SMS
Disk Controller IC Family
OMTI Components overview
July 27th, 1987

Scientific Micro Systems, Inc.

SMS
Disk Controller IC Family
OMTI Components overview
July 27th, 1987

PUBLICATION NUMBER: **3001659-0001A.1**

SMS
Disk Controller IC Family

OMTI Component Overview

SCIENTIFIC MICRO SYSTEMS, INC.
339 North Bernardo Avenue
P.O. Box 7777
Mountain View
California 94039
TEL: 415-964-5700
TLX: 184160 SMS MNTV
FAX: 415-968-4861

REV	REVISION HISTORY	PRINT DATE
A	INITIAL RELEASE	JULY 27th, 1987

DOCUMENT NUMBER : **3001659-0001A.1**

REVISION : **A.1**

DATE OF THIS REVISION : **JULY 27th, 1987**

CONTENTS

About This Manual

This manual is the introductory volume in a complete set of application engineering documentation for the SMS series of disk controller integrated circuits. Its purpose is to give a technical overview and description to the engineer or programmer who will design these ICs into a new controller. It does not include timing diagrams, flow charts, register maps, or electrical specifications. Rather, it is provided to give the designer a higher level view of the ways in which the individual components of an OMTI chip set based controller can work together to provide the capabilities and features required by a particular application.

This manual shows, at the block level, the hardware design of a typical disk controller. The description of each block in this design illustrates the role it plays in the complete controller system and how it interacts with neighboring blocks. Although a particular application of the OMTI chip set may not contain exactly the same components, after reviewing this example the hardware design engineer will have the information and understanding necessary to quickly and fully utilize the power of the set in his own specific product.

For the firmware programmer, this manual provides an equivalent block level description of firmware appropriate for the same controller application. Much of the unique 'personality' of each new design using the OMTI chips is a result of the particular firmware system developed. In fact, a key facet of the OMTI chip family is the flexibility it affords the firmware designer in providing the specific capabilities and features best suited to his product environment. All OMTI chip based designs, however, require a core of firmware capabilities which changes little from implementation to implementation. This manual provides a functional description of these required firmware components along with guidelines and suggestions regarding possible enhancements for specific situations.

MANUAL OVERVIEW

Unlike manuals for the individual components of the OMTI chip family, this introduction is nearly all tutorial, containing no detailed engineering parameters, specifications, or diagrams. It is divided into three sections. The first is a discussion of the component pieces of a typical disk controller based on the OMTI chip set. It primarily discusses the high level roles played by each functional block and their relationship to each other. The second section contains step by step descriptions of typical controller operations. It is provided to illustrate the flow of data through the controller system and to point out specific areas of component interaction. In particular, this section highlights those controller functions which are most effected by the specific hardware and firmware design of the individual controller. The final section of this manual contains a block level discussion of a typical firmware implementation and a collection of checklists and design guides for the firmware developer.

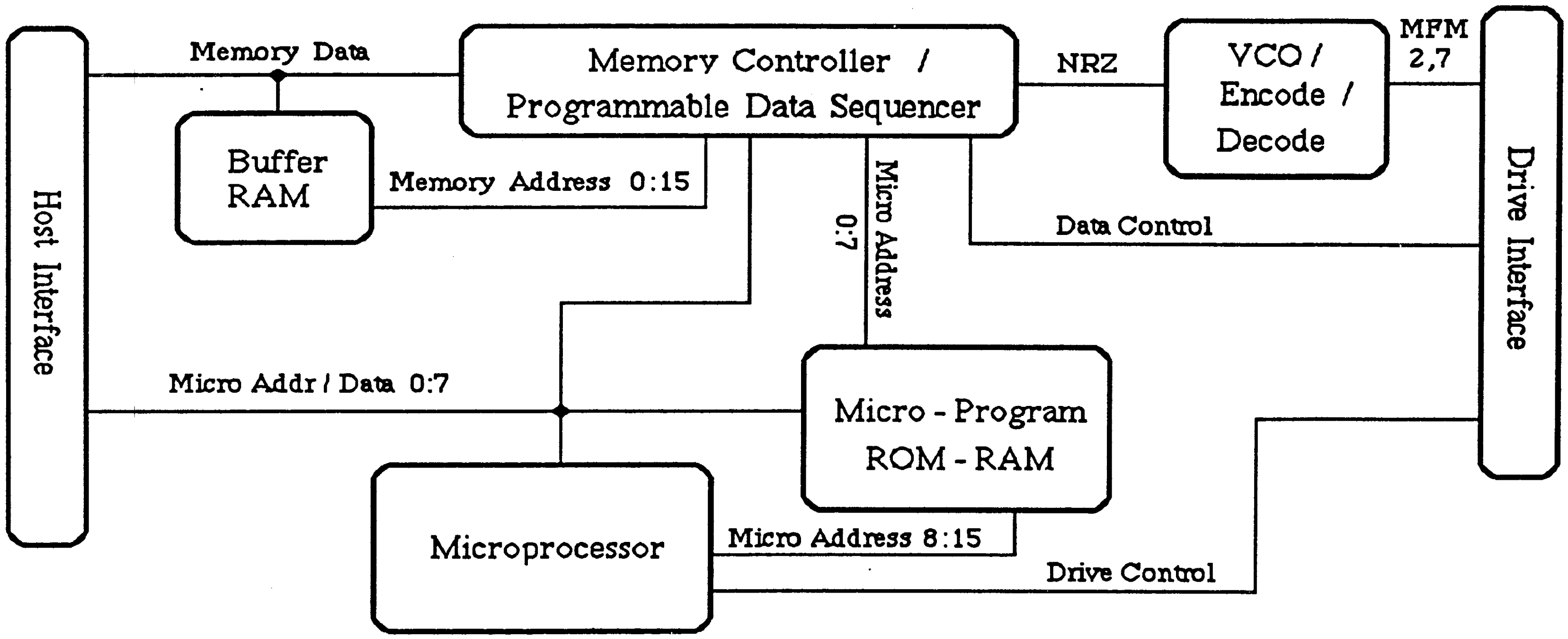
HOW TO USE THIS MANUAL

The engineer or programmer working with the OMTI disk controller chip set for the first time should read through this entire manual once before beginning detailed design activities. The time spent on such an introduction will not only familiarize the reader with the component ICs in the family, and how they relate to one another, but will also serve two other purposes. First it will familiarize the reader with the specific terms and names used in the complete catalog of OMTI chip set documentation. Second, and perhaps more important, it will help the new designer to understand how the chip set's designers themselves assumed the components were to be used. This insight should help to explain the specific features of each component and help the design engineer most easily utilize its full potential and capability. Finally, regardless of specific design responsibility, both hardware and firmware engineers are encourages to read all of the sections of this manual. Because of the very close interaction between the hardware components of a disk controller and the microcomputer firmware which configures and controls these components, hardware design engineers and firmware programmers will each want to be familiar with the work and design objectives of the other.

I.

Controller Architecture

On the next page is the block diagram of a typical Winchester disk controller based on the SMS family of disk controller ICs. Each of the main functional hardware components is represented as a block in the diagram and, in fact, nearly all of the blocks represent single integrated circuits in an actual OMTI chip set based controller. Following the figure is a functional description of each of the blocks. First, however, notice that the block diagram is effectively arranged in two rows. The top row contains those functional blocks which are in the path of user data from the host to the disk and back again. These blocks actually process the data and thus operate at the controller data rate. The bottom row of the diagram contains the components of the controller's microprocessor system. The microprocessor manages the data path components, processes the controller's high level commands and status, and is generally responsible for the 'personality' presented by the controller to the host. Although the microprocessor is not directly in the path of system data to or from the disk drive, the proper operation of the controller often depends upon time critical support from the microprocessor. The various data paths between blocks of this sample controller design are explained in the section of this manual which follows the block description. The critical timing aspects of the microprocessor's operation is covered in detail in the final chapter of this manual. The final section in this chapter further explains the three levels of controller to drive interface.



Disk Controller Block Diagram

Block Level Organization

HOST INTERFACE:

The host interface block in the Disk Controller Block Diagram represents the controller's hardware interface to the host computer system. This interface may connect the controller to the host system's memory or I/O bus as, for example, in the case of a controller built for the IBM PC-XT or IBM PC-AT buses. Alternatively, many host systems require that the host interface be made compatible with an industry interface standard such as SCSI, the Small Computer System Interface. Note that in each of these cases, single integrated circuits from the OMTI family may be used in a controller design to provide this complete function. Available ICs for this role include:

5090	---	IBM PC / PC-XT 8 bit bus
5098	---	IBM PC-AT 16 bit bus
5080	---	SCSI ASYNC
5086	---	SCSI Sync / Async

In addition to providing the direct electrical interface required between the host system and the controller, the host interface generally supports some level of arbitration and communication protocol between the host and the controller for the exchange of commands, data, and status. This is true for each of the ICs listed above.

RAM BUFFER:

The RAM buffer provides controller storage space for user data as it passes throught the controller from the host to the disk or from the disk to the host. In addition, a portion of the RAM buffer is used by the controller itself to save command information, track format sequences, and controller status. In most controllers, the RAM buffer is provided by a single static RAM IC.

MEMORY CONTROLLER / PROGRAMMABLE DATA SEQUENCER:

This functional block of the controller may easily be though of as the heart of the data path circuitry. It serves as the central manager of all user data as it flows through the controller. Under the programmed control of the microprocessor, this controller component is responsible for:

Data transfer between the host interface and the RAM buffer
Data transfer between the RAM buffer and the disk via the encode / decode chip
Microprocessor access to data in the RAM buffer
Disk track format / sector size
Disk data location and transfer
CRC / ECC generation & verification

Despite the complexity of services required of this block of the controller, OMTI chip set based designs provide all these capabilities in as little as one integrated circuit:

For most Winchester disk drive controllers -
5055 --- Memory Controller / Programmable Data Sequencer

For systems which also support other peripherals -

5050	---	Programmable Data Sequencer
5060	---	Direct Memory Access Controller

VCO / ENCODE / DECODE:

The bit serial data interface between the Data Sequencer and the disk read/write electronics is provided by this component of the controller. Although not programmable, and invisible to the controller's microprocessor and the host, this component is the crucial link between the syncronous data of the host and controller systems and the asyncronous data stream of the physical disk. When writing to the disk, it provides proper data encoding and pre-compensation. When reading from the disk, it syncronizes to the varying disk data stream and provides address mark detection and data decoding. When combined with a small number of discrete components, a single OMTI family component provides these capabilities:

5070	---	Encode / Decode / VCO	for MFM data
5027	---	Encode / Decode / VCO	for 2 of 7 RLL code

MICROPROCESSOR:

As described earlier, the blocks in the Disk Controller Block Diagram which make up the first row, and which were detailed above, constitute the complete data path for a typical Winchester disk controller. The balance of the controller is made up of the microprocessor and its support components. The microprocessor coordinates and controls the actions of the data path components. It also provides the higher level control and computational capabilities. The tasks undertaken by the microprocessor include:

Programmable control of data path components
Controller command acceptance and execution
Disk drive mechanical positioning control
Data stream error recognition and handling
Controller status generation and delivery

The OMTI chip family is designed to connect directly with any of a selection of available microprocessors which employ the bus structure of either the Zilog Inc. Z8 or the Intel Corp. 8051.

MICRO-PROGRAM ROM / RAM:

The final block in the Disk Controller Block Diagram is the ROM and RAM directly associated with the microprocessor. The size and composition of memory in this block is largely determined by the specific capabilities and features included in each controller implementation.

Data Paths

In order to understand the internal organization of any disk controller, it is just as important to know about the data paths between the functional hardware blocks as it is to be familiar with the capabilities of the blocks themselves. In fact, a major characteristic of the internal structure of those blocks is the way in which they implement the various data paths of the controller.

The Disk Controller Block Diagram used previously to outline the component hardware blocks of the sample controller also displays these interconnecting data paths. As might be expected, each of these paths also represents a real electrical interface between components of the SMS/OMTI controller IC family. The Disk Controller Block Diagram of this manual may thus also serve as a kind of "roadmap" to position each of the components in relation to the others as the hardware designer scans the manuals for the individual components of the family.

The interconnections of the sample Winchester controller constitute nine data paths of various widths along with accompanying control signals:

MEMORY DATA 0:7

The byte wide memory data bus is the main path for all parallel user data as it flows into or out of the controller. All such data is stored at various times in the Buffer RAM. This buffering capability absorbs the timing differences between the host and the disk and between the host and the microprocessor. For example:

> Data may be transferred between the host interface and the buffer RAM at high speed, without regard for disk latency or transfer rate.
>
> Data may be transferred between the buffer RAM and the disk drive unimpeded by arbitration or access delays at the host interface.
>
> Extended command and status information may be transferred, as a block, quickly between the host interface and the buffer RAM independent of the immediate activity of the microprocessor or the time it requires to process this data.

The data buffering facility provided by the memory data bus and the buffer RAM also provides the hardware support necessary for such capabilities as data read-ahead, caching, and ECC sector data error correction.

Access to the memory data bus is under the exclusive control of the memory controller section of the memory controller / programmable data sequencer block of the controller. It generates the control signals associated with accesses on the bus and also supplies the buffer RAM address for each access. The host interface block of the controller is allowed to transfer data on the bus via a typical DMA controller REQ / ACK handshake with the memory controller.

MEMORY ADDRESS 0:15

As noted above, the memory data bus is controlled exclusively by the memory controller section of the memory controller / programmable data sequencer block of the controller. Part of that control is the generation of a buffer RAM address for each access. The memory address bus is the path for these addresses. It is 16 bits wide, thus allowing the buffer RAM to be as large as 64 Kbytes.

MICRO ADDRESS / DATA 0:7

All data tranferred between the microprocessor and other blocks of the controller passes along the byte wide micro address / data bus. In addition, this multiplexed address and data bus also carries the least significant byte of microprocessor address during each bus access cycle. This bus is simply the external data bus of the microprocessor, which controls it at all times. Transfers on the bus are under control of the microprocessor's normal read and write signals. This bus is used by the microprocessor to configure and control the host interface and the memory controller / programmable data sequencer. For controller command and status transfers, the microprocessor then used this same path for communications through the host interface. It is also the bus over which the microprocessor fetches its instructions from the micro-program ROM. Finally, the microprocessor uses this bus as it transfers its local data to and from local RAM.

MICRO ADDRESS 0:7

The micro address / data bus described above is a multiplexed address and data bus. In order to use the contents of this bus properly, the address information present at the beginning of each bus access cycle must be latched and saved to be used as part of a complete address for the subsequent bus data as it is read or written by the microprocessor. Some components on that bus, the host interface and the memory controller / programmable data sequencer, perform this function internally. For the micro-program ROM and RAM, however, this de-multiplexing must be done externally. Since the memory controller / programmable data sequencer already contains this hardware function for its own internal use, it is a simple matter to provide the same capability for these external devices. The resulting de-multiplexed address byte is provided on these low order bits of the micro address bus. This bus is thus always sourced by the memory controller / programmable data sequencer and is controlled, indirectly, by the microprocessor.

MICRO ADDRESS 8:15

While the least significant byte of the microprocessor's address is de-multiplexed and provided on micro address 0:7, the most significant byte is provided directly by the microprocessor on micro address 8:15. This half of the address bus is not multiplexed and is, obviously, controlled by the microprocessor.

DRIVE CONTROL

As will be described in more detail in the next section of this chapter, the microprocessor is directly responsible for controlling mechanical movement portions of the disk drive interface. The control signals associated with this function connect the microprocessor directly with the drive control electronics and may be thought of as a control bus in their own right.

DATA CONTROL

This intermediate level of drive interface control, also described in the next section, is controlled by the data sequencer portion of the memory controller / programmable data sequencer. Much like the drive control signals, the data control signals connect the data sequencer directly to the disk drive read/write channel.

NRZ - MFM - 2,7

The final two data interconnections forming the sample Winchester controller are the serial data connections from the data sequencer to the VCO/encoder/decoder and then from the VCO/encoder/decoder to the disk drive's raw data read/write channels.

Disk Drive Interface

The interface between a typical Winchester disk drive and its associated controller may be logically divided into three areas. Going from one area to the next is characterized by a reduction in the dependence on programmable flexibility and an increase in data or processing rates required. The three interface areas and the blocks of the sample controller which handle them are:

DRIVE CONTROL:

The area of disk drive interface most removed from the actual high speed disk data transfer is drive control. This level of control is responsible for mechanical positioning of drive heads (seeking) and read/write head selection. Since drive control is the least time critical of the three areas and since its implementation is the most depend on variable factors such as the number of drive platters, this interface area is best supported directly by the microprocessor.

DATA CONTROL:

Once a particular cylinder and head have been selected by the drive control section of the controller, the data control section is responsible for the management of the data on each track. This includes the organization of sector header and sector data, sync fields, address marks and error detecting and correcting codes. The drive interface signals controlled or monitored at this level include write gate, index, and sector. Related to the timing of the drive's rotation at this level are also signals to the VCO/encode/decode block to control address mark generation or detection and data clock selection. Control timing at the data control level must be precise to the individual bit level. As a result, this timing may not be provided by the microprocessor. Rather, the general format for the data on each disk track is specified by the microprocessor to the programmable data sequencer portion of the memory controller / programmable data sequencer as programmable parameters. The processing of these controls in real time is then accomplished by the data sequencer under its own timing.

SERIAL DATA:

The final level of controller to drive interface is at the level of the serial data stream. There are actually two parts of this data stream in the sample controller. The first is the NRZ data path between the data sequencer and the VCO/encode/decode block. The second part is the encoded data between this block and the physical drive read and write channels. Depending on the particular encoding chosen for the specific controller, this may either be MFM data or 2,7 RLL data. Since timing at this level for such subtle items as bit level write precompensation is less than a single bit time, it is controlled entirely by fixed hardware. This fixed control is integrated along with sub-byte level processing within the VCO/encode/decode block of the controller.

II.

Controller Operations

Each host command issued to a typical disk controller results in a well defined sequence of execution steps within the controller. This manual section discusses a selection of typical host commands. In each case, the steps taken by the controller are detailed along with a description of the data which flows through the system as the command is executed. These command summaries illustrate two important facets of controller operation. First they demonstrate the way in which the capabilities of the OMTI chip set are applied to carry out controller operations. In particular, they show the portions of each typical host command which are performed fairly directly by the chip set and which portions are primarily the responsibility of the microprocessor and its firmware. These distinctions are important to the firmware designer who must provide control functions for the first portion and complete support for the later items. These details are also important to the hardware designer who must insure that the microprocessor is supported with appropriate control and status signals to implement these functions. Second, the command summaries give a clear description of internal controller data flow. This is also of critical importance to firmware designer's since much of the job performed by the controller's microprocessor has to do with data flow: command data, user data, and status data. At the same time, the hardware designer must understand the required data paths in order to insure that the proper channels are available for data movement.

INITIALIZE:

Initialization commands from the host to the disk controller may be of several sorts. In systems where the function of the controller is tightly defined, a command of this type may only amount to the equivalent of a software RESET. In other circumstances, the controller's microprocessor may receive a great deal of information about its operation from an initialization command. In either case, these commands are generally characterized by the transfer of data from the host to the controller microprocessor only. A typical initialization command consists of three stages. The host sends the command to the controller, the initialization data is moved, and the controller returns appropriate completion status. Two of these stages, command and status, are likely to be common to all of the controller's commands. The steps in each of these stages are described below.

Command: Under most circumstances, the initiation of a command by the host is processed immediately and directly by the microprocessor. Unless complicated command overlap is part of the specific controller's design, the microprocessor is idle at the time the command is begun and is free to devote full attention to receiving it. In order to collect the series of command bytes directly from the host, the microprocessor uses the path through the host interface block which allows direct host - microprocessor communications. Once the series of bytes which describe the command and its parameters has been moved into the microprocessor system, it is decoded and command execution is directed.

Data: Depending on the particular command protocol implemented by the controller, the amount of data actually passed with the initialization command may vary from none to a significant block. For small amounts, the command data path through the host interface may be used directly. For larger blocks, the microprocessor may program the memory controller portion of the memory controller / programmable data sequencer to quickly transfer the data into buffer RAM where it may be accessed after the command is completed.

Status: In most respects, the returning to the host of command status by the controller is typically the functional inverse of command acceptance. As in that case, the controller's microprocessor is generally devoted at this point to completing the transfer as directly and quickly as possible in order to interrupt the host for as short a period as possible. The direct data path from the microprocessor through the micro address / data bus and the host interface is again an appropriate path for data.

READ:

Commands to read data from the controller / disk sub-system are generally the most commonly executed controller functions. At the same time, a complete transfer of data from the disk media to the host interface is one fo the most complex activities undertaken by the controller. In addition, since this class of command is undertaken so often, it is important for it to be performed with a minimum of unnecessary controller activity and overhead. Finally, disk read commands must be supported by the most comprehensive error detection and recovery capabilities of the controller.

The steps required for a typical disk to host transfer command are:

The command is passed to control as described above for initialization commands.

The microprocessor checks the command data for errors such as illegal disk media address, etc. If an error exists, processing transfers immediately to status reporting.

The microprocessor checks the disk drive to make sure it is ready for data transfer. Again, if an error exists status is reported and the command abandoned.

If the command protocol specifies media addresses logically rather than physically, the microprocessor translates the specified address into physical cylinder, head, and sector. This translation will often involve the consideration of media areas not available for normal data because of surface flaws. Depending on the particular system, this "flaw management" may be very sophisticated and involve a significant amount of microprocessor firmware.

The microprocessor executes appropriate routines to mechanically position the read/write heads of the drive to the requested cylinder. Since this process may involve significant delays, this process is often overlapped with other operations or commands.

The proper head (track) within the cylinder is selected by the microprocessor.

The microprocessor programs the memory controller portion of the memory controller / programmable data sequencer to initialize it to transfer sector data as it is received from the data sequencer portion into the proper address range in the buffer RAM.

The microprocessor programs the data sequencer portion of the memory controller / programmable data sequencer with the proper sequencer read command, with the complete starting sector header data, and with the number of sectors to be read in sequence.

The data sequencer is started by the microprocessor.

As each required sector is read from the disk into the buffer RAM, the microprocessor verifies that no ECC error has been detected and starts the buffer RAM to host interface DMA controller within the memory controller portion of the memory controller / programmable data sequencer to transfer the sector data to the host.

As the complete operation proceeds, the microprocessor must monitor block sizes to insure that the proper number of sectors is transferred and that any partial sector byte count required to complete the transfer is processed correctly.

After the final data is transferred, status is reported and the command is completed.

Multiple track transfers:

If the amount of data requested is not exhausted as the last sector of the track is read, the microprocessor must break the transfer into sections at physical track boundaries. In some cases, these breaks will only require a change of selected read/write head before the transfer can continue. In others, a physical cylinder change will be required. In either case, quick response from the microprocessor is important.

Read command error conditions:

No other command processed by the controller has as many potential error conditions as a disk read command. An important part of the particular strength and "personality" of any specific controller implementation is reflected in its capabilities in this area. The responses to any error condition are generally a combination of re-tries and corrective actions. Among the possible errors and their common responses are:

Wrong track: Recalibrate heads - reseek - reselect heads
Sector ID sync field or address mark not found: Retry
Sector ID ECC/CRC error: Retry
Data sync field or address mark not found: Retry
Data ECC error: Retry - compute correction and correct buffer data

WRITE:

Second only to read commands in both frequency and complexity are user data write commands. Fortunately, most of the controller capabilities required for read commands apply equally to write commands. In particular, all of the drive positioning and data area location facilities described above for disk read activities apply fairly directly to disk write operations.

The steps required for a typical host to disk transfer command are:

> The command is passed to the controller as before.
> The microprocessor checks the command data for errors and the drive for readiness as for the read command. As before, if an error exists status is reported and the command abandoned.
> If the command protocol specifies media addresses logically rather than physically, the microprocessor translates the specified address into physical cylinder, head, and sector exactly as for the read command.
> The microprocessor executes appropriate routines to mechanically position the read/write heads of the drive to the requested cylinder.
> The proper head (track) within the cylinder is selected by the microprocessor.
> Since the user data transferred through the write command comes from the host instead of the disk, as was the case for the read command, each sector length data block must be moved from the host interface into the buffer RAM before a transfer is begun to write the corresponding sector from the buffer RAM to the disk. If this is not done, it is possible for the buffer to empty before the sector is complete.
>
> The microprocessor programs the memory controller portion of the memory controller / programmable data sequencer to transfer data from the host interface into the proper address range in the buffer RAM.
>
> The microprocessor programs the data sequencer portion of the memory controller / programmable data sequencer with the proper sequencer write command, with the complete starting sector header data, and with the number of sectors to be written in sequence.
>
> When sufficient data is available in the buffer RAM, the data sequencer is started by the microprocessor.
>
> As the complete operation proceeds, the microprocessor must monitor block sizes to insure that the proper number of sectors is transferred and that any partial sector byte count required to complete the transfer is processed correctly.
>
> After the final data is transferred, status is reported and the command is completed.

Multiple track transfers:

> Just as for the read command, if the amount of data requested is not exhausted as the last sector of the track is written, the microprocessor must break the transfer into sections at physical track boundaries.

COMPARE:

Not all disk controllers support a compare instruction. When it is fully supported, its function is to read data from the disk as in a read command and from the host as in a write command and then to compare the two data streams within the controller. This function allows the contents of the disk to be compared exactly with host data. Media addresses and transfer lengths are specified in the same way as in a read or write command. Hardware support for a compare command is fully provided by the OMTI chip set. The memory controller / programmable data sequencer may be programmed by the microprocessor to perform the data transfers and to compare the data internally.

A less complex version of the compare instruction causes the controller to scan the specified sectors on the disk as above without reading data from the host system or actually comparing data. It relies on the fact that if no ECC errors occur when scanning the disk media, the data is likely to have been correctly written.

STATUS:

In many respects typical status commands are like the status reporting phase of other controller commands. In some controller implementations the status command is, in fact, identical to the status phase of other commands. A variety of other implementations are possible, returning a variety of status data structures. In each case, however, the status command is much like the inverse of the initialization command. It passes controller data from the controller microprocessor to the host. The choice of transfering extended status to the host through the buffer RAM or through the direct microprocessor to host interface path is determined by the complexity of status data and system performance considerations.

FORMAT:

Nearly all disk controllers provide some media formatting capability. This function is provided to allow the host to direct the controller to write data sector locating information on the disk media. The particular information written is partially determined by the OMTI chip set components themselves, but within a broad framework they support a wide selection of data formats. In many respects, a track format command is much like a multiple sector track write command. The fundamental differences are that variable sector data is not written and rather than reading headers to locate specific sectors, the entire serial track is written. Nearly all of the control and timing information required to format a track is either built into the data sequencer part of the memory controller / programmable data sequencer or is programmed into control registers there by the controller's microprocessor. The contents of the four data bytes in each sector header, however, are specified in a table located in the buffer RAM. This allows a great deal of flexibility in such things as sector interleaving, etc. Depending on the requirements for any specific controller implementation, some flexibility may be allowed within the host command regarding the contents of the data in the sector headers or all of the data for sector header may be written to the buffer RAM by the microprocessor itself through the memory controller / programmable data sequencer.

The steps required for a typical format command are:

The command is passed to the controller.

The microprocessor checks the command data for errors and the drive for readiness, as for read or write commands. As before, if an error exists status is reported and the command abandoned.

If the command protocol specifies media addresses logically rather than physically, the microprocessor translates the specified address into physical cylinder and head. The sector portion of the translated logical address is ignored.

The microprocessor executes appropriate routines to mechanically position the read/write heads of the drive to the requested cylinder.

The proper head (track) within the cylinder is selected by the microprocessor.

If the sector header data is to be supplied by the host, the microprocessor programs and starts the memory controller to transfer the table data from the host interface to the buffer RAM.

The microprocessor programs the various timing, data, and error correcting parameters into the data sequencer portion of the memory controller / programmable data sequencer and then starts the sequencer.

The data sequencer waits for the drive to signal its rotational index and then writes a complete track of data.

The microprocessor returns appropriate status to the host.

READID:

Many disk controllers do not provide a command to read sector header information. When this command is provided it performs header read operations under much the same protocol as normal read operations. Its fundamental applications are the determination of disk rotational position and in media flaw mapping schemes which use header data as pointers or flags.

SEEK:

The seek command is provided to allow the host to specify a disk media location before that location is actually accessed to read or write data. It serves two functions. First it allows the host to anticipate a future data transfer and to begin any necessary disk mechanical movement in advance. Second, it is sometimes used to simplify other disk access commands by separating the specification of disk cylinder or head from the specification of sector address or host memory transfer address.

III.

Firmware Guidelines

The first two chapters of this manual have described the architecture of a typical Winchester controller based on the SMS disk controller chip family and common commands which might be executed by such a controller. Included in these discussions have been many references to the fundamental responsibilities of the controller resident microprocessor and its firmware. Beyond these basic definitions, nearly all of the unique characteristics and system strengths of any specific controller based on these chips is primarily a function of the microprocessor firmware included. Depending on the system requirements of the individual controller design, emphasis may be placed on a variety of performance or command capabilities. In some applications, minimum command flexibility with maximum performance may be appropriate. In others, a broad and flexible command set may be desirable. Certain environments may require more robust error handling or media defect management than others.

No general application manual can specify which emphasis is best for a particular system environment. Those decisions are necessarily the unique responsibility of the engineers and programmers developing the controller. This final manual chapter is provided, rather, as a basis for firmware specification. It contains an outline for the key points of a typical firmware implementation to complete a controller built on the Disk Controller Block Diagram of the first chapter. For each major section of that outlline it also contains a short checklist of design items which pertain to that section as an aide to initial firmware planning. Along the way, various guidelines are also added to highlight specific points which may be easy to overlook.

FIRMWARE OUTLINE

- Reset Processing
  - Microprocessor test
  - ROM checksum
  - RAM test
  - Timer checks
  - Host interface programming
  - Default memory controller / programmable data sequencer programming
  - Recalibrate disk heads
  - Load disk resident data / flaw table
  - Time spin up
- Main loop
  - Manage timers
  - Watch for command
  - Command / status queuing

- Command processor
  - Handshake command
  - Decode command
  - Process command errors
  - Branch to command
    - Initialize command
    - Read command
    - Write command
    - Compare command
    - Readid command
    - Status command
    - Format command
    - Seek command
  - Encode status
  - Handshake status

- Command support utilities
  - Program data sequencer
  - Convert logical media address to physical address
  - Media flaw management
    - Flaw map storage
    - Redundant map checks
    - Reserved locations
  - Buffer RAM space allocation
    - Host - buffer allocation
    - Buffer - device allocation
    - Cache / Read ahead

- Error processing
  - Error status return formatting
  - Error recovery / retry counters
  - ECC correction computation

- Self test / manufacturing test
  - Extended ROM / RAM testing
  - Random media read / write / read
  - Media scan

RESET PROCESSING:

This module contains the firmware which is executed once whenever the controller is powered up or hardware reset. It has two parts. The first part consists of local hardware self tests. They verify that the controller itself and selected additional parts of the storage system are operating correctly. The level of testing required and the response to any errors is system dependent. In some cases it may be worthwhile for the controller to attempt to continue operation even after a fault has been detected. In others, especially where high reliability is important or where redundant systems are available, immediate error flagging and controller halting may be appropriate. Typical tests include:

Microprocessor test
ROM checksum
RAM test
Timer checks

== Guideline: A policy, defined early in the controller development cycle, regarding the controller's reaction to errors under differing conditions will insure that consistant responses are experienced by the host system.

== Guideline: Extensive local testing may not be worthwhile if the host system is not equipped to handle an error intelligently once discovered.

The second part of reset processing covers the initialization of programmable hardware, initial drive mechanical conditioning, and loading or initialization of media flaw management data:

Host interface programming
Default memory controller / programmable data sequencer programming
Recalibrate disk heads
Load disk resident data / flaw table
Time spin up

== Guideline: If power up processing implies extended delays for items such as drive spin up and if the controller may also experience frequent hardware resets from the running host system, a hardware or firmware ability to recognise the difference between the two events will probable be required to avoid crippling, unnecessary storage system delays.

MAIN LOOP:

Depending on the level of command or status queuing or overlap, this main controlling code module may range from a very simple endless loop to a sophisticated timer driven monitor. It may include:

Manage timers
Watch for command
Command / status queuing

== Guideline: Once the host system has begun the process of issuing a command, delays or inefficiencies in this firmware will result directly in system delays.

== Guideline: High reliability systems may execute self test firmware while waiting for commands.

COMMAND PROCESSOR:

This module is responsible for accepting, decoding, and executing each command. Typical components are:

- Handshake command
- Decode command
- Process command errors
- Branch to command
    - Initialize command
    - Read command
    - Write command
    - Compare command
    - Readid command
    - Status command
    - Format command
    - Seek command
- Encode status
- Handshake status

== Guideline: Since many commands share common execution phases such as mechanical positioning, logical to physical address translation, etc., careful attention to command decoding routines can significantly reduce code space requirements with minimum impact on execution time.

COMMAND SUPPORT UTILITIES:

This collection of subroutines provides common support capabilites for the command execution routines. Depending on the particular controller implementation, some of these routines may either not exist or be very extensive. The collection includes:

- Program data sequencer
- Convert logical media address to physical address
- Media flaw management
    - Flaw map storage
    - Redundant map checks
    - Reserved locations
- Buffer RAM space allocation
    - Host - buffer allocation
    - Buffer - device allocation
    - Cache / Read ahead

== Guideline: Simple subroutines to convert media addresses from logical to physical, especially with flaw management, often take far longer to execute than anticipated. Design carefully.

== Guideline: System integration problems can be caused by a failure to determine early and clearly whether the first logical block on a logically addressed device is number zero or number one.

ERROR PROCESSING:

A very significant portion of the typical controller's firmware is dedicated to error detection and recovery. These routines are responsible for error retry processing and recovery. They include:

Error status return formatting
Error recovery / retry counters
ECC correction computation

== Guideline: Some of the most complex firmware in many controllers is the portion which computes error correction patterns from the ECC results. Detecting errors and producing the required data for this computation is computationally very simple when compared to the actual generation of offset and pattern information.

SELF TEST / MANUFACTURING TEST

Not all controllers will include these self test routines. For others, however, they provide an important burn-in and test capability. Typical components are:

Extended ROM / RAM testing
Random media read / write / read
Media scan