I. Physical Interface - Pinout and Signal Definition

The pinout of the Macintosh DB19 connector is defined below. The third column defines the use of those pins for this interface.

Pin #	Macintosh name	DCD usage	Comment	Macintosh connection
1	GND	GND		Cold ground
2	GND	GND		Cold ground
3	GND	GND		Cold ground
4	GND	GND		Cold ground
5	-12v	N/C	DCD to be self-powered!	
6	+5v	N/C	DCD to be self-powered!	
7	+12v	N/C	DCD to be self-powered!	
8	+12v	N/C	DCD to be self-powered!	
9	N/C	N/C		
10	PWM	N/C		
11	PH0	Phase0	Phase0-2 used for handshake	IWM
12	PH1	Phase1		IWM
13	PH2	Phase2		IWM
14	PH3	Phase3	Used to allow multiple DCD's	IWM
15	/WrReq	N/C		
16	HDSel	N/C		VIA/6522
17	/ENBL2	/Enable		IWM
18	RD	ReadData	Also connected to Sense on IWM	IWM
19	WR	WriteData		IWM

Note that all DCD lines are outputs (from the perspective of the Macintosh) except ReadData, which is input.

WriteData (from the point of view of the Macintosh) provides serial data transmission at 489.58K bps (2.042 microsecond data cell). This is because the Macintosh clock is 7.8333 instead of a full 8.0 MHz. The high-order bit of each byte is always 1--a net data rate of 428.38K bps. ReadData is bi-modal as a function of Phase0-2 (described below). In read mode ReadData functions in symmetric fashion to WriteData. In Sense mode it has a 0 or 1 "constant" value. For further information see Dwg. #343-0041-B.

As described below, the eighth bit of each data byte is collected into a group (7 bits to a group) and then transmitted separately (with the high-order bit set as required by the IWM).

II. Handshake and Data Transmission

Data is transmitted on the WriteData and read from the ReadData lines. The most significant bit (MSB) of each byte transmitted by the IWM is always set (this is a requirement of the IWM chip). For terminology, we will therefore distinguish between "data bytes" and "transmitted bytes"; data bytes have a full eight bits of information while transmitted bytes have sevenbits of information with the MSB set. In order to send seven data bytes, eight transmitted bytes must be sent. We therefore talk about a "group" of seven data bytes (or eight transmitted bytes).

The least significant bit (LSB) of each data byte in a group is collected into the seven low order bits of an eighth byte (which will become the eighth transmitted byte). The seven MSB's of each data byte are shifted right one bit. The MSB of all of these eight bytes are then set, becoming a "group" of eight transmitted bytes. Figure 1 on the next page shows some example data.

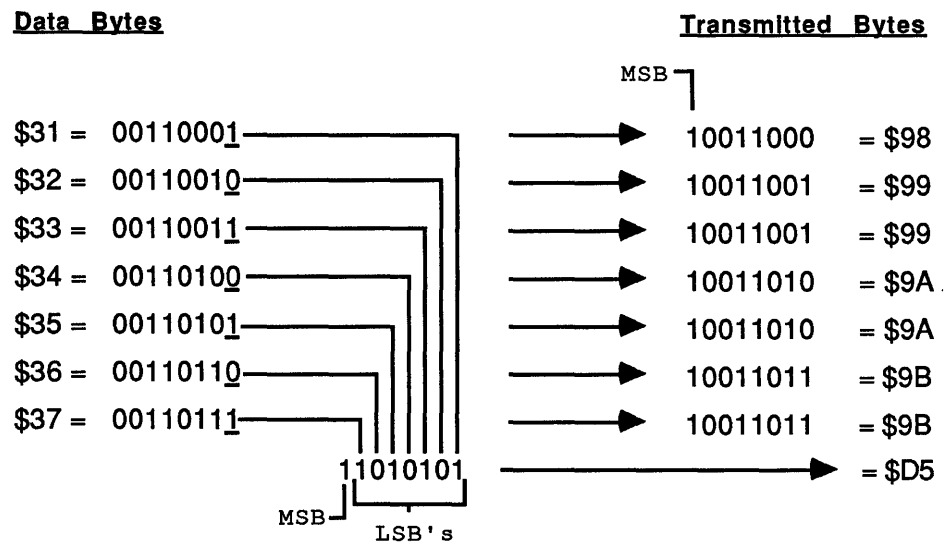


Figure 1.
Transforming 7 data bytes into 8 transmitted bytes.

When sending data from the Macintosh to the DCD, the LSB-byte is sent first, followed by the other seven transmitted bytes, in order. Using the sample data from figure 1:

```
$D5 $98 $99 $99 $9A $9A $9B $9B
```

When sending data from the DCD to the Macintosh, the LSB-byte is sent last, following the other seven transmitted bytes, in order. Using the sample data from figure 1:

```
$98 $99 $99 $9A $9A $9B $9B $D5
```

The phase lines plus the sense status of ReadData are used during startup to determine the presence of a drive and its type. Subsequently they are used to handshake between the Macintosh and the DCD to coordinate data transmission. Note that the Macintosh acts as the master. The states are shown in the table below. Note that the phase lines can only change one at a time--one can't go instantly from state 0 to 3, for example.

State	Phase2	Phase1	Phase0	Interpretation
0	0	0	0	HOST asserted; HOFF asserted; ReadData in sense mode for /HSHK
1	0	0	1	HOST asserted; ReadData in data mode
2	0	1	0	Idle; ReadData in sense mode for /HSHK
3	0	1	1	HOST asserted; ReadData in sense mode for /HSHK
4	1	0	0	RESET asserted (DCD performs equivalent of power-up reset)
5	1	0	1	ReadData in sense mode; DCD must set ReadData to 0 (unused in Sony)
6	1	1	0	ReadData in sense mode; DCD must set ReadData to 1 (# sides in Sony)
7	1	1	1	ReadData in sense mode; DCD must set ReadData to 1 ("drive connected")

After startup the Macintosh will transition through states 6, 7, and 5 to determine that a drive is connected and that it is a DCD as opposed to a 1- or 2-sided Sony.

In fact, it is possible to connect more than one DCD to a Macintosh if they have the proper "flow-through" circuit (or if a T-connector with such a circuit is employed). As noted in section I, the Phase3 line is reserved for connecting to multiple DCD's with the possibility of an external Sony drive at the end of the chain. Figure 2 on the next page shows a possible flow-through circuit (courtesy of Gary Marten).

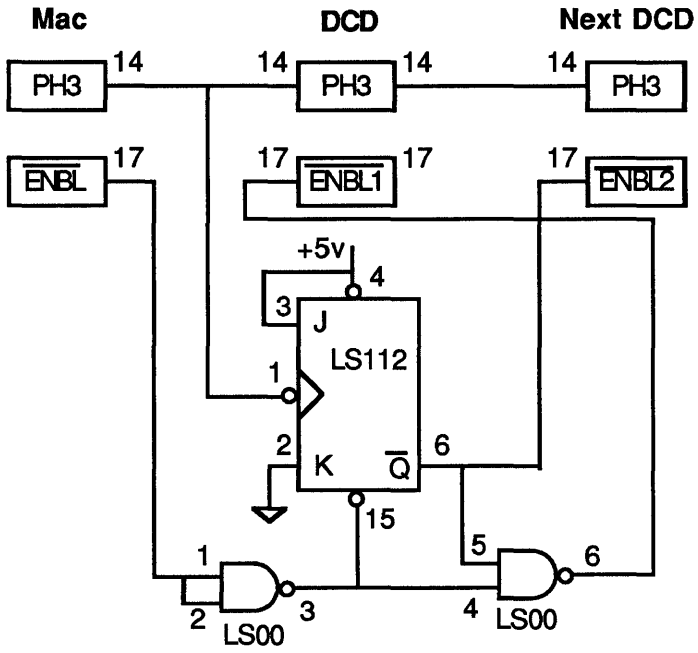


Figure 2.
Flowthrough Schematic

As is shown in Figure 2, whenever the /ENBL line is raised (i.e., de-asserted) then the flip-flop in the flowthrough circuit will clear, thus clearing any flowthrough circuits down the line. When the /ENBL line is lowered (asserting it), then the first DCD is enabled. At this time, the Macintosh will go through the ID states (6, 7, and 5) to determine if the device selected is a DCD or an external Sony drive. If it is a DCD, then the Macintosh may toggle Phase3 to enable the next device in the chain (and disable the current DCD). Once again, the Macintosh should check the ID states to see if another device exists, and if so, what kind. This can be repeated through all the devices chained together. **Note:** Any DCD connected to the Macintosh *must* support this flowthrough circuit so that the Macintosh does not assume there is an infinite sequence of DCD's connected. If a DCD does not support additional chaining, it must support "phantom" states (i.e., returning a 1 for all states 5, 6, and 7) after the "next" DCD has been selected. In this way, the Macintosh can know it has reached the end of the chain.

Following the startup handshake, states 0-3 are used to transmit data between the Macintosh and DCD. As described in the next section, transmissions are always paired--Macintosh to DCD followed by a response from the DCD to Macintosh. Thus it is always clear which is the reader and which is the writer.

Macintosh to DCD Handshake

A diagram of the Macintosh-to-DCD handshake is shown in figure 3 on the next page.

When the Macintosh wants to transmit it transitions to state 3 via the pahse lines (asserting that it wants to send a command) and polls ReadData (the IWM Sense bit). When it goes low indicating that the DCD is ready to receive, the Macintosh transitions to state 1 and sends a sync byte (either $AA or $96) followed by transmitted bytes on the WriteData line. Typically this continues to the end of the transmission at which point the Macintosh waits for the last transmitted byte to go out, and then transitions to state 3 in order to sense the DCD's /HSHK line again. When it goes high, the Macintosh transitions to the idle state, state 2.

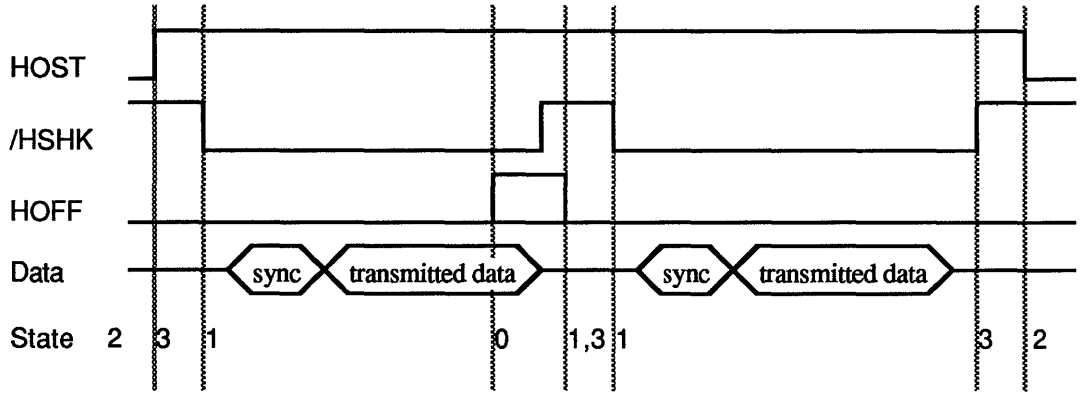


Figure 3.
Macintosh to DCD Handshake.

If the Macintosh wants to interrupt the transmission (e.g., to service the SCC chip) it sets the phase lines to state 0 (HOFF asserted). After completing the transmission of the current group of eight transmitted bytes, WriteData becomes invalid. The DCD indicates its acknowledgement of the Hold-off by de-asserting ReadData (/HSHK). When the Macintosh is ready to resume transmission it cycles briefly through state 1 to state 3, waits for /HSHK, transitions to state 1, sends the same sync byte as at the beginning of the transmission, and then restarts transmitting with the group that was interrupted. **Note:** If interrupts are occuring with high frequency, then it is possible for the Macintosh to do a hold-off in the middle of the same (restarted) group over and over again, potentially forever.

In order to abort a command, the Macintosh must do a hold-off first (by transitioning to state 0) and then de-assert *both* HOST and HOFF by transitioning straight to state 2. The only place in which a command may not be aborted is when the DCD is waiting for a sync byte.

DCD to Macintosh Handshake

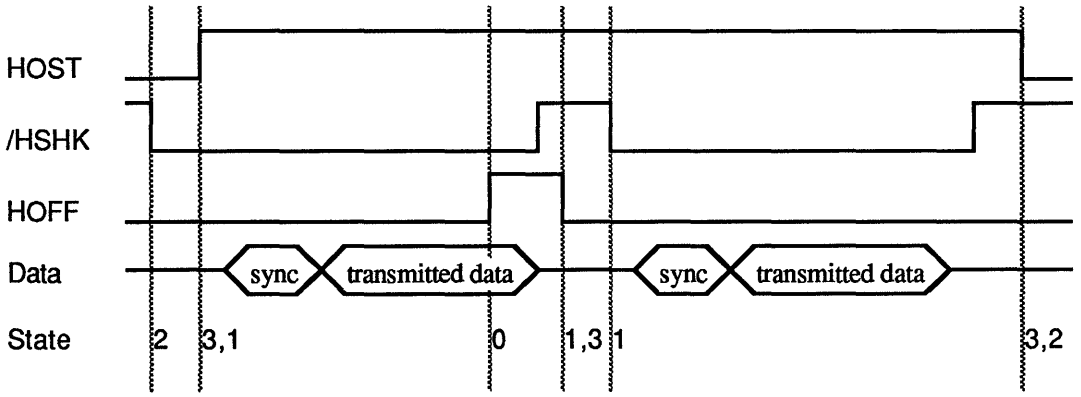


Figure 4.
DCD to Macintosh Handshake.

When the DCD wants to transmit (a response to a command) it asserts /HSHK. When the Macintosh senses this on ReadData (sense mode) it transitions briefly through state 3 to state 1 and begins reading data (from ReadData in data mode). Following reception of a sync byte (which is always $AA) actual data is received until /HSHK is de-asserted. The Macintosh transitions briefly through state 3 back to the idle state 2.

If the Macintosh must interrupt its reception it transitions to state 0 (HOFF asserted) and can then

ignore the rest of the group in progress. The DCD de-asserts /HSHK following its transmission of the last byte of the group. When the Macintosh is ready to resume it de-asserts HOFF, transitioning briefly through state 1 to state 3. When /HSHK is asserted it transitions to state 1 where it reads a sync byte ($AA again) and then restarts reading data with the group that was interrupted. **Note:** If interrupts are occuring with high frequency, then it is possible for the Macintosh to do a hold-off in the middle of the same (restarted) group over and over again, potentially forever.

III. Command, Status and Data Formats

Provided here are the formats for Status, MultiBlock Read, and MultiBlock Write. The synchronizing bytes ("sync-bytes") are shown in bold-face before the data bytes (NOTE: since the sync-bytes are sent "raw" over the IWM, they will always have the hi-bit set).

For the status byte that is returned on some commands, only three bits are defined. The most significant bit indicates "operation failed". The Macintosh will only retry or fail when this is set.

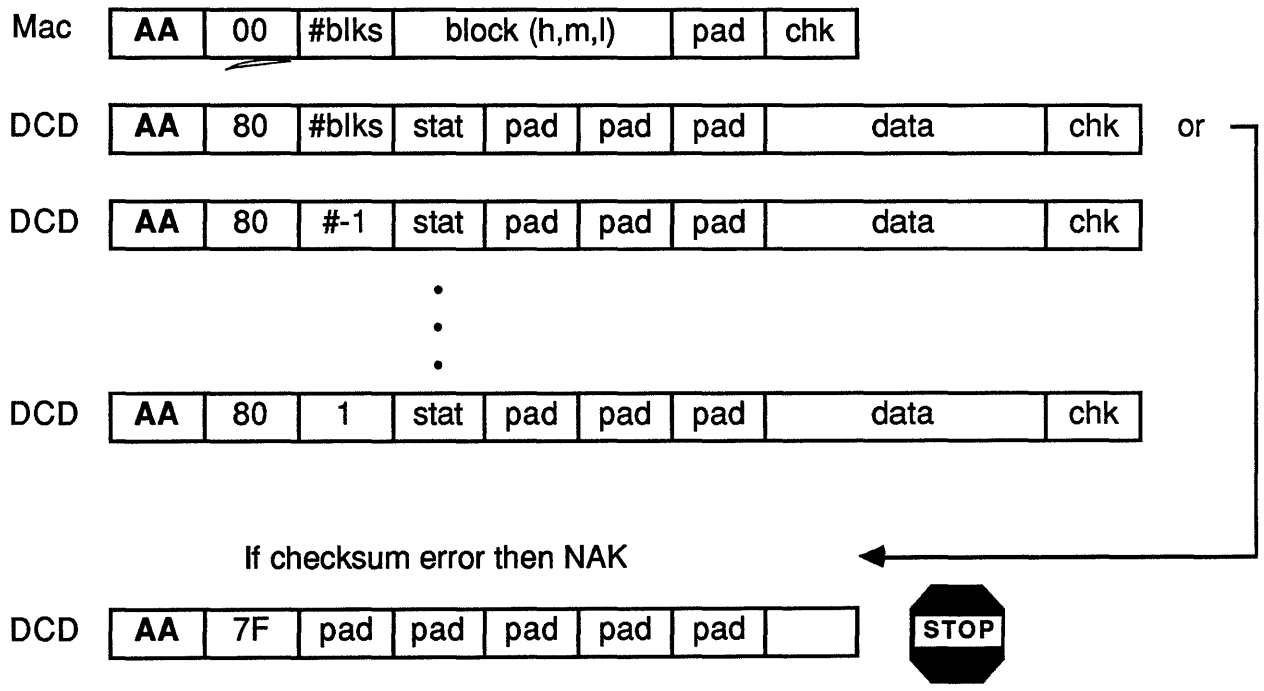
Const

```
{status bits}
stat_opfailed = $80;
stat_softerr  = $40;
stat_warn     = $20;
```

The "softerr" bit indicates some other exceptional condition. This would indicate to a user diagnostic that further information should be queried from the drive. Some examples of exceptional conditions that are not fatal are 1) a block was spared during the last operation, 2) a recoverable seek error occured, or 3) a recoverable ECC error occured.

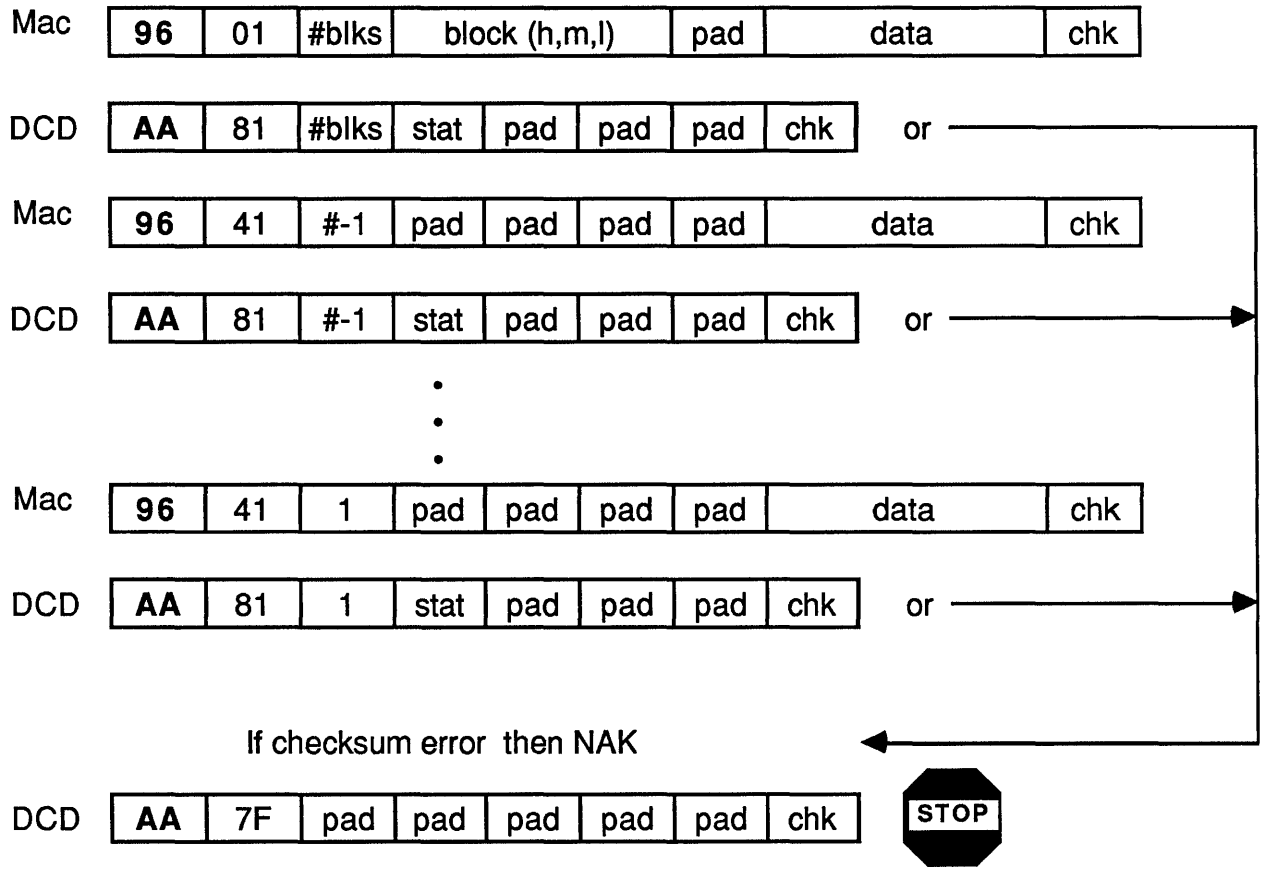
The "warn" bit indicates that the condition of the disk currently is in doubt. Currently, the only condition that would set this bit is having less than five spare blocks. The operating system can take this warning and notify some application (perhaps the Finder) to post a warning to the user.

MultiBlock Read



Note that the command includes only one byte for the block count. The (possibly multiple) reponses will decrement the sequence #, starting at the number of blocks. For example, if the Macintosh requests 10 blocks, the sequence numbers will go from 10 to 1.

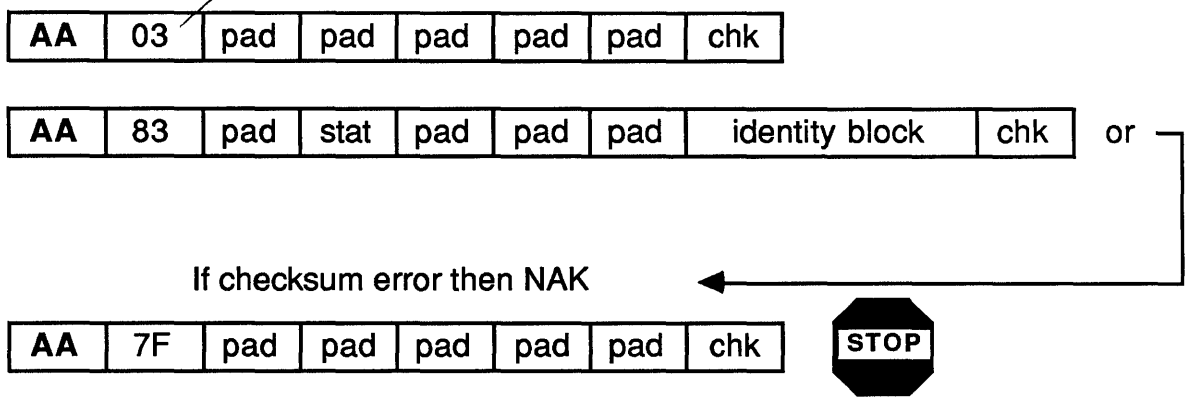
MultiBlock Write/Write-Verify - 02



In this case, when the block count is greater than 1, the sequence numbers start at one less than the block count and continue down to one this is because the first block (sequence number "count") is included with the command. The first block of data is sent with the command in order to optimize one-block writes, of which there seem to be a lot in the Macintosh.

Note that there are three bytes of padding between the sequence number and the write data when sending more than one block. This is so that the data will line up the same during each write-transmission, which should make it easier for coding.

Status

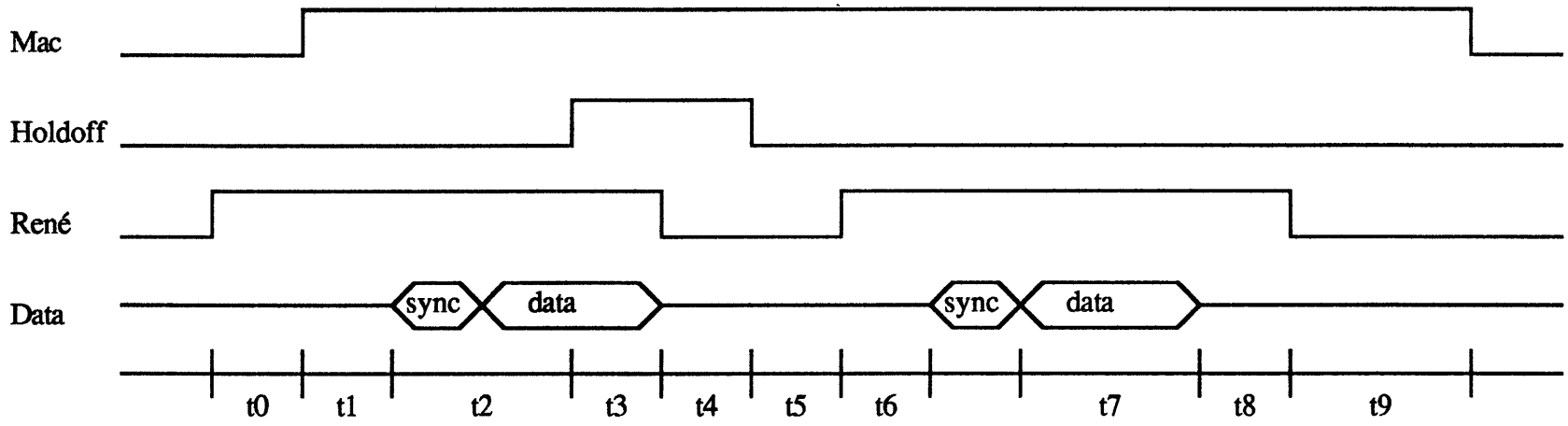


The identity block need be only 288 bytes long, but a total of 532 bytes are still sent. The manufacturer may place any other information in the last 244 bytes. Assuming a truly "packed" structure, the ID block can be defined as follows:

```
CONST
{ Device Characteristics }
        Mountable           = $80;      Write_protected     = $08;
        Readable            = $40;      Icon_Included       = $04;
        Writable            = $20;      Disk_In_Place       = $02;
        Ejectable           = $10;

TYPE
        byte                = 0..$FF;
        threebytes          = 0..$FFFFFF;
ID_Block = packed record
        Device_Type:        word;                           { Device type }
        Device_Manuf:       word;                           { Device Manufacturer, Apple = 1}
        Device_Character:   byte;                           { Characteristics of device }
        Num_Blocks:         threebytes;                     { Number of blocks on device }
        Num_Spares:         word;                           { Number of spare blocks left }
        Num_BadBlocks:      word;                           { Number of bad blocks currently }
        Manuf_Reserved:     packed array [0.51] of byte;    { Reserved for manufacturer info}
        Icon:               packed array [0.255] of byte;   { Icon + mask for device }
        Filler:             packed array [0..191] of byte;  { Filler bytes to make it 512 bytes }
        end;
```

Data Transmission From Drive to Mac



t0 - René will wait forever for Mac to respond to handshake.

t1 - René will send sync within 33us.

t2 - Mac may assert holdoff anywhere in a group. That group will be ignored and will not be included in the checksum.

t3 - René will acknowledge the holdoff immediately after the last byte of the group is sent

t4 - Must be a minimum of 100us. After that, René will wait forever for holdoff to de-assert.

t5 - René will respond to de-assertion of holdoff within 18us after t4.

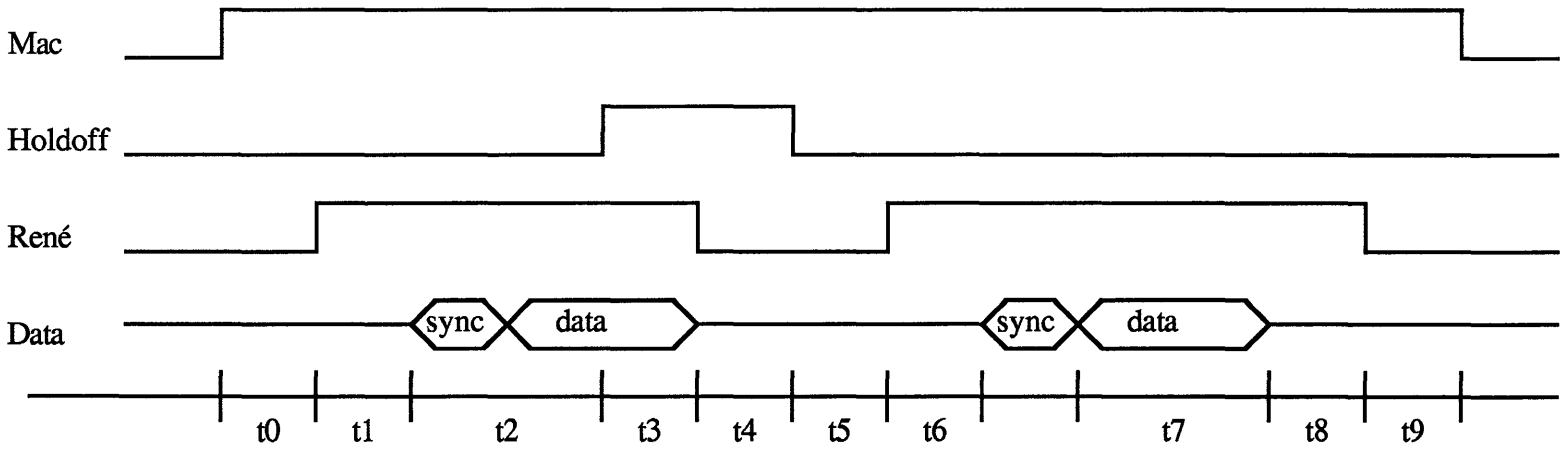
t6 - Transmission starts with group that was interrupted by holdoff within 34us.

t7 - Data transmission time is dependent on the number of groups sent (8 bytes * byte time * number of groups).

t8 - René signals end of transmission within 3us of last byte loaded into IWM. Mac will received it one byte time later.

t9 - René will wait forever for Mac to de-assert.

Data Transmission From Mac to Drive



t0 - René normally responds to Mac in 14us, but may take as long as 2 seconds if doing self test.

t1 - René will wait forever for a valid sync byte.

t2 - Mac may assert holdoff anywhere in a group. That group will be ignored and will not be included in the checksum.

t3 - René will acknowledge the holdoff immediately after the last byte of the group is received.

t4 - Must be a minimum of 70us. After that, René will wait forever for holdoff to de-assert.

t5 - René will respond to de-assertion of holdoff within 14us after t4.

t6 - Same as t1. Transmission starts with group that was interrupted by holdoff.

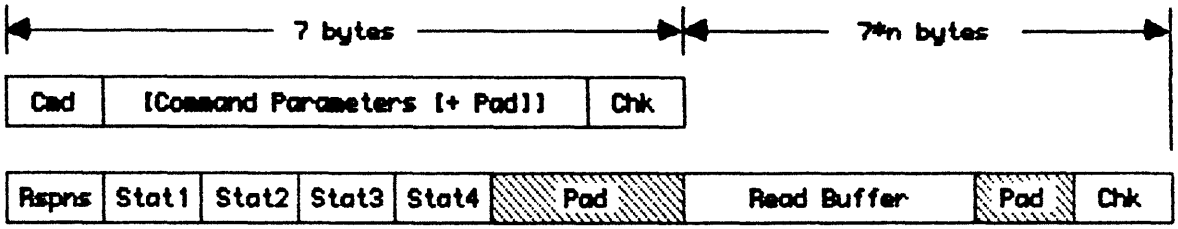
t7 - Data transmission time is dependent on the number of groups sent (8 bytes * byte time * number of groups).

t8 - René acknowledges end of transmission within 3us of last byte received.

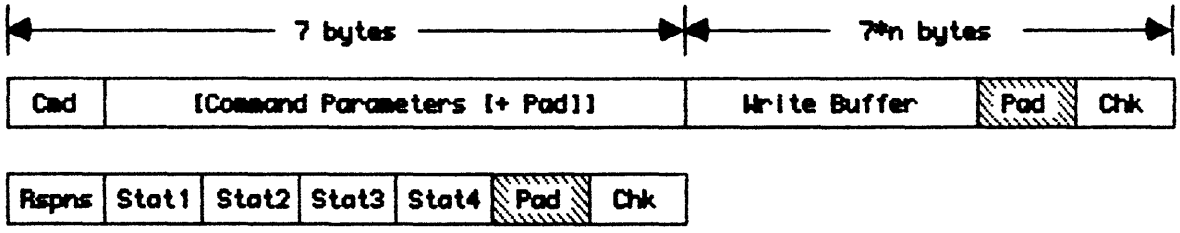
t9 - René will wait forever for Mac to de-assert.

Rene Command Format

Read Commands have the format:

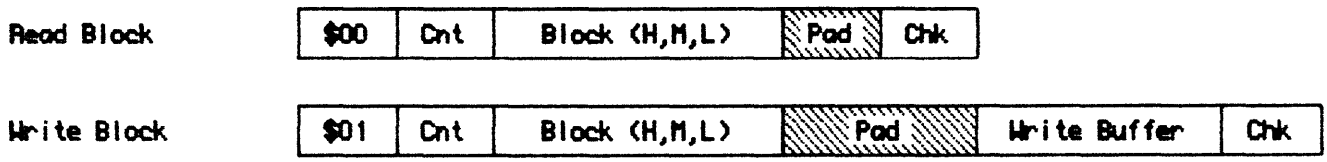


Write Commands have the format:

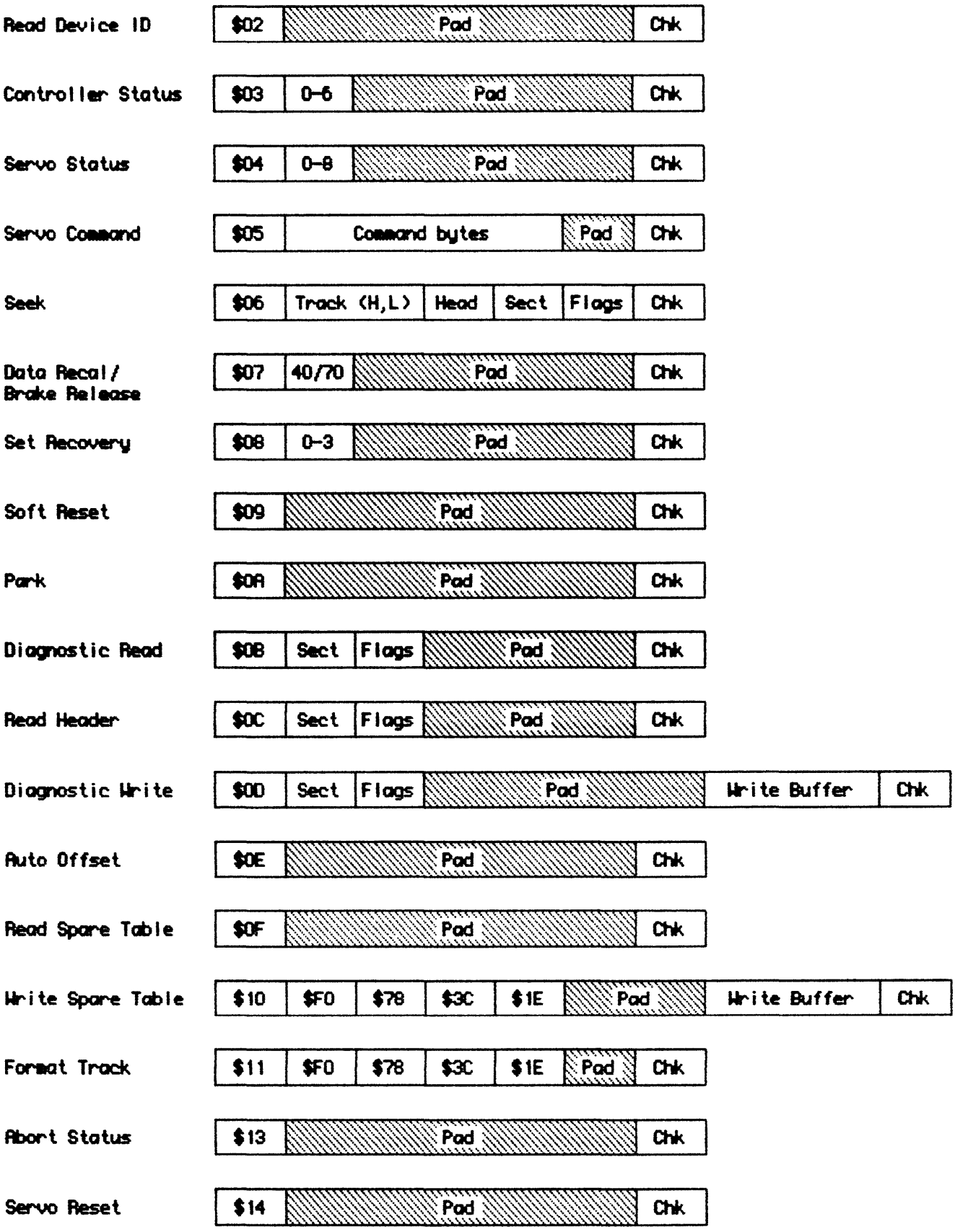


Cmd — a unique number for each command Rene can do.
Pad — 0-6 bytes used to pad a block out to a multiple of 7 bytes.
Chk — a checksum on the entire block of data being transferred.
Rspns — command response byte=Cmd+$80.
Statx — standard status bytes returned by almost all commands.

System Commands:



Diagnostic Commands:



Diagnostic Commands (new stuff):