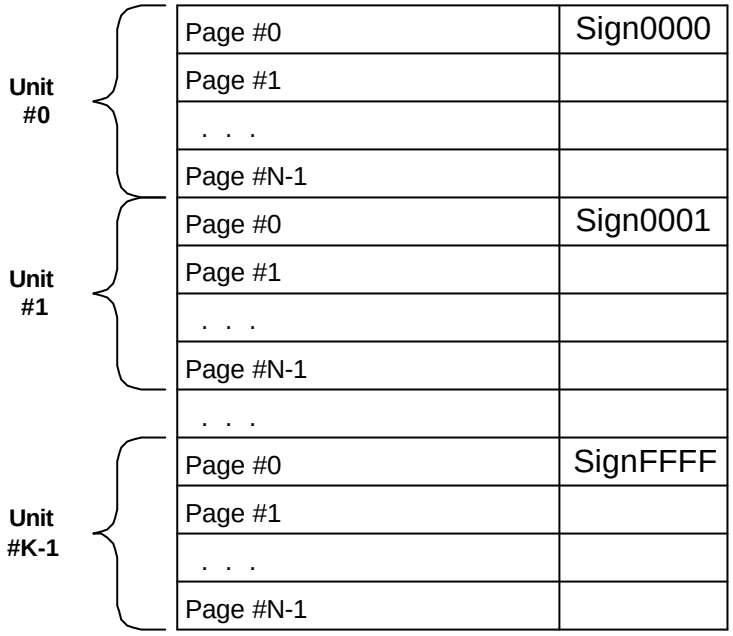

Figure 4 – Block and Page Numbers

4.2 The Boot Loader Code

The simple boot loader code includes a stripped down version of the DiskOnChip driver, consisting only of the modules that read data from the DiskOnChip. The code is customizable to a great extent, and can be adapted to fit your target platform. The function that provides the entry point is named **`copyBootArea`**. For details on its parameters, see the function header. All functions and user types are placed in the `DOC_BDK.H` header file. All functions return 0 (or flOK) after successful processing. Error codes are described in the header of every function in the file `DOC_BDK.C`.

Following are the steps taken by the boot loader:

1) Search for the DiskOnChip window in memory. Two `#DEFINE` statements control the search range. They should be set to reflect the memory region in which the DiskOnChip can reside. `DOC_LOW_ADDRESS` determines the start address, while `DOC_HIGH_ADDRESS` determines the end address. In addition there is a global variable, `docWindow`, that can be used during runtime to define a specific search address. By setting the value of `docWindow` to something other than zero you can force the search routine to scan only one address. This is convenient for special tools or if you can retrieve the base address of the DiskOnChip from some external storage area.

   Note: To use the variable `docWindow` you need to define in your application the following:
   `EXTERN UNSIGNED LONG docWindow;`

   `DOC_WINDOW_SIZE` determines the window size used to access the

DiskOnChip. This is typically 8KB, but the size is modified by the value of `DOC_ACCESS_TYPE`. For example, in a system which uses 32-bit access the default window size will be 32KB.

All these variables are placed in the header file `DOC_BDK.H`. When setting the first two variables, keep in mind that the end address actually determines the last address that can be used as an ending address for a window. If both address variables are equal, no search will be performed and it is assumed that the DiskOnChip resides in that address. The function **findDiskOnChip** can be used separately to identify the DiskOnChip in memory.

Function usage:

```
FLStatus findDiskOnChip(unsigned long FAR2 *docAddress,
                        unsigned long FAR2 *docSize );
```

Where ***docAddress*** will return the physical DiskOnChip base address and ***docSize*** will return the unformatted size of the DiskOnChip in bytes.

2) Search for the storage units as described in section 4.1. The units are copied into RAM. A byte-checksum is calculated while copying. It may be accessed using the pointer transferred to the entry point function.

Function usage:

```
FLStatus copyBootArea( unsigned char FAR1 *startAddress,
                       unsigned short startBlock,
                       unsigned long areaLen,
                       unsigned char FAR2 *checkSum,
                       char FAR2 *signature );
```

Where ***startAddress*** is a pointer to the starting location area in the RAM, ***startBlock*** is the number of the first block of the specified BDK Partition on the DiskOnChip (usually 0 block), ***areaLen*** is the length of the specified BDK Partition you want to read, ***checkSum*** is the pointer that will return the checksum of the image and ***signature*** is signature of the BDK Partition as described in section 4.1.

Note: *you can use this function to copy any area of the DiskOnChip Boot Image.*

4.3 Boot Loader Code Customization

A few issues in the boot loader code need special attention. They all appear at the beginning of the source file.

4.3.1 Far pointers

Far pointers are usually associated with Intel 80x86 architectures, and are usually irrelevant on other architectures. If your processor/compiler has no far pointers, i.e. it uses a flat memory model, then define the variable `FAR_LEVEL` as `0`.
On 80x86 architectures, you need to define which pointers are far, by specifying a value of `0` to `3` for the variable `FAR_LEVEL`. If you are using a flat 32-bit addressing model, define `FAR_LEVEL` as `0`. Otherwise:

- If the RAM window and the DiskOnChip window are near (in the boot loader data segment), define `FAR_LEVEL` as 0.
- If only the DiskOnChip window is far, define `FAR_LEVEL` as 1.
- If both the DiskOnChip window and the RAM window are far, define `FAR_LEVEL` as 2.
- If the DiskOnChip window, the RAM window and the pointer(s) transferred to the copying function are far, define `FAR_LEVEL = 3`.

4.3.2 Pointer arithmetic

Here, you are asked to supply two macros (or short routines) for two pointer arithmetic operations that may depend on your processor or compiler.

`physicalToPointer(address, length)`

This is a macro that takes a number that specifies a physical address, and returns a (far) pointer that points to that address. In a real-mode Intel 80x86 architecture, this macro would convert a linear address (say hex 0D8000), to a segment:offset far pointer (say 0D800:0).

In a system that uses virtual addressing, this macro would be more complex. It would need to know the correspondence between physical and virtual addresses, and would have the task of mapping the physical address to the virtual address-space so that a pointer can be returned for it. For this reason the second parameter `length` is supplied. This parameter reflects the amount of space starting at the address that should be mapped to the virtual address-space.

`addToFarPointer(pointer, increment)`

This macro defines how to add an offset to a (far) pointer and returns a new (far) pointer. The increment may be as large as the socket window size. In most cases this can be coded simply as ((char *) (pointer)) + (increment), but sometimes this might be inappropriate. For example: in a real-mode 80x86 system, where the increment might be larger than 64KBytes, there is a need to perform “huge” pointer arithmetic, because segment offsets may not exceed 64KBytes.

`freePointer(pointer)`

This macro frees an allocated pointer. It might be required only for environments that use virtual mapping.

4.3.3 Delay

A short delay function is required for the operation of the DiskOnChip. This can either be implemented by a library function, having a millisecond resolution, or by a specifically calibrated delay loop. The required delay is no more than a few milliseconds.

4.3.4 DiskOnChip Access

DiskOnChip is 8-bit device, therefore it demands 8-bit data access. In some systems the DiskOnChip window has 16-bit or 32-bit access. Besides hardware changes you need to access the DiskOnChip correctly in software. For compile-time version you simply need to put the correct bus access width into the file `DOC_BDK.H` file as follows: `#DEFINE DOC_ACCESS_TYPE 8`
In order to get the ability to change bus access width in run-time, you need to uncomment `#DEFINE USE_FUNC_FOR_ACCESS` and to define in your module
`EXTERN UNSIGNED SHORT DOC_ACCESS_TYPE`
You need to assign to `DOC_ACCESS_TYPE` the bus access width (8, 16 or 32) before any call to the BDK functions.

Note: *Default value is 8.*

Note: *It is possible to customize DiskOnChip access functions in module DOC_BDK.C for BIG_ENDIAN processor or NON_MEMORY_MAPPED DiskOnChip access.*

4.4 Advanced Boot Image Processing

When you need more complicated DiskOnChip boot image processing or you need to implement several of your own functions like the Windows CE functions `ReadRegistryFromOEM` and `WriteRegistryToOEM` for example, you can use the BDK write functions.
By deleting the remarks from `#define WRITE_IMAGE` in the file `DOC_BDK.H`, you will be able to use the following functions for writing to the BDK Partition of the DiskOnChip:

- ***`getBootPartitionInfo`*** (obtaining information about the BDK partition)
- ***`writeBootAreaInit`*** (initializing the required data structures)
- ***`writeBootAreaBlock`*** (writing data to the BDK partition)

Note: Be careful. These functions erase blocks before writing, therefore you can loose your data!

Function Usage:

```
FLStatus getBootPartitionInfo( unsigned long FAR2 partitionSize,
                               unsigned long FAR2 *unitSize,
                               char FAR2 *signature );
```

Where ***partitionSize*** returns the current size of the BDK Partition in bytes, ***unitSize*** returns the minimum unit size that can be updated, ***signature*** is the signature of the BDK partition.

Function usage:

```
FLStatus writeBootAreaInit( unsigned short startBlock,
                            unsigned long areaLen,
                            unsigned char updateFlag,
                            char FAR2 *signature );
```

Where ***startBlock*** is the block of the specified BDK Partition you want to start from, ***areaLen*** is the specified BDK Partition length and ***signature*** is the BDK Partition signature. ***updateFlag*** can get one of the two following values: `BDK_COMPLETE_IMAGE_UPDATE` or `BDK_PARTIAL_IMAGE_UPDATE`. The first is used in order to update for example NK.BIN storage (*see Figure-3*) and the second one is used in order to update for example Permanent Registry Entries storage (*see Figure-3*). The difference between these two modes of update is that the first one finishes the update with [Signature]FFFF (i.e. `BIPOFFFF`) and the second one only updates the image, setting current numbers to the blocks.

Function usage:

```
FLStatus writeBootAreaBlock( unsigned char FAR1 *buffer,
                             unsigned short bufferLen );
```

Where ***buffer*** is the current buffer you want to write to and ***bufferLen*** is the current buffer length.

Note: The image is updated one buffer at a time. The buffer size must be less than or equal to the Unit Size. Most buffers should be equal between themselves and should be equal to Page Size (512 Bytes) or a multiple integer of Page Size. However the last buffer may be less than the previous.

4.5 Example source code

In the source code of the BDK you can find two function examples that show you how to read the BDK Partition into a file and how to copy a file into the BDK Partition. To compile these functions you need to delete the remarks from:
`#define BOOT_TO_FILE` in the file `DOC_BDK.H`.

Function Usage:

```
FLStatus copyBootAreaFile( char FAR2 *fname,
                           unsigned short startBlock,
```

```
                                unsigned long areaLen,
                                unsigned char FAR2 *checkSum,
                                char FAR2 *signature );
```

Where ***fname*** a is pointer to a file name, ***startBlock*** is the starting block of the specified BDK Partition on the DiskOnChip (usually block 0), ***areaLen*** is the length of specified BDK Partition you want to read, ***checkSum*** is the pointer that will return the checksum of the image and ***signature*** is the signature of the BDK Partition as described in section 4.1.

Function Usage:

```
FLStatus writeBootAreaFile( char FAR2 *fname,
                            unsigned short startBlock,
                            unsigned long areaLen,
                            char FAR2 *signature );
```

Where ***fname*** is a pointer to a file name, ***startBlock*** is the starting block of the specified BDK Partition on the DiskOnChip, ***areaLen*** is the length of specified BDK Partition you want to write and ***signature*** is the signature of the BDK Partition as described in section 4.1.

5. Utilities

In order to create the BDK partition, you need to use one of the following DOS utilities.

5.1 DFORMAT

Before the DiskOnChip can be used, it must be low-level formatted. Formatting initializes the media and writes a new and empty DOS file system on it. When formatting is complete, the media contains only a root directory.

In order to prepare the DiskOnChip for booting, the following steps are taken by DFORMAT:

1. Format the DiskOnChip, preserving the area for the BDK Partition.
2. Install the DiskOnChip firmware.
3. Copy the Operating System image to the BDK Partition on the DiskOnChip.

Note: The region reserved for the Operating System image is automatically set, according to the image file size. To prepare for future versions of the Operating System, versions that might require a larger image file, it is recommended to artificially increase the size of the image file.

Following is a description of the command line options that can be used with the DFORMAT utility. Please refer to the DiskOnChip User's Manual for detailed information about the standard usage and command line options of this utility.

/S:Firmware	The firmware file (driver) to be written to DiskOnChip. Usually the file extension is .EXB
/BDKF:<file name>	This flag specifies the Operating System image file name.
/BDKN:XXXX	This flag specifies the 4-character prefix (signature) of the BDK Partition. XXXX can be replaced by any combination of 4 ASCII characters. Use capital letters. Default: BIPO
/BDKL:partition size	Size of BDK Partition.
/O:signature offset	This flag specifies the BDK Partition signature offset. Can be 0 or 8. Default: 8. *Note: From firmware version 1.20 and onwards, the default value is 8. Previous firmware versions hold a default value of 0.*
/NODOS	Do not create a DOS FAT file system while formatting. Only low-level format is performed. This is useful for non-DOS applications.

Note: *All sizes specified in DFORMAT options are in bytes if specified as simple numbers, in KBytes if specified with the suffix **K**, or in megabytes if specified with the suffix **M**.*

Examples

1. Formats the DiskOnChip, located at address D000h, with the firmware named `docimage.bin`, an Operating System image file named NK.BIN and the signature "`WCE*`". Signature offset is 0.

   ```
   DFORMAT /WIN:D000 /S:docimage.bin /BDKF:nk.bin /BDKN:WCE* /O:0
   ```

2. Formats the DiskOnChip, located at address D000h. Creates a BDK Partition of 5MB size.

   ```
   DFORMAT /WIN:D000 /BDKL:5M
   ```

3. Formats the DiskOnChip, located at address D000h. Creates a BDK Partition of 5MB and writes the file NK.BIN to it.

```
DFORMAT /WIN:D000 /BDKF:nk.bin /BDKL:5M
```

Note: The size of the file NK.BIN can be up to 5MB. Operation will fail if the size is larger than the BDK Partition size.

4. Formats the DiskOnChip, located at address D000 without firmware or BDK Partition. If they previously existed, they will be deleted.

```
DFORMAT /WIN:D000 /BDKF:! /S:!
```

5.2 DUPDATE

This utility updates the DiskOnChip firmware and the Operating System image without altering the file-system area. The usage is similar to that of the DFORMAT utility. The same command line parameters can be used.
Since DUPDATE does not create a new BDK Partition, the only useful flags are : **/S:** and **/BDKF:**

*Note: **DUPDATE** does not access the file-system area, therefore user files are not erased.*

Example

Updates the BDK Partition with the file newnk.bin and the firmware on the DiskOnChip, located at address D000h.

```
DUPDATE /WIN:D000 /S:firmware.exb /BDKF:newnk.bin
```

Note : The file newnk.bin cannot be larger than the BDK Partition.

6. Accompanying Files

The following files accompany this application note:

DOC_BDK.H .....: The boot loader H header code
DOC_BDK.C .....: The boot loader C source code
DFORMAT.EXE .....: DiskOnChip formatting utility
DUPDATE.EXE .....: DiskOnChip updating utility

7. Additional information and Tools

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-21	Using the DiskOnChip 2000 with Linux O/S
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

USA - M-Systems Inc., Phone: 510-413-5950, Fax: 510-413-5980, email: info@m-sys.com
Taiwan - M-Systems Asia, Phone: 886-2-2550-1741, Fax: 886-2-2550-1745
Japan – M-Systems Japan, Phone: 81-3-3445-9042, Fax: 81-3-3445-9045
Israel - M-Systems Ltd, Phone: 972-3-647-7776, Fax: 972-3-647-6668
http://www.m-sys.com