

Nisha Firmware Specification

Revision 1.0-0

Dec 9, 1984

Written by Rodger Mohme

MS-190 x4879

Some Useful Definitions:

The following is an explanation of the symbols that will be used throughout this document to describe the operation of the various firmware commands.

'<, >': The bracket symbols mean that the information inclosed within them is manditory.

'[,]': The square bracket symbols mean that the information inclosed within them is optional.

'|' : The vertical bar symbol is used to indicate an alternative or "OR" condition. For example, A|B can be thought of as "Either A or B".

'::=': This symbol is used to indicate a definition or equivelence.

'{ }': Curly brackets are used to denote commemnts.

'+' : The plus sign is used as an addition symbol or logical or'ing.

'\$' : The dolar sign is used to indicate that a value is radix 16 {in other words, the number is in hexadecimal}. Values that are not preceded by '\$' are assumed to be decimal.

'NULL': This key word indicates the empty set, or in some cases the fact that the function whose value is NULL can be ignored. An example is:

Argle_Bargle ::= <NULL>

Essentially you can forget that Argle_Bargle exists for this context.

Command Types:

Widget commands are broken up into 3 categories:

1. ProFile commands

These commands are used emulate a ProFile mass storage device and provide for downward compatibility.

2. Diagnostic commands

These commands are used to separate the various subfunctions of the drive and provide a means to troubleshoot a Widget without the controller of performing any retrying of it's own.

3. System commands

These commands are used to operate a Widget at it's maximum efficiency. Blocks are transfered logically in a multiple block fashion, up to 255 blocks.

ProFile Commands:

Widget is designed to be backwards compatible with the current ProFile Driver, and to that end there exists the three ProFile System commands {Read, Write, and Write_Verify} within the firmware.

<u>Opcode</u>	<u>Definition</u>
\$00	Read Logical Block
\$01	Write Logical Block
\$02	Write_Verify Logical Block

The three ProFile commands behave in exactly the same fashion as do the corresponding instructions on ProFile, with one small exception: the Read Logical command does not include information concerning Retry Count or Sparring Threshold {however, because of a side effect in the way that the Host/Controller interface was designed, the Host may write as many command bytes to the controller as it chooses. The Controller will only decode the first four.}. The form of each command is:

<\$00|\$01|\$02> <3 bytes of Logical Block Address>

There are two 'special' logical address defined in the ProFile protocol, namely \$FFFFFF {-1} and \$FFFFFFE {-2}. Logical address (-1) returns as it's value Device_ID {as explained under the section titles Diagnostic Commands} and logical address (-2) returns as it' s value Widget's spare table structure in it's raw form.

It should be noted that if *at any time* Widget can not pass it's self test that it will refuse to communicate via logical commands {both ProFile and System type commands}; Widget will respond to Diagnostic commands at all times, however.

The rest of the commands available on Widget are a complete departure from the way that ProFile was implemented. The new form of any command is:

(<Command_Byte>
 <Instruction_Byte>

[Instruction_Parameter]

<CheckByte>)

Command_Byte ::= <CommandType_Nibble + CommandLength_Nibble>

CommandType_Nibble ::= <Diagnostic_Command|System_Command>

Diagnostic_Command ::= <\$10>

System_Command ::= <\$20>

CommandLength_Nibble ::= <Count of all the bytes in the command string *NOT* including the first one. For example, the command string to read Device_ID is: (<\$12> <\$00> <\$ED>). The commandlength_nibble in this case is 2.>

System_Command ::= <Sys_Read|Sys_Write|Sys_WrVer>

Diagnostic_Command ::= (<Read_ID|
Read_Controller_Status|
Read_Servo_Status|
Send_Servo_Command|
Send_Seek|
Send_Restore|
Set_Recovery|
Soft_Reset|
Send_Park|
Diag_Read|
Diag_ReadHeader|
Diag_Write|
Auto_Offset|
Read_SpareTable|
Write_SpareTable|
Format_Track|
Read_Abort_Stat|
Reset_Servo|
Read_Track|
Write_Track>)

Instruction_Parameter ::= { This value is instruction dependent, and will be formally defined at the same time as the individual instructions }

CheckByte ::= { This byte is the ones-complement of the sum, in MOD-256 arithmetic, of all the bytes in the instruction string *including* the Command_Byte. }

Diagnostic Commands:

Widget's personality, or manner in which it behaves in a specific Host environment, can be thought of as having two distinct parts: 1) that portion that is dictated by the hardware and 2) that portion that is controlled by the firmware. As trite as that last statement may seem, the fact remains that the part of Widget that is the hardware is not easily molded to adapt to different conditions. The same is true, but not quite in the same manner, for the firmware: the code is locked in a ROM of some sort and costs a lot to change. How then can Widget's "personality" be changed {on-the-fly} to "adapt" to a new environment? The answer in this case was to architect the firmware in a layered fashion: build the intelligence required to operate Widget in its normal system mode from a pool of discrete, primitive functions; these primitive functions having just one specific task that they are capable of completing. The implication of this architecture is that with very little effort these same primitive functions are available to the Host system.

Read_ID

Read_ID ::= <\$00>

Instruction_Parameter ::= <NULL>

This diagnostic command requires Widget to deliver to the host some device specific information. The structural layout of the data returned is:

STRUCTURE Identity_Block

This identity block is defined by the data structures contained within it; you will note, however, that a comment is given explaining the type of structure for a given element and range of bytes - if the structure is thought of as a linear array of bytes - that include the structure. An example is NameString. It is a 13-character ascii string, and is located in bytes \$0:C.

NameString ::= <Lisa/Nisha2 {13 bytes/\$0:C; Ascii String}>

Device_Type ::= <\$000110 {3 bytes/\$D:F}>

Firmware_Revision ::= <{2 bytes/\$10:11}>

Capacity ::= <\$9836 {3 bytes/\$12:14}>

Bytes_Per_Block ::= <532 {2 bytes/\$15:16}>

Number_Of_Cylinders ::= <610 {2 bytes/\$17:18}>

Number_Of_Heads ::= <2 {1 byte/\$19}>

Number_Of_Sectors ::= <32 {1 byte/\$1A}>

Number_Of_Possible_SpareBlocks ::= <\$00004C {3 bytes/\$1B:1D}>

Number_Of_SpareBlocks ::= <{3 bytes/\$1E:20, range 0..\$4B}>

Number_Of_BadBlocks ::= <{3 bytes/\$21:23, range 0..\$4B}>

Read_Controller_Status

Read_Controller_Status ::= <\$01>

Every time an operation completes {normally or abnormally} Widget will return Standard_Status. This allows the Host system to change it's flow of execution based on the state of the value returned in the Status. Normally, Standard_Status is all that is necessary to ensure continuous operation. In the exceptional case, or when the Host system is emulating the controller's functions, additional information concerning the state of Widget is mandatory: without it the Host simply could not make an optimum choice in deciding a course of action.

Controller_Status is then a means for the Host system to interrogate Widget further. Each Status {with the exception of Abort_Status, which is a separate command and is discussed later in this document} belongs to a homogeneous data structure: namely a four byte quantity containing a bit map representing the various exceptional conditions that are available as the first four bytes read from the controller upon completion of the current command.

There are eight status' available to the Host system. The Host requests a specific status by setting the Instruction_Parameter to the value corresponding to the status needed.

```
IF (Instruction_Byte = Read_Controller_Status)
  THEN Instruction_Parameter ::= (<Standard_Status|
                                Last_Logical_Block|
                                Current_Seek_Address|
                                Current_Cylinder|
                                Internal_Status|
                                State_Registers|
                                Exception_Registers|
                                Last-Seek_Address>)
```

The four byte response to each of the above status requests is of the form:

Status_Response ::= (<Byte0> <Byte1> <Byte2> <Byte3>)

Standard_Status ::= <\$00>

**Byte0 ::= < Bit7: Other than \$55 response from Host
Bit6: Write Buffer OverFlow
Bit5: {not used}
Bit4: {not used}
Bit3: Read Error
Bit2: No Matching Header Found
Bit1: Servo Error
Bit0: Operation Failed >**

**Byte1 ::= < Bit7: {not used}
Bit6: Spare Table OverFlow
Bit5: 5 or Less Spare Blocks Available
Bit4: {not used}
Bit3: Controller SelfTest Failure
Bit2: Spare Table has been Updated
Bit1: Seek Error
Bit0: Controller Aborted Last Operation >**

**Byte2 ::= < Bit7: First Status Response since Power-On
Bit6: Logical Block Number Out of Range
Bit5: 0 : {not used}>**

**Byte3 ::= < Bit7: Read Error Detected by Ecc circuitry
Bit6: Read Error Detected by Crc circuitry
Bit5: Header timeout
Bit4: {not used}
Bit3: 0 : Number of unsuccessful retries {out of 10}>**

Last_Logical_Block ::= <\$01>

Byte0 ::= {not used}

Byte1 ::= <Most Significant Block Address>

Byte2 ::= <Next Most Significant Block Address>

Byte3 ::= <Least Significant Block Address>

Current_Seek_Address ::= <\$02>

Byte0 ::= <Most Significant Cylinder Address>

Byte1 ::= <Least Significant Cylinder Address>

Byte2 ::= <Head Address>

Byte3 ::= <Sector Address>

Current_Cylinder ::= <\$03>

Byte0 ::= <Most Signifioant Cylinder Address>

Byte1 ::= <Least Significant Cylinder Address>

Byte2 ::= <Head Address>

Byte3 ::= <Sector Address>

Internal Status ::= <\$04>

Byte0 ::= <Bit7: Recovery On
Bit6: Spare Table Almost Full
Bit5: Buffer Structure is Contaminated
Bit4: Power reset has just occurred
Bit3: Current Standard Status is non-zero
Bit2: 1 : {not used}
Bit0: Controller LED is on>

Byte1 ::= <Bit7: On_Track
Bit6: Read Headers after data recal
Bit5: Current operation is a write operation
Bit4: Heads are parked
Bit3: Sequential look-ahead table search
Bit2: {not used}
Bit1: Seek_Complete
Bit0: Auto_Offset is ON>

Byte2 ::= {this status is valid ONLY after a ProFile or System Command}
<Bit7: Seek_Needed
Bit6: Head_Change_Needed
Bit5: 2 {not used}
Bit1: Current block is a BAD block
Bit0: Current block is a SPARE block>

Byte3 ::= <SpareTable_Type|UserData_Type>
SpareTable_Type ::= <\$08>
UserData_Type ::= <\$02>

State_Registers ::= <\$05>

Byte0 ::= {not used}

Byte1 ::= <Bit7: Ram_Failure
Bit6: Eprom_Failure
Bit5: Disk_Speed_Failure
Bit4: Servo_Failure
Bit3: Sector_Count_Failure
Bit2: State_Machine_Failure
Bit1: Read_Write_Failure
Bit0: No_SpareTable_Found>

Byte2 ::= <Bit7: Disk Read/-Write
Bit6: SioRdy
Bit5: Msel1
Bit4: Msel0
Bit3: Bsy
Bit2: Cmd
Bit1: EccError {active low}
Bit0: Start {active low}>

Byte3 ::= <Bit7: CrcError {active low}
Bit6: Write_Not_Valid {active low}
Bit5: ServoReady
Bit4: ServoError
Bit3: 0 : Current state of the state-machine>

Exception_Registers ::= <\$06>

Byte0 ::= <Bit7: Read error
Bit6: Servo error while reading
Bit5: At least one successful read in last retry sequence
Bit4: Header Timeout
Bit3: CrcError or EccError
Bit2:0 : {not used}>

Byte1 ::= <Bit7 ::= EccError
Bit6 ::= CrcError
Bit5 ::= Header Timeout
Bit4 ::= {not used}
Bit3:0 : {number of bad retries out of 10}>

Byte2 ::= <Bit7: Write Error
Bit6: Servo Error while writing
Bit5: At least one successful write in last retry sequence
Bit4: Header Timeout
Bit3:0 : {not used}>

Byte3 ::= {number of bad retries out of 10}

Read_Servo_Status

Read_Servo_Status ::= <\$02>

Instruction_Parameter ::= <0..8>

This status command is used to interrogate the Servo Processor in much the same way that Read_Controller_Status is used. In fact, the form of the result is the same four byte-mapped quantity.

This command is of the particular value to a diagnostician that is interested in 'scoping-out' the servo subsystem.

A more complete description of the servo commands can be read in the document titled "Widget Servo Functional Objective" written by Jim Reed.

Send_Servo_Command

Send_Servo_Command ::= <\$03>

Instruction_Parameter ::= (<Byte0> <Byte1> <Byte2> <Byte3>)

Normally, the Host will allow the controller to manipulate the servo processor in order to perform useful work. For example, let's suppose that the Host system wishes to move drive's heads from one track to another. Under normal operating conditions the preferred way to perform this task is to use the Send_Seek command {explained later}. However, the Host has the capability to bypass the controller and direct the servo processor. Indeed, the Host can issue the servo command to position the heads so that the seek is completely transparent to the controller. The implication of this command is that the Host can gain even more control of the system if it so chooses.

A more complete description of the servo commands can be read in the document titled "Widget Servo Functional Objective" written by Jim Reed.

Byte0 ::= <S_Command + S_Direction + Hi_Magnitude>

S_Command ::= <Offset|
Diagnostic|
DataRecal|
BrakeRelease|
Access|
Access_Offset|
Home>

Offset ::= <\$10>
Diagnostic ::= <\$20>
DataRecal ::= <\$40>
BrakeRelease ::= <\$70>
Access ::= <\$80>
Access_Offset ::= <\$90>
Home ::= <\$C0>

S_Direction ::= <Positive|Negative>

Positive ::= <\$08 {towards inside diameter}>
Negative ::= <\$00 {towards outside diameter}>

Hi_Magnitude ::= <0..3 {move heads in multiples of 256}>

Byte1 ::= <Low_Magnitude ::= 0..255>

{note: Hi_magnitude, Low_magnitude, and S_Direction establish the *relative* distance the heads must move to arrive at the target track}

Byte2 ::= <Offset_Direction + Auto_Offset_Switch + Offset_Magnitude>

Offset_Direction ::= <Positive|Negative>

Positive ::= <\$80 {towards outside diameter}>

Negative ::= <\$00 {towards inside diameter}>

Auto_Offset_Switch ::= <ON|OFF>

ON ::= <\$40 {assert fine positioning}>

OFF ::= <\$00>

Offset_Magnitude ::= <0..32>

Byte3 ::= <StatusRquest>

Send_Seek

Send_Seek ::= <\$04>

Instruction_Parameter ::= <(HiCyl) <(LoCyl) <(Head) <(Sector) <(AutoOffsetFlag)>

HiCyl, LoCyl, Head, Sector ::= <\$00..\$FF>

AutoOffsetFlag := <(ON|OFF)>

ON ::= <\$01>

OFF ::= <\$00>

Widget's Send_Seek command allows the Host system to place the heads over any track on the disk. The value of the seek address is sent as the Instruction_Parameter, and each parameter is a byte in length. For example, for the Host to seek to (Cylinder 1, Head 0, Sector 18) without AutoOffset a seek command would be issued with the following Instruction_Parameter: (\$0000, \$00, \$12, \$00).

Send_Restore

Send_Restore ::= <\$05>

Instruction_Parameter ::= <DataRecal|BrakeRelease>

DataRecal ::= <\$40>

BrakeRelease ::= <\$70>

The **Send_Restore** command is used by the Host to initialize the servo processor and to put the heads in a known location. This command is the same as performing a Data/Format Recal except that the controller updates it's internal state to account for the new servo position.

Set_Recovery

Set_Recovery ::= <\$06>

Instruction_Parameter ::= <ON|OFF>

ON ::= <\$01>

OFF ::= <\$00>

The exception handling characteristics of Widget approximate a binary set: either Widget handles everything, or the Host system does. The command 'Set_Recovery' is the Host's link with this protocol in that it is through this instruction that the Host can gain control of the media. When Widget comes up after being reset, it assumes control and sets *Recovery* to be ON. The Host system must overtly change this state if it wishes to emulate a different exception handling criteria. Once Recovery is OFF, the controller will always fail in an operation if an exception occurs: the Host *must* assume responsibility for ALL error handling.

Soft_Reset

Soft_Reset ::= <\$07>

Instruction_Parameter ::= <NULL>

This command instructs the Widget firmware to restart its flow of execution at its initialization point. The results should be the same as a power reset.

Send_Park

Send_Park ::= <\$08>

Instruction_Paramter ::= <NULL>

When the Host issues a Send_Park command to the controller the results are that the heads are moved off the data surface and held very near the inside diameter crash stop. The difference between this command and the Send_Servo_Command: Home, is that Home is performed 'open-loop' with the crash stop as its reference point, while Send_Park is an access command to a specific track. The net result is a fairly hefty savings of time.

Diag_Read

Diag_Read ::= <\$09>

Instruction_Parameter ::= (<Sector><SeqValue>)

Sector ::= (<0..31>)

SeqValue ::= (<NewSector|IncSector><Long>)

NewSector ::= \$80 {selects 'Sector' as the sector to be operated on}

IncSector ::= \$40 {increments the last sector value}

Long ::= \$20 {if Long then the ECC syndrome will be ignored and the checkbytes will be included at the end of the data}

The Diag_Read command is used to read the block on the disk pointed to by the last seek address. The form of the returned data is exactly the same as that of ProFile_Read or Sys_Read in that 4 bytes of Standard_Status precede the block of data.

Diag_ReadHeader

Diag_ReadHeader ::= <\$0A>

Instruction_Parameter ::= (<Sector><SeqValue>)

Sector ::= (<0..31>)

SeqValue ::= (<NewSector|IncSector><Long>)

NewSector ::= \$80 {selects 'Sector' as the sector to be operated on}

IncSector ::= \$40 {increments the last sector value}

Long ::= \$20 {if Long then the ECC syndrome will be ignored and the checkbytes will be included at the end of the data}

When the heads are positioned over an unknown location, or when it is suspected that a block's header is shot, it is time to use the Diag_ReadHeader command. This instruction allows the host to 'suck-up' both whatever information is residing in the block's header field as well as the data from the block. The form of the result is:

Result ::= (<Header {bytes/\$00:05}>
 <Gap {bytes/\$06:0C}>
 <Data {bytes/\$0D:21F}>)

Header ::= (<HiCyl> <LowCyl> <HdSct> <-HiCyl> <-LowCyl> <-HdSct>)

HiCyl ::= <Most significant byte of cylinder address>

LowCyl ::= <Least significant byte of cylinder address>

HdSct ::= <Bit7:6 : Head address
 Bit5:0 : Sector address>

-HiCyl ::= <ones-complement of HiCyl>

-LowCyl ::= <ones-complement of LowCyl>

-HdSct ::= <ones-complement of HdSct>

Gap ::= <\$00>

Diag_Write

Diag_Write ::= <\$0B>

Instruction_Parameter ::= <NULL>

Sector ::= (<0..31>)

SeqValue ::= (<NewSector|IncSector><Long>)

NewSector ::= \$80 {selects 'Sector' as the sector to be operated on}

IncSector ::= \$40 {increments the last sector value}

Long ::= \$20 {if Long then the ECC checkbytes are to be supplied at the end of the write data}

This instruction allows the Host to write a block of data to the location on the disk pointed to by the last seek address. Diag_Write is valid for all states that the controller may wind up in, but is recommended that a Send_Seek command precede the write command to ensure that the correct block will be written.

Auto_Offset

Auto_Offset ::= <\$OC>

Instruction_Parameter ::= <NULL>

This command is used by the Host to fine-position the heads after they are on-track. The `auto_offset` function can also be implemented by using the `Send_Servo_Command` instruction; the difference is that the controller will update some internal information {remember, servo commands are transparent} as well as select the correct head to offset off of {the Widget system uses head 1 only for fine positioning}.

Read_SpareTable

Read_SpareTable ::= <\$0D>

Instruction_Parameter ::= <NULL>

Reading {and writing} the Widget's sparetable is an absolute must for diagnostic purposes, and if the Host wishes to emulate the controller. The result of this instruction is identical to performing a ProFile_Read from block -1 {<FFFFFFE>} and has the form:

```
Result ::= ( <Fence {bytes/$00: 03}>
             <RunNumber {bytes/$04: 07}>
             <Format_Offset {byte/$08}>
             <Format_InterLeave {byte/$09}>
             <HeadPtr_Array {bytes/$0A: 49}>
             <SpareCount {byte/$4A}>
             <BadBlockCount {byte/$4B}>
             <BitMap {bytes/$4C: 55}>
             <Heap {bytes/$56: 185}>
             <InterLeave_Map {bytes/$186: 1A5}>
             <Checksum {bytes/$1A6: 1A7}>
             <Fence {bytes/$1A8: 1AB}>
             <Zone_Table {bytes/$1AC: 1C1}>
             <Fence {bytes/$200: 203}> )
```

Fence ::= (<\$F0> <\$78> <\$3C> <\$1E>)

RunNumber ::= <32-bit integer>

This integer is incremented once each time the spare table is written to the disk. Because two copies are kept on the the disk, the RunNumber is used to indicate which is the more recent of the two, should both copies not be updated.

Format_Offset ::= <0..NumberOfSectors>

Format_Offset is the number of physical sectors there are from index mark until logical sector 0.

Format_InterLeave ::= <0..6>

This number is the interleave factor for this disk and is used in calculating where each of the logical sectors are relative to actual sector locations.

HeadPtr_Array ::= <ARRAY[0..63] of HeadPtr>

HeadPtr ::= <Nil+Ptr>

Nil ::= <\$80 {if Nil the end-of-chain}>

```
Ptr ::= <$00..$7F {address of next element}>
```

A Ptr is a 7-bit structure that 'points' to a specific location within the Heap. To arrive at the actual index value within the Heap, the Ptr must first be multiplied by 4 {the length of each element}.

When a disk is formatted and being written to for the first time, each logical block is assigned the first available physical block on the disk. Therefore you would expect that LogicalBlock(0) would occupy PhysicalBlock(0), L(1) --> P(1), etc. There are instances, however, when a block of data must be relocated to another space on the disk that does not follow the original progression (for example, the original space was defective). In order to 'find' these relocated blocks in the future a record must be kept as to where all these relocated blocks have been put. This record takes the form of 128 linked lists having the form:

```
HeadPtr[n] --> LinkedList[n], where n ::= [0..127]
```

The algorithm for deciding whether or not a logical block has been relocated is to extract bits 10:16 from the LogicalBlockNumber and use it as an index into the HeadPtrArray:

```
IF (HeadPtr[LogicalBlockNumber/bits 10:16].Nil)
  THEN LogicalBlock has not been relocated
  ELSE use HeadPtr[].Ptr to begin searching the chain for a matching
       element {refer to the structure of ListElement for more detail}
  IF no matching ListElement
    THEN LogicalBlock has not been relocated
    ELSE the element position in the Heap corresponds to the new
         physical block location
```

```
SpareCount ::= <$00..$4B>
```

```
BadBlockCount ::= <$00..$4B>
```

```
BitMap ::= <ARRAY[$00..$4B] of Bits>
```

The bit map is used to keep a record of which spare blocks are occupied.

```
Heap ::= <ARRAY[$00..$4B] of ListElement>
```

```
ListElement ::= (<Nil+Used+Useable+Spr_Type+Data_Type>
                 <Token>
                 <Ptr>)
```

```
Used ::= <$40>
```

```
Useable ::= <$20>
```

```
Spr_Type ::= <Spare|BadBlock>
```

```
  Spare ::= <$10>
```

```
  BadBlock ::= <$00>
```

```
Data_Type ::= <Data|SpareTable>
```

```
  Data ::= <$02>
```

SpareTable ::= <\$08>

Token ::= <Bits 0:9 of LogicalBlock>

InterLeave_Map ::= <ARRAY[0..31] of [0..31]>

The InterLeave_Map is used to logical re-interleave the drive so that Widget can be run optimally on any system without having different manufacturing or formatting processes.

Check_Sum ::= <sum of all bytes in the spare table from the first fence to beginning of this structure, in MOD-65536 arithmetic>

Zone_Table ::= <ARRAY[0..NumberOfZones] of Zone_Element>

Zone_Element ::= <Offset_Direction+Offset_Magnitude>

Write_SpareTable

Write_SpareTable ::= <\$OE>

Instruction_Parameter ::= (<\$F0> <\$7B> <\$3C> <\$1E>)

This command allows the Host to 'force' a new spare table on the controller, and is executed just like any of the other write commands (data, in this case, **MUST** conform to the structure presented in Read_SpareTable}. The data sent to the controller is written to the two spare table locations on the disk.

Format_Track

Format_Track ::= <\$0F>

Instruction_Parameter ::= (<PassWord>)

PassWord ::= (<\$F0> <\$78> <\$3C> <\$1E>)

The format command is used to:

1. Operate on the track that is currently beneath the heads - this implies that the Host had best perform a Send_Seek and Auto_Offset command prior to formatting a track.
2. New headers will be layed down in every sector of the track.

Read_Abort_Status

Read_Abort_Status ::= <\$11>

Instruction_Parameter ::= <NULL>

Read_Abort_Status will return valid data only AFTER the controller has aborted (identified by Standard_Status.Byte1.Bit0). The form of the result is a 16 byte string, and its contents are the contents of the controller's registers at the time of the abort - with the exception of byte \$0F, which is the value of the Abort taken.

Reset_Servo

Reset_Servo ::= <\$12>

Instruction_Parameter ::= <NULL>

Reset_Servo allows the Host to initialize the servo processor without having to power the device down. The controller will automatically reset the Servo, set the baud rate at 57.6K, and check for valid initial conditions.

Read_Track

Read_Track ::= <\$13>

Instruction_Parameter ::= (<Sector><SeqValue>)

Sector ::= (<0..31>)

SeqValue ::= (<NewSector|IncSector><Long>)

NewSector ::= \$80 {selects 'Sector' as the sector to be operated on}

IncSector ::= \$40 {increments the last sector value}

Long ::= \$20 {if Long then the ECC syndrome will be ignored and the
checkbytes will be included at the end of the data}

Write_Track

Write_Track ::= <\$13>

Instruction_Parameter ::= (<Sector><SeqValue>)

Sector ::= (<0..31>)

SeqValue ::= (<NewSector|IncSector><Long>)

NewSector ::= \$80 {selects 'Sector' as the sector to be operated on}

IncSector ::= \$40 {increments the last sector value}

Long ::= \$20 {if Long then the ECC checkbytes are to be supplied at the end of the write data}

This diagnostic command is used mainly to facilitate the Nisha FST program. The entire track (as defined by the last Seek address, and beginning with Sector or Last_Sector + 1 if NewSector or IncSector is set) is written. Data, however is sent for only the first sector written (implying that the whole track will be written with the same data pattern). This diagnostic command is used mainly to facilitate the Nisha FST program.

System Commands:

System commands have been implemented for essentially two reasons:

1. It was important for Widget to add one more check on the CMD/BSY handshake: namely the addition of a checkbyte following the command string.
2. In order to increase the performance of the system without modifying the hardware it was critical to introduce another level of parallelism into the Host/Controller interface. Most of the reads for a specific block on the disk are followed by a read for the next logically sequential block. Therefore the command decoding and checkbyte comparison for all but the first block has been suppressed into a multiblock-type command. The implementation for this added parallelism is to send an extra parameter with the (first) LogicalBlock indicating the number of blocks to be read sequentially.

Sys_Read

Instruction_Parameter ::= (<BlockCount> <LogicalBlock>)

BlockCount ::= <\$01..\$FF>

This parameter is the number of blocks to be read that follow sequentially from LogicalBlock. It is assumed that one block (LogicalBlock) will be read.

LogicalBlock ::= <\$000000..009835>

Sys_Write

Instruction_Parameter ::= (<BlockCount> <LogicalBlock>)

BlockCount ::= <\$01..\$FF>

This parameter is the number of blocks to be read that follow sequentially from LogicalBlock. It is assumed that one block (LogicalBlock) will be read.

LogicalBlock ::= <\$000000..009835>

Command Summary

Profile_Commands:

Profile_Read ::= (<\$00> <3 bytes LogicalBlock>)
Profile_Write ::= (<\$01> <3 bytes LogicalBlock>)

Diagnostic_Commands:

Read_Id ::= (<\$12> <\$00> <\$ED>)
Read_Controller ::= (<\$13> <\$01> <StatusRequest> <CheckByte>)
Read_Servo_Status ::= (<\$13> <\$02> <StatusRequest> <CheckByte>)
Send_Servo_Command ::= (<\$16> <\$03> <CommandRequest> <CheckByte>)
Send_Seek ::= (<\$17> <\$04> <SeekAddress> <AutoOffset Flag> <CheckByte>)
Send_Restore ::= (<\$13> <\$05> <RecalType> <CheckByte>)
Set_Recovery ::= (<\$13> <\$06> <On/Off> <CheckByte>)
Soft_Reset ::= (<\$12> <\$07> <\$E6>)
Send_Park ::= (<\$12> <\$08> <\$E5>)
Diag_Read ::= (<\$14> <\$09> <Sector> <SeqValue> <CheckByte>)
Diag_ReadHeader ::= (<\$14> <\$0A> <Sector> <SeqValue> <CheckByte>)
Diag_Write ::= (<\$14> <\$0B> <Sector> <SeqValue> <CheckByte>)
Auto_Offset ::= (<\$12> <\$0C> <\$E1>)
Read_SpareTable ::= (<\$12> <\$0D> <\$E0>)
Write_SpareTable ::= (<\$16> <\$0E> <PassWord> <CheckByte>)
Format_Track ::= (<\$16> <\$0F> <PassWord> <CheckByte>)
Read_Abort_Status ::= (<\$12> <\$11> <\$DC>)
Reset_Servo ::= (<\$12> <\$12> <\$DB>)
Read_Track ::= (<\$14> <\$13> <Sector> <SeqValue> <CheckByte>)
Write_Track ::= (<\$14> <\$14> <Sector> <SeqValue> <CheckByte>)

System_Commands:

Sys_Read ::= (<\$26> <\$00> <BlockCount> <LogicalBlock> <CheckByte>)
Sys_Write ::= (<\$26> <\$01> <BlockCount> <LogicalBlock> <CheckByte>)
Sys_WrVer ::= (<\$25> <\$02> <LogicalBlock> <CheckByte>)

PassWord ::= (<\$F0> <\$78> <\$3C> <\$1E>)

Abort_Status_Variables

There are occasions when the Nisha Controller will detect that something is radically wrong with the Nisha SubSystem, i. e., the ram on board the controller goes on vacation, or the positioning system gives up the ghost, etc. In one of these cases the controller will abort its current instruction and return control to the Host, hopefully with enough information that the Host can make an intelligent decision concerning the state of Nisha.

The Host can read some information concerning the abort that the controller took by requesting Read_Abort_Status. This command returns a result that is 20 bytes long: 4 bytes of standard status and 16 bytes of abort status. The contents of the abort status are dependent upon the actual abort taken, and is determined by examining the contents of byte 16: the value of the abort taken.

- \$01: Illegal interface response, or Host Nak
Byte/\$09: Response byte that caused abort
- \$02: Illegal Ram Bank select
Byte/\$00: Bank number
- \$03: Format Error: illegal state-machine state
Byte/\$0A: state of state-machine at time of abort
- \$04: Illegal Rom Bank Select
Byte/\$00: Bank number
- \$05: Illegal interrupt or DeadMan_Timeout
Bytes/\$0A: 0B: Address of routine at time of timeout
- \$06: Format Error: Error while writing sector
Byte/\$09: Error status from FormatBlock
- \$08: Command Checkbyte Error
- \$09: ProFile or System command attempted while SelfTest Error
- \$0A: Illegal Command
- \$0B: Unrecoverable Servo Error while reading
- \$0C: Sparring attempted on non-existent spare block
- \$0D: Sparring attempted while sparetable full
- \$0E: Deletion attempted of non-existent bad block
- \$0F: Illegal exception instruction
- \$10: Write buffer overflow
- \$11: Unrecoverable servo error while writing
- \$12: Servo status request sent as Servo command
- \$13: Restore Error: Non-Recal parameter
Byte/\$00: Value of illegal parameter sent
- \$14: Illegal password sent to Write_SpareTable_Command
- \$15: Illegal password sent to Format command
- \$16: Illegal format parameters
Bytes/\$09: 0A: illegal parameters
- \$17: Illegal password sent to Init_SpareTable_Command
- \$18: Zero block count sent to System_Command
- \$19: Write Error: Illegal state-machine state
Byte/\$0A: State-machine state at time of abort
- \$1A: Read Error: illegal state-machine state
Byte/\$0A: State-machine state at time of abort

\$1B: ReadHeader Error: illegal state-machine state
Byte/\$0A: State-machine state at time of abort
\$1C: Request for illegal logical block
Bytes/\$00: 02: logical block number
\$1D: External Stack overflow
Bytes/\$04: 07: stack history
\$1E: Search for SpareTable failed
\$1F: No sparetable structure found in sparetable
\$20: Update of sparetable failed
\$21: Illegal sparecount instruction
Bytes/\$09: value of illegal instruction
\$22: Unrecoverable servo error while seeking
\$23: Unable to transmit command to servo
\$24: Unable to receive status from servo
\$25: Unable to find any headers after DataRecal
\$26: Servo error after servo reset
Byte/\$0A: value of controller status port
\$27: Servo communication error after servo reset
\$28: Scan attempted without sparetable
\$29: Illegal Bank Call
\$2A: Illegal Bank Return
\$2B: Illegal Sector value detected in LocateSector
\$2C: Illefal Sector value detected while remapping
\$2D: Control/Status register in GA is non-functional
\$2E: Read gate not active
\$2F: Read always active
\$30: Statemachine single step error
\$31: Data compare mismatch in r/w selftest
\$32: SpareTable Ptr is addressing out-of-bounds
\$33: New Spare not found by OverLap

REV.	ZONE	ECO #	REVISION	APPD	DATE
A		P855	INITIAL RELEASE (was SK-L170-00)	<i>S. PATEL</i>	4/85 4/85

NISHA DRIVE SPECIFICATION

PRODUCTION RELEASE
 This revision supersedes all previous versions.
 Please destroy all old copies.
NOTICE OF PROPRIETARY PROPERTY
 The information contained herein is the
 proprietary property of Apple Computer, Inc.
 The disclosure herein is limited to the following:
 (1) To manufacturers and designers in accordance
 with the license agreement of Apple II.
 (2) Not to reproduce or distribute in whole or part.

TOLERANCES
 UNLESS OTHERWISE SPECIFIED
 DIMENSIONS ARE IN INCHES.

DECIMALS .X ± _____
 .XX ± _____
 .XXX ± _____

ANGLES XX.X ± _____

FRACTIONS ± _____

DIMENSIONS IN PARENTHESIS
 ARE IN MILLIMETERS.

DRAWN BY	DATE
HAAHR/R.J.B.	11/84
CHECKED BY	DATE
<i>[Signature]</i>	4/85
APPROVED BY	DATE
S. PATEL	4/85
RELEASED BY	DATE
<i>[Signature]</i>	4/85



TITLE
 SPECIFICATION, NISHA DRIVE

SIZE	DRAWING NUMBER
A	062-0287-A

NEXT ASSY.	FINISH:	SCALE: -----	SHEET 1 OF 30
-------------------	----------------	---------------------	----------------------

DRAWING NUMBER
062-0287-A
SH 1 OF 30

APPLICABLE DOCUMENTS

Engineering Specification:

The HDA components shall meet all requirements and specifications set forth in the following documents unless separately specified herein.

NISHA Assembly	678-5007
Head Disk Assembly	678-5006
Disk Specification	062-0236
Head Specification	062-0231
Spindle Motor Assembly	699-5034
P.C.B. Assembly, Motor Control	678-0102



apple computer inc.

SIZE
A

DRAWING NUMBER

062-0287-A

SCALE:

SHEET 2 OF 30

CONTENTS

1.0	SCOPE
2.0	CAPACITY
3.0	TRANSFER RATE
4.0	ACCESS TIME
5.0	ROTATIONAL SPEED
6.0	RECORDING DENSITY
7.0	RELIABILITY
8.0	ERROR RATE
9.0	POWER REQUIREMENT
10.0	ENVIRONMENTAL
11.0	MOUNTING
12.0	NISHA INTERFACE
13.0	SERVO
	13.1 COMMUNICATION FUNCTIONS
	13.2 SERVO PROTOCOL
	13.3 ERROR HANDLING

1.0 SCOPE

The NISHA Drive consists of the NISHA HDA Assembly and one Electronic Board Assembly. The scope of this specification is to define the NISHA Drive without controller.

2.0 CAPACITY

No. of Disks	1
Recording Surface Per Drive	2 20.7MB(Formatted)
No. of Cylinders	610
Total No. of Tracks	1220
No. of Sector/Track	32
Bytes/Sector	532(Formatted)
Total No. of Blocks (Data)	38,964
Spare Blocks	76

3.0 TRANSFER RATE 7.5MHZ

4.0 ACCESS TIME

Track to Track	10 ms
Average	50 ms
Maximum	150 ms
Average Latency	10.9 ms

5.0 ROTATIONAL SPEED 2749 RPM

6.0 RECORDING DENSITY

Bits Per Inch	19,065 (MAX)
Flux Changes Per Inch	12,710 (MAX)
Tracks Per Inch	600

7.0 RELIABILITY

MTBF (Typical Usage)	15,000 Hours
Component Life	5 Years
Media Life	10,000 Starts/Stops



SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 4 OF 30

8.0

ERROR RATE

Soft Read Errors (9 retries minimum)	1 per 10 ⁹ bits
Hard Read Errors	1 per 10 ¹² bits
Seek Errors	1 per 10 ⁴ seek

9.0

POWER REQUIREMENT

+5V DC 5%	620 ma
-12.5V DC 5%	180 ma
+12.5V DC 5%	470 ma (on Track)
	670 ma (Average Seek- R/W)
	1200 ma (Start for 5 Sec)
Standby Power	11.2 Watts
Typical Power	13.7 Watts

10.0

ENVIRONMENTAL

Ambient Temperature	
Operating	10°C to 60°C
Shipping	-40°C to 70°C
Storage	-40°C to 70°C
Maximum Temperature Gradient	
Operating	10°C per hour
Non-operating	Below that causing Condensation
Relative Humidity (Non-Condensing)	
Relative Humidity	20% to 80%
Stray Magnetic Field (1 inch from casting)	20 Gauss(Max)
Attitude	10,000 Feet
Shock and Vibration	
Shock (Operating)	0.5G, 11 msec, half sine
Shock (shipping)	50G, 11msec, half sine
Vibration (Operational)	0.5g peak; 20-200Hz
Vibration(Non-operational)	2g peak; 20-200Hz

11.0

MOUNTING AND PHYSICAL CONFIGURATION See Figure 1



apple computer inc.

SIZE
A

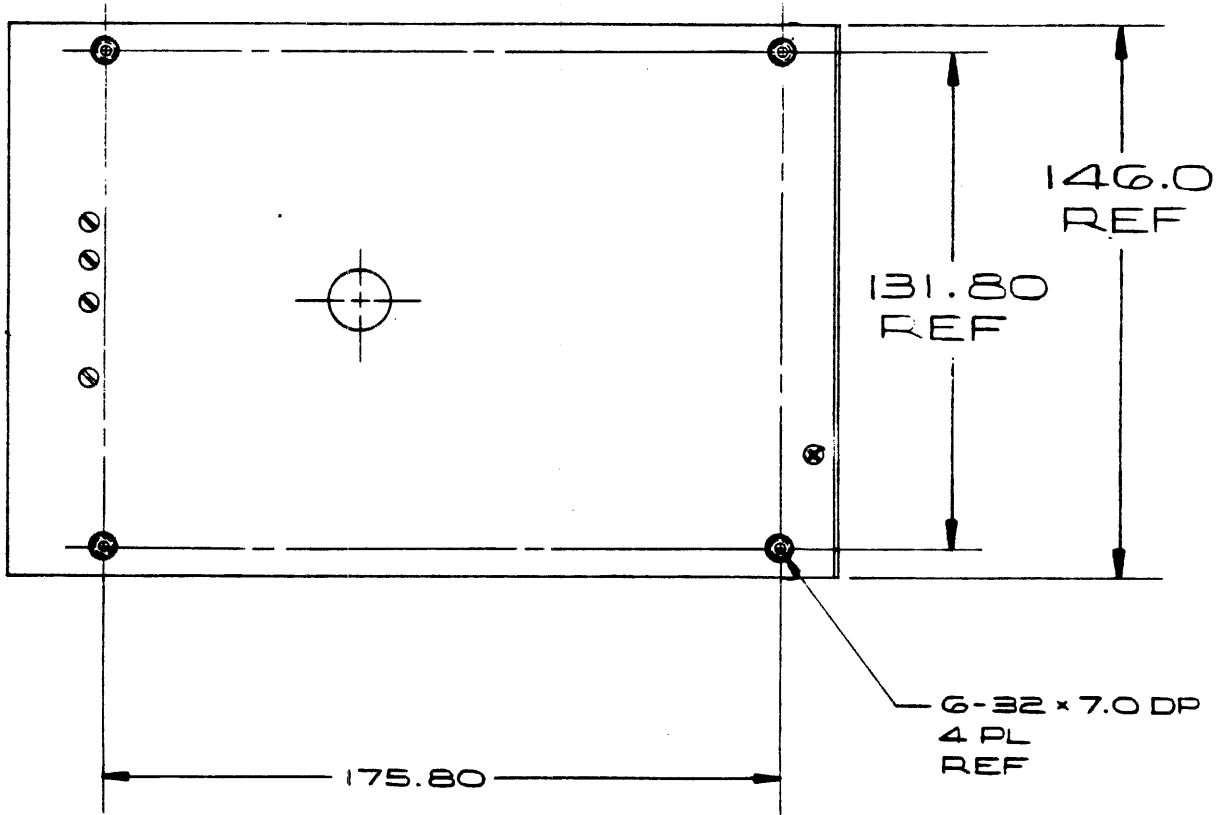
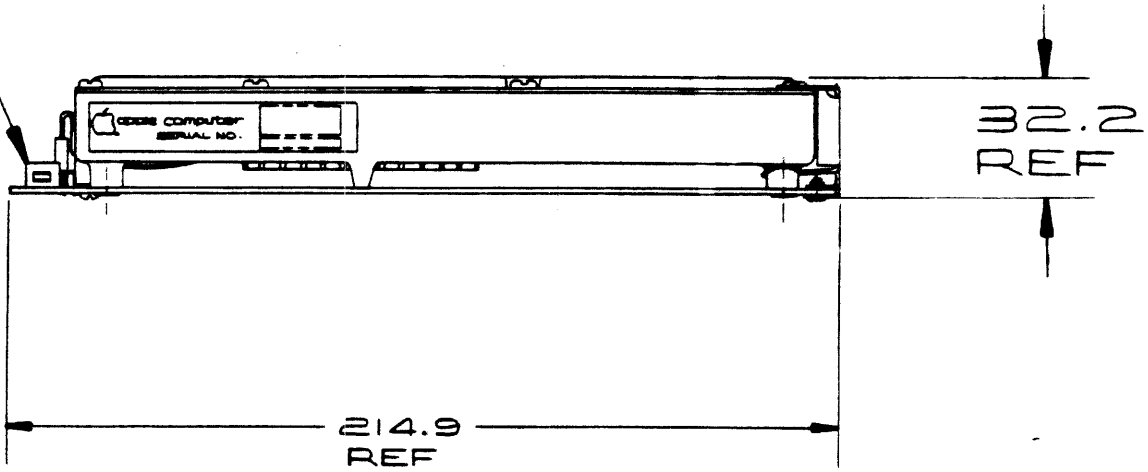
DRAWING NUMBER

062-0287-A

SCALE: _____

SHEET 5 OF 30

26 PIN RIGHT
ANGLE CONN.



 apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: —

SHEET 6 OF 30

12.0 NISHA INTERFACE

12.1 GENERAL DESCRIPTION:

PIN	SIGNAL NAME	I/O	FUNCTION
1	INDEX	0	THIS SIGNAL INDICATES THE BEGINNING OF SERVO SECTOR AND END OF DATA SECTOR 31. ONE PER REVOLUTION (2749RPM \pm 1.5%).
3	SECTOR	0	THIS SIGNAL INDICATES THE BEGINNING OF DATA SECTORS. TOTAL DATA SECTORS IS 32 STARTING FROM SECTOR 0 TO 31. EACH SECTOR HAS 620 \pm 16 BYTES OR 661 \pm 17 US.
5	RWCLK	0	READ/WRITE CLOCK IS THE SIGNAL TO SYNCHRONIZE THE DRIVE READ/WRITE DATA. CLOCK FREQUENCY IS 7.5 MHZ \pm 40%. WITH 50 \pm 10% DUTY CYCLE.
7	/RDATA	0	NRZ READ DATA FROM DRIVE IS SYNCHRONIZED TO THE RISING EDGE OF THE READ/WRITE CLOCK (RWCLK) READ DATA CAN CHANGE AT A MIN. OF 40NS AND AT A MAX. OF 90NS FROM THE RISING EDGE OF THE RWCLK.
9	WDATA	1	NRZ WRITE DATA FROM CONTROLLER IS SYNCHRONIZED TO THE RISING EDGE OF THE RWCLK. WRITE DATA CAN CHANGE AT A MIN. OF 40NS AND AT A MAX. OF 90NS FROM THE RISING EDGE OF THE RWCLK.



apple computer inc.

SIZE
A

DRAWING NUMBER

062-0287-A

SCALE: _____

SHEET 7 OF 30

11	/WTGT	1	WRITE GATE FROM CONTROLLER, WHICH INDICATES THAT WRITE DATA IS SHIFTING FROM CONTROLLER TO DRIVE.
12	HS0	1	HEAD SELECT 0 INDICATES THE SELECTION OF HEAD 0. HEAD SWITCHING RECOVERY TIME IS 1 US.
13	RDGT	1	READ GATE FROM CONTROLLER, WHICH MUST BE TURNED ON AT GLITCH FREE DATA SYNCHRONIZATION AREA. READ DATA WILL BE AVAILABLE AFTER VCO IS LOCKED ON THE DATA SYNCHRONIZATION BYTES(00). VCO LOCK ON TIME IS 12 BYTES MAXIMUM.
14	HS1	1	HEAD SELECT 1, A RESERVED PIN FOR FUTURE ADDITION OF READ WRITE HEADS.
15	SO	1	SERVO SERIAL DATA OUT IS AN OUTPUT SIGNAL THAT SHIFTS SERVO STATUS FROM DRIVE TO CONTROLLER AT A MAX. RATE OF 58.6K BITS/SEC.
16	/RWI	1	REDUCED WRITE CURRENT IS A SIGNAL THAT INDICATES TO DRIVE THAT LESS WRITE CURRENT IS REQUIRED AT THE INSIDE TRACKS WHICH HAVE MUCH HIGHER FLUX CHANGE PER INCH (FCI).
17	SI	1	SERVO SERIAL DATA IN IS AN INPUT SIGNAL THAT SHIFTS SERVO COMMANDS FROM CONTROLLER TO DRIVE AT A MAX. RATE OF 58.6K BITS/SEC.



SIZE
A

DRAWING NUMBER

062-0287-A

SCALE: _____

SHEET 8 OF 30

18	/RESET	1	SERVO RESET IS AN INPUT SIGNAL THAT RESETS THE SERVO CONTROL FUNCTIONS.
19	SIORDY	0	SERVO INPUT/OUTPUT READY IS AN INDICATION FROM DRIVE TO CONTROLLER THAT SERVO COMMUNICATION IS READY.
21	SERVORDY	0	SERVO READY IS AN OUTPUT SIGNAL THAT DRIVE INDICATES TO CONTROLLER THAT READ/WRITE HEAD IS ON TRACK AND IS READY FOR READ/WRITE DATA TRANSFER.
23	/WRITESAFE	0	WRITESAFE WHEN LOW INDICATES FROM DRIVE TO CONTROLLER THAT WRITING IS SAFE TO BE PROCEEDED. WRITESAFE WHEN HIGH INDICATES FROM DRIVE TO CONTROLLER THAT WRITING IS PROHIBITED OR SERVO COMMUNICATIONS HAS AN ERROR. WRITESAFE CAN GO HIGH DURING THE WRITING CYCLE AND WILL NOT BE LATCHED. IT IS THE RESPONSIBILITY OF THE CONTROLLER TO LATCH THE STATUS AND REWRITE THE DATA SECTOR.

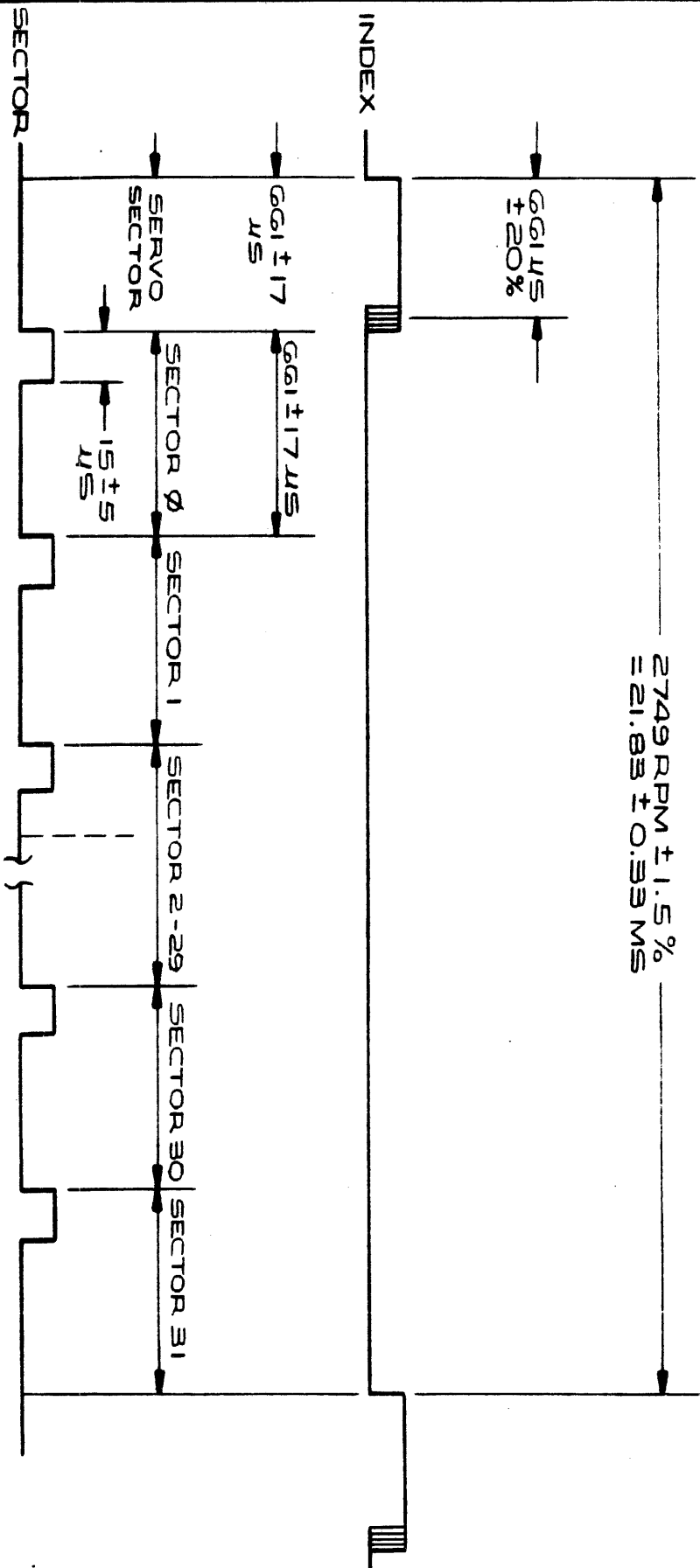
12.0 NISHA INTERFACE (con't)

PIN	SIGNAL NAME	I/O	FUNCTION
8	+5V	1	POWER SUPPLY INPUT
10	+5V	1	POWER SUPPLY INPUT
2	GND	1	GROUND
4	GND	1	GROUND
6	GND	1	GROUND
20	+12V	1	POWER SUPPLY INPUT
22	+12V	1	POWER SUPPLY INPUT
24	-12V	1	NEGATIVE POWER SUPPLY INPUT
25	MOTOR 12V	1	POWER SUPPLY TO MOTOR (ISOLATED FROM +12V IN NISHA)
26	MOTOR GND	1	MOTOR GROUND (ISOLATED FROM GND IN NISHA)

12.1 DC ELECTRICAL CHARACTERISTICS
 (TEMPERATURE=10 DEG TO 60 DEG CELSIUS)

PARAMETER	MIN	TYP	MAX	UNITS	COMMENTS

VIL INPUT					
LOW VOLTAGE	-0.3		0.6	V	10L MAX=-400UA
VHI INPUT					
HIGH VOLTAGE	2.0		VCC	V	10H MAX=+40UA
VOL OUTPUT LOW					
VOLTAGE			0.4	V	3 LSTTL=+1.2MA
VOL OUTPUT HIGH					
VOLTAGE	2.4			V	3 LSTTL=-120MA
5V POWER SUPPLY					
VOLTAGE	4.75	5.00	5.25	V	1A MAX
+12V POWER SUPPLY					
VOLTAGE	12.0		13.2	V	
-12V NEG POWER					
SUPPLY VOLTAGE	-12.0		-13.2	V	
MOTOR +12V MOTOR					
SUPPLY VOLTAGE	12.0		13.2	V	
CABLE LENGTH			1	FT	
CABLE CAPACITANCE		15	25	PF	
RISE TIME		35	40	NS	
FALL TIME		20	25	NS	

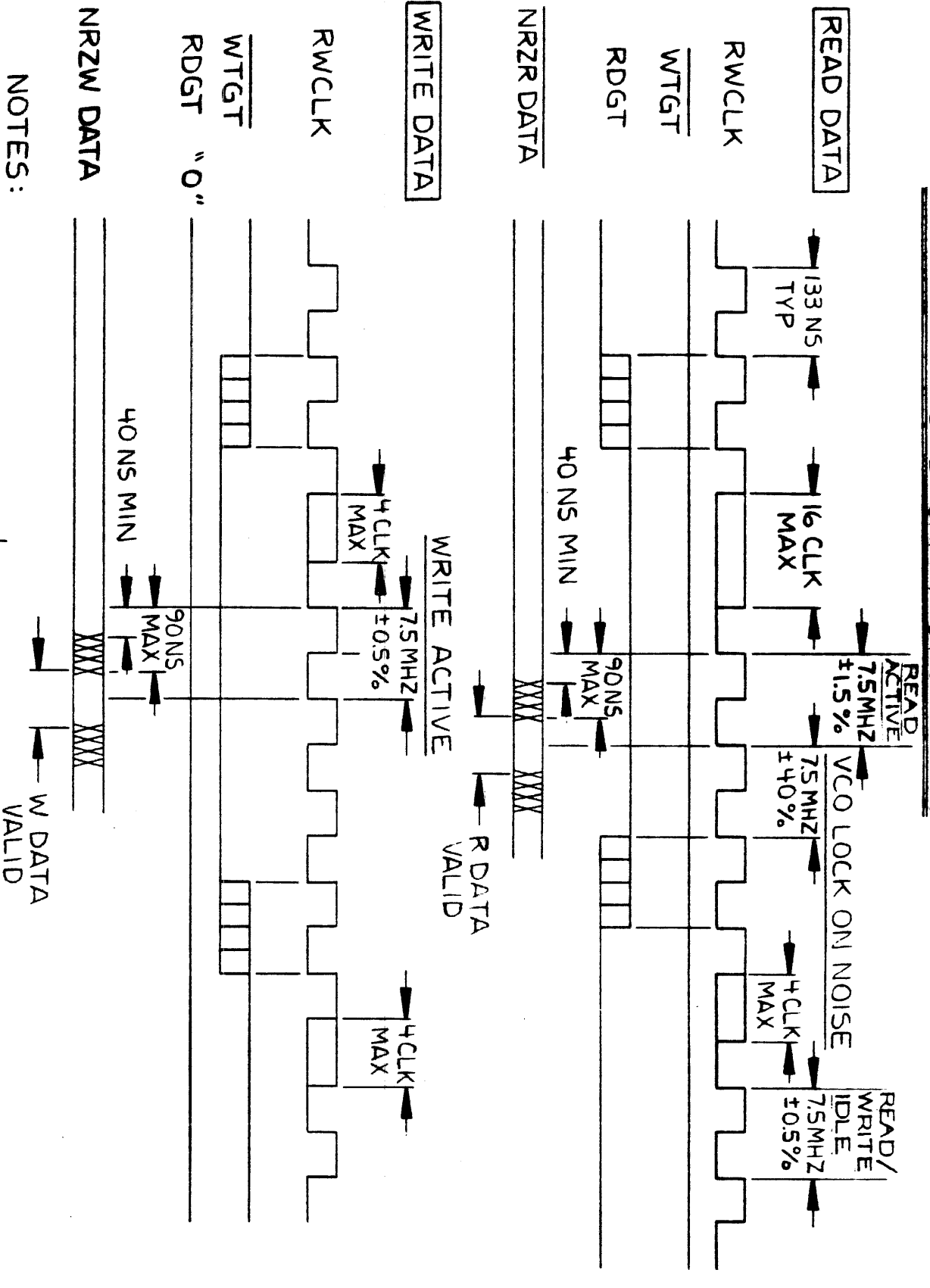


NISHA INTERFACE
AC ELECTRICAL CHARACTERISTICS



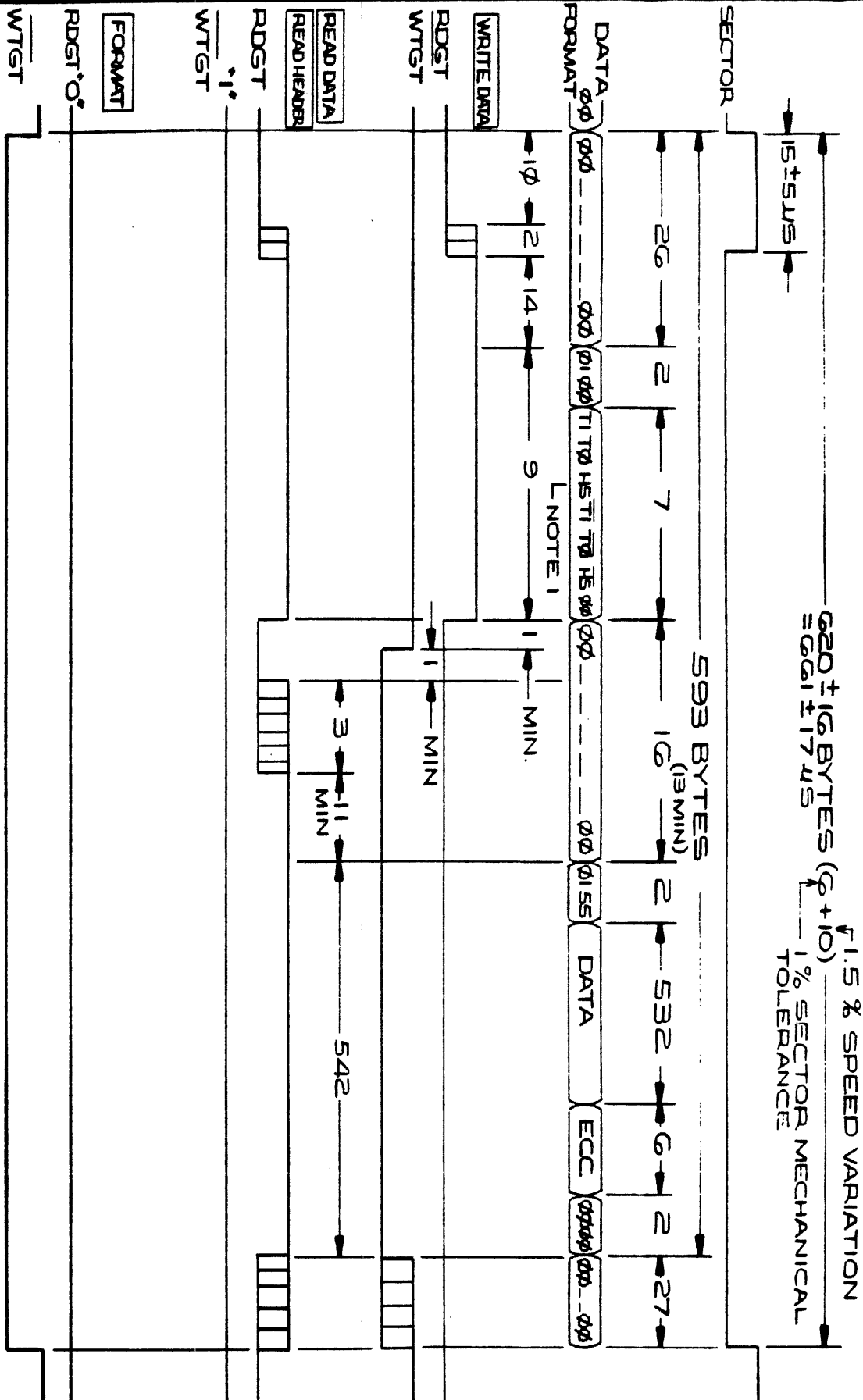
SIZE	DRAWING NUMBER	
A	062-0287-A	
SCALE:	_____	SHEET 12 OF 30

AC ELECTRICAL CHARACTERISTICS



NOTES:

- 1 - RWCLK PERIOD = $\frac{1}{7.5 \text{ MHz}} = 133.33 \text{ NS}$ (WITH $50 \pm 10\%$ DUTY CYCLE).
- 2 - HEAD SWITCHING RECOVERY TIME = $1 \mu\text{s}$.
- 3 - WRITE TO READ RECOVERY TIME = $80 \pm 20 \mu\text{s}$.



±1.5 % SPEED VARIATION
 620 ± 16 BYTES (G + 10)
 661 ± 17 μS
 1% SECTOR MECHANICAL TOLERANCE

NISHA INTERFACE
 AC ELECTRICAL CHARACTERISTICS

12.3 AC ELECTRICAL CHARACTERISTICS

NOTE 1: ID BYTES FORMAT

M	L
S	S
B	B
7 6 5 4 3 2 1 0	

T1 =	TRACK BYTE 1	
T0 =	TRACK BYTE 0	
HS =	HEAD SECTOR	
/T1 =	INVERSE OF TRACK 1	
/T0 =	INVERSE OF TRACK 0	
/HS =	INVERSE OF HD,STR	
00 =	LAST 00 BYTE	

12.4

-RESET P-18 Input. Resets the Servo z8 and the TL7705 (U24) power monitor I.C.

RESET Output from the TL7705 I.C.

RESET Output from the TL7705 I.C. complement of RESET

12.5 NORMAL POWER UP

(SEE FIG. A)

When +5V crosses a trigger threshold of 4.3V the TL7705 initiates a 140MS timing period.

At the end of this timing period RESET will go high and RESET will go low and the Spindle Motor will restart.

If +5V goes below the 4.3V threshold RESET will go low, RESET will go high and the TL7705 will initiate a new 140MS timing cycle.



apple computer inc.

SIZE
A

DRAWING NUMBER

062-0287-A

SCALE: _____

SHEET 15 OF 30

12.6 POWER UP WITH SPINDLE MOTOR HOLDOFF (SEE FIG. B)

If the $\overline{\text{RESET}}$ line is held low the TL7705 is held in a RESET state. $\overline{\text{RESET}}$ is low and RESET is high. Upon releasing the $\overline{\text{RESET}}$ line the TL7705 will continue to hold $\overline{\text{RESET}}$ low and RESET high for approximately 100MS. The Spindle Motor drive circuitry is enabled only when RESET from the TL7705 is low. Sending a $\overline{\text{RESET}}$ longer than $1\text{MS} + 10\%$ will cause the TL7705 to go into a RESET state.

12.7 SPECIFICATIONS

$\overline{\text{RESET}}$ $1\text{MS} - 0\% + 10\%$

TL7705 Timeout Period.

Power Up - $140\text{MS} \pm 10\%$

External RESET - $100\text{MS} \pm 15\%$

TL7705 Threshold $4.3\text{V} \pm 10\%$



SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 16 OF 30

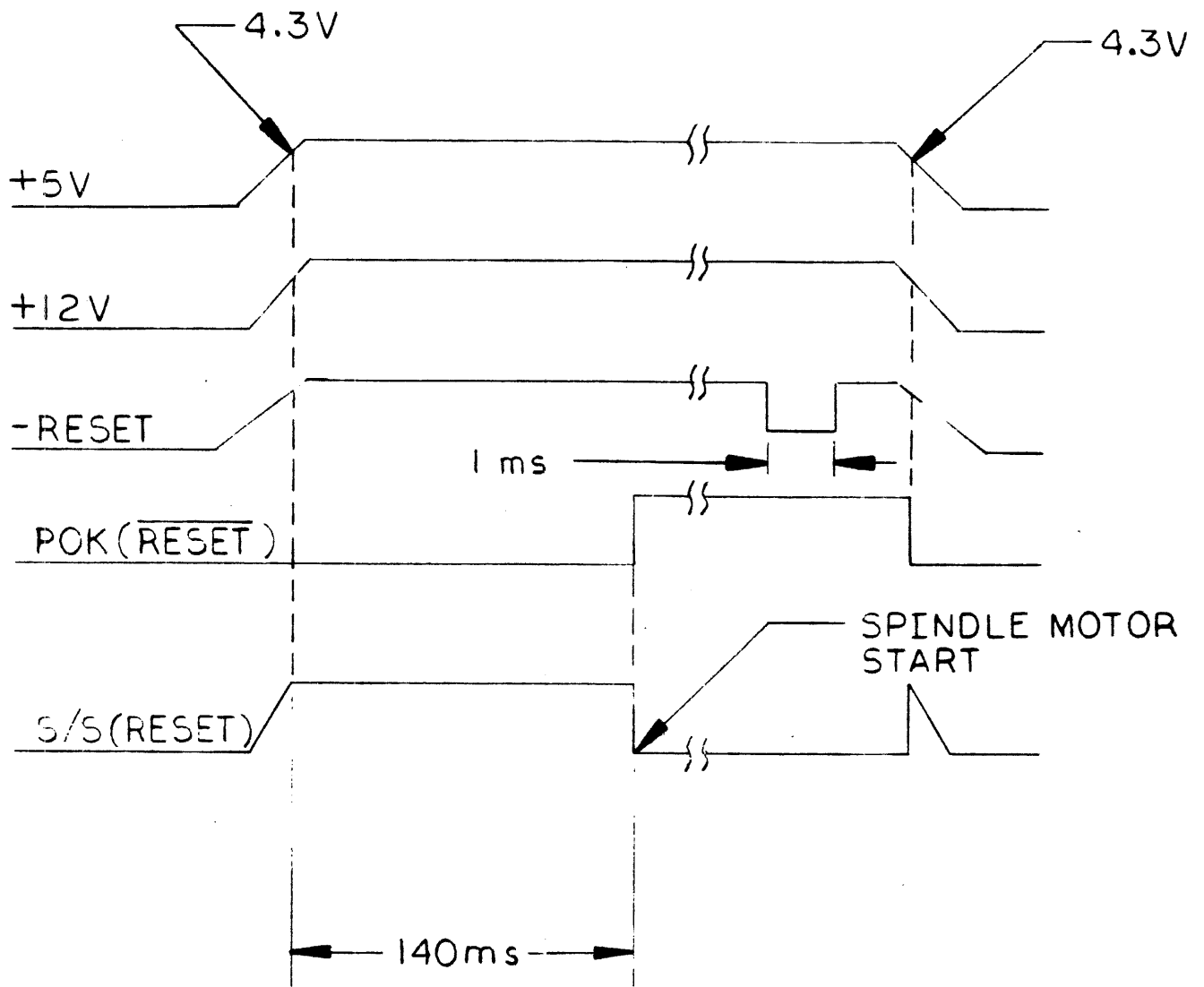


FIGURE "A"

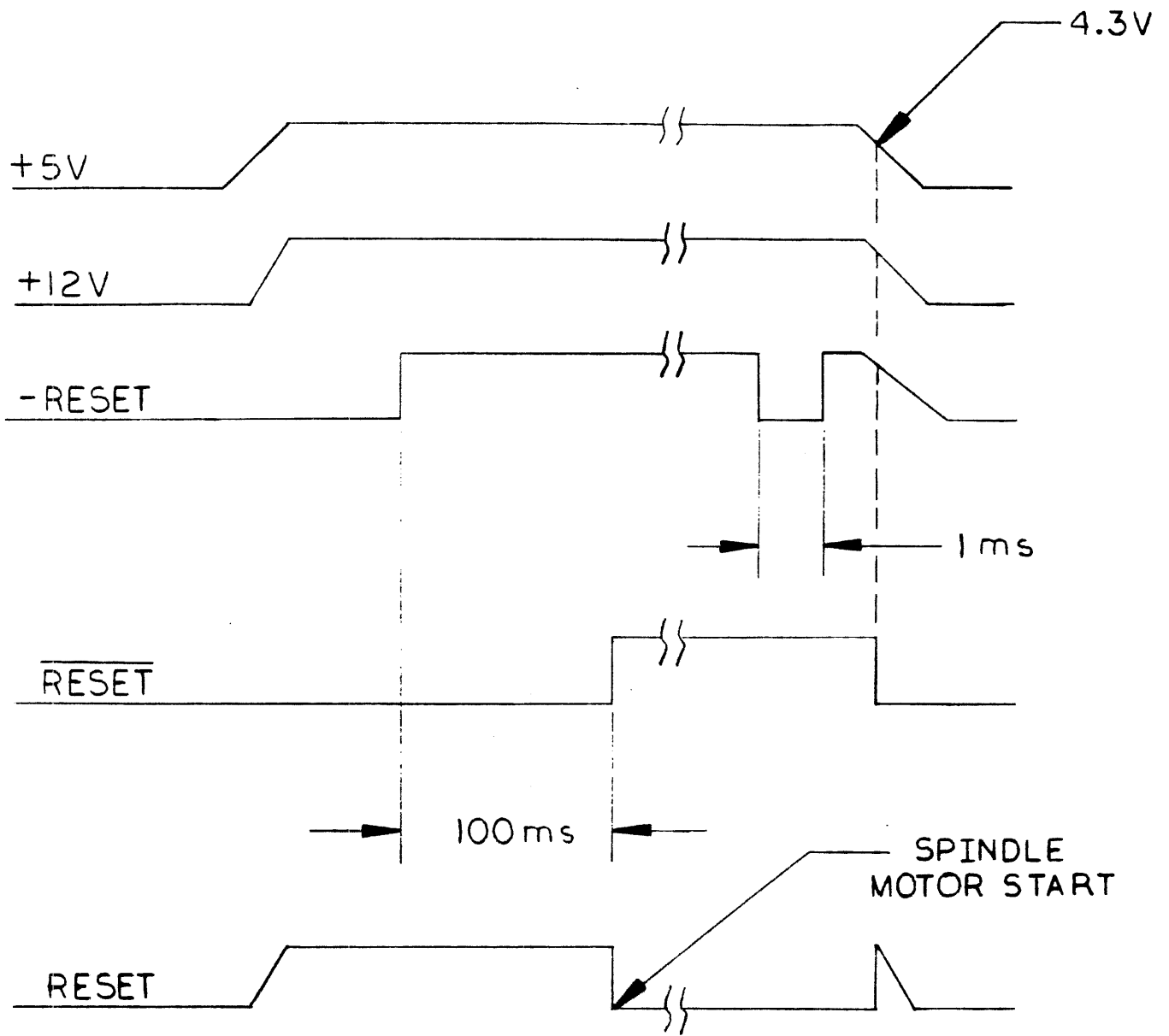


FIGURE "B"



apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: —

SHEET 18 OF 30

13.0 SERVO

13.1 BASIC SERVO FUNCTIONS

Nisha servo control functions are handled by a Z8 microprocessor. The Z8 handles all I/O operations, timing operations and communication with a host controller. Control functions to the Z8 Servo Controller are made through the serial I/O.

The following commands for the Nisha servo are:

- A. INHIBIT SERVO - not detented, heads off data zones located at the inner stop. Inhibit Servo is also generated following a reset of Z8.
- B. SET BRAKE - performs recal operation all the way to outer crash stop, holds brake to insure latching then inhibits servo back to inner crash stop.
- C. RECAL - 72, -0, +3 tracks from HOME. Used to initialize into data zone.
- D. ACCESS - coarse track positioning of data head to any desired track location.
- E. OFFSET TRACK FOLLOWING - controlled microstepping of fine position system during TRACK FOLLOWING (two modes).
 - 1. COMMAND OFFSET - direction and amount of offset is specified to the servo.
 - 2. AUTO OFFSET - command allows the servo to automatically move off track by the amount indicated by the embedded servo signal on the data surface (disk).
- F. STATUS - command can read servo status.



apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 19 OF 30

See Table 1 (Section IV) for the actual command description. With the present command structure a SEEK COMMAND can be augmented with an OFFSET COMMAND. Upon completion of a seek, the offset command bit is tested to determine if an offset will occur following a seek (either auto or command offset).

When a SERVO ERROR occurs the Z8 SERVO will attempt to do a short RECAL (ERROR RECAL). Two attempts are made by the system to do the ERROR RECAL function. If either of the two RECAL operations terminate successfully the protocol status will be SERVO READY, SID READY and -WRITE SAFE. Should the ERROR RECAL fail then the system will complete the error recovery by a inhibit servo function.

The two OFFSET commands will be described. First COMMAND OFFSET is a predetermined amount of microstepping of the fine position servo. Included in the OFFSET BYTE (STATREG) bit B6=0 is a COMMAND OFFSET. Bit B7-1 is a forward offset step (toward the spindle); B7=0 is a reverse step. If bit B6-1, the OFFSET command is AUTO OFFSET.

AUTO OFFSET command normally occurs during a write operation. When the HDA was initially formatted at the factory, special encoded servo data was written on each track "near" the index zone. The reason for this follows:

"Normal coarse and fine position information for the position servos is derived from an optical signal relative to the actual data head-track location. Over a period of time, the relative position (optical signal) will be misaligned to the absolute head-track position by some unknown amount (less than 100 uln). This small change is important for reliability during the write operation. Write/Read reliability can be degraded due to this misalignment. The special disk encoded servo signal is available to the fine position servo. It will correct the difference between the relative position signal of the optics and the absolute head to track position under the data head only a index time. The correction signal can be held indefinitely or updated (if desired at each index time) until a new OFFSET command or move command (SEEK or RECAL) occurs."



apple computer inc.

SIZE
A

DRAWING NUMBER

062-0287-A

SCALE: —

SHEET 20 OF 30

13.1 COMMUNICATION FUNCTIONS

The servo functions described in the previous section only occur when the servo Z8 microprocessor is in the communication state. Communication states occur immediately after a system reset, upon completing head setting after a recal, seek, offset, read servo status or set servo diagnostic command. If + SID READY is not active, no communication can exist between the external controller and the servo Z8 processor.

Servo commands are serial bits grouped as five separate bytes total. Refer to Table 1 parts I through V for the total communication string. The first byte is the command byte (i.e. seek, read status, recal, etc.). The second byte is the low order difference for a seek (i.e. Byte 2 = \$0A is a ten track seek). The third byte is the offset byte (AUTO or COMMAND OFFSET and the magnitude/direction for command offset). The fourth byte is the status byte (use for reading internal servo status) Byte five is the check sum byte used to check verify that the first four bytes were correctly transmitted (communication error checking).

Part of the communication function requires a specific protocol between the servo Z8 processor and the external controller.



apple computer inc.

SIZE
A

DRAWING NUMBER

062-0287-A

SCALE: _____

SHEET 21 OF 30

13.2 Z8 SERVO PROTOCOL

The protocol between the Z8 SERVO microcomputer and the CONTROLLER is based on five I/O lines. Two of the I/O lines are serial input (to Z8 servo from controller) serial output (from Z8 servo to controller). Data stream between the Z8 servo and controller is 8 bit ASCII with no parity bit (the fifth byte of the command string contains check sum byte use for error checking). There are three additional output lines between the Z8 servo used as control lines to the controller. Combining the two serial I/O lines and the three unidirectional port lines generates the bases of the protocol between the Z8 servo and controller. The important operations between the Z8 servo and controller are:

1. Send commands to Z8 servo.
2. Read Z8 servo status.
3. Check validity of all four command bytes.
4. I/O timing signals between the Z8 servo and controller.
5. Z8 servo reset.

Sequencing the Z8 servo controller is an important process following a Power Up or if the controller should issue a Z8 Servo Reset at any time. After a Z8 Servo Reset is inhibited, the Z8 I/O ports and internal register are initialized. This takes approximately 75 msec after the Z8 Servo Reset is inhibited. The protocol baud rate is automatically set to 58.59KB and then the system is parked at HOME position by inhibiting the SERVO and SIO READY is set active.

Before the controller transmits the command byte the controller must poll the SIO READY line from the Z8 servo to determine if it is active (+5 volts). If the line is active then a command can be transmitted to the Z8 servo. The program in the Z8 servo will determine what to do with the command bytes (depending upon the current status of the Z8 servo). After the command (five bytes long) has been transmitted to the Z8 servo, the program in the Z8 servo will determine if the command bytes (first four bytes) are in error by evaluating the check sum byte (fifth byte transmitted). After the controller has transmitted the last serial string and -WRITE SAFE is true (0V) it must wait 250 U sec then test for -WRITE SAFE.



apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: —

SHEET 22 OF 30

If -WRITE SERVO is high the command was rejected (check sum error or invalid command). If -WRITE SAFE is set inactive 600 U sec after the command is sent (and not 250 U sec), this was a command reject. This unsafe condition must be cleared by a READ STATUS COMMAND or RECAL COMMAND before transmitting another command. If SERVO READY is low then this protocol is not valid since SERVO READY is gated with $\overline{\text{SERVO ERROR}}$ and on track to generate -WRITE SAFE. This protocol is only effective if the drive is track following (SERVO READY is true).

As long as SID READY is active the controller can communicate with the Z8 Servo Controller. If SERVO READY is not active the only command that will cause the Nisha Servo to set SERVO READY active is a RECAL COMMAND. Read Status will only clear SERVO ERROR if SERVO READY is also true, and all other commands will be rejected.

Next, if SERVO READY is high and -WRITE SAFE is also high, -WRITE SAFE can be cleared by:

1. Any READ STATUS COMMAND.
2. Any RECAL COMMAND.
3. Any other commands will be rejected and maintain -WRITE SAFE

If a SEEK COMMAND is transmitted with both SERVO READY and $\overline{\text{-WRITE SAFE}}$ active, the command will be rejected.

It is important to check the status of all three status lines from the Z8 Servo. It is best to avoid sending a SEEK COMMAND with SERVO READY and $\overline{\text{-WRITE SAFE}}$ inactive. If a seek length of 1024 or great is sent, the command will be rejected and -WRITE SAFE will go high.

Chart V, parts A-1, illustrate some of the serial communication commands and error conditions that can occur between the controller and Z8 SERVO.



apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 23 OF 30

13.3 ERROR HANDLING

-WRITE SAFE will be generated during the following conditions:

1. During **Recal** mode (velocity control only) access time-out. If a **Recal** function exceeds 220 msec then an access timeout occurs.
2. During **Seek** mode (velocity control only) access time-out. If a **Seek** function exceeds 220 msec then an access time-out occurs.
3. During **Settling** mode (following a **Recal**, **Seek**, or **Offset**) if there is excessive **On Track** pulses (3 crossings) indicating excessive head motion, a **Settling** error check will occur.
4. During a command transmission if a communication error occurs (check sum error).
5. During a command transmission if a invalid command is sent.



apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 24 OF 30

28 SERVO COMMAND BYTES

TABLE 1

I. BYTE 1: COMMAND BYTE (DIFCNTH)

		B7	B6	B5	B4	FUNCTIONS
command bits	---	1	0	0	0	access only
	B7	1	0	0	1	access with offset
	B6	0	1	0	0	data recal
	B5	0	1	1	1	set brake
	B4	0	0	0	1	offset-trk following
	---	1	1	0	0	inhibit servo
	---	0	0	0	0	read status command

access bits

B3 - access direction
 B2 - NOT USED(command will be rejected -WRITE SAFE will go high)
 B1 - hi diff2 (512)
 B0 - hi diff1 (256)

MAXIMUM seek length is +/-1023

access direction = 1 (FORWARD: toward the spindle)
 = 0 (REVERSE: away from the spindle)

hi diff2 (512) = 1 (512 tracks to go)
 = 0 (not set)

hi diff1(256) = 1 (256 tracks to go)
 = 0 (not set)

II. BYTE 2: DIFF BYTE (DIFCNTH)

command BYTE 2 contains the LOW ORDER DIFFERENCE COUNT for a seek

B7-bit7 = 128 tracks
 B6-bit6 = 64 tracks
 B5-bit5 = 32 tracks
 B4-bit4 = 16 tracks
 B3-bit3 = 8 tracks
 B2-bit2 = 4 tracks
 B1-bit1 = 2 tracks
 B0-bit0 = 1 track



SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: —

SHEET 25 OF 30

III. BYTE 3: OFFSET BYTE (STATREG)

command BYTE 3 contains the INSTRUCTION for an OFFSET COMMAND (seek or during track following)

B7-offset direction

B6-auto offset function

B5-not used

B4-offset Bit4=16

B3-offset Bit3=8

B2-offset Bit2=4

B1-offset Bit1=2

B0-offset Bit0=1

1. if offset command from BYTE 1 is followed by bit6 set (auto offset); offset direction (bit7) read offset (bit5) and bits 4-0 are ignored but should be set to 0 if not used.
2. OFFSET DIRECTION = 1 (FORWARD OFFSET: toward the spindle)
= 0 (REVERSE OFFSET: away from the spindle)
3. AUTO OFFSET = 1 (normally used preceding a write operation)
= 0 (manual offset: MUST send direction and magnitude of offset)

IV. BYTE 4: STATUS BYTE (CNTREG)

B7 - NOT USED

B6 - NOT USED

B5 - NOT USED

B4 - NOT USED

B3 - status or diagnostic bits

B2 - status or diagnostic bits

B1 - status or diagnostic bits

B0 - status or diagnostic bits

Status call = \$00,\$00,\$00,\$02: The return will be 4 bytes (exclude check sum) = \$01 (Drive ID for Nisho), \$0A (ROM CODE Version), \$5A (Byte filler), \$A5 (Byte filler).

 apple computer inc.

SIZE
A

DRAWING NUMBER
062-0287-A

SCALE: —

SHEET 26 OF 30

V. BYTE 5: CHECKSUM BYTE (CKSUM)

[B7 B6 B5 B4 B3 B2 B1 B0]

results of the transmitted CHECKSUM BYTE are derived as:

(BYTE 1 + BYTE 2 + BYTE 3 + BYTE 4) = CHECKSUM BYTE

(+) is defined as the addition of each BYTE

(BYTE) is defined as the compliment of the BYTES (1-4)

VI. The SERVO STATUS lines (SIO RDY, SERVO RDY, SERVO ERROR) must have the following conditions in order to send the listed Z8 COMMANDS:

SERVO STATUS

(-)

S S W
I R R
Q Y T

R R S
D D A
Y Y F

Z8 SERVO CMD HEX _____

access (only)	8X	1	1	0
access(offset)	9X	1	1	0
recal(data)	40	1	X	X
set brake	70	1	X	X
inhibit servo	C0	1	X	X
offset(detent)	10	1	1	0
status	00	1	X	X

X = either 0, 1



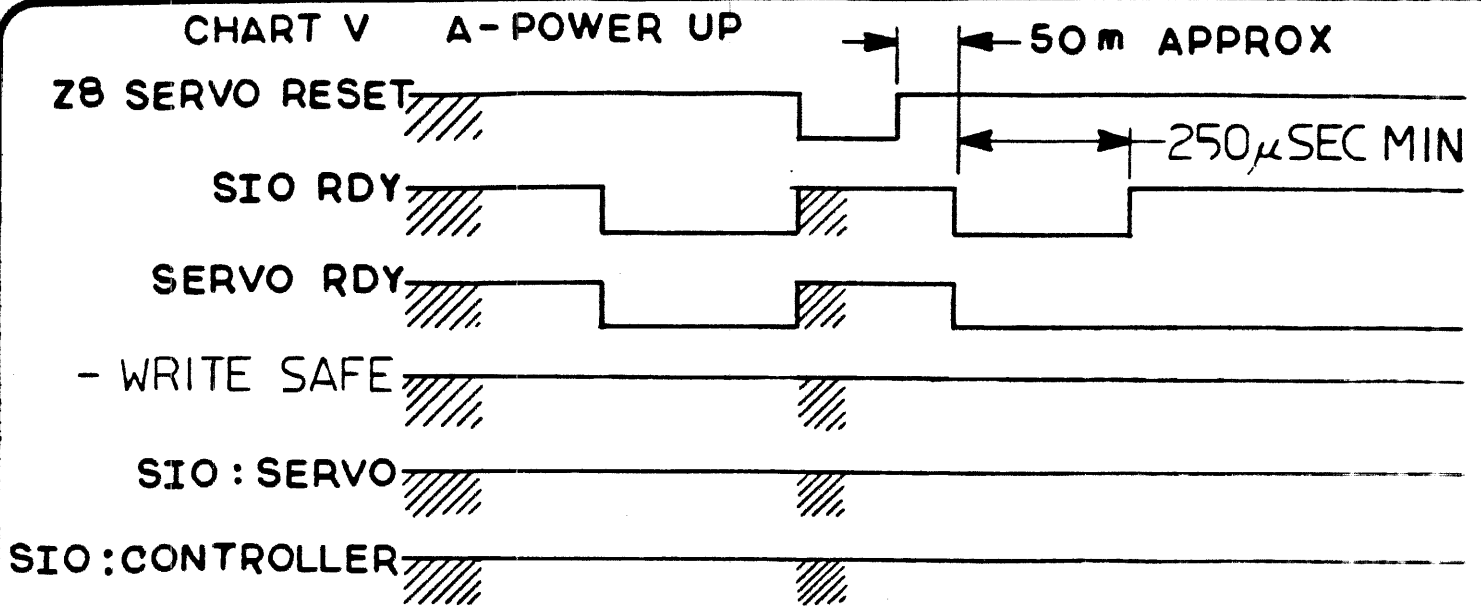
apple computer inc.

SIZE
A

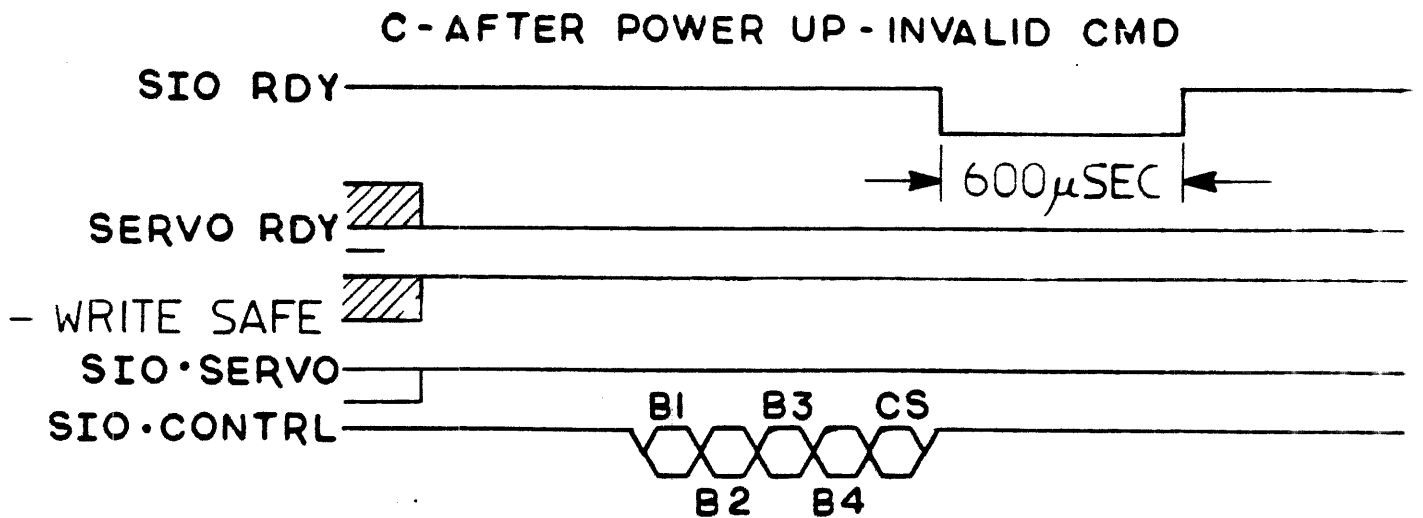
DRAWING NUMBER
062-0287-A

SCALE: _____

SHEET 27 OF 30



N/S - NOT SPECIFIED



 apple computer inc.

SIZE
A

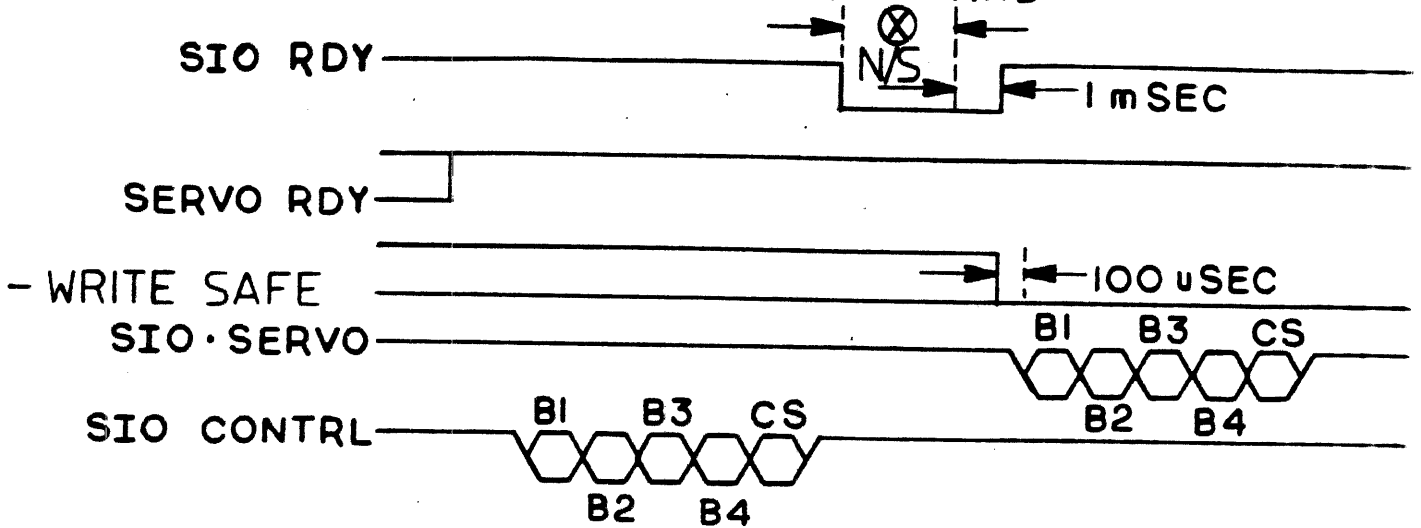
DRAWING NUMBER
062-0287-A

SCALE: _____

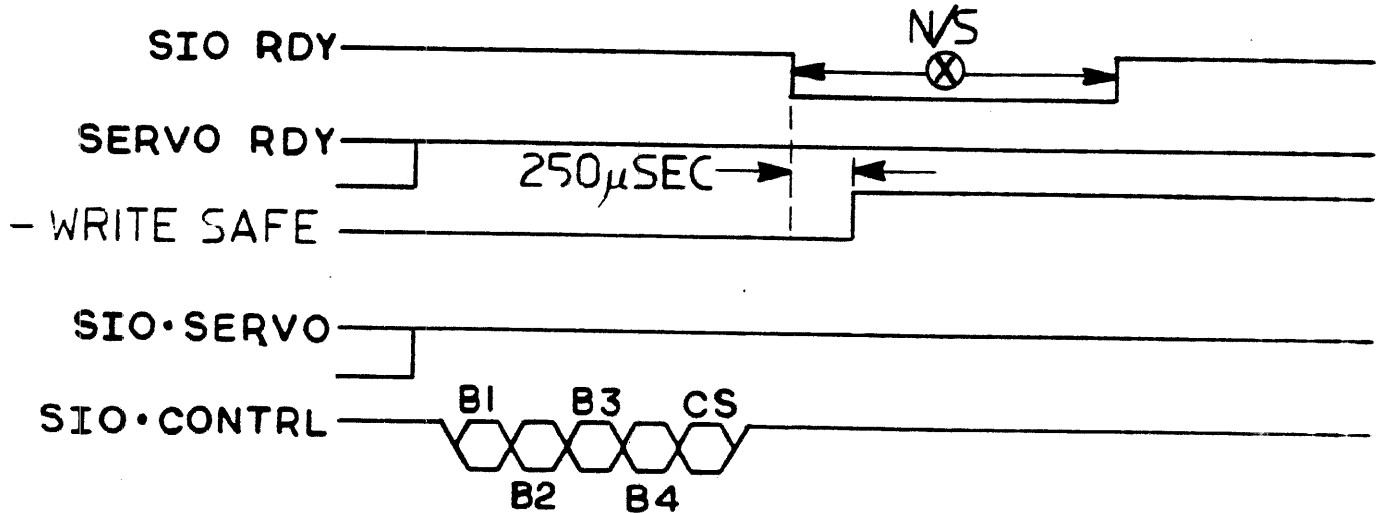
SHEET 28 OF 30

CHART V

D-READ STATUS COMMAND



E-TRACK FOLLOWING SERVO ERROR - INVALID COMMAND



F-TRACK FOLLOWING SERVO ERROR - READ STATUS

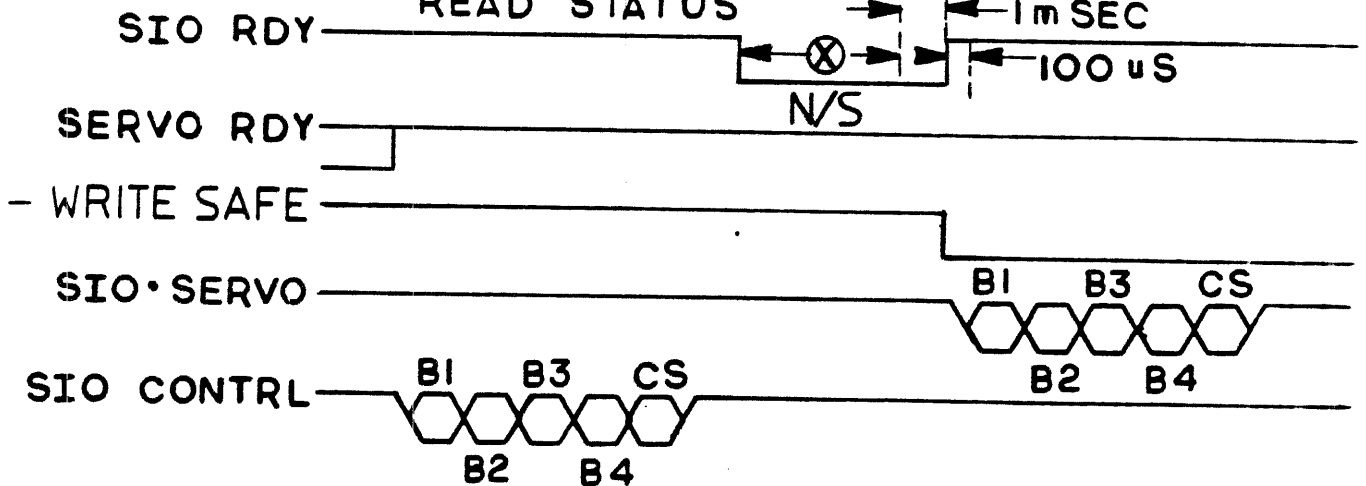
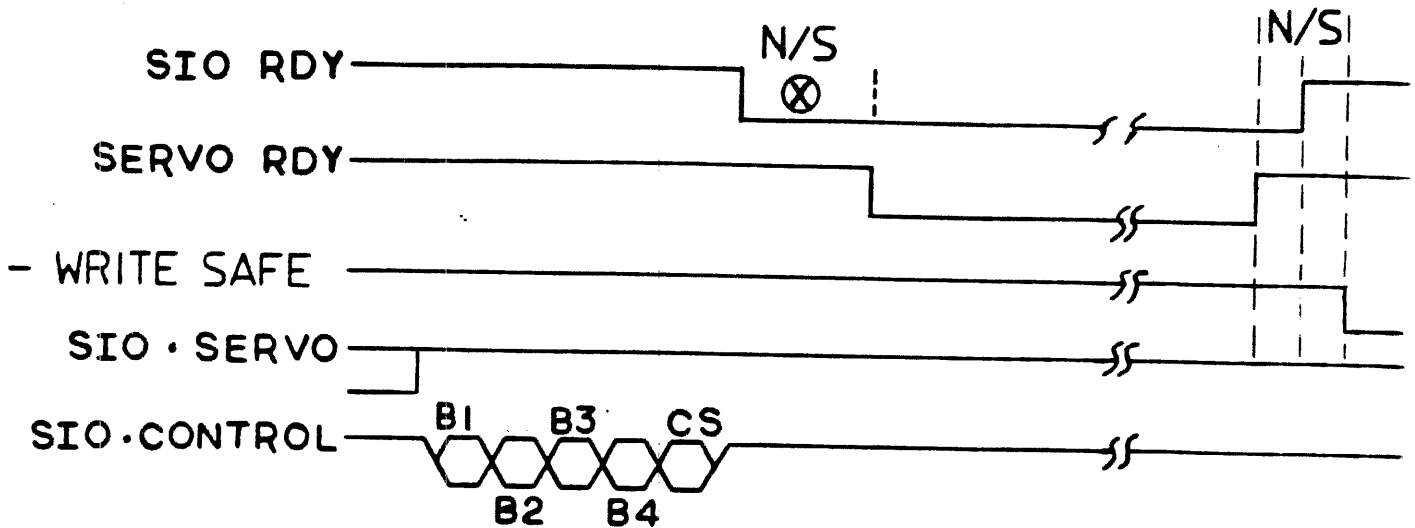
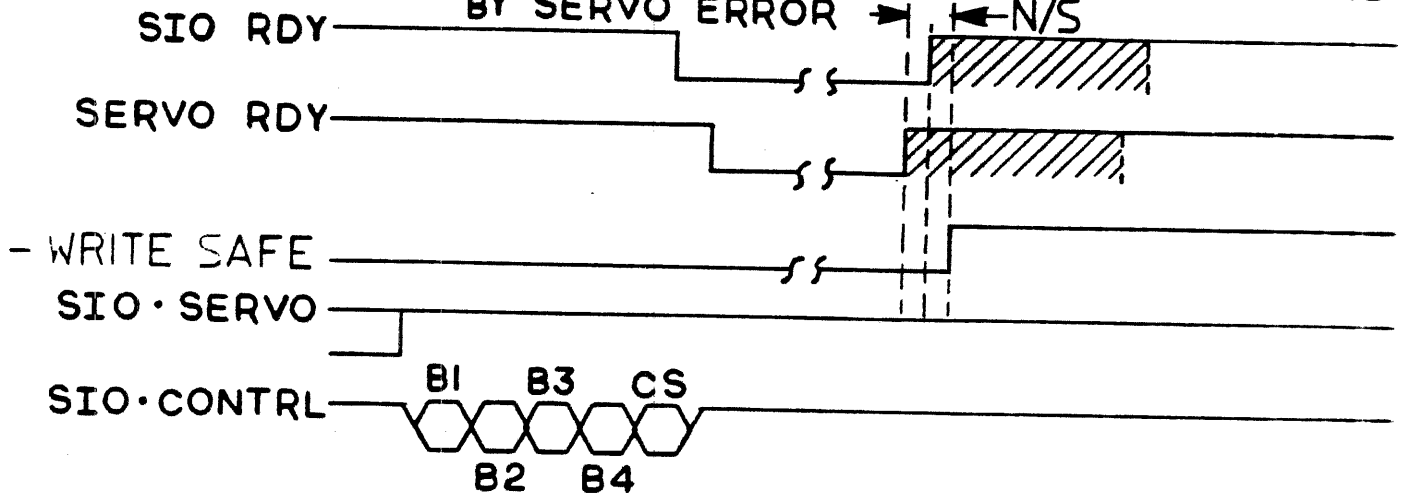


CHART V

G-TRACK FOLLOWING VALID COMMAND (MOVE)



H-TRACK FOLLOWING (MOVE CMD) FOLLOWED BY SERVO ERROR → ← N/S



I-TRACK FOLLOWING (NO CMD) SERVO ERROR

