Cortland Implementation of SmartPort

Revision History.....

March 18, 1986	Ver. 00.10	R. Montagne	Initial release.
March 20, 1986	Ver. 00.11	R. Montagne	Quark QC-10 support.
April 17, 1986	Ver. 00.12	R. Montagne	Dynamic deivce assignment.
April 29, 1986	Ver. 00.13	R. Montagne	Interface to ROMDISK drivers. Extended and Non Extended Control calls. Typos in examples on page 2.

Written by Ray Montagne - April 23, 1986

PREFACE

This document describes the implementation of both the extended and non extended SmartPort Interface in Cortland. For specific information on the non extended and extended CBUS architecture, please refer to the CBUS specifications.

This document includes descriptions of the command sets, and specifics of passing parameters in absolute zero page for built in driver support of devices not residing on the CBUS. These devices include Unified Disk3.5, RamDisk and RomDisk. For detailed information on the command sets supported by these devices, refer to the device specifications.

GENERAL

The SmartPort provides a method of attaching a series of devices to the external disk port of the Cortland. The SmartPort is a program that converts the calls made to it into a format which can be transmitted over the disk port to control intellegent devices such as the UniDisk3.5. Within the SmartPort, calls fall into one of two groups. Extended calls, and non extended calls. This document details all calls supported by the SmartPort in Cortland (e.g. extended and non extended).

To use the SmartPort interface, a program issues calls in a manner similar to that used for ProDOS Machine Language Interface (MLI) calls. The topmost 'level' of one of these calls is a 'JSR' to the SmartPort entry point (DISPATCH), followed by a single byte which specifies the SmartPort command type, followed by a pointer to a table which contains the parameters necessary for the call. For a non extended call this will be a word pointer, while for an extended call it will be a longword pointer.

Here is an example of an non extended call:

```
SP_CALL     JSR     DISPATCH    ; Call SmartPort command dispatcher
            DFB     CMDNUM      ; This specifies the command type
            DW      CMDLIST     ; word pointer to the parameter list in bank $00
            BCS     ERROR       ; Carry is set on an error
```

Here is an example of a extended call:

```
SP_EXT_CALL JSR     DISPATCH .  ; Call SmartPort command dispatcher
            DFB     CMDNUM+$40  ; This specifies the extended command type
            DW      CMDLIST     ; Low word pointer to the parameter list
            DW      ^CMDLIST    ; High word pointer to the parameter list
            BCS     ERROR       ; Carry is set on an error
```

Information concerning the DISPATCH address, the CMDNUM and the CMDLIST appears below.

Upon completion of the call, execution returns to the RTS address plus three for a non extended command, or the RTS address plus five for an extended command (the 'BCS' statement in the examples on the previous page). If the call was successful, the C flag is cleared, and the A register is set to 0. If the call was unsuccessful, the C flag is set and the A register contains the error code. The complete register status upon completion is summarized below.

REGISTER STATUS ON RETURN FROM SMARTPORT

	65816 Status byte N V 1 B D I Z C	Acc	Xreg	Yreg	PC	SP
Successful Nonextended Call	X X 1 X 0 U X 0	0	n	n	JSR+3	U
Successful Extended Call	X X 1 X 0 U X 0	0	n	n	JSR+5	U
Unsuccessful Nonextended Call	X X 1 X 0 U X 1	Error	X	X	JSR+3	U
Unsuccessful Extended Call	X X 1 X 0 U X 1	Error	X	X	JSR+5	U

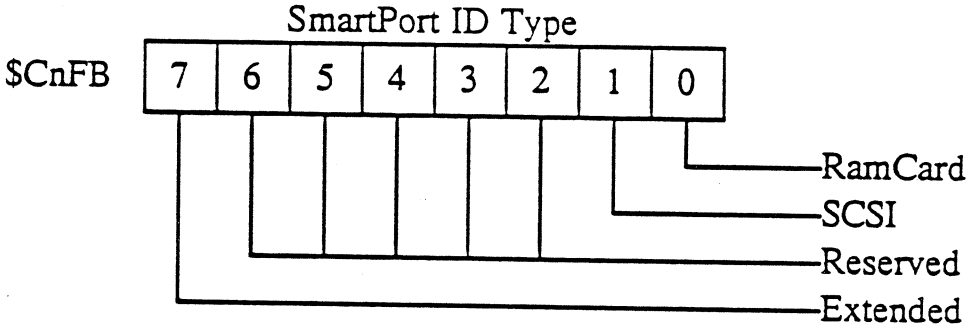
(Note: X = undefined, U = unchanged, n = undefined for tranfers to the device or number of bytes transferred when the transfer was from the device to the host)

Determining the DISPATCH address

Regardless of whether you are operating in a machine with built in disk support or a machine containing cards supporting the SmartPort Interface, the existance of the SmartPort Interface may be determined by examining the signature bytes as follows;

$Cn01 = $20 $Cn03 = $00 $Cn05 = $03 $Cn07 = $00

where n = the slot number. All cards or built in slot support firmware with these values support SmartPort calls. In addition, the SmartPort ID Type byte can be examined to obtain more information about what special support may be built into the SmartPort firmware driver. The SmartPort ID Type byte located at $CnFB has been encoded to indicate the type of devices that can be supported by the SmartPort firmware. Note that a driver that supports the extended modes will also support the existing non extended modes.



Once having determined that a SmartPort interface exists in a slot or port, you need to determine the entry point, or DISPATCH address, for this slot or port. This address is determined by the value found at $CnFF, where n is the slot number. By adding the value at $CnFF to $Cn00, one obtains the standard ProDOS entry point for block devices. Information about this entry point is described in the ProDOS Technical Reference manual. The SmartPort entry point, DISPATCH, is located three bytes after the ProDOS entry point. Therefore, the SmartPort entry point is $Cn00 plus 3 plus the value found at $CnFF.

For example, if a the signature bytes for the SmartPort interface are found in slot 5, and $C5FF contains a $0A, the ProDOS entry point will be $C50A, and the SmartPort entry point is three larger than $C50A, or $C50D.

ProDOS Interface

A new implementation of ProDOS called ProDOS16 will be available for the Cortland CPU. This new implementation will recognize and be able to communicate directly with a SmartPort driver. The current implementation of ProDOS does not recognize SmartPort Interfaces. In order to have SmartPort interfaces supported by ProDOS, it is necessary to support the ProDOS entry point as well as ProDOS comands.

ProDOS Implementations

ProDOS entry points are supported by converting the input to the ProDOS driver into a format compatable with the non extended SmartPort, and then have the ProDOS driver call the SmartPort driver. The non extended format is chosen because the majority of devices support the non extended protocol rather than the extended format. Also, ProDOS is runs in a 6502 environment that may not easily support direct tranfers between the SmartPort and any bank of memory. In Cortland, the ProDOS device driver for slot 5 has been implemented as a block device. All calls to the slot 5 ProDOS device driver will be mapped into non extended SmartPort calls. On return from the SmartPort, the ProDOS driver maps all non fatal SmartPort errors into a non error condition (ACC=$00 with carry clear). In addition, any fatal SmartPort error is converted by the ProDOS device driver into standard ProDOS error codes.

CMDNUM and CMDLIST

The CMDNUM specifies the command type and is in the range of $00 - $09 for non extended commands, or $40 - $49 for extended commands. The parameter list, CMDLIST, varies for each call, though the first byte in the list always specifies the number of parameters in the call. This is NOT the physical number of bytes; it is the logical number of pieces of information passed in the list. A summary of command numbers and their associated parameter lists can be found in an appendix.

Unit Numbers

A part of every parameter list is the Unit number. The unit number specifies which device connected to the SmartPort will respond to the command you are giving. Calls which allow you to reference the SmartPort itself require the specification of Unit $00 (Status, Init, and Control allow you to do this). Cortland assigns unit numbers to devices in ascending order starting with a unit number of $01. Devices are assigned unit numbers starting with the Unified Disk 3.5, RamDisk, RomDisk and finally the UniDisk3.5 type devices. Then, depending on the device that is selected as the boot device, the unit numbers for the first four devices may be remapped to place the boot device as the first device in the chain (unit $01). If remapping is required, then all devices previously located in front of the boot device are relocated after the boot device. This will be discussed in greater detail later in this document.

Determining the Number of Devices

You can determine the number of devices connected to the SmartPort by executing a SmartPort status call (described later) with a unit number of $00.

SmartPort Limitations

There are two major constraints on the use of the SmartPort. The first is that its stack usage is 30-35 bytes. Programs should allow this much stack space on a call. Second, the SmartPort cannot generally be used to put anything into absolute Zero Page locations. Absolute Zero Page is defined as Direct Page when the Direct Register is set to $0000.

STATUS CMDNUM = $00

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit number
	Byte 2 :	status list pointer low
	Byte 3 :	status list pointer high
	Byte 4 :	status code

This call returns the status information about a particular device or the SmartPort itself. There are defined status calls for returning general information. In general, device specific calls can be implemented by a device for diagnostic or other information.

On return from a status call, the eight bit X and Y registers contain a count of the number of bytes tranferred to the host. X contains the low byte of the count, while Y contains the high byte value of the count.

Required Parameters

unit num: 1 byte value
Range : $00, $01 - $7E

This is the unit number. Each device has a unique number assigned to it at initialization time. The numbers are assigned according to the device's position in the chain. A status call with a unit number of $00 specifies a call for the overall SmartPort status. (The status list returned on this call is explained below.)

status list: pointer

This is a word pointer to the buffer where the status is to be returned in bank $00. Note that the length of the buffer will vary depending on the status request being made.

status code: 1 byte value
Range: $00 - $FF

This is the number of the status request being made. All devices respond to the following requests:

Code	Status returned
$00	Return device status
$01	Return device control block
$02	Return newline status (character devices only)
$03	Return device information block (DIB)

Although devices must respond to the status requests listed above, the device may not be able to support the request. In this case, the device should return an Invalid Status Code error ($21).

Statcode = $00
The device status consists of four bytes. The first is the general status byte:

Bit	Function
7	1 = Block device, 0 = Character device
6	1 = Write allowed
5	1 = Read allowed
4	1 = Device online, or disk in drive
3	1 = Format allowed
2	1 = Media Write Protected (block devices only)
1	1 = Device currently interrupting (not supported on Cortland)
0	1 = Device currently open (character devices only)

If the device is a block device, the next three bytes are the size in 512 byte blocks. The least significant byte is first. The maximum size is $FFFFFF blocks, or about 8 giga bytes. If the device is a non block device, these bytes are set to zero.

Statcode = $01
The device control block is device dependent. The DCB is typically used to control various operating characteristics in a device. The DCB is set with the corresponding control call. The first byte will be the number of bytes in the control block. A value of $00 returned in this byte should be interpretted as a DCB length of 256, while a value of $01 would be a DCB lenth of 1 byte. The length of the DCB will always be in the range of 1 to 256 bytes excluding the count byte.

Statcode = $02
There are currently no character devices implemented for use on the SmartPort, and therefore the Newline status is presently undefined.

Statcode = $03
This call returns the device's information block. It contains information identifying the device, its type, and various other attributes. The returned status list has the following form:

```
STATLIST  DFB  DEVICE_STATBYTE1   ;Same as byte 1 in status code = $00
          DFB  DEVICE_SIZE_LO     ;Number of blocks on device (block device)
          DFB  DEVICE_SIZE_MED    ;Number of blocks on device (middle byte)
          DFB  DEVICE_SIZE_HI     ;Number of blocks on device (high byte)
          DFB  ID_STR_LEN         ;Length in bytes (16 max)
          ASC  'device name'      ;(16 bytes) upper case ascii, msb=0, blanks added
          DFB  DEVICE_TYPE        ;
          DFB  DEVICE_SUBTYPE     ;Bit 7 = Extended Support
          DW   VERSION            ;Device firmware version number
```

Note that the first four bytes of the DIB are the same bytes returned in the Device Status call. Also, since the name string is always 16 characters, the position of the bytes in the list is always fixed relative to the beginning of the DIB for non extended calls. Bit 7 of the DIB subtype byte will be returned as a '1' if the device supports extended calls. Bit 7 of the DIB subtype byte will be returned as a '0' if the device does not supports extended calls. Bit 6 of the DIB subtype byte will be returned as a '0' if the device does not support disk switched errors (error code $2E). Bit 6 of the DIB subtype byte will be returned as a '1' if the device does support disk switched errors.

SmartPort Status

A status call with a unit number of $00 and a status code of $00 is a request to return the status of the SmartPort as a whole. The number of devices as well as the current interrupt status is returned. The Format of the status list returned is as follows:

```
STATLIST   byte 0:   Number of devices
           byte 1:   Interrupt Status (Bit 7 clear => no interrupt)
           byte 2:   Reserved
           byte 3:   Reserved
           byte 4:   Reserved
           byte 5:   Reserved
           byte 6:   Reserved
           byte 7:   Reserved
```

The number of devices byte tells the caller the total number of devices connected to this slot or port. This number will always be in the range of 0 to 127.

The interrupt status byte is used by programs to determine if the SmartPort was the source of an interrupt. If the most significant bit of this byte is set, there is a device (or devices) in the chain that requress interrupt servicing. Which device is actually interrupting cannot be determined from this value. The user's interrupt handler, having determined that A SmartPort interrupt has occured must poll each device on the chain to find out which device requires service.

The Extended SmartPort interface is currently not supported on either the Apple//c or the Apple// SmartPort Interface card, however, if future revisions of these products were to implement the Extended SmartPort, interrupts should be handled as they currently are with the non extended implementation.

Possible Errors

$06	BUSERR	Communications error
$21	BADCTL	Invalid status code
$30-$3F	$50-$7F	Device specific error

(Some user defined status calls may use other error codes.)

READ BLOCK cmdnum = $01

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer low
	Byte 3 :	data_buffer pointer high
	Byte 4 :	block_num low
	Byte 5 :	block_num med
	Byte 6 :	block_num high

This call reads one 512 byte block from the block device specified by unit_num into bank $00 memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a word pointer to the user's buffer in bank $00 that the data is to be read into. The buffer must be 512 or more bytes in length.

block num: LongWord number

This is the logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a three byte number.)

Possible Errors

$06 BUSERR	Communications error
$27 IOERROR	I/O Error
$28 NODRIVE	No Device Connected
$2D BADBLOCK	Invalid block number
$2F OFFLINE	Device off line or no disk in drive

WRITE_BLOCK cmdnum = $02

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer low
	Byte 3 :	data_buffer pointer med
	Byte 4 :	block_num low
	Byte 5 :	block_num med
	Byte 6 :	block_num high

This call writes one 512 byte block to the block device specified by unit_num from bank $00 memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a word pointer to the user's buffer that the data is to be read into in bank $00. The buffer must be 512 or more bytes in length.

block num: number

This is the logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a three byte number.)

Possible Errors

$06 BUSERR	Communications error
$27 IOERROR	I/O Error
$28 NODRIVE	No Device Connected
$2B NOWRITE	Disk write protected
$2D BADBLOCK	Invalid block number
$2F OFFLINE	Device off line or no disk in drive

FORMAT cmdnum = $03

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call formats a block device. It should be noted that the format done by this call is NOT linked to any operating system: it simply prepares all blocks on the medium for reading and writing. Operating system specific catalog information such as bitmaps and catalogs are not laid down by this call.

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	Disk Write Protected
$2F	OFFLINE	Device off line or no disk in drive

CONTROL cmdnum = $04

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	control_list low
	Byte 3 :	control_list high
	Byte 4 :	control_code

This call sends control information to the device. The information can be either general or device specific. A control call with a unit number of zero has special significance as noted below.

Required parameters

unit num: 1 byte value
Range: $01-$7E

control list: pointer

This is a word pointer to the user's buffer in bank $00 where the control information is to be read from. The first two bytes specify the length of the control list; the low byte is first. A control list is mandatory even if the call being issued does not pass information in the list. A length of zero is used for the first two bytes in this case.

control code: 1 byte value
Range: $00-$FF

This is the number of the control request being made. This number and function i device specific, with the exception that all devices must reserve the following codes for specific functions.

Code	Control Function
$00	Reset the device.
$01	Set device control block
$02	Set newline status (character devices only)
$03	Service device interrupt

Code = $00
Performs a soft reset of the device. Generally returns 'housekeeping' values to some reset value.

Code = $01
Allows the user to set the device control block. Devices generally use the bytes in this block to control global aspects of the device's operating environment. Since the length is device dependent, the recommended way to set the DCB is to first read in the DCB (with the STATUS call), alter the bits of interest, and then write out the same string with this call. The first byte is the length of the DCB (excluding the byte itself). A value of $00 in the length byte corresponds with a DCB size of 256 bytes, while a count value of $01 corresponds with a DCB size of 1 byte. A count value of $FF corresponds with a DCB size of 255 bytes.

Unit $00 Control Calls

A control call with a unit number of $00 specifies that the call applies to the SmartPort as a whole. There are two calls currently available.

Code = $00 Enable Interrupts from the SmartPort
This call is used to enable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort driver on Cortland, and will return a bad control code error ($21).

Code = $01 Disable Interrupts from the SmartPort
This call is used to disable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort driver on Cortland, and will return a bad control code error ($21).

Possible Errors

$06 BUSERR	Communications error
$21 BADCTL	Invalid control code
$22 BADCTLPARM	Invalid parameter list
$30-$3F	Device specific error

INIT

cmdnum = $05

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call provides the application with a way of resetting the SmartPort.

Required parameters

unit num: 1 byte value
Value: $00

The SmartPort will go through it's initialization sequence, hard resetting all devices and sending each their device numbers. This call is made internally at first access and it should never be necessary for an application to make this call. (Though it is never recommended to connect new devices when the CPU power is on, this call provides a method for the SmartPort to communicate with devices connected midstream).

Possible Errors

$06 BUSERR	Communications error
$28 NODRIVE	No Device Connected

OPEN

cmdnum = $06

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to prepare a character device for reading or writing.

Note that block devices do not accept this call, and will return a BADCMD error code ($21) to be returned.

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

CLOSE

cmdnum = $07

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to tell a character device that a sequence of reads or writes is over. In the case of a printer, this call could have the effect of flushing the print buffer.

Note that block devices do not accept this call, and will return a invalid command error ($01).

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

READ cmdnum = $08

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer low
	Byte 3 :	data _ buffer pointer high
	Byte 4 :	byte_count low
	Byte 5 :	byte_count high
	Byte 6 :	address pointer low
	Byte 7 :	address pointer med
	Byte 8 :	address pointer high

This call reads a number of bytes from the device specified by unit_num into bank $00 memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by character devices, a block device might use this call to read a block of a non standard size (greater than 512 bytes per block). The UniDisk3.5 will return 524 bytes with this call (Macintosh block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is the word pointer to the user's buffer in bank $00 that the data is to be read into. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. The Cortland implementation of the SmartPort has a limitation of 767 bytes for this call.

address: Word

This is a device specific parameter. An example of how this call might be implemented with a block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes).

Possible Errors

$06 BUSERR	Communications error
$27 IOERROR	I/O Error
$28 NODRIVE	No Device Connected
$2B NOWRITE	DISK WRITE PROTECTED
$2F BADBLOCK	Invalid block number
$2F OFFLINE	Device off line or no disk in drive

WRITE cmdnum = $09

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer low
	Byte 3 :	data _ buffer pointer high
	Byte 4 :	byte_count low
	Byte 5 :	byte_count high
	Byte 6 :	address pointer low
	Byte 7 :	address pointer med
	Byte 8 :	address pointer high

This call writes a number of bytes to the device specified by unit_num from bank $00 memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by extended character devices, an extended block device might use this call to write a block of a non standard size (greater than 512 bytes per block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: Word pointer

This is the word pointer to the user's buffer in bank $00 that the data is to be written from. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. The Cortland implementation of the SmartPort has a limitation of 767 bytes for this call.

address: Word

This is a device specific parameter. An example of how this call might be implemented with a block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes such as a Macintosh block).

Possible Errors

$06 BUSERR	Communications error
$27 IOERROR	I/O Error
$28 NODRIVE	No Device Connected
$2B NOWRITE	DISK WRITE PROTECTED
$2F BADBLOCK	Invalid block number
$2F OFFLINE	Device off line or no disk in drive

EXTENDED STATUS CMDNUM = $40

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit number
	Byte 2 :	status list pointer (low byte, low word)
	Byte 3 :	status list pointer (high byte, low word)
	Byte 4 :	status list pointer (low byte, high word)
	Byte 5 :	status list pointer (high byte, high word)
	Byte 6 :	status code

This call returns the status information about a particular device or the SmartPort itself. There are defined status calls for returning general information. In general, device specific calls can be implemented by a device for diagnostic or other information.

On return from a status call, the X and Y registers contain a count of the number of bytes tranferred to the host. X contains the low byte of the count, while Y contains the high byte value of the count.

Required Parameters

unit num: 1 byte value
Range : $00, $01 - $7E

This is the unit number. Each device has a unique number assigned to it at initialization time. The numbers are assigned according to the device's position in the chain. A status call with a unit number of $00 specifies a call for the overall SmartPort status. (The status list returned on this call is explained below.)

status list: pointer

This is a long word pointer to the buffer where the status is to be returned. Note that the length of the buffer will vary depending on the status request being made.

status code: 1 byte value
Range: $00 - $FF

This is the number of the status request being made. All devices respond to the following requests:

Code	Status returned
$00	Return device status
$01	Return device control block
$02	Return newline status (character devices only)
$03	Return device information block (DIB)

Although devices must respond to the status requests listed above, the device may not be able to support the request. In this case, the device should return an Invalid Status Code error ($21).

Statcode = $00
The device status consists of four bytes. The first is the general status byte:

Bit	Function
7	1 = Block device, 0 = Character device
6	1 = Write allowed
5	1 = Read allowed
4	1 = Device online, or disk in drive
3	1 = Format allowed
2	1 = Media Write Protected (block devices only)
1	1 = Device currently interrupting (not supported on Cortland)
0	1 = Device currently open (character devices only)

If the device is a block device, the next four bytes are the size in 512 byte blocks. The least significant byte is first. The maximum size is $FFFFFFFF blocks, or about 2.2 trillion bytes. If the device is a non block device, these bytes are set to zero.

Statcode = $01
The device control block is device dependent. The DCB is typically used to control various operating characteristics in a device. The DCB is set with the corresponding control call. The first byte will be the number of bytes in the control block. A value of $00 returned in this byte should be interpretted as a DCB length of 256, while a value of $01 would be a DCB lenth of 1 byte. The length of the DCB will always be in the range of 1 to 256 bytes excluding the count byte.

Statcode = $02
There are currently no character devices implemented for use on the SmartPort, and therefore the Newline status is presently undefined. Since the Apple// SCSI Interface card can support both character devices and block devices, and utilizes the SmartPort interface, some definition may come out of that design.

Statcode = $03
This call returns the device's information block. It contains information identifying the device, its type, and various other attributes. The returned status list has the following form:

```
STATLIST  DFB  DEVICE_STATBYTE1   ;Same as byte 1 in status code = $00
          DW   DEVICE_SIZE_LO     ;Number of blocks on device
          DW   DEVICE_SIZE_HI     ;Number of blocks on device
          DFB  ID_STR_LEN         ;Length in bytes (16 max)
          ASC  'device name'      ;(16 bytes) upper case ascii, msb=0, blanks added
          DFB  DEVICE_TYPE        ;
          DFB  DEVICE_SUBTYPE     ;Bit 7 = Extended Support
          DW   VERSION            ;Device firmware version number
```

Note that the first five bytes of the DIB are the same bytes returned in the Device Status call. Also, since the name string is always 16 characters, the position of the bytes in the list is always fixed relative to the beginning of the DIB for extended devices. Bit 7 of the DIB subtype byte will be returned as a one if the device supports extended calls. Bit 7 of the DIB subtype byte will be returned as a zero if the device does not supports extended calls. Bit 6 of the DIB subtype byte will be returned as a '0' if the device does not support disk switched errors (error code $2E). Bit 6 of the DIB subtype byte will be returned as a '1' if the device does support disk switched errors.

SmartPort Status

A status call with a unit number of $00 and a status code of $00 is a request to return the status of the SmartPort as a whole. The number of devices as well as the current interrupt status is returned. The Format of the status list returned is as follows:

```
STATLIST   byte 0:   Number of devices
           byte 1:   Interrupt Status (Bit 7 clear => no interrupt)
           byte 2:   Reserved
           byte 3:   Reserved
           byte 4:   Reserved
           byte 5:   Reserved
           byte 6:   Reserved
           byte 7:   Reserved
```

The number of devices byte tells the caller the total number of devices connected to

this slot or port. This number will always be in the range of 0 to 127.

The interrupt status byte is used by programs to determine if the SmartPort was the source of an interrupt. If the most significant bit of this byte is set, there is a device (or devices) in the chain that requress interrupt servicing. Which device is actually interrupting cannot be determined from this value. The user's interrupt handler, having determined that A SmartPort interrupt has occured must poll each device on the chain to find out which device requires service.

The Extended SmartPort interface is currently not supported on either the Apple//c or the Apple// SmartPort Interface card, however, if future revisions of these products were to implement the Extended SmartPort, interrupts should be handled as they currently are with the non extended implementation.

Possible Errors

$06	BUSERR	Communications error
$21	BADCTL	Invalid status code
$30-$3F	$50-$7F	Device specific error

(Some user defined status calls may use other error codes.)

EXTENDED READ BLOCK cmdnum = $41

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer (low byte, low word)
	Byte 3 :	data_buffer pointer (high byte, low word)
	Byte 4 :	data_buffer pointer (low byte, high word)
	Byte 5 :	data_buffer pointer (high byte, high word)
	Byte 6 :	block_num (low byte, low word)
	Byte 7 :	block_num (high byte, low word)
	Byte 8 :	block_num (low byte, high word)
	Byte 9 :	block_num (high byte, high word)

This call reads one 512 byte block from the block device specified by unit_num into memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a longword pointer to the user's buffer that the data is to be read into. The buffer must be 512 or more bytes in length.

block num: LongWord number

This is the logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a four byte number.)

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED WRITE BLOCK cmdnum = $42

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer (low byte, low word)
	Byte 3 :	data_buffer pointer (high byte, low word)
	Byte 4 :	data_buffer pointer (low byte, high word)
	Byte 5 :	data_buffer pointer (high byte, high word)
	Byte 6 :	block_num (low byte, low word)
	Byte 7 :	block_num (high byte, low word)
	Byte 8 :	block_num (low byte, high word)
	Byte 9 :	block_num (high byte, high word)

This call writes one 512 byte block to the block device specified by unit_num from memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a longword pointer to the user's buffer that the data is to be read into. The buffer must be 512 or more bytes in length.

block num: LongWord number

This is the logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a four byte number.)

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	Disk write protected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED FORMAT cmdnum = $43

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call formats a block device. It should be noted that the format done by this call is NOT linked to any operating system: it simply prepares all blocks on the medium for reading and writing. Operating system specific catalog information such as bitmaps and catalogs are not laid down by this call.

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	Disk Write Protected
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED CONTROL cmdnum = $44

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	control_list (low byte, low word)
	Byte 3 :	control_list (high byte, low word)
	Byte 4 :	control_list (low byte, high word)
	Byte 5 :	control_list (high byte, high word)
	Byte 6 :	control_code

This call sends control information to the device. The information can be either general or device specific. A control call with a unit number of zero has special significance as noted below.

Required parameters

unit num: 1 byte value
Range: $01-$7E

control list: pointer

This is a longword pointer to the user's buffer where the control information is to be read from. The first two bytes specify the length of the control list; the low byte is first. A control list is mandatory even if the call being issued does not pass information in the list. A length of zero is used for the first two bytes in this case.

control code: 1 byte value
Range: $00-$FF

This is the number of the control request being made. This number and function is device specific, with the exception that all devices must reserve the following codes for specific functions.

Code	Control Function
$00	Reset the device.
$01	Set device control block
$02	Set newline status (character devices only)
$03	Service device interrupt

Code = $00
Performs a soft reset of the device. Generally returns 'housekeeping' values to some reset value.

Code = $01
Allows the user to set the device control block. Devices generally use the bytes in this block to control global aspects of the device's operating environment. Since the length is device dependent, the recommended way to set the DCB is to first read in the DCB (with the STATUS call), alter the bits of interest, and then write out the same string with this call. The first byte is the length of the DCB (excluding the byte itself). A value of $00 in the length byte corresponds with a DCB size of 256 bytes, while a count value of $01 corresponds with a DCB size of 1 byte. A count value of $FF corresponds with a DCB size of 255 bytes.

Unit $00 Control Calls

A control call with a unit number of $00 specifies that the call applies to the SmartPort as a whole. There are two calls currently available.

Code = $00 Enable Interrupts from the SmartPort
This call is used to enable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort support on Cortland, and will return a bad control code error ($21).

Code = $01 Disable Interrupts from the SmartPort
This call is used to disable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort support on Cortland, and will return a bad control code error ($21).

Possible Errors

$06 BUSERR	Communications error
$21 BADCTL	Invalid control code
$22 BADCTLPARM	Invalid parameter list
$30-$3F	Device specific error

EXTENDED_INIT cmdnum = $45

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call provides the application with a way of resetting the SmartPort.

Required parameters

unit num: 1 byte value
Value: $00

The SmartPort will go through it's initialization sequence, hard resetting all devices and sending each their device numbers. This call is made internally at first acces and it should never be necessary for an application to make this call. (Though it is never recommended to connect new devices when the CPU power is on, this call provides a method for the SmartPort to communicate with devices connected midstream).

Possible Errors

$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED OPEN cmdnum = $46

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to prepare a character device for reading or writing.

Note that extended block devices do not accept this call, and will return a invalid command error ($01).

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED CLOSE cmdnum = $47

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to tell an extended character device that a sequence of reads or writes is over. In the case of a printer, this call could have the effect of flushing the print buffer.

Note that extended block devices do not accept this call, and will return a invalid command error ($01).

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED READ cmdnum = $48

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer (low byte, low word)
	Byte 3 :	data _ buffer pointer (high byte, low word)
	Byte 4 :	data _ buffer pointer (low byte, high word)
	Byte 5 :	data _ buffer pointer (high byte, high word)
	Byte 6 :	byte_count low
	Byte 7 :	byte_count high
	Byte 8 :	address pointer (low byte, low word)
	Byte 9 :	address pointer (high byte, low word)
	Byte 10 :	address pointer (low byte, high word)
	Byte 11 :	address pointer (high byte, high word)

This call reads a number of bytes from the device specified by unit_num into memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by extended character devices, an extended block device might use this call to read a block of a non standard size (greater than 512 bytes per block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: LongWord pointer

This is the four byte pointer to the user's buffer that the data is to be read into. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. All of the current implementations of the SmartPort utilizing CBUS have a limitation of 767 bytes for this call. Other peripheral cards supporting the SmartPort interface, and using this call may not have this limitation.

address: LongWord

This is a device specific parameter. An example of how this call might be implemented with an extended block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes).

Possible Errors

$06 BUSERR	Communications error
$27 IOERROR	I/O Error
$28 NODRIVE	No Device Connected
$2B NOWRITE	DISK WRITE PROTECTED
$2F BADBLOCK	Invalid block number
$2F OFFLINE	Device off line or no disk in drive

EXTENDED WRITE cmdnum = $49

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer (low byte, low word)
	Byte 3 :	data _ buffer pointer (high byte, low word)
	Byte 4 :	data _ buffer pointer (low byte, high word)
	Byte 5 :	data _ buffer pointer (high byte, high word)
	Byte 6 :	byte_count low
	Byte 7 :	byte_count high
	Byte 8 :	address pointer (low byte, low word)
	Byte 9 :	address pointer (high byte, low word)
	Byte 10 :	address pointer (low byte, high word)
	Byte 11 :	address pointer (high byte, high word)

This call writes a number of bytes to the device specified by unit_num from memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by extended character devices, an extended block device might use this call to write a block of a non standard size (greater than 512 bytes per block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: LongWord pointer

This is the four byte pointer to the user's buffer that the data is to be written from. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. All of the current implementations of the SmartPort utilizing CBUS have a limitation of 767 bytes for this call. Other peripheral cards supporting the SmartPort interface, and using this call may not have this limitation.

address: LongWord

This is a device specific parameter. An example of how this call might be implemented with an extended block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes).

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	DISK WRITE PROTECTED
$2F	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

ProDOS Entry Point

The ProDOS entry point can be thought of as simply a 'front end' to the non extended SmartPort which is tuned to the whims of the ProDOS device manager. The parameters are passed in fixed absolute zero page locations, and device numbers are mapped from ProDOS conventional unit numbers to the sequential unit numbers required by the non extended SmartPort. All errors which the non extended SmartPort can return are mapped on ProDOS calls to the following error codes:

$27 I/O Error
$28 No Device Connected
$2B Device is Write Protected
$2F Device is Offline (No media)

Any 'fatal' error (bit 6 clear, bits 5-0 non-zero) that is not $28, $2B, or $2F is mapped to I/O Error ($27). All 'non-fatal' errors (bit 6 set; codes $50-$7F) is not considered an error and is mapped to $00.

Passing Parameters to Device Specific Drivers

Call parameters are passed to device specific drivers from SmartPort through fixed memory locations in absolute zero page. All input to device specific drivers are passed in an extended format. This is done so that no matter what type of call the device specific driver is receiving, the input parameters will always be found in fixed locations. This does not mean that a non extended call will be changed to an extended call. Only the organiztion of parameters is affected. Some parameters do not occupy contiguous memory when presented in an extended format. This occurs because the order of parameters has been prepared so that the parameters can be transmitted over CBUS to intellegent devices. Absolute zero page locations $40-62 have been saved by SmartPort prior to dispatching to the device specific driver, and will be restored by SmartPort after returning from the device specific driver. This means that these locations are available for use by the device specific driver. Input parameters are passed to the device specific driver as shown below:

Location	Parameters	Call Type
$42	Buffer Address (bits 0-7)	All
$43	Buffer Address (bits 8-15)	All
$44	Buffer Address (bits 16-23)	All
$45	Command	All
$46	Parameter Count	All
$47	Buffer Address (bits 24-31)	All
$48	Extended Block (bits 0-7) Status Code or Control Code Byte Count (bits 0-7)	ReadBlock & Writeblock Status & Control Read & Write
$49	Extended Block (bits 8-15) Byte Count (bits 8-15)	ReadBlock & Writeblock Read & Write
$4A	Extended Block (bits 16-23) Address Pointer (bits 0-7)	ReadBlock & Writeblock Read & Write
$4B	Extended Block (bits 24-31) Address Pointer (bits 8-15)	ReadBlock & Writeblock Read & Write
$4C	Address Pointer (bits 16-23)	Read & Write
$4D	Address Pointer (bits 24-31)	Read & Write

About the RamDisk Driver

The RamDisk driver has been implemented as a ToolSet (tool set #13). The input parameters for this tool are passed through zero page as shown on the previous page. The SmartPort will make a tool call to function #10 of the RamDisk Tool Set when SmartPort determines that the call is directed toward the RamDisk. The Direct Page Register and the Data Bank Register are set to a value of zero prior to dispatching to the RamDisk tool. The tool passes output information back to the SmartPort driver through zero page as follows:

Location	Output Parameter Passed
$000050	Error Code
$000051	Low byte of count of bytes tranferred to host
$000052	High byte of count of bytes tranferred to host

All output information being passed to the application making the call to the SmartPort driver will be passed through the buffer specified by the application in the parameter list. For more information on calls supported by the RamDisk driver, see the RamDisk ERS.

About the Unified Disk 3.5

The Unified Disk 3.5 driver resides at a fixed location on a page boundry within the firmware rom, and is contiguous to the SmartPort driver. By definition, this is a Cortland specific driver. The Unified Disk 3.5 recieves call parameters exactly the same way as the RamDisk. Output is passed to the application through the buffer specified in the parameter list. Output is passed to the SmartPort Driver as follows:

Location	Output Parameter Passed
$000050	Error Code
$E10FB4	Low byte of count of bytes tranferred to host
$E10FB5	High byte of count of bytes tranferred to host

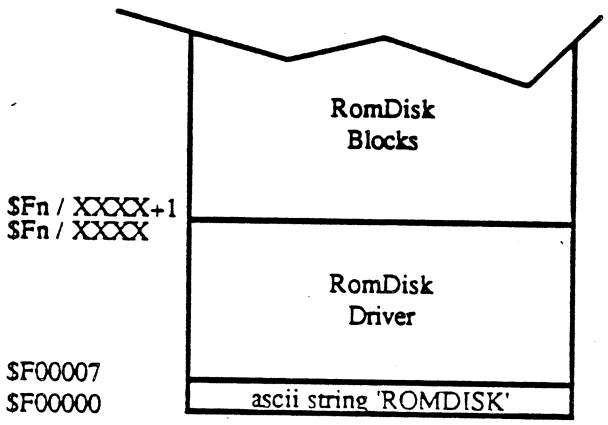
For more information on calls supported by the Unified Disk 3.5 driver, see the Unified Disk Firmware ERS.

About the RomDisk

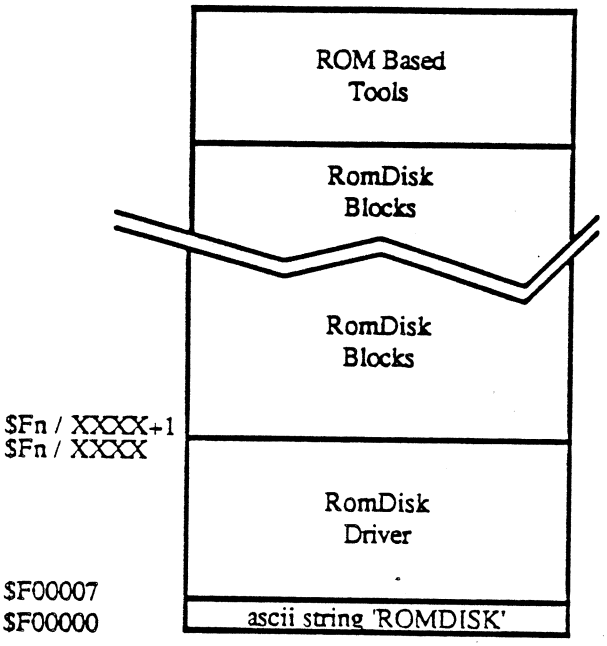
This document describes how the SmartPort firmware locates a RomDisk driver, and how parameters are passed to and from that driver. It also describes how a RomDisk driver might be implemented.

Installing a RomDisk Driver

A RomDisk driver must reside at address $F0/0000. The base address of the driver must contain the ascii string 'ROMDISK' in upper case with the MSB on. Entry to the RomDisk driver will be through address $F0/0007 (immediately following the ascii string). The SmartPort firmware will search for a RomDisk driver during device initialization time. This could occur during system boot at powerup, as a result of CTRL-RESET or by dispatching through $C500. If the ascii string 'ROMDISK' is found at address $F0/0007, an initialization call will be executed the via the RomDisk entry point. If the RomDisk returns with no error, the RomDisk driver will be installed into the SmartPort device chain. If the RomDisk initialization call returns an error, the RomDisk driver will not be installed in the SmartPort device chain.



It is possible to use the expansion ROM for both a RomDisk and ROM based extensions to the tool set by partitioning the ROM into three areas (Driver, Blocks and Tools) as shown below:



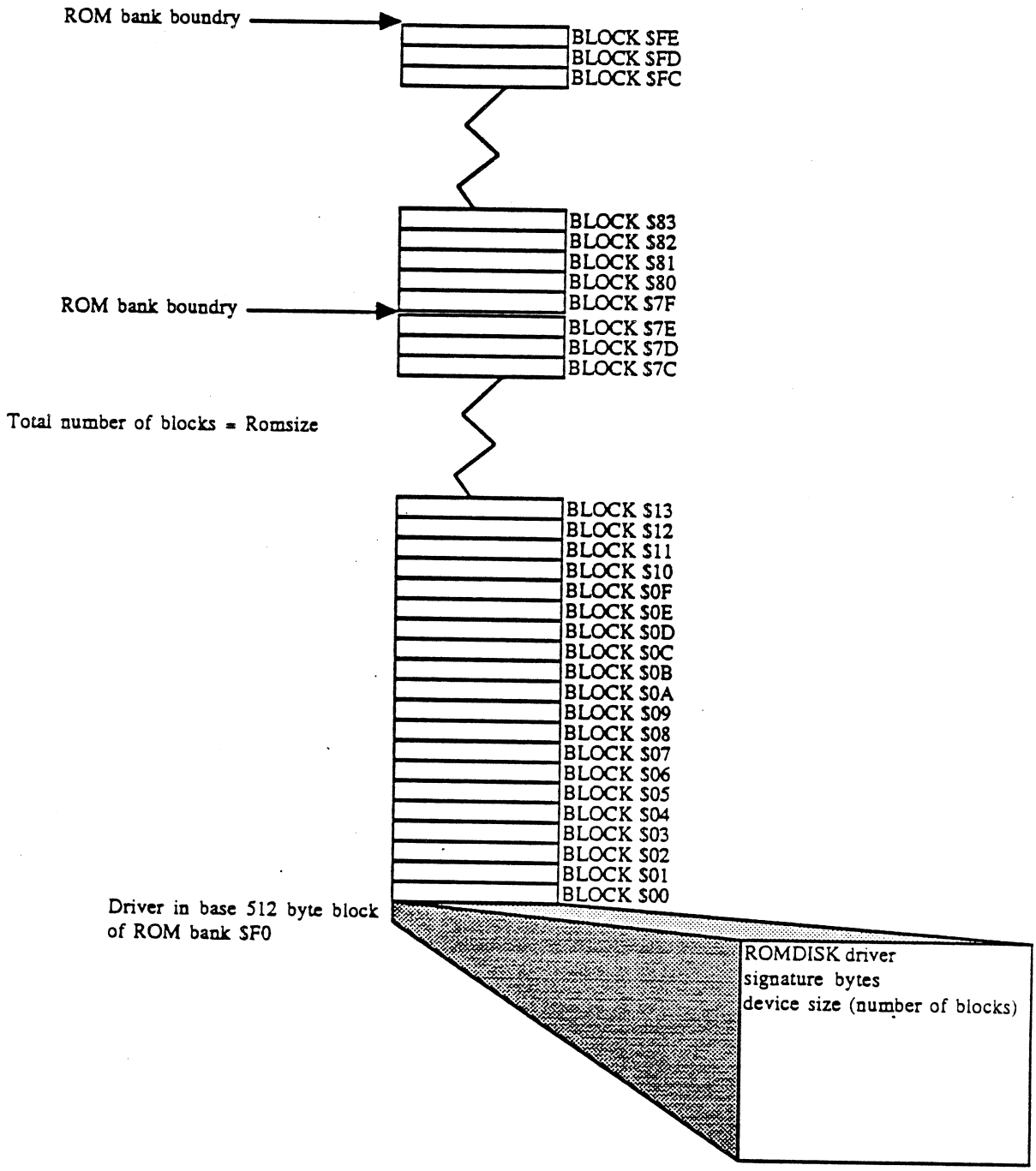
The initialization call made to the RomDisk driver should make a call to the ToolLocator to install the ROM based tool set extensions. This then allows the tool locator to dispatch to the ROM based tool set extensions directly rather than down loading the tool set to RAM.

Call parameters are passed to the RomDisk driver in a manner identical to the RamDisk driver. The Direct Page Register and the Data Bank Register are set to a value of zero prior to dispatching to the RamDisk tool. The tool passes output information back to the SmartPort driver through zero page as follows:

Location	Output Parameter Passed
$000050	Error Code
$000051	Low byte of count of bytes tranferred to host
$000052	High byte of count of bytes tranferred to host

All output information being passed to the application making the call to the SmartPort driver will be passed through the buffer specified in the parameter list.

A block diagram of a RomDisk that occupies 128k of ROM (including the driver itself) is shown below;



Note that no ROM space has been reserved for toolset expansion in this example.

Dynamic Allocation of Device Unit Numbers

The Cortland implementation of SmartPort must interact with the control panel selection of boot devices. For any given port, a boot can only occur from the first logically connected device on that port. SmartPort must support booting from any of three types of devices which include:

RamDisk
RomDisk
Disk type device (Unified Disk 3.5 or UniDisk 3.5)

At device initialization time, unit numbers are assigned to devices in ascending order according to the device type as follows:

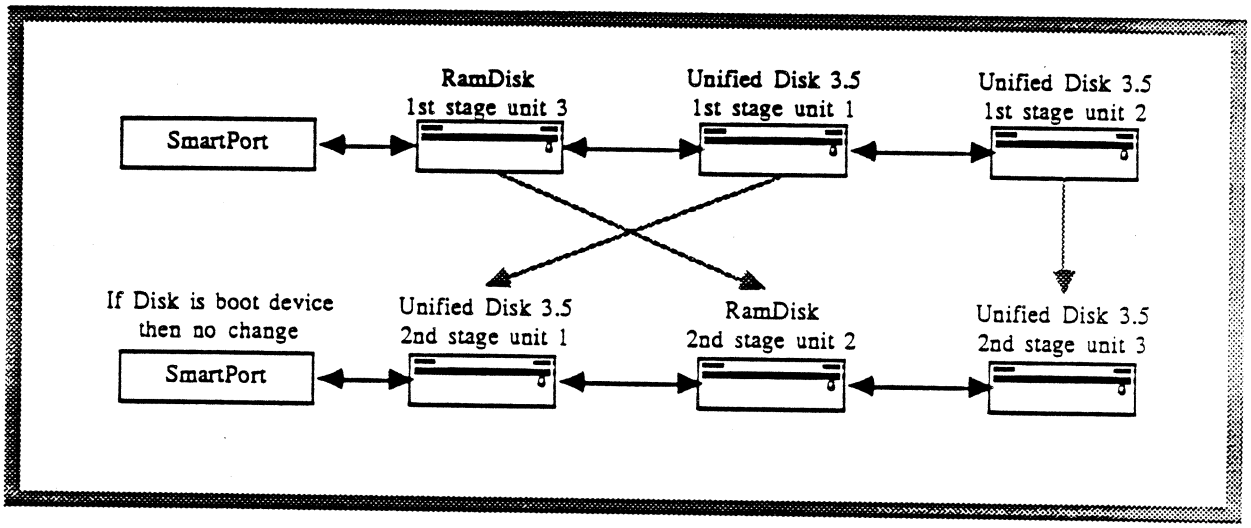
RamDisk
RomDisk
Unified Disk 3.5
UniDisk 3.5

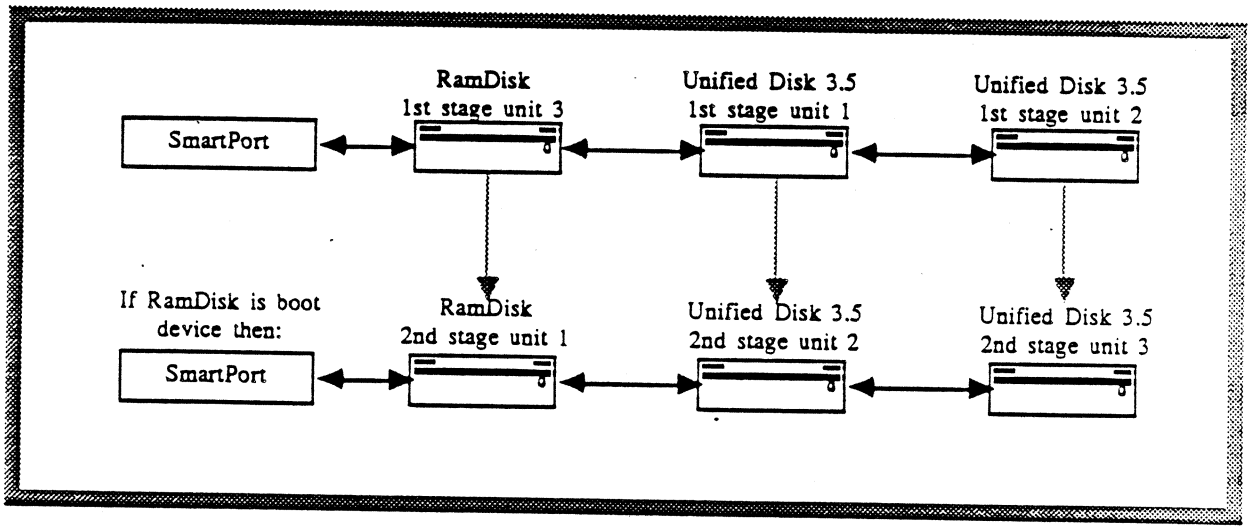
Depending on what devices are connected to the SmartPort, the selected boot device may not be the first logical device in the chain. In order to boot from the selected device, the selected device must be moved logically to the first unit in the device chain. This means that all devices that were previously ahead of the selected boot device must now be moved logically so that they are now located behind the selected boot device. This is handled by the initialization call by actually assigning unit number in two stages. The first stage assigns unit numbers as described above. The second stage remaps the units so that the selected boot device is always the first logical device in the chain. If the 'scan' is selected as the boot option in the control panel, SmartPort will place the first physical disk device as the first logical device in the device chain.

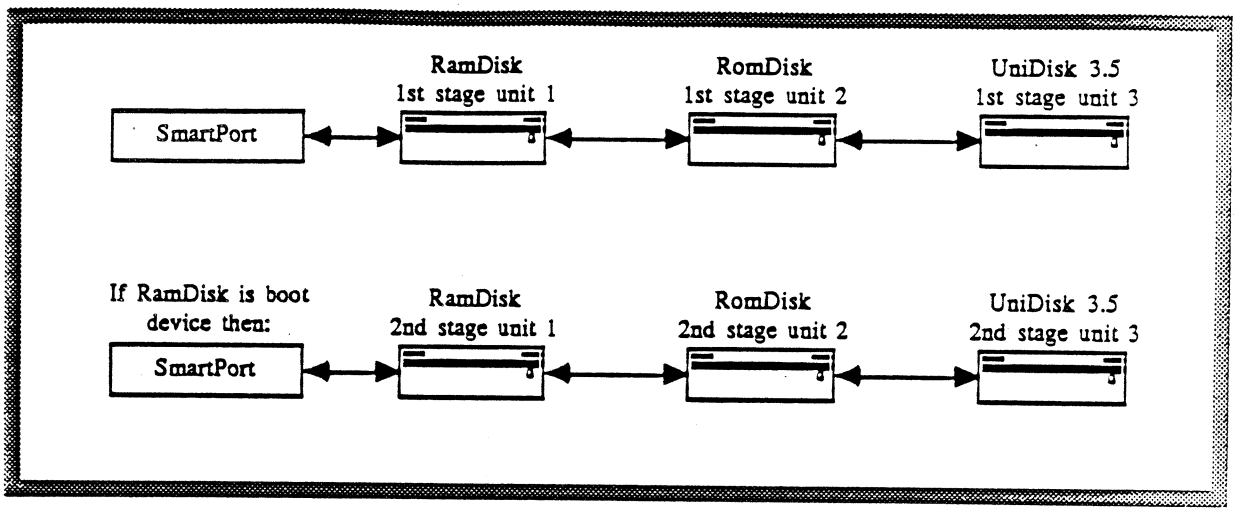
Remapping of devices has some interresting implications when running with ProDOS 1.1.1. Current implementations of ProDOS only support two devices per port or slot. If more than two devices are logically connected to the device chain, devices beyond the second device can not be accessed with ProDOS 1.1.1. The interim version of ProDOS for Cortland that will be available before ProDOS'16 is ProDOS 1.2. ProDOS 1.2 will support up to four devices on SmartPort. ProDOS 1.2 will map the to two devices beyond the second device in the device chain so that the additional devices will appear as if they are connected to slot 2. Due to the affects of the logical remapping that places the boot device as the first device in the chain, the relationship of devices and slots with ProDOS 1.2 varies with the boot configuration as set by the control panel. The affects of this is best illustrated by experimenting with the boot configuration, and listing volumes with the FILER or FINDER.

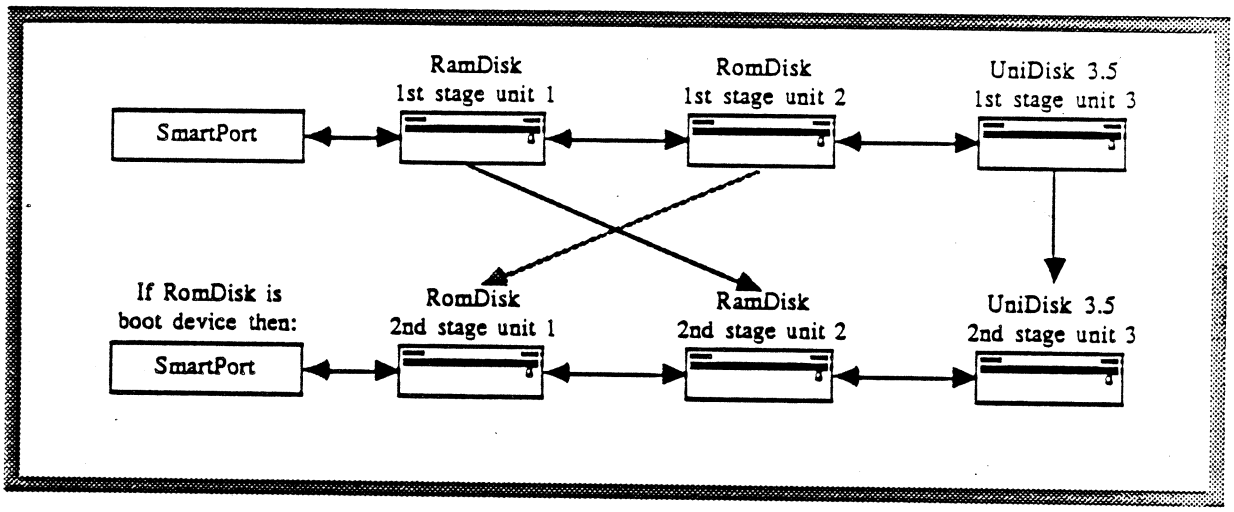
Interaction between the control panel and the logical assignment of unit numbers to devices on the SmartPort device chain will also be visable with ProDOS'16, however all the devices will appear in slot 5. No remapping of units to slot 2 will be neccessary with ProDOS'16 since ProDOS'16 will support more than two devices per port or slot.

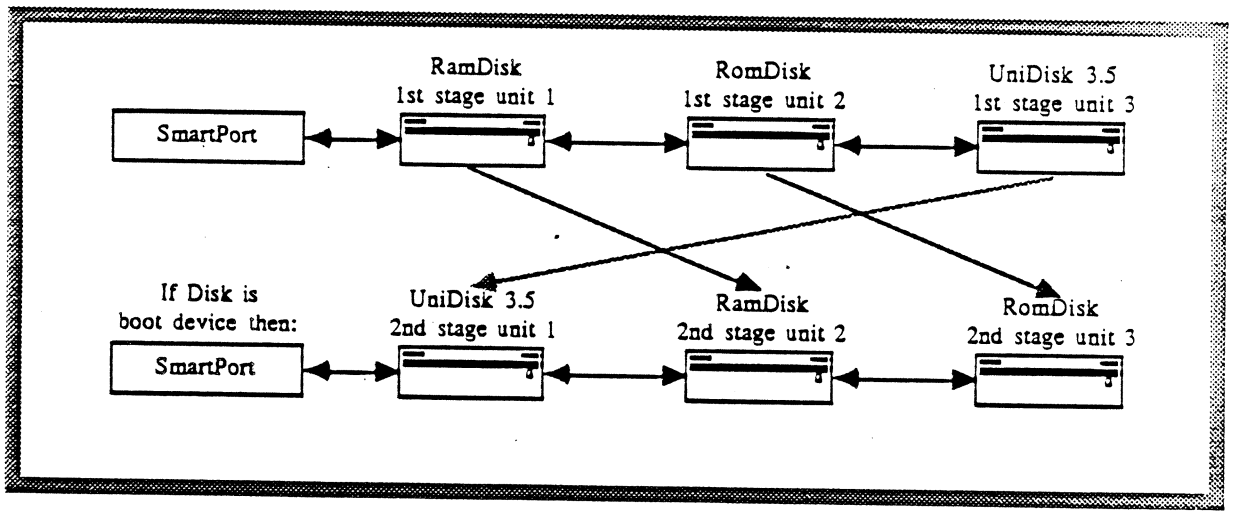
Several illustrations follow which shows remapping of devices based on the selected boot device v.s. the device configuration. Only a few of the derivations of the device mapping are shown.






RamDisk
1st stage unit 1
RomDisk
1st stage unit 2
UniDisk 3.5
1st stage unit 3
SmartPort
If RamDisk is boot device then:
RamDisk
2nd stage unit 1
RomDisk
2nd stage unit 2
UniDisk 3.5
2nd stage unit 3
SmartPort


RamDisk
1st stage unit 1
RomDisk
1st stage unit 2
UniDisk 3.5
1st stage unit 3
SmartPort
If RomDisk is boot device then:
RomDisk
2nd stage unit 1
RamDisk
2nd stage unit 2
UniDisk 3.5
2nd stage unit 3
SmartPort


RamDisk
1st stage unit 1
RomDisk
1st stage unit 2
UniDisk 3.5
1st stage unit 3
SmartPort
If Disk is boot device then:
UniDisk 3.5
2nd stage unit 1
RamDisk
2nd stage unit 2
RomDisk
2nd stage unit 3
SmartPort

Quark QC-10 Support

Support of the QC-10 presents some interresting problems for SmartPort. The QC-10 uses several phase line combinations that are in conflict with SmartPort devices such as the Unified Disk 3.5 and the UniDisk 3.5. By definition, Unified Disk 3.5 devices must be the first physical devices in the device chain. The Unified Disk hardware blocks the phase lines from being passed to devices further down the device chain during any access to the Unified Disk devices. This should prevent any phase collisions with the QC-10. However, one of the active phase conditions used by the QC-10 will reset all devices physically connected to the device chain such as the UniDisk 3.5. The net effect of an access to the QC-10 is that all intellegent devices connected to the SmartPort device chain will no longer communicate with SmartPort unless the device chain is reinitialized. This presents some problems with the normal initialization sequence. If a full initialization sequence is executed on SmartPort, dynamic reallocation of unit numbers based on the control panel selected boot device could cause the device chain order to change if the control panel setting had changed. What really is neccessary is to restore the original state of the device chain. In order to support the QC-10, a flag byte is maintained in the slot 5 screen holes ($0006FD) with a value of $A5. It is neccessary for the QC-10 device driver to modify this flag byte after each access to the QC-10. The SmartPort firmware will examine this byte on each call to the SmartPort device driver. If this byte is not set to $A5, then SmartPort will reassign the unit numbers only to the intellegent devices physically connected to the device chain. SmartPort will not reassign unit numbers based on control panel settings since this reassignment has already occured. This is done to restore the device chain to the same state that existed before the QC-10 was accessed.

Summary of Non Extended Commands and Parameter Lists						
Command	Status	ReadBlock	WriteBlock	Format	Control	Init
CMDNUM	$40	$41	$42	$43	$44	$45
CMDLIST Byte 0:	$03	$03	$03	$01	$03	$01
1:	Unit #	Unit #	Unit #	Unit #	Unit #	Unit #
2:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
3:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
4:	StatusCode	Block Addr	Block Addr		Ctrl Code	
5:		Block Addr	Block Addr			
6:		Block Addr	Block Addr			
7:						
8:						

Summary of Non Extended Commands and Parameter Lists						
Command	Open	Close	Read	Write		
CMDNUM	$46	$47	$48	$49		
CMDLIST Byte 0:	$01	$01	$04	$04		
1:	Unit #	Unit #	Unit #	Unit #		
2:			Buffer Ptr	Buffer Ptr		
3:			Buffer Ptr	Buffer Ptr		
4:			Byte Count	Byte Count		
5:			Byte Count	Byte Count		
6:			*	*		
7:			*	*		
8:			*	*		

* This parameter is device specific

Notes:

1) The read byte count and the Control call list contents cannot be larger than 767 bytes.
2) Upon return from the Read call, the byte count bytes will contain the number of bytes actually read from the device.

Summary of Extended Commands and Parameter Lists						
Command	Status	ReadBlock	WriteBlock	Format	Control	Init
CMDNUM	$40	$41	$42	$43	$44	$45
CMDLIST Byte 0:	$03	$03	$03	$01	$03	$01
1:	Unit #	Unit #	Unit #	Unit #	Unit #	Unit #
2:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
3:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
4:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
5:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
6:	StatusCode	Block Addr	Block Addr		Ctrl Code	
7:		Block Addr	Block Addr			
8:		Block Addr	Block Addr			
9:		Block Addr	Block Addr			
10:						
11:						

Summary of Extended Commands and Parameter Lists						
Command	Open	Close	Read	Write		
CMDNUM	$46	$47	$48	$49		
CMDLIST Byte 0:	$01	$01	$04	$04		
1:	Unit #	Unit #	Unit #	Unit #		
2:			Buffer Ptr	Buffer Ptr		
3:			Buffer Ptr	Buffer Ptr		
4:			Buffer Ptr	Buffer Ptr		
5:			Buffer Ptr	Buffer Ptr		
6:			Byte Count	Byte Count		
7:			Byte Count	Byte Count		
8:			*	*		
9:			*	*		
10:			*	*		
11:			*	*		

* This parameter is device specific

Notes:

1) The read byte count and the Control call list contents cannot be larger than 767 bytes.
2) Upon return from the Read call, the byte count bytes will contain the number of bytes actually read from the device.

SUMMARY OF SMARTPORT ERROR CODES

Acc Value	Error Type	Description
$00		No error
$01	BADCMD	A nonexistent command was issued.
$04	BADPCNT	Bad call parameter count. This error will occur only if the call parameter list was no properly constructed.
$06	BUSERR	A communications error with the IWM occured.
$11	BADUNIT	An invalid unit number was givien.
$1F	NOINT	Interrupt devices not supported.
$21	BADCTL	The control or status code is not supported by the device.
$22	BADCTLPARM	The control list contains invalid information.
$27	IOERROR	The device encountered an I/O error.
$28	NODRIVE	The device is not connected. This can occur if the device is not connected but its controller is.
$2B	NOWRITE	The device is write protected.
$2D	BADBLOCK	The block number is not present on the device.
$2E	DISKSW	Disk was switched.
$2F	OFFLINE	Device off line or no disk in drive.
$30-$3F	DEVSPEC	These are device specific error codes
$40-$4F	RESERVED	
$50-$5F	NONFATAL	A device specific 'soft' error. The operation completed successfully, but some exception condition was detected.