Cortland Unified Disk 1.0 Firmware

Control Call Extensions

Rev. 1.0

Developed by:

Gus Andrade ext. 6254

02-24-86

Introduction.

Six new control calls have been defined for the Unified drive; sethook, resethook, setmark, resetmark, setsides and set interleave. Each of these calls are designed to let an application customize the firmware without having to rewrite it.

Each call is invoked through the control call parameter passing conventions set up by the "Protocol Converter" and "Extensions to the Protocol Converter" documentation.

The SetHook call lets an application program point the entry address into several low level routines into user provided code.

The ResetHook call restores specific hooks to the firmware entry points.

The SetMark call lets an application program change the Mark tables used by the firmware to values defined by the application program

The ResetMark call restores mark table values to their default values defined in the firmware.

The SetSides call lets an application change the number of sides for a disk accessed by the unified disk drive firmware.

The SetInterleave call sets the sector interleave on a track of disk used in a unified disk drive.

The control call number are as follows:

Sethook = $05
Resethook = $06
SetMark = $07
ResetMark = $08
SetSides = $09
SetInterleave = $0A

Before making any of these calls, a status call should be made to insure that the device ID we are talking to is a Unified disk drive.

1. SETHOOK = $05

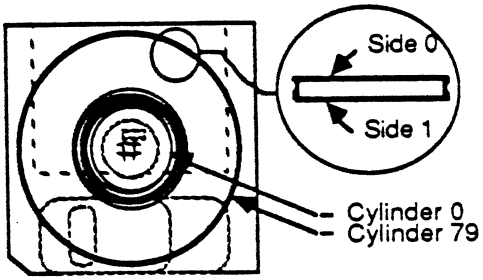
The Sethook call will allow an application to change the entry point address for several firmware routines used by the Unified 3.5" drive driver firmware. The list of hooks for the firmware routines are; Read address, Read data, Write data, Seek, Format, Write track, Verify, and Vector hook.

The read address routine will read bytes from the disk until it finds the address marks and a sector number specified by the caller.

The read data routine will read a 524 byte block of data from the disk which corresponds to the block number the caller specified. If the block read call is for Apple //e block read, the first twelve bytes will be discarded. If the block read call is for a Macintosh block read then all 524 bytes will be return to the caller in the buffer specified.

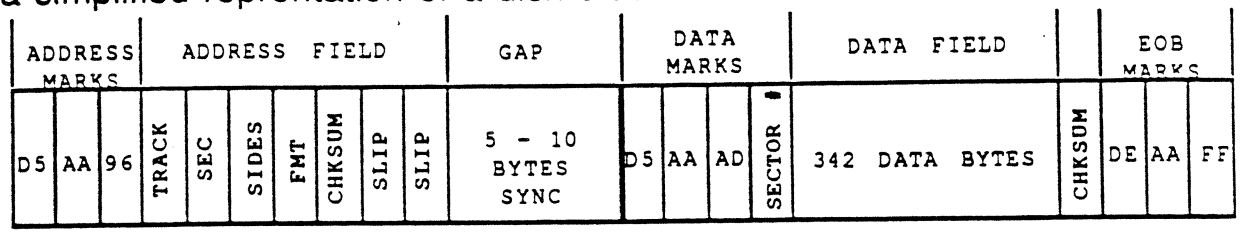
Write data will write 524 bytes of data to the disk in a disk position corresponding to the block number the user specified. If the call is an Apple //e block write call, the firmware will write twelve bytes of zeroes to the disk before writing the data provided by the user. If the call is a Macintosh block write call, all 524 data bytes written to the disk are provided by the caller.

The seek routine is called when the read/write head must be moved to another cylinder on the disk. The cylinder numbers range in value between zero and seventy nine. The following picture represents one side of a 3.5" disk.



The format call is made when a disk is to be initialized with new

address marks, data marks, zeroed data blocks and trailer marks. This call writes information to the disk which lets the read and writes routines find any block of data on the disk. The following pictorial diagram shows a simplified reprentation of a disk block.



The write track routine is called by the formatter to write out one track of empty blocks. The number of blocks on a track will vary depending on which cylinder the read write head is positioned on.

The verify routine is called by the foramtter to verify that the data written by the write track routine was writtern correctly.

The vector routine is a high level piece of code which validates the format of protocol converter calls parameters and dispatches to the code which corresponds to each of the protocol converter calls. Please refer to the "Protocol converter specification" and the "Extensions to the protocol converter" documents for detailed descriptions of each call.

Each of the routines mentioned above are assigned a hook number. This hook number will be used by the firmware to change the specific entry point address when a call is made to set a hook or reset a hook. The Sethook call will set the entry address field corresponding to a hook number, to a specific value defined by the caller. Once the callers code is in control it must save the state of the processor. The first thing the code which has been hooked into the firmware is to save the state of the emulation bit, the state of the M and the X bits and the whether interrupts are enabled or disabled. Upon exiting from a hooked routine the state of the e, m, x and interrupt bit must be restored, the carry bit must be cleared and the accumulator must be zero. If hooked code must pass

status information back to the application, it is the responsibility of teh routine to store status information in an area of memory the application knows about. This will be followed by a RTL instruction. This RTL will return cotrol back to the firmware which will in turn return back to higher levels or continue running the firmware routines corresponding to this call. When a set hook call is made and the high order bit of the hook number is zero, the firmware routine corresponding to the hook number will be executed upon returning from the users code. If the high bit is set, the firmware will not be executed after returning from the users code and control will be passed back to higher levels of the firmware. Any routines the user may have hooked into the fimware will not be able to make other protocol converter or extended protocol converter calls. The parameter list is defined as follows:

Sethook parameter list format:

$00XXYYZZ	Count low	= $04
	Count high	= $00
	Hook #	With high bit set or cleared
	Addr low	
	Addr high	
	Bank no.	

* %1XXXXXXX for a hook which will NOT execute the default firmware routine when it returns from the caller.

%0XXXXXXX for a hook which WILL execute firmware code after returning from the caller's code.

The list of hook numbers is defined as follows:

Hook #1 = Read address
Hook #2 = Read data
Hook #3 = Write data
Hook #4 = Seek
Hook #5 = Format disk

Hook #6 = Write track
Hook #7 = Verify track
Hook #8 = Vector

Error codes:

Invalid parameter count = $22
Bad hook # = $30

Hook numbers other than the ones defined, will return an error status when a call is made to set hook or reset hook. Also, the parameter count in the first two bytes has to be as defined or the hook address will not be modified and a control parameter list error will be returned to the caller.

2. Resethook = $06

The Resethook call will restore the default entry address defined in the firmware for a specific hook. The format of the parameter list for the Reset hook call is as follows:

ResetHook parameter list:

$00XXYYZZ	Count low	= $01
	Count high	= $00
	Hook #	= Hook #

The list of hook numbers is defined as follows:

Hook #0 = Restore ALL hooks to firmware defaults.
Hook #1 = Read address
Hook #2 = Read data
Hook #3 = Write data
Hook #4 = Seek
Hook #5 = Format disk
Hook #6 = Write track
Hook #7 = Verify track
Hook #8 = Vector

Error codes:

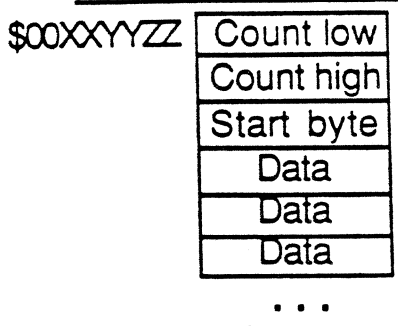
Invalid parameter count = $22
Bad hook # = $30

3. Setmark = $07

The Setmark call lets an application program change individual bytes in the mark tables to values of their choice. The parameters are stored in the table in reverse order. Bounds checking will be performed to make sure the byte count from a specific start position will not go past the end of the Mark table. The Marks table is defined as follows:

```
Marktab   equ     *
afdmtab   dfb     $FF     ; byte 0 (Sector number)
          dfb     $AD     ; byte 1
          dfb     $AA     ; byte 2
          dfb     $D5     ; byte 3
          dfb     $FF     ; byte 4
synctab   equ     *       ; Sync byte table
          dfb     $FC     ; byte 5
          dfb     $F3     ; byte 6
          dfb     $CF     ; byte 7
          dfb     $3F     ; byte 8
          dfb     $FF     ; byte 9
bsmarks   equ     *       ; bit slip marks table
          dfb     $FF     ; byte 10
          dfb     $AA     ; byte 11
          dfb     $DE     ; byte 12
adrguts   equ     *       ; inter-header gap
          dfb     $FF     ; byte 13
          dfb     $FF     ; byte 14
          dfb     $FF     ; byte 15
          dfb     $FF     ; byte 16
          dfb     $FF     ; byte 17
adrmarks  equ     *       ; Address marks
          dfb     $96     ;
          dfb     $AA     ;
          dfb     $D5     ;
          dfb     $FF     ;
```

Parameter list for SetMark call:



Number of bytes to set in the Mark table+1 = $00

Error code:
Invalid parameter count = $22

4. Resetmark = $08

The Resetmark call will change the individual bytes in the Mark table to their default values defined in the firmware. The parameters are stored in the table in reverse order. Bounds checking will be performed to make sure the byte count from a specific start position will not go pas the end of the Mark table. The Marks table is defined as follows:

```
Marktab   equ   *
afdmtab   dfb   $FF     ; byte 0
          dfb   $AD     ; byte 1
          dfb   $AA     ; byte 2
          dfb   $D5     ; byte 3
          dfb   $FF     ; byte 4
synctab   equ   *       ; Sync byte table
          dfb   $FC     ; byte 5
          dfb   $F3     ; byte 6
          dfb   $CF     ; byte 7
          dfb   $3F     ; byte 8
          dfb   $FF     ; byte 9
bsmarks   equ   *       ; bit slip marks table
          dfb   $FF     ; byte 10
          dfb   $AA     ; byte 11
          dfb   $DE     ; byte 12
adrguts   equ   *       ; inter-header gap
          dfb   $FF     ; byte 13
          dfb   $FF     ; byte 14
          dfb   $FF     ; byte 15
          dfb   $FF     ; byte 16
          dfb   $FF     ; byte 17
adrmarks  equ   *       ; Address marks
          dfb   $96     ;
          dfb   $AA     ;
          dfb   $D5     ;
          dfb   $FF     ;
```

Parameter list for ResetMark call:

$00XXYYZZ

Count low
Count high
Start byte

Number of bytes to set in the Mark table+1

= $00

Error code:

Invalid parameter count = $22

5. Setsides = $09

This call lets an application program change the number of sides defined on a specific disk drive. The only call affected by the SetSides call is the format call. Read an write calls use the number of sides encoded in the block header on the disk. The SetSides call can be made before a format call in order to format single sided disks. The high order bit will define the number of sides for any unified disk drives in the chain. A zero defines single sided and a one defines double sided. The firmware defaults this value to double sided. This call will not do any checking to see if any unified drives are connected. The parameter list is defined as follows:

Parameter list for SetSides call:

$00XXYYZZ

Count low	
Count high	= $00
Sides byte	= %1XXXXXXX double sided = %0XXXXXXX single sided

Error codes:

NONE

6. Set Interleave = $0A

This call lets an application program set the block interleve on a disk to a value between one an twelve inclusive. The value set will apply to all unified drives connected in the chain. An error status will be returned to the caller if an invalid interleave or an invalid parameter count is passed to the firmware. This call applies only to format calls. The sector number set down on each data block of a disk during a format call is calculated from the interleave value. The parameter list for a Set interleave call is as follows:

Parameters list for SetInterleve:

$00XXYYZZ	Count low	
	Count high	= $00
	Interleave	= $01 to $0C (Interleave value)

Error codes:

Invalid parameter count = $22

Invalid Interleave = $32