The DLion Low Level Disk Drivers

Last revised: 26-Jun-84 23:13:59 by Mitch Lichtenberg

This file is an attempt to explain the operation of the Dandelion rigid disk interface, the microcode, and how Lisp constructs and uses disk IOCBs to perform disk operations.

DISK DRIVE INTERFACE

The Dandelion's central processor divides its time among the high speed I/O devices: the ethernet, the rigid disk, the I/O processor, and the display. The "I/O Page" is located in a well-known (to the microcode and Lisp) area of virtual memory, and it holds locations for communication between the different "micro-tasks."

Currently, the Dlion's Disk IOCB is the second word on the I/O page (that is, `(\ADDBASE \IOPAGE 1)`). Memory locations placed on this page must be up to 16 bits long, which constrains the address to be within the first 256 pages.

The IOCB page is used to store parameters and other information that is picked up by the microcode. When one wants to initiate an I/O operation, it can be done by depositing the parameter block onto the IOCB page (somewhere) and then placing the location (16 bits) of the parameter block onto the device's "mailbox" on the I/O page. When the device notices that something has been deposited onto its special I/O page location, it will read in the parameters, execute the operation, and reset the flag to zero to indicate that the operation is complete. (*Note: This is not completely true for the disk.). So, device I/O in Lisp usually looks like the following:

```
(\BLT (\ADDBASE \IOCBPAGE IOCBDisplacement) IOCB IOCBLen)
... or an alteration of an existing IOCB on the IOCB page ...
(\PUTBASE \IOPAGE DeviceCSBDisplacement
        (\LOLOC (\ADDBASE \IOCBPAGE DeviceIOCBDisplacement)))
(until (ZEROP (\GETBASE \IOPAGE DeviceCSBDisplacement)))
```

The `\PUTBASE`s and `\GETBASE`s usually come in the form of record package macros.

The reason why the disk does not follow the usual `\IOPAGE` convention of resetting its CSB to zero is because the drivers were meant to cause interrupts when disk I/O is finished. Lisp does not currently utilize this feature, so to poll the IOCB to detect when it has been completed, the IOCB status is set with some unused bits activated, and when the IOCB completes, the disk microcode will fill in the status and wipe out those bits as a side effect.

DISK IOCBS

To make life easier on the Dandelion's disk microcode, the implementors thought it to be a good idea to have the microcode emulate some kind of primitive "instruction set." So, when you want disk I/O, you have to write a little "program" in "IOCB Machine Language" to accomplish it.

Fortunately, these IOCBs are still around after booting, and Lisp leaves them in place but changes the fields to do more complicated operations.

The entire IOCB page is divided into three sections:

1. The Data Field:

This portion of the IOCB page contains mostly scratch space for the microcode and information for the programmer. Of particular interest here are the Header and Label "template" fields. For read operations, these fields are modified by the microcode, whereas on verify and write operations, they are just read.

The header field corresponds to the header records that were written on the disk when it was formatted. They contain identification information for the microcode - the sector's track, cylinder, and head numbers. These records are *never* written to. The usual operation on this field is verify, and it is primarily used to indicate to the microcode which sector is which on a given track, and to provide a security mechanism for the microcode. This template is also used to store the current cylinder number for the disk drive, and it is the place where the cylinder, track, and sector numbers are stored prior to a disk operation.

The label field is more general-purpose. In the pilot world, the ID number of a file and the page's relative location within the file are stored in the label field. For booting, a coded pointer is also stored here to lead the boot microcode from one sector to the next.

Note that the header and label fields *must* be in these locations on the IOCB page. (they may not be somewhere else in VM). The pointers to these fields from within the parameter blocks are only 16 bits long, so the headers and labels must be kept here.

2. The Parameter Area

The second portion of the IOCB page is the parameter areas for the IOCB programs. There are two of these IOCB parameter areas left over after booting, but Lisp only uses one of them. The information contained in the parameter block includes the run length, the type of operation to use (which operation, read/write/verify, to use on each field), the virtual page number of the disk buffer, and information on how to handle errors. They must be aligned on 16 word boundaries, due to the way that they are loaded into the micro-registers inside the CP.

3. The IOCBs

The third portion of the IOCB page is contains the actual IOCBs themselves. There are two basic types of IOCBs:

1. Seek IOCB:

The Seek IOCB is complete as it stands. (it has no parameter areas to read in like the transfer IOCB has). The Lisp code fills in the fields for the number of cylinders and the direction to seek, and the IOCB's code steps the drive head in the given direction for the given number of steps. There are no verify operations on seeks, so it is the programmer's responsibility to remember which track number the head is currently positioned at. If a disk drive gets lost, a recalibrate operation is necessary. This can be accomplished by setting up an IOCB to step out one track, and continually running it until the `Track00` bit of the disk controller status register becomes T. This register can be read with the function `(\DEVICE.INPUT 3)` and the fields can be found in the record `DLDISK.STATUS` on `DISKDLION`.

2. Transfer IOCB:

The Transfer IOCB reads in a parameter area of 17 words, executes the transfer operation, and exits.

DISK IOCB MACHINE LANGUAGE

As was mentioned before, the disk microcode emulates a very small instruction set (to keep the code size down and increase flexibility). This instruction set is as follows:

Opcode	Operation
`8000 xxxx`	Send word `xxxx` to disk controller register `KCtl`.
`0007 ssss`	Set status bits from `ssss`
`0000 aaaa`	Increment number in location `aaaa`, and skip if zero.
`0002 aaaa`	Unconditional jump to location `aaaa`.
`0006`	Finish up IOCB.
`0400 aaaa`	Write status to location `aaaa`.
`0005 aaaa`	Load parameter table from locations starting at `aaaa`.
`0800`	Transfer a run of pages, skip of no error

Some interesting "`8000 xxxx`" commands follow:

Opcode	Operation
`8000 0422`	Wait for pending seeks to complete. (InsureSeekComplete)
`8000 0420`	Seek step IN (positive direction)
`8000 04A0`	
`8000 0460`	Seek step OUT (negative direction)
`8000 04E0`	

Inside the parameter areas, the following "code numbers" are important:

Code	Meaning
`001E`	Abort on NotReady, WriteFault, Overrun, or CRC errors
`001C`	Abort on NotReady. WriteFault, Overrun
`001F`	Abort on NotReady, WriteFault, Overrun, CRC, or Verify

OTHER NOTES, RESTRICTIONS, ETC.

To specify the length of the data field, "`8100`" is used instead of "`0100`". Setting the high bit of the data length field causes the microcode to increment the virtual page number after each page is transferred. This is used primarily for multiple page runs.

The length of the header and label fields must be decremented for verify operations.

It is impossible to follow a write operation with anything other than a write operation. That is, if you write the label field you *must* write the data field. Otherwise, the tail of the label write operation will trash the data field. (Something in the microcode or disk controller causes this, and it cannot be avoided!)

The files `[Eris]<Lispcore>Dlion>DiskBootIOCBs.bravo` and `[Eris]<Lispcore>Dlion>DiskTest.dm` contain many sample IOCBs. They are invaluable anyone tinkering with the Dlion disk system.

THE LISP DLION DISK HEAD

The heads for the DLion disk are stored on the file `DISKDLION` in the sources directory. The following is a description of the functions in this file and their purposes:

`(\DL.DISKINIT)` [Function]

Determines the shape of the disk drive and sets up variables as follows:

```
\DISKTYPE:     One of \SA4000, \SA1000, \Q2040, \Q2080
SEC/HD:        Sectors per head on this disk drive
SEC/CYL:       Sectors per cylinder on this disk drive
```

The data for each drive follows:

Drive	Sec/Hd	Sec/Cyl	Heads
SA4000	28	224	8
SA1000	16	64	4
Q2040	16	128	8
Q2080	16	112	7

`(\DL.RECALIBRATE)` [Function]

Attempts to find track zero of the disk drive by repeatedly stepping the drive out and checking the status word for `Track00` indication. If more than 512 steps are made and `Track00` still is not true, a call to `RAID` is made.

`(\DL.DISKSEEK CYL)` [Function]

Seeks disk drive to cylinder `CYL`, and updates information in the header template.

`(\DL.TRANSFERPAGE DA BUFFER MODE LABEL RUNLENGTH NORAIDFLG)` [Function]

"User" entry (that is, DLion file system entry) to the disk head. `DA` is the disk address, which may be a fixp. The remaining args are the same as those for `\DL.XFERDISK`, as described below:

`(\DL.XFERDISK CYL HD SEC BUFFER MODE LABEL RUNLENGTH NORAIDFLG)` [Function]

Starts a Disk I/O operation. The argument format is meant to be compatible with the old `\DL.XFERDISK`. This minimizes the confusion with changing the swapper. New features include the ability to work with labels and an error recovery mechanism. If a disk error occurs, the `\DL.XFERDISK` function will retry the operation up to ten times. If it fails, it will do a `(\DL.RECALIBRATE)` first and try ten more times before finally calling `RAID`. Arguments are as follows:

`CYL`	Cylinder number of disk address
`HD`	Head number of disk address
`SEC`	Sector number of disk address

Note: These numbers will be normalized automatically. For example, it is permissible to transfer Cylinder 0, Head 440, Sector 1215. `\DL.XFERDISK` will change that into a meaningful value. This is how the swapper works - see below.

`BUFFER` A pointer to the first `page` that will be used in the disk operation. **Note:** The page must be *locked down*, *touched* (referenced), and *dirty*, or else the swapper will not perform properly!

`MODE` One of the following:

`NIL`	Read pages , read labels (`VRR` operation)
`T`	Write pages, read labels (`VRW` operation)
`VRR`	Read pages, read labels
`VVR`	Read pages, verify labels
`VVW`	Write pages, verify labels
`VWW`	Write pages, write labels
`VRW`	Write pages, read labels (used by swapper)

`LABEL` A pointer to the label record (10 words), or `NIL` if you don't want to use a label record. The label must be locked down to prevent page faults inside the `\DL.XFERDISK` routine.

`RUNLENGTH` The number of consecutive pages to transfer, or `NIL` for one. There are restrictions on multiple page runs: To do a multiple page run, the virtual page numbers of the buffer pages must be sequential, and the run may not cross cylinder boundaries. (See function `\CYLBOUNDCROSSP` below).

`NORAIDFLG` Normally, `\DL.XFERDISK` will bomb after failing to do an I/O operation ("failing" does not include verify errors). (It will call `RAID`). To supress this, set `NORAIDFLG` to `T` and disk errors will be returned as status to the caller.

`(\CYLBOUNDCROSSP DA1 DA2)` [Function]

Predicate returns T if DA1 and DA2 are on different cylinders, NIL otherwise. Note: This function is not locked down.

`(\DL.DISKOP IOCB)` [Function]

Passes IOCB to the disk microcode (which starts the I/O operation), waits for it to complete, and returns the status.

`(\D2V CYL HDSEC)` [Function]

Returns the disk address of the page on cylinder CYL and with encoded head and sector information in HDSEC (left byte is head number, right byte is sector number)

`(\V2HDSEC DA)` [Function]

Returns encoded head and sector information from disk address.

`(\V2CYL DA)` [Function]

Returns cylinder number from disk address.

`(\DL.ACTONVMEMPAGE FILEPAGE BUFFER WRITEFLG)` [Function]

Performs a file operation on the virtual memory file. `FILEPAGE` is a file relative page number to transfer, `BUFFER` is the page number for the transfer, and `WRITEFLG` is passed to `\DL.XFERDISK` as the `MODE` parameter. It is usually `T` or `NIL`. This function is implemented by figuring the starting address triple of the beginning of the VMEM file and computing the number of pages into the disk from there that the page is located (skipping bad pages), then supplying this information as the sector number to `\DL.XFERDISK`, which normalizes it internally to a real disk address.

`(\DL.ACTONVMEMFILE FILEPG BUFFER NPGS WRITEFLG)` [Function]

Performs multiple file operations on the virtual memory file. `FILEPG` is the starting file page number (relative to the start of the VMem file). `BUFFER` is a pointer to the first page in the group to be transferred. NPGS is the number of pages to transfer. `WRITEFLG` passed to `\DL.ACTONVMEMPAGE` as the `WRITEFLG` parameter. This function will transfer a run of pages to or from the virtual memory file.

`(\DLDISK.GETSTATUS)` [Macro]

Returns the status of the disk controller in a smallp. Use the record definition `DLDISK.STATUS` to understand its contents. This macro expands to `(\DEVICE.INPUT 3)`

DLDISK.STATUS [Record Definition]

Record defintion (access functions) for reading the result of (\DLDISK.GETSTATUS). Contains the followng fields:

TRACK00	True if on track zero
HEADSELECT	Curreny head number
SA1000	True if controller is in SA1000 mode
DRIVENOTREADY	True if drive is not ready
WRITEFAULT	True if last operation caused a write fault
OVERRUN	True if last operation caused an overrun
CRCERR	True if last operation caused a CRC error
VERIFYERR	True if last operation caused a verify error

IOCBPAGE [Record Definition]

This record contains the layout of the IOCB page. The fields are as follows:

LASTIOCBSTATUS	Last status reported while running IOCB
NEXTIOCG	Short pointer to next IOCB in chained IOCBS. This is not currently used.
SEEKIOCBLOC	Contains the location of the SEEK IOCB.
XFERIOCBLOC	Contains the location of the TRANSFER IOCB
VRRIOCBLOC	Contains the location of the VRR Parameter block
VVRIOCBLOC	Contains the location of the VVR Parameter block
HCYLINDER	Header Template: Contains current cylinder number. Changed in all operations.
HHEAD	Header Template: Contains current head number. Changed in all operations.
HSECTOR	Header Template: Contains current sector number. Changed in all operations.
LID	Label Template: 5 words of ID number for the label.
LPAGELO	Label Template: Low 16 bits of page-within-file information in the label.
LPAGEHI	Label Template: High 7 bits of page number within file.
LFLAGS	Label Template: Flag storage for boot code
LTYPE	Label Template: Type of page (type of file in which the page is a part) (16 bits)
LBOOTLINKCHAIN1	Label Template: Boot chain info
LBOOTLINKCHAIN2	Label Template: Boot chain info
PRUNLENGTH	Parameter Block: Run length (number of pages)
PLABELCMD	Parameter Block: Code for operation on label field
PLABELLEN	Parameter Block: Length of label field
PLABELABORT	Parameter Block: Conditions for aborting transfer & error codes to scan for
PDATACMD	Parameter Block: Code for operation on data field
PDATALEN	Parameter Block: Length of data field
PVPAGE	Parameter Block: Virtual page number of memory buffer

PDATAABORT	Parameter Block: Conditions for aborting transfer & error codes to scan for
PTERMCOND1	Code to halt hardware after transfer
PTERMCOND2	Code to halt hardware after transfer
SCYLINDERDISPLACEMENT	Seek IOCB: Number of cylinders to move in seek operation.
SSEEKCMD1	Seek IOCB: First part of seek command
SSEEKCMD2	Seek IOCB: Second part of seek command

DISK IOCB PAGE

This section contains the contents of the IOCB page.

Displacements are relative to the start of the IOCB page.

Lines with asterisks following the opcode indicate fields for the user to fill in.

Address Op/Data Comment

```
0100:      000B         ; Special Block Type
0101:      00FE         ; Word count
0102:      0000         ; Not used
0103:      0000         ; IOCB Status (filled in by uCode)
0104:      0000   *     ; Next IOCB address (or 0 for last one)
0105:      0120         ; Address of seek IOCB
0106:      0132         ; Address of transfer IOCB
0107:      0140         ; Address of verify-read-read parameter area
0108:      0160         ; Address of verify-verify-read parameter area
0109:      0180         ; Address of verify-verify-write parameter area
010A:      01A0         ; Address of verify-write-write parameter area

010B:      0000   *     ; Header template: Cylinder number
010C:      0000   *     ; Header Template: Head[0..7], Sector[0..7]

010D:      0000   *     ; Label Template: Word 0 \
010E:      0000   *     ; Label Template: Word 1  \
010F:      0000   *     ; Label Template: Word 2   > ID Number for page
0110:      0000   *     ; Label Template: Word 3  /
0111:      0000   *     ; Label Template: Word 4 /
0112:      0000   *     ; Label: Page # low [bits 7..22]
0113:      0000   *     ; Label: [Pg# Hi 0..6, Pad 7..12, Flags 13..15]
0114:      0000   *     ; Label: Type #
0115:      0000   *     ; Label: Unused
0116:      0000   *     ; Label: Unused
0117:      0000         ; Filler for 16 wrd boundary lineup
0118:      0000         ; Filler for 16 wrd boundary lineup
0119:      0000         ; Filler for 16 wrd boundary lineup
011A:      0000         ; Filler for 16 wrd boundary lineup
011B:      0000         ; Filler for 16 wrd boundary lineup
011C:      0000         ; Filler for 16 wrd boundary lineup
011D:      0000         ; Filler for 16 wrd boundary lineup
011E:      0000         ; Filler for 16 wrd boundary lineup
011F:      0000         ; Filler for 16 wrd boundary lineup
```

Parameter Area for Verify-Read-Read IOCB

```
0140:       0000   *      ; Number of sectors to read
0141:       0031          ; Max #+1 of secs that may be skipped searching
0142:       0432          ; Verify header field
0143:       0001          ; word count-1 of header field
0144:       010B          ; address of header field
0145:       001C          ; Abort on NotReady. WriteFault, Overrun
0146:       0003          ; skip to next sector if CRC/Vrfy err on header
0147:       0430          ; Read label field
0148:       000C          ; word count of label field
0149:       010D          ; Address of label field in IOCB
014A:       001E          ; Abort on NotReady, WriteFault, Overrun, CRC
014B:       0430          ; Read Data Field
014C:       8100          ; Length of data field (256 words)
014D:       0000   *      ; virtual page # of buffer
014E:       001E          ; Abort on NotReady, WriteFault, Overrun, CRC
014F:       0420          ; control word to halt hw after each field
0150:       0426          ; control word to find sector mark for header
```

Parameter Area for Verify-Verify-Read IOCB

```
0140:       0000   *      ; Number of sectors to read
0141:       0031          ; Max #+1 of secs that may be skipped searching
0142:       0432          ; Verify header field
0143:       0001          ; word count-1 of header field
0144:       010B          ; address of header field
0145:       001C          ; Abort on Not Ready. Write Fault, Overrun
0146:       0003          ; skip to next sector if CRC/Vrfy err on header
0147:       0432          ; Verify label field
0148:       000B          ; word count of label field (-1 for verify)
0149:       010D          ; Address of label field in IOCB
014A:       001F          ; Quit on NotRdy, WrtFlt, Ovrrn, CRC, Verif Err
014B:       0430          ; Read Data Field
014C:       8100          ; Length of data field (256 words)
014D:       0000   *      ; virtual page # of buffer
014E:       001E          ; Quit on NotReady, WriteFault, Overrun, or CRC
014F:       0420          ; control word to halt hw after each field
0150:       0426          ; control word to find sector mark for header
```

Parameter Area for Verify-Verify-Write IOCB

```
0140:       0000   *      ; Number of sectors to read
0141:       0031          ; Max #+1 of secs that may be skipped searching
0142:       0432          ; Verify header field
0143:       0001          ; word count-1 of header field
0144:       010B          ; address of header field
0145:       001C          ; Abort on Not Ready. Write Fault, Overrun
0146:       0003          ; go to next sector if CRC/Verfy err on header
0147:       0432          ; Verify label field
0148:       000B          ; word count of label field (-1 for verify)
0149:       010D          ; Address of label field in IOCB
014A:       001F          ; Stop on NotRdy, WrtFlt, Ovrrn, CRC, Verif Err
014B:       043B          ; Write Data Field
014C:       8100          ; Length of data field (256 words)
014D:       0000   *      ; virtual page # of buffer
014E:       001C          ; Abort on NotReady, WriteFault, Overrun
014F:       0420          ; control word to halt hw after each field
0150:       0426          ; control word to find sector mark for header
```

Parameter Area for Verify-Write-Write IOCB

```
0140:       0000   *      ; Number of sectors to read
```

```
0141:       0031          ; Max #+1 of secs that may be skipped searching
0142:       0432          ; Verify header field
0143:       0001          ; word count-1 of header field
0144:       010B          ; address of header field
0145:       001C          ; Abort on Not Ready. Write Fault, Overrun
0146:       0003          ; skip to next sector if CRC/Vrfy err on header
0147:       043B          ; Write label field
0148:       000C          ; word count of label field
0149:       010D          ; Address of label field in IOCB
014A:       001C          ; Abort on Not Ready. Write Fault, Overrun
014B:       043B          ; Write Data Field
014C:       8100          ; Length of data field (256 words)
014D:       0000   *      ; virtual page # of buffer
014E:       001C          ; Abort on NotReady, WriteFault, Overrun
014F:       0420          ; control word to halt hw after each field
0150:       0426          ; control word to find sector mark for header
```

Parameter Area for Verify-Read-Write IOCB

```
0140:       0000   *      ; Number of sectors to read
0141:       0031          ; Max #+1 of secs that may be skipped searching
0142:       0432          ; Verify header field
0143:       0001          ; word count-1 of header field
0144:       010B          ; address of header field
0145:       001C          ; Abort on Not Ready. Write Fault, Overrun
0146:       0003          ; skip to next sector if CRC/Vrfy err on header
0147:       0430          ; read label field
0148:       000C          ; word count of label field
0149:       010D          ; Address of label field in IOCB
014A:       001C          ; Abort on Not Ready. Write Fault, Overrun
014B:       043B          ; Write Data Field
014C:       8100          ; Length of data field (256 words)
014D:       0000   *      ; virtual page # of buffer
014E:       001C          ; Abort on NotReady, WriteFault, Overrun
014F:       0420          ; control word to halt hw after each field
0150:       0426          ; control word to find header sector mark
```

Disk IOCB program for Seek

```
0152:       0000   *      ; Number of cylinders to move (negative)
0151:       8000          ; Insure that the seek
0152:       0422          ; completed from before
0153:       0007          ; clear out the status bits that
0154:       0000          ; are not used in a seek operation
0155:       8000          ; send a step pulse
0156:       0000   *      ; (direction is filled in)
0157:       8000          ; finish sending step pulse
0158:       0000   *      ; (direction is filled in)
0159:       0000          ; increment remaining distance
015A:       0152          ; in IOCB field, and skip if zero
015B:       0002          ; Jump back to the step
015C:       0155          ; loop.
015D:       0006          ; Clean up and finish IOCB
015E:       0000
015F:       0000
0160:       0400          ; quit and write status back
0161:       0103          ; into status word
```

Disk IOCB program for Transfer

```
0162:       0005          ; Load parameters from
0163:       0000   *      ; parameter table
```

```
0164:       8000         ; Wait for any pending seeks
0165:       0422         ; to complete.
0166:       0800         ; transfer run of pages
0167:       0002         ; if there was an error,
0168:       0169         ; finish up anyways (we're done!)
0169:       0006         ; Clean up and finish IOCB
016A:       0000
016B:       0000
016C:       0400         ; quit and write status back
016D:       0103         ; into status word.
```

End of file {Eris}<LispCore>Internal>Doc>DLionDiskDriver.TEdit.