The TI floppy disk controller card

This card was originally developped by Texas Instruments to control upto three single-density, double-sided drives. One of the drives was meant to be installed inside the PE-box, the other two being external, with their own power supply. Several other controller cards were developped later on, to allow for use of double-density disks or even quad-density. This page will concentrate on the original TI card, but I will also discuss the FD179x controller used in double-density cards.

The TI controller card consists in: the FD1771 FDC, a 8Kb ROM containing the DSRs and subprograms, and logic circuits to access each of these. The ROM maps at >4000-5FEF, and the FDC registers at >5FF0-5FFF. There is also quite a bit of electronics to deal with the drive interface.

Here is an annotated picture of the card.

Theory of operation

Disk physical organisation

Magnetic media

Floppy disks are made of a thin layer of ferromagnetic medium coating a plastic film. Contrarily to paramagnetic or diamagnetic media, a ferromagnetic element (or alloy, e.g. iron, cobalt, nickel, $Fe_{65}Co_{35}$, MnBi, etc) has the property to amplify and possibly to "remember" an externally applied magnetic field. This is due to the number of electrons in the outside layer of ferromagnetic atoms: each atom behaves like a tiny magnet. Since magnets attract/repell each other, an atom can recruit its neighbours and force them to point in the same direction than itself. These atoms can in turn recruit their neighbours, and the process goes on... until it clashes with another cluster of atoms, oriented differently and too large to be recruited. A ferromagnetic medium is thus made of millions of tiny domains, into which the atoms are oriented in the same direction. Each domain acts as a little magnet, but since they are randomly oriented the gobal result is magnetically "neutral".

However, if an external magnetic field is applied to a ferromagnetic medium, it can force some of the domains to align themselves in the direction of the field. The easier domains to recruit will of course be those whose orientation is already close to that of the field. But if the field is strong enough, and if it lasts long enough, it will eventually recruit all domains. When the field is interrupted the domains remain frozen as they were, which means that the medium is not globally neutral any more: it now behaves like a magnet. The intensity of the remaining field with respect to the field that created it is known as *remanence*, and depends on the element/alloy used.

There are two ways to reverse this magnetization: the first one is to apply an external magnetic field with the opposite orientation. The intensity needed to cancel a magnetization is known as *coercivity*. Again it depends on the type of atom involved: as you may suspect, elements that have a high remanence also have a high coercivity, they are known as magnetically hard elements. The other way to cancel a magnetisation is to warm up the medium above its "Curie point": the temperature at which the molecular motion becomes strong enough so that atoms can overcome their dependance to their neighbours and freely change their orientation (this property is used in magneto-optic disks).

With floppy disks, a magnetic head generates a magnetic field strong enough to magnetize a small area in the ferromagnetic layer of the disk. Originally some drives used to orient these "magnets" vertically (i.e whithin the thickness of the layer itself), but nowaday all drives arrange the magnetic domains horizontally, along the tracks. When the disk moves, these magnetized area generate a current in the reading head and can therefore be detected. Note however that only *changes* in magnetic fields generate current, thus if a disk does not spin it can still be written (at the bit currently under the head), but nothing can be read back.

Not all floppy disks are born equal however: high-density disk (HD) are made of a medium with higher remanence and coercivity than single- or double-density disks. This allows to pack bits closer to each other,

but requires a writing head capable of generating stonger magnetic fields. HD drives can do that, and they can also reduce their write current to handle SD or DD diks. However, SD or DD drives cannot handle HD disks: their read/write head just can't generate enough magnetic field to overcome the coercivity of the medium.

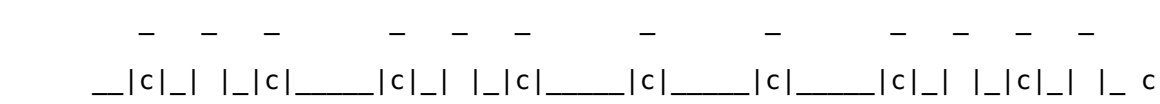
A big problem with magnets is that they attract and repell each other. This is also true for the small magnetic areas used to encode bits! With certain bit patterns the attraction/repulsions of several successive bits combine is such a way that it can result in "bit shifting", i.e. a bit moves slightly ahead, or behind the positions it should occupy on a track. This is especially marked towards the center of the disk, since the inner tracks are shorter but must hold the same amount of information than the outer tracks: the distance between bits is thus smaller and the attraction they apply to each other is stronger.

To overcome that phenomenon, that could give rise to reading problems, some disk controllers use a special trick known as "write precompensation": depending on the bit pattern, critical bits can be written slighlty earlier or slightly later than they should. When shifting occurs, these bits end up precisely where they are supposed to be. Nifty, isn't it?

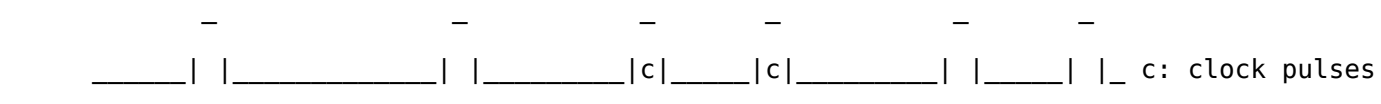
Data encoding

Data can be stored on magnetic media using various encoding schemes. The original one was FM (frequency modulation), wich was later modified into MFM (modified frequency modulation). Nowadays, more sophisticated methods are used (such as RLL: run length limited), but the TI controller cards don't know about them.

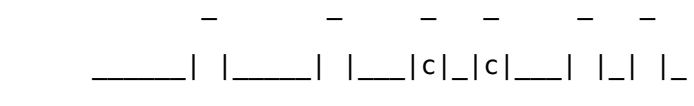
FM
In FM, clock bits are recorded at regular intervals. In between two clock pulses, there can be a data pulse or not. If there is a pulse, it encodes a "1" bit, else it encodes a "0". This is how a byte (>A3: 10100011) is encoded using FM:



MFM
Things are a tad more complicated with MFM: the data bits are encoded as above, with a pulse for "1" and no pulse for "0". But clock bits are only issued in between two "0" data bits. Here is how the same byte is encoded with MFM:



As you can see, this method leaves more room between pulses, which means that we can compress the above by a factor of two and still have the same minimum interval between pulses than with FM:

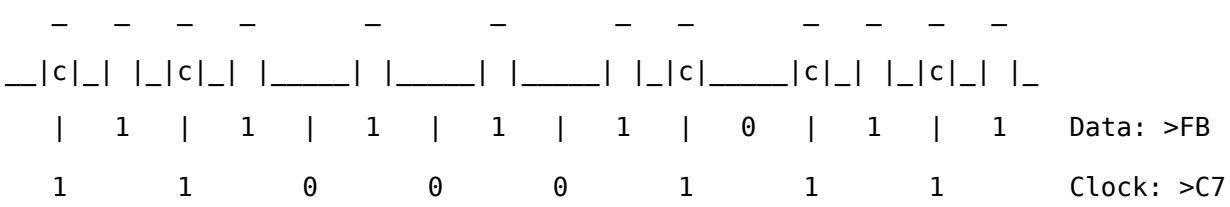


Marks
The controller is also able to record some bytes in a special way. These bytes will be used as synchronisation marks, to define the beginning or the end of a data area for instance.

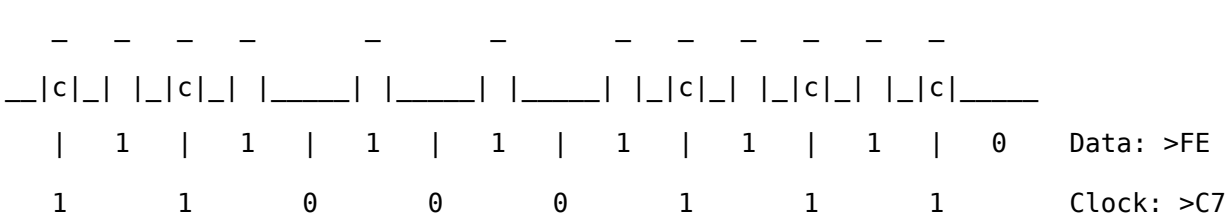
The way it is done in FM is by omitting two or three clock pulses in the middle of the byte. We saw above that there are normally 8 clock bits per byte. If we represent each clock bit with a "1", the normal clock pattern is thus 11111111, or >FF. This clock pattern is interlaced with the data bits for synchronisation purposes. The FD179x can encode marks with a >C7 (11000111) or a >D7 (11010111) clock pattern. To prevent the controller from completely loosing sync, the bytes encoded with such a clock pattern should contain a lot of 1s: the FD179x uses >F8-FB or >FE with a >C7 clock pattern, and >FC with a >D7 clock pattern.

In the TI-99/4A disk format described below, a >FE mark signals the ID block in a sector, whereas a >FB mark signals the data block. A >F8 indicates a deleted data mark. Finally, the IBM format uses >FC at the beginning of the track, to mark the location of the index hole on the track. The unique >D7 clock pattern of the >FC mark tells the CRC logic not to reset the CRC count, as it normally does with sector marks. Track marks are not part of the TI-99/4A format.

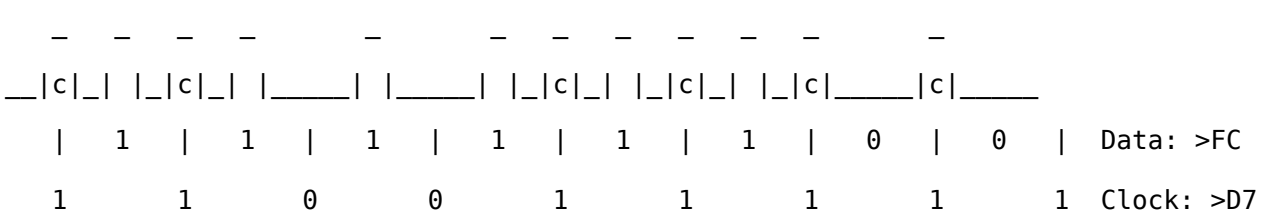
Data mark:



ID mark:

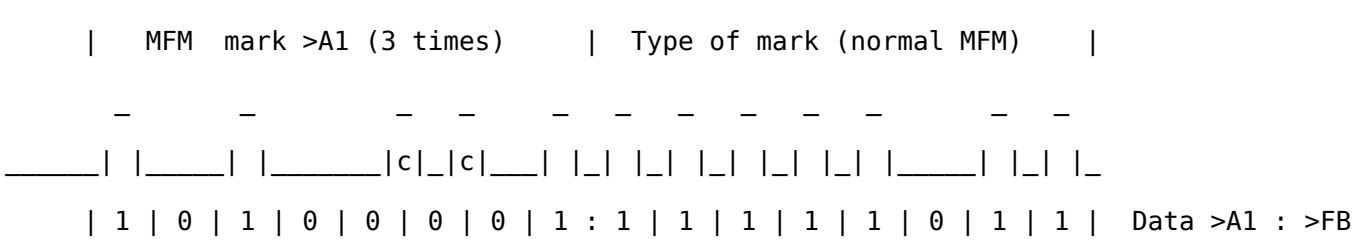


Track mark:

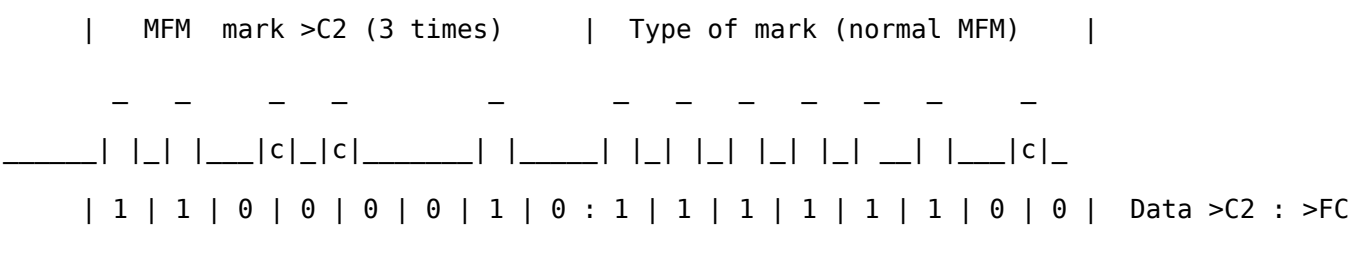


In MFM, marks are generated by omiting a clock transition between bits 4 and 5, or between bits 3 and 4 of the data byte. Obviously, these data bits must be 0s, otherwise no clock bit would be expected there. The two MFM marks are >A1 (missing a clock bit between bits 4 and 5) for sector marks, and >C2 (missing a clock bit between bits 3 and 4) for track marks. An MFM mark is repeted 3 times and followed by a normally encoded byte, with the value of the corresponding FM mark (>FB, >FE, >FC, etc). Byte >FC comes after three >C2 track marks, all other bytes after three >A1 sector marks.

Sector marks



Track mark



Tracks and sectors

A disk is divided into a number of concentric circles called tracks. Each track is in turn divided into several data segments called sectors. Finally, most floppy disks can be written on both sides.

The number of sides per disk and the number of tracks per side depend on the hardware: you need two reading heads to access both sides of a disk. And the number and positions of the tracks depends on the mechanics that move the reading head in the drive. Typically, TI-drives have 35 or 40 tracks per side. IBM drives can have upto 76 tracks.

On the other hand, the number and the size of the sectors do not depend on the hardware. It is said that floppies are "soft sectored" (hard-sectored disks have an index hole for each sector, soft-sectored disks have only one hole). In the TI single-density format, there are 9 sectors per track, each 256 bytes in length. Double-density format has 18 sectors per track. But this is no obligation: you could elect to have only one big sector per track : this will result in faster data transfer, but may cause a tremendous waste of space if your file is just 1 byte larger than a sector.

Each sector is composed of two parts: a sector ID block and a data block. The sector ID block contains the track number, the side, the sector number and a code for the sector size. When you access a sector, the FDC checks each ID block until it finds one that with the right sector, track and side numbers. If it does not find one after 5 disk revolutions, it returns a "record not found" error. Not surprisingly, the data block contains data, to be transfered to or from the CPU. Each block ends with a CRC byte (cyclic redundancy check) which is a way for the FDC to ensure that the information on the disk has not been corrupted. Special marks (recorded in a special way) are used to define the begining of each block after a stretch of zeros intended to allow for synchronisation.

A track begins shortly after the index hole has been detected by the controller: there are a few >FF filler bytes before the first sector. Similarly, any unused space after the last sector is filled by >FF bytes. There are also filler bytes between the ID and data blocks.

In summary, here is how a single-density track looks like in the TI format. The part in bold type is rewritten during sector write operations. The number of bytes preceded by a ~ may vary when a sector is rewritten, due to slight variations in the speed of the drive motor (especially, if the sector was written with another drive than the one used to initialize the disk).

	Name	Bytes	Value	Description
	Index gap	12	>FF	Index mark filler
x9	ID sync	6	>00	Synchronizes the controller for ID mark
	ID mark	1	>FE	Recorded in a special way (clock >C7)
	Track	1	>00-FF	Track number (normally 0-39)
	Side	1	>00-01	>00=side A, >01=side B
	Sector	1	>00-FF	Sector number (normally 0-8)
	Length	1	>01-04	Sector size (normally >01: 256 bytes)
	CRC	2	crc	Cyclic redundance check

	Separator	~11	>FF	Prevents "Write sector" from erasing the ID block
	Data sync	6	>00	Synchronizes the controller for data mark
	Data mark	1	>FB	Recorded in a special way (clock >C7)
	Data	256	data	This is the data contained in the sector
	CRC	2	crc	Cyclic redundancy check
	Separator	~36	>FF	Prevents "Write sector" from erasing next ID block
	End filler	~240	>FF	Varies with motor speed (+/- 32 bytes)

Total data bytes: 9 x 256 = 2304 bytes
Total sector bytes: 9 x 325 = 2925 bytes
Total per track: about 3177 bytes

And here is a double-density track.

	Name	Bytes	Value	Description
	Index gap	32	>4E	Index mark filler
x18	ID sync	12	>00	Synchronizes the controller for ID mark
	MFM mark	3	>A1	Recorded in a special way (missing clock 4-5)
	ID mark	1	>FE	Recorded in normal MFM
	Track	1	>00-FF	Track number (normally 0-39)
	Side	1	>00-01	>00=side A, >01=side B
	Sector	1	>00-FF	Sector number (normally 0-17)
	Length	1	>01	Sector size (normally >01: 256 bytes)
	CRC	2	crc	Cyclic redundance check
	Separator	~22	>4E	Prevents "Write sector" from erasing the ID block
	Data sync	12	>00	Synchronizes the controller for data mark
	MFM mark	3	>A1	Recorded in a special way (missing clock 4-5)
	Data mark	1	>FB	Recorded in normal MFM
	Data	256	data	This is the data contained in the sector
	CRC	2	crc	Cyclic redundancy check
	Separator	~28	>4E	Prevents "Write sector" from erasing next ID block
	End filler	~190	>FF	Varies with motor speed: (+/- 32 bytes)

Total data bytes: 18 x 256 = 4608 bytes
Total sector bytes: 18 x 346 = 6228 bytes
Total per track: about 6450 bytes

Cyclic Redundancy Check

Cyclic redundancy check (CRC) is a very powerfull method to detect a data alteration the disk, or a transmission error. Unfortunately, it's not as simple to explain as a checksum, but I'll do my best.

We saw above that the data block of a sector is normally 2048-bit long. Traditionnally, we think of these bits as 256 bytes of 8 bits, but nothing prevents us to consider them as 128 words of 16 bits or even as a single 2048-bit number. The lattest provides us with a nice way to verify data integrity: let's just divide this huge number by an arbitrary value and store the result (the quotient or the remainder) toghether with the data. The problem with the quotient is that it may well be a very large number in itself and may require many bits of storage space. The remainder, on the other hand, is by definition smaller than the divisor by at least 1 bit.

Therefore, if we were to use a 17-bit divisor, we would be sure to get a 16-bit remainder that can conveniently be stored as two extra bytes, toghether with the sector data. It's not completely impossible that an alteration of the data will result in a number that provides the same remainder, but it's very unlikely: for a 16-bit remainder (17-bit divisor) the odds are $1/2^{16}$, i.e. one chance in 65536.

The only problem with this technique is that long divisions are difficult to perform for electronic circuits. For neuronal circuits also: just try to divide 12,146,753,456 by 1,247. You have five seconds. Answer.....now!

One way to make it easier is to split the operation into a series of partial divisions, and to combine them with shifts and substratctions. This is the traditional way we learn at school:

```
153 / 12 =           Divide 153 by 12

153                  First shift 12 all the way to the left

12                   Now, how many times 12 in 15? Answer=1

153 / 12 =  1        Write down partial result

12                   Get the remainder (i.e. substract 12 from 15)

 33                  Bring down the next digit

153 / 12 = 1

12

 33                  Shift the divisor (12) by 1 position to the right

 12                  Now, how many times 12 in 33? Answer=2 (remainder=9)

153 /12  = 12        Write down the result: this is the quotient

12

 33

 12

  9                  Calculate the remainder (substract 2*12 from 33)
```

We could to the same with binary numbers. It would even be easier since there are only two posible answers to the question "how many times X in Y?", these are 0 and 1. Thus we can replace partial divisions with comparisons: "is X greater than Y?", which is much easier to perform for electronic circuits. What's less easy to obtain is the remainder since substactions (and additions for that matter) require moderately complicated circuits, due to the need to borrow (or carry) bits from the next position on the left. Things would be easier if we could replace substractions with a bitwise operation such as AND, OR or XOR.

But wait, nothing prevents us from doing it! The result may not be very meaningfull, but as long as it retains the same properties than a division that's all we need. CRC codes are generated with such "pseudo-divisions", that use an XOR operation instead of a substraction. Just to refresh you memory: a XOR operation on two bits results in 1 when one or the other bit is 1 (but not both). The truth table is:

```
0 xor 0 = 0

0 xor 1 = 1

1 xor 0 = 1

1 xor 1 = 0
```

Now let's "divide" 111001 by 1101 (both being binary values: 57 and 11 in decimal) using our XOR-division scheme:

```
111001 % 1101 =          Pseudo-divide (%) 111001 by 1101
1101                     Is 1110 greater than 1011? Answer=1 (yes)
111001 % 1011 = 1        Write down the partial result (i.e. 1)
1101                     Calculate the "remainder"...
0011                     ...but use XOR instead of substract
111001 % 1011 = 1
1101
00110                    Bring down the next digit
 1011                    Shift the divisor. Is 0110 greater than 1011? No.
111001 % 1011 = 10       Write down the partial result (i.e. 0)
1101
001101                   Bring down the next digit
  1011                   Shift the divisor. Is 1101 greater than 1011? Yes.
111001 % 1011 = 101      Write down the result. This is the pseudo-quotient.
1101
001101
  1011                   Perform another XOR (ignore leftmost digits)
  0110                   And this is the final pseudo-remainder.
```

As you see, this kind of operation is fairly easy to perform for a human brain, and even more so for an electronic circuit. The final "remainder" is our CRC value. Just like the remainder of a division, it is not unique to a given dividend, but the probability that an altered dividend gives the same CRC than the original one is very low. More precisely it is $1/2^n$, where n is the number of bits in the CRC value.

This means that our 16-bit CRC scheme will pick up 65535 errors out of 65536, or 99.9985 %. Not bad, eh? But we can make it even better if we consider the way errors appear on a disk. They can be due to spontaneous demagnetisation, in which cas only one or two bits at a time will be affected, unless you let a disk sit for 10 years. Or a precise spot on the disk may be damaged and the faulty bits will be grouped toghether (at the place you put your greasy fingers on the magnetic medium). So is often the case with serial transmissions: a burst of statics scrambles a whole bunch of bits. It turns out that some CRC "divisors" (the official name is "generator") are better than other at detecting such burst errors. For instance, the widely-used CCITT generator 1,0001,0000,0010,0001 results in the following impressive performances:

Type of error	Detection rate
Single bit errors	100%
Two bits errors	100%
Any even number of faulty bits	100%
Burst errors of 16 bits or less	100%
Burst errors of 17 bits	99.9969%
All other burst errors	99.9984%

This happens to be the CRC generator used by the FD197x controller.

By the way, did you notice that it's not very convenient to "spell out" a generator, even if we use comma to group the bits 4 by 4? Therefore, the traditional way of writing down a CRC generator is to give its

polynomial, i.e. which of the bits are set to 1. In the case of the CCITT generator, the polynomial is $x^{16}+x^{12}+x^5+1$, i.e. bits 16, 12, 5 and 0 (x^0 =1) are set to 1. For obvious reasons the leftmost bit is always 1 in CRC generators (otherwise we would have 15-bit CRC values and lower error detection rates). For reasons that are much less obvious to me, the rightmost bit should also be 1.

One last detail. You may think that when reading data from the disk, the controller calculates the CRC and compares it with the CRC value written on disk. You're close, but no cigar. You see, this would require a special comparison circuit. On the other hand, one can check if two numbers are identical by XORing them: A xor B is always 0 if, and only if, A=B. It is thus possible to use the existing XOR logic in the CRC circuitery to perform the check. This is done as follow:

When writing a sector, the FDC considers it as a 2064-bit number, made of 256 8-bit bytes plus a trailing >0000 16-bit word. Upon writing, it replaces the final >0000 with the CRC value. When reading, it just calculate the CRC of the 2064 bits: the result should be 0 since the CRC value will be XORed with itself.

This may remind you of the method used to check parity in serial transmission chips. No wonder: the parity bit is nothing else than a 1-bit CRC. Odd parity uses a 10 generator, and even parity a 11 generator (i.e. the polynomial is x+1). Of course, the error detection rate is fairly low since $1/2^1$ is only 50%...

Sector interleaving

A word about numbering. On each side tracks are numbered from 0 to 39 starting from the outside of the disk. On each track, sectors are numberd from 0 to 8 (or to 17 for DD disks). When you try to a sector, the software determines on which track it is, and what sector number is will have on this track. Note that sector numbers grow inwards on side A, but outwards on side B (even though track numbers grow inwards on both sides).

However, this numbering scheme is only a software convention, you are allowed to use another numbering scheme provided your software knows how to deal with it. For instance, some protection programs assign random numbers to their sectors (41, 12, 88, 26, etc). The loading software knows which sector to call on which track, but plain-vanilla disk management softwares are completely lost and cannot copy such a disk (see goofy formats, below).

Even though sectors are numbered from 0 to 8, they need not to be arranged in this order on the track. In fact, it is generally better that they are not! This is because it will take some time for the software to deal with a sector before trying to access the next one. In the mean time, the disk will have spun and may have skipped several sectors. To optimize access speed, it is thus advisable to figure out an interleaving (a.k.a. interlacing) scheme designed in such a way that reading head arrives on the next sector just when the software is ready to access it.

For instance, assume that the disk spins by three sectors before the software is ready to proceed. The ideal sequence of sectors would be something like this: 0 x x x 1 x x x 2 x x x 3 x x x 4 x x x 5 x x x 6 x x x 7 x x x 8. Since tracks are circular, this sequence results in the following interleave: 0 7 5 3 1 8 6 4 2. You can verify that consecutive sectors are always 4 sectors apart on the disk (this happens to be the best interlace pattern for the TI disk controller card).

Disk logical organisation

Volume information block

The first two sectors on the disk are reserved for important information about the disk and its content. One uses the first sectors because they are on the outside of the disk, where tracks are physically longer than in the center. Since floppies rotate at a constant speed (unlike CDs), there will be more space to write the same amount of information on track 0 than on track 39. As a result, the probability of data corruption is less on the outer tracks.

Sector 0

This sector is called the VIB (volume information block) and contains various information about the disk: name, size, protection, and most importantly a sector bitmap.

This bitmap is used by the software to determine whether a sector is free or used. Each sector is represented by one bit (starting with the least significant bit of each byte): when the bit is 0, the sector is free and can be used to create a new file or appended to an existing file. When the bit is 1, the sector is either used or damaged. There are 200 bytes available in the bitmap, which is enough to map 1600 sectors. That's more than enough, since even DS/DD disks have only 1440 sectors.

```
Exemple:

Byte  Sectors

>38:  7  6  5  4  3  2  1  0

>39: 15 14 13 12 11 10  9  8
```

Content of sector 0:

Bytes	Contents	Typical values
>00-09	Disk name	"DISKNAME01"
>0A-0B	# of sectors	SS/SD: >168 DS/SD: >2D0 SS/DD: >2D0 DS/DD: >5A9
>0C	Sectors/track	SD: >09 DD: >12
>0D-0F	DSR mark	"DSK"
>10	Protection	Unprot: " " Protected: "P"
>11	Tracks/side	>23 / >28
>12	Sides	>01 / >02
>13	Density	SS: >01 DS: >02
>14-37	(reserved)	>00
>38-EB	Bitmap	SS/SD: >38-64 DS/SD and SS/DD: >38-91 DS/DD: >38-EB
>EC-FF	(reserved)	Must be >FF

Some reserved bytes are sometimes used by disk managers to store extra information about the disk: for instance, DISKU by the late John Birdwell stores an 8-char date string into bytes >20 to >27.

Sector 1

This sector contains a list of pointers (i.e. sector numbers) to file descriptor records, sorted in alphabetical order. Each pointer is 2 bytes long (since there are more that 256 sectors on a disk) and the list ends with >0000, which allows us to list 127 files per disk.

Interesting things happen when you play around with that sector:

- If you shuffle the pointers out of order: the directory can still be listed, but accessing a file by name generally result in a "file not found" error (although this depends on the disk manager you are using).
- Inserting a >0000 at the top of the list has the opposite effect: files are not listed in the directory, but can still be accessed by name (as the search algorithm begins in the middle of the list).
- Duplicating a pointer lists that file twice in the directory.

File descriptor record

Each file consists of at least one sector, even when empty. This sector is known as the File Descriptor Record (FDR) and contains various informations about the file: its name, its type, its size, etc. It also contains a list of the sectors where the file data is to be found. The list does not enumerate each sector as this would drastically limitate the maximum file size. Rather it lists clusters, i.e. chunks of consecutive sectors belonging to the file.

Ideally, a file should consist in only one big cluster. However, as other files are written on the disk, it may be that the next sector is not available when it is time to increase the file size. It is thus necessary to start a new cluster, in a free area of the disk: the file is now fractured. Disks that contain a lot of files that have often been modified may end-up in a awfully fractured way, which results in decreasing the access speed (as the reading head must move from one cluster to the next) and may impose a limit on the file size (as only 76 clusters can be defined in the FDR). Therefore, most disk managers will rearrange the files in single clusters when copying a disk.

The FDR dedicates three bytes per cluster, that are used to define the sector number where the cluster begins and the total file size (in sectors, minus one) reached with this cluster. As both numbers may be bigger than 256, each is encoded using one byte and a half (3 nibbles), as follow: UM SN OF Where N U M are the three nibbles forming the sector number (in this order) and O F S are the three nibbles forming the total file offset (counting from zero).

Exemple:
A file consists in 7 sectors: sectors >02E and >02EF and sectors >192-196 (5 more sectors).
This makes two clusters: the first starts at sector >02E and ends with an offset of >001, the second starts at sector >192 and ends with a total of >006 sectors.
The cluster list would thus look like this: >2E >10 >00 >92 >61 >00.

Note: the DISKU disk manager uses the end of the FDR to store a user defined comment about the content of the file, in bytes 220 to 254. I suppose John Birdwell assumed no file will be so hopelessly fractured as to require more that 64 clusters...

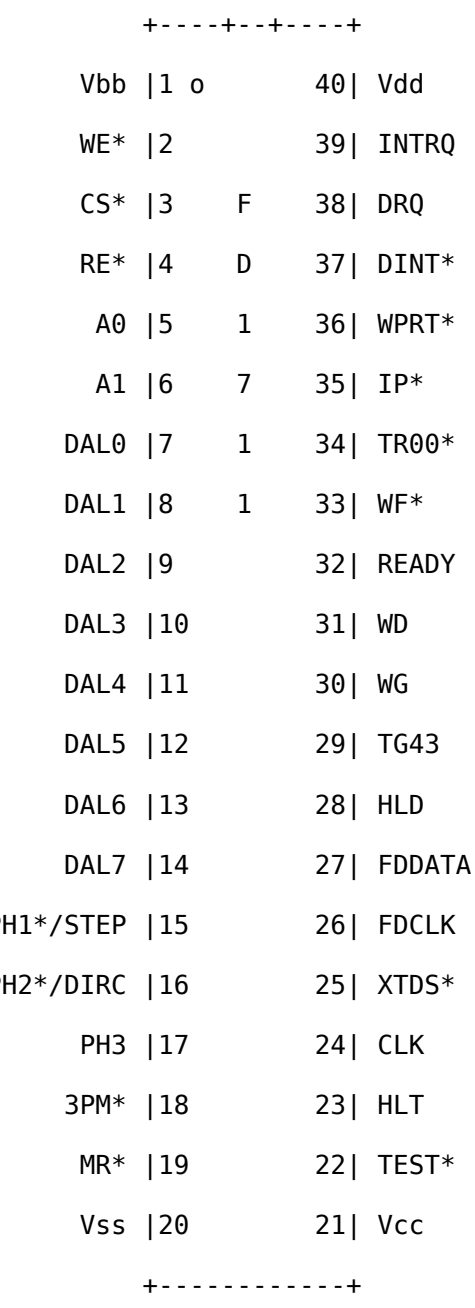
Bytes	Contents	Comments
>00-09	File name	"MYFILE01"
>0A-0B	(reserved)	>00
>0C	File type	>80: variable >08: write protected >02: internal >01: program
>0D	Records/sector	>00 for program files
>0E-0F	# of sectors in file	Not counting FDR
>10	Last byte in last sector	>00 for fixed files
>11	Record length	>00 for program files
>12-13	Fixed: number of records Var: number of sectors Program: >00	! Bytes are swapped !
>14-1B	(reserved)	>00
>1C-FF	Cluster list	>UM >SN >OF == >NUM >OFS

The FD1771 Floppy disk controller

The Floppy Disk Controller (FDC) is the microprocessor that carries out all disk operations: it starts and stops the motor, moves the reading head to the required track, searches for the specified sector, and reads or writes it one bit at a time. It transfers these bits to/from the CPU one byte at a time.

The FD1771 encodes data in FM (frequency modulation) and can therefore only handle single-density disks.

Pinout



Power supply
Vdd: +12 Volts
Vcc: +5 Volts
Vbb: -5 Volts
Vss: Ground

Computer interface
MR*: Master Reset. When hold low for at least 50 usec, this input pin resets the FDC and resets the "Not-ready" status bit. When MR* becomes high again, the FDC executes a Restore (>03) command and loads

>01 into the Sector register.
CS*: Chip Select. Enables FDC access when low.
A0,A1: Address lines. Select the register to be accessed. 0=Command (write) or Status (read), 1= Track, 2=Sector, 3=Data.
DAL0-DAL7: Data Access Lines. Inverted bidirectional data bus, enabled by WE* and RE*.
WE*: Write Enable. Used by the CPU to tell the FDC that data has been sent on the data bus. Active low.
RE*: Read Enable. Used by the CPU to read cause the FDC to place data on the data bus. Active low.
CLOCK: This input pin must receive a 2 MHz +/- 1% square wave signal, with a 50% duty cycle.
DRQ: Data Request. This open collector output is used by the FDC during read operations to indicate that a byte of data is ready in the Data register (high signal). During write operations, a high level signals that the Data register is now empty and ready to receive another byte. Should be pulled up to +5V with a 10K resistor.
INTRQ: Interrupt request. This open collector output (to be pulled up with a 10K to +5V) becomes low each time the FDC has completed a command. Reset when the command register is loaded with a new command.

Disk drive interface

The FD1771 can be handle two types of drive interface: it can either command 3 stepper motors by sending successive three phase pulses on pins PH1, PH2 and PH3, or it can send stepping pulses on the PH1/STEP pin and a direction control on the PH2/DIRC pin. Pin 3PM is used to determine the type of interface used.

PH1*/STEP: If 3PM is high. the FDC sends a 4 microseconds high pulse on this output pin, causing the drive to move the reading head by one track. If 3PM is low, the FDC sends one active low signal out of three on this pin.
PH2*/DIRC: When 3PM is high, this pin determines the direction in which the head will move. Low=outwards, high=inwards. If 3PM is low, the FDC send one active low signal out of three on this pin.
PH3: When 3PM is low, the FDC sends one active low signal out of three on this pin. Not used if 3PM is high. Note that PH3 needs an external inverter.
3PM: This pin should be hardwired low to use a 3-step drive interface: pulses will be sent as PH1-PH2-PH3-PH1 to step in and as PH1-PH3-PH2-PH1 to step out. Alternatively, 3PM can be wired high for a more sophisticated interface: a single stepping pulse will be sent on STEP. whith DIRC indicating the direction.
TG43: Track greater than 43. This output pin tells the drive that the head is positioned beyond track 43 (some drives use it to compensate for the higher density of data bits on these tracks). Valid only during read and write commands.
HLT: Head Loading Time. When high, the magnetic head is assumed to be engaged. This input is typically driven by a one-shot timer triggered by HLD but could also be connected to +5V if the internal delay is sufficient (command bit "E": the HLT line will only be sampled 10 ms after HLD went high).
HLD: Head load. Tells the drive to load the read/write head on the magnetic medium. Typically used to trigger a one-shot that fires back to HLT.

The FD1771 can either receive a mixed signal from the drive, i.e clock and data bits together, or receive the clock bits on pin FDCLK and the data bits on pin FDDATA. Pin XTDS is hardwired to determine the mode to be used.

FDCLK: When XTDS* is low, this pin receives the clock bits from the drive, processed by an external separator. If XTDS* is low, this pin should be tied to a logic high.
FDDATA: Data input from the drive. When XTDS* is low, this pin receives externally separated data bits from the drive. If XTDS* is high, this pin receives raw data from the drive and an internal separator is used.
XTDS*: This pin should be hardwired low if an external data separator is used. It should be tied high or left open to make use of the internal data separator.
WG: Write Gate. Becomes active before write operations.
WD: Write Data. This output pin sends 500 ns pulses to the drive for each flux transition. This signal always incorporates data and clock bits (including marks), no matter the status of XTDS*.

READY: Input pin used by the drive to tell the FDC that it is ready for the next operation.
WF*: Write fault. The drive can bring this pin low to indicate wrinting faults. If WG was high, the current write command is terminated and the "write fault" status bit is set. This line should remain inactive (high) when WG is low.
TR00*: Track 0. The drive uses this pin to inform the FDC that the head is on track 0 (to the outside of the

disk). Active low..
IP*: Index pulse. Used by the drive to signal index holes. Low=hole.
WPRT*. Write protected. Used by the drive to signal a write protection. A low level aborts any write command and sets the "write protected" status bit.
DINT*: Disk Init. Used to prevent disk formating: the FD1771 samples this input when a Write Track command is received. If the pin is low, the operation is terminated and the "write protected" status bit is set.

TEST*: Ouput pin for test purposes. Should be left open or pulled up to +5V.

The FD179x-02 Floppy disk controller

The Floppy Disk Controller (FDC) is the new version of the FD1771. The main improvement is that it can handle double-density disks, by performing MFM encoding.

There are six, slightly different FD179x models. The FD1791, FD1792 and FD1795 have an inverted data bus. The FD1791, FD1793, FD1795 and FD1797 can deal with double-density. The FD1795 and FD1797 have a side selection output.

Pinout

```
       +----+--+----+
    nc |1 o       40| Vdd
   WE* |2         39| INTRQ
   CS* |3    F    38| DRQ
   RE* |4    D    37| DDEN*
    A0 |5    1    36| WPRT*
    A1 |6    7    35| IP*
  DAL0 |7    9    34| TR00
  DAL1 |8    x    33| WF*/VFOE*
  DAL2 |9    -    32| READY
  DAL3 |10   0    31| WD
  DAL4 |11   2    30| WG
  DAL5 |12        29| TG43
  DAL6 |13        28| HLD
  DAL7 |14        27| RAWREAD*
  STEP |15        26| RCLK
  DIRC |16        25| RG/SSO
 EARLY |17        24| CLK
  LATE |18        23| HLT
   MR* |19        22| TEST*
   Vss |20        21| Vcc
```

```
        +------------+
```

Power supply
Vdd: +12 Volts
Vcc: +5 Volts
Vss: Ground

Computer interface
MR*: Master Reset. When hold low for at least 50 usec, this input pin resets the FDC and resets the "Not-ready" status bit. When MR* becomes high again, the FDC executes a Restore (>03) command and loads >01 into the Sector register.
CS*: Chip Select. Enables FDC access when low.
A0,A1: Address lines. Select the register to be accessed. 0=Command (write) or Status (read), 1= Track, 2=Sector, 3=Data.
DAL0-DAL7: Data Access Lines. Bidirectional data bus. Inverted on the FD1791, FD1792 and FD1795.
WE*: Write Enable. Used by the CPU to tell the FDC that data has been sent on the data bus (pins DAL0-DAL7).
RE*: Read Enable. Used by the CPU to read cause the FDC to place data on the data bus.

CLOCK: This input pin must receive a 1 MHz (2 MHz for 8" floppies) square wave signal, with a 50% duty cycle.
DRQ: Data Request. This open collector output is used by the FDC during read operations to indicate that a byte of data is ready in the Data register (high signal). During write operations, a high level signals that the Data register is now empty and ready to receive another byte. Should be pulled up to +5V with a 10K resistor.
INTRQ: Interrupt request. This open collector output (to be pulled up with a 10K to +5V) becomes low each time the FDC has completed a command.

Disk drive interface
STEP: The FDC sends a pulse on this output pin, causing the drive to move the reading head by one track.
DIRC: Direction. This output pin determines in which direction the head will move. Low=outwards, high=inwards.
TG43: Track greater than 43. This output pin tells the drive that the head is positioned beyond track 43. Valid only during read and write commands.
HLT: Head Loading Time. When high, the magnetic head is assumed to be engaged. This input is typically driven by a one-shot timer triggered by HLD but could also be connected to +5V if an internal delay is used (command bit "E")..
HLD: Head load. Tells the drive to load the read/write head on the magnetic medium. Typically used to trigger a one-shot.
SSO: Side Select Ouput. Tells the drive which side of the disk to access (for FD1795/97 only).
RG: Read Gate. Used for synchronisation of external data separators (for FD1791-94 only). Goes high after two >00 bytes in SD (4 bytes of >00 or >FF in DD).
DDEN*. Double Density Enable. This input pin puts the FDC in double-density mode when low, in single-density mode when high. Must be left open on the FD1792/1794 (that cannot handle double density).
EARLY: When active (high) indicates that the write data pulse should be shifted early for write precompensation.
LATE: When active (high) indicates that the write data pulse should be shifted early for write precompensation.

RCLK: Read Clock. The drive sends a nominal square wave signal derived from the data stream on this pin. The polarity of the signal (+/-) does not matter, only phasing (i.e. transitions) does.
RAWREAD*: Data input from the drive. Active low, i.e. each recorded flux transition should send a negative pulse.
WG: Write Gate. Becomes active before write operations.
WD: Write Data. This output pin sends a 500 ns (FM) or 200 ns (MFM) pulse to the drive for each flux transition. This signal incorporates data and clock bits (including marks).

READY: Input pin used by the drive to tell the FDC that it is ready for the next operation.

WF*/VFOE*: Compined input/output pin. During write operations (when WG is high) this pin is used as input for a Write Fault signal: when active (low) it interrupts the current write command. During read operations (when WG is low) this pin becomes a VFO enable output signal.On the FD1791/93 it goes low once the head has settled (HLT is high) and remains low until the end of the data field. Additionally, on the FD1795/97 it temporarily goes high between the end of the ID CRC and 4 bytes (8 in MFM) before the data mark. This pin has an internal 100K pull-up resistor.
TR00: Track 0. The drive uses this pin to inform the FDC that the head is on track 0 (to the outside of the disk).
IP*: Index pulse. Used by the drive to signal index holes. Low=hole.
WPRT*. Write protected. Used by the drive to signal a write protection. A low level aborts any write command.

TEST*: Ouput pin for test purposes. Should be left open or pulled up to +5V. Used with voice-coil actuated steppers (i.e. linear stepper motors. But appart for some antiques, all drives use rotating stepper motors to move the read/write head).

Internal structure

The FDC contains five 1-byte registers: a write-only Command register, a read-only Status register, a Data register, a Track register and a Sector register. These registers are mapped at addresses >5FF0-5FFE in the TI controller card.

The write-only **Command register** accepts 11 different commands from the CPU. They are described below.

The meaning of the various bits in the read-only **Status register** depends on the current command. Several bits reflect the state of output pins ("Busy" for INTREQ, "DRQ pin" for DRQ, etc), which gives the designer the option to physically tie those pin to the CPU, or to read them via the Status register. See "Commands" for more details.

The **Data register** is used as a holding register during read or write operations. It communicates with an internal **shift register** that allows to serialize data to send it to the drive (on the WD pin). Conversely, during read operations, the shift registers accepts data from the drive (on the RAWREAD* or FDDATA pin), assembles it into a byte and transfers this byte to the data register.

The **Track register** contains the number of the track the magnetic read/write head is on. Valid values are 0 to 255, although I don't think any TI-99/4A-compatible drive ever went beyond 76 tracks. The FDC automatically updates the Track register when stepping the magnetic head. The content of this register is compared to the track number recorded on the disk during read, write and verify operations. The track register can also be modified directly by the CPU.

The **Sector register** holds the number of the desired sector, as loaded by the CPU. The content of this register is compared to the sector number recorded on the disk during read, write and verify operations.

In addition to these five register (and the shift register), the FDC also contains several functional units that are not directly accessible by the CPU:
An **ALU** (Arithmetic and Logic Unit) used to perform comparisons and register modifications.
An **address mark detector**, that checks the clock pulses and detects the ID and data marks.
A **CRC logic** (Cyclic Redundancy Check), used to write and verify a CRC code on disk. The polynomial is $x^{16}+x^{12}+x^5+1$, which is the standard CCITT CRC generator (see above).
And last but not least, a **timing and control circuit**, triggered by an external clock signal, that controls all drive operations.

Commands

The FDC accepts 11 different commands. Their syntax is summarized in the table below. Be aware that there are subtle differences between the FD1771 and the newer FD179x. Furthermore, there are differences between the various subtypes of FD179x.

Most commands will abort if the drive is not ready: the FDC checks for that by sampling the READY pin at the beginning of the command. The meaning of the various bits in the Status register depends on the command being executed, but one constant is the Busy bit (weight >01). No new command should be sent to the Command register as long as this bit is set (apart for the "Force interrupt" command). Once the command has been completed, the Busy bit is cleared and an interrupt is issued on the INTREQ pin.

Command	>80	>40	>20	>10	>08	>04	>02	>01
Restore	0	0	0	0	h	V	r1	r0
Seek	0	0	0	1	h	V	r1	r0
Step	0	0	1	T	h	V	r1	r0
Step-in	0	1	0	T	h	V	r1	r0
Step-out	0	1	1	T	h	V	r1	r0
Read sector	1	0	0	m	S	E	C/0	0
Write sector	1	0	1	m	S	E	C/a1	a0
Read ID	1	1	0	0	0	E'	0	0
Read track	1	1	1	0	0	E'	0	s*
Write track	1	1	1	1	0	E'	0	0
Force interrupt	1	1	0	1	I3	I2	I1	I0

r0,r1: stepping motor rate (see below)
V: verify track number flag. 1=verify, 0=don't.
h: head load flag. 0=unload head at beginning, 1=load head at beginning.
T: track update flag. 1=update track register, 0=don't.

a0: data mark flag. For FD179x: 0=data mark (>FB), 1=deleted data mark (>F8). For FD1771: combined with C to select one in 4 possible data marks: 00 = >FB, 01 = FA, 10 = >F9, 11 = >F8.
E: delay flag: 0=no delay, 1=15 msec delay before sampling the HLT pin (10 msec for FD1771).
E': same as E, but always 1 with FD1771.
C/a1: For FD1795/97: compare side number flag. 0=disable, 1=enable: set value of SSO pin (for all read/write commands). For the FD1771: always 0 for read command, combines with a0 for write commands.
S: For FD 1791/2/3: expected side, to be compared to side number on disk if C=1. For FD1771 and FD1795/97: changes the meaning of the sector size code.
m: multiple record flag. 0=read/write one sector. 1=keep reading/writing sectors on that track until Force interrupt is issued.
s*: For FD1771 only. Synchronize to address marks if 0, ignore address marks if 1. Always enabled with the FD179x.

I3: issue an interrupt now. This bit can only be reset with another "Force interrupt" command.
I2: issue an interrupt at the next index pulse.
I1: issue an interrupt at the next ready to not-ready transition of the READY pin.
I0: issue an interrupt at the next not-ready to ready transition of the READY pin.
(If I0-I3 are 0: don't issue any interrupt, but still abort the current command).

Status register

The meaning of the status bits for the different commands is summarized below:

Command	>80	>40	>20	>10	>08	>04	>02	>01

All steppings + Force interrupt	Not ready	Write protect	Head loaded	Seek error	(CRC error)	Track 0	Index pulse	Busy
Read ID	Not ready	0	0	Rec not found	CRC error	Lost data	DRQ pin	Busy
Read sector	Not ready	0 (Mark type)	Mark type	Rec not found	CRC error	Lost data	DRQ pin	Busy
Read track	Not ready	0	0	0	0	Lost data	DRQ pin	Busy
Write sector	Not ready	Write protect	Write fault	Rec not found	CRC error	Lost data	DRQ pin	Busy
Write track	Not ready	Write protect	Write fault	0	0	Lost data	DRQ pin	Busy

Not-ready: inverted copy of the READY input pin ("ORed" with MasterReset pin). A 1 indicates the drive in not ready.
Write protect: inverted copy of the WPRT* input pin. A 1 indicates the disk is write protected.
Head loaded: logical "and" of HLD and HLT pins. A 1 indicates the head is loaded and engaged.
Seek error: if 1, the desired track did not match. Reset when updated.
CRC error: if 1 the CRC found on disk did not match the calculated one. For stepping commands: CRC found in an ID field..
Track 0: inverted copy of the TR00* input pin. A 1 indicates the magnetic head is on track 0.
Index pulse: inverted copy of the IP* pin. A 1 indicates an index hole.
Busy: a 1 indicates a command in progress.
Mark type: For FD179x a 0 indicates a normal data mark (>FB), a 1 indicates a deleted data mark (>F8). For FD1771, two bits are used: 00 = >FB, 01 = >FA, 10 = >F9, 11 = >F8.
Write fault: a 1 indicates a write fault detected by the drive. Reset when updated.
Rec not found: a 1 indicates the desired track+sector+side ID combination was not found in any ID block.
Lost data: a 1 indicates that the CPU did not read/write the data register in time for the next byte to be processed.
DRQ pin: copy of the DRQ pin. For read operations, a 1 indicates that the Data register is full and must be read. For write operations, a 1 indicates that the Data register is empty and must be filled. Reset when updated.

Step

This command causes the FDC to issue a pulse on the STEP pin, which results in moving the magnetic head by one track in the currently selected direction. If the h flag is set, the head will be loaded onto the magnetic medium at the beginning of the command. After a delay specified by the r0+r1 bits, the track number is verified from the first encountered ID field, provided the V flag was set (no verification is done with the FD1795/97). Finally, if the T flag is set, the Track register will be incremented/decremented by one.

Head step times (the usual value for the TI controller card is in bold)

Clock		2 MHz	2 MHz	1 MHz	1 MHz	2 MHz	1 MHz	2MHz	1 MHz	2MHz	1MHz
DDEN*		low	high	low	high	any	any	**FD1771**	**FD1771**	**FD1771**	**FD1771**
TEST* r1 r0		high	high	high	high	low	low	high	high	low	low
0	0	3 ms	3 ms	6 ms	6 ms	184 us	368 us	6 ms	12 ms	~400 us	~600 us
0	1	6 ms	6 ms	12 ms	12 ms	190 us	380 us	6 ms	12 ms	~400 us	~600 us
1	0	10 ms	10 ms	20 ms	20 ms	198 us	396 us	**10 ms**	20 ms	~400 us	~600 us
1	1	15 ms	15 ms	30 ms	30 ms	208 us	416 us	20 ms	40 ms	~400 us	~600 us

NB The CorComp card provides a way to determine the proper head-step time by reading CRU bits 18 and 22. You can determine which card is in use by trying to bank-switch the card ROM with CRU bit 11: if the ROM changes, it's probably a CorComp card. Don't forget that it does not invert the data bus!

```
* This routine tests for the presence of the CorComp card
* It returns with Eq set if the card is not a CorComp (JNE if CorComp)
CORCOM LI   12,>1100    CRU address of the controller
       SBO  0           Turn ROM on
       SBO  11          Try bank-switching it
       CLR  0
       MOVB @>4000,0    Get the first byte (>AA in bank 1)
       SBZ  11          Back to bank 1
       CI   0,>AA00     Did the byte change?
       B    *11

* This routine selects the proper head-step time for a CorComp card
* It return with the proper bit values for commands in HSTEP
HSTIME CLR  @HSTEP
       SBZ  7
       SBZ  8
       SBO  10
       LI   0,>0100
       TB   18
       JNE  SK1
       SOC  0,@HSTEP    Copy bit 18
SK1    LI   0,>0200
       TB   22
       JNE  SK2
       SOC  0,@HSTEP    Copy bit 22
SK2    B    *11
```

Step-in

This commands sets the current direction as "inward" (pin DIRC is high), then executes a Step command.

Step-out

This commands sets the current direction as "outward" (pin DIRC is low), then executes a Step command.

Seek

The CPU uses this command to move the magnetic head on a specific track. The desired track number should be loaded in the Data register and it is assumed that the Track register contains an up-to-date track number. The latter can be a problem when using multiple drives: the current track number for a drive must be saved by the software before to operate another drive. Upon reception of the command, the FDC compares these two registers to determine the adequate direction and sets the DIRC pin accordingly. It then executes the required number of Step commands to make the Track register match the Data register.

Restore

This command seeks track 0. It sets the Track register as 255 and the Data register as 0. Then is executes a Seek command, which will result in moving the magnetic head outwards. The Restore command constantly monitors the TR00* pin: when the head reaches track 0 this pin goes low and the command is aborted. The Track register is set as 0, and the track number is verified (by reading an ID block) if the V flag was set. If TR00* does not become active after 255 steps, a Seek error is issued (provided the V flag was set) and the command terminates. This command is automatically executed after a master reset.

Read sector

This commands reads one or more sectors from disk (only the data part of the sector is transfered). The desired sector number should be placed in the Sector register.

For obvious reasons, the FDC always activated the HLD pin to load the magnetic head at the beginning of a read or write command. This operation takes some time and there are two ways to wait for its completion. Before to perform any read/write operation, the FDC waits for the HLT pin to becomes high, a feature meant for use with an external "one-shot" timer. If the "E" bit is set, the FDC first wait 15 msec (10 msec with the FD1771) before to check the HLT pin, a feature meant for use with a HLT pin permanently tied to +5V.

If the content of the Track register is greater than 43, the TG43 pin is set as high, otherwise it is set as low.

The Read sector command then checks every ID block on the current track until it encouters one that matches the required parameters. It first verifies that the track number matches the content of the Track register, then that the sector number matches the content of the Sector register, then that the side number matches the "S" bit in the Command register. If there is a mismatch the next ID block is checked. If no matching ID block (or no ID block at all) is found whithin 5 disk revolutions, as detected by the index hole pulses on the IP* pin, the command ends with the "rec not found" bit set in the Status register. If a matching ID block is found, the FDC loads the sector size and check the CRC. If the CRC does not match, the search continues.

When a matching ID field is found, the FDC expects a data mark within the next 30 bytes (43 for double-density disks). If none occured, the search procedure resumes. Once the data mark has been found, the CRC counter is reset and bytes read from disk are transfered to the Data register, via the shift register. Once a byte is ready, the DRQ pin is activated and the CPU must read the Data register before the next byte arrives (which resets the DRQ pin). If this did not happen, the " lost data" bit is set in the status register, but reading continues normally.

Once the appropriate number of bytes have been read, the CRC field is read and compared with the calculated CRC value. If they don't match, the command aborts and sets the "CRC error" bit in the Status register. Otherwise the commands terminates normally and the type of address mark is saved in the "mark type" bit of the status register.

If the m bit is set in the Command register, the FDC increments the Sector register and proceeds with reading the next sector. This goes on until either a sector cannot be found on the track, or the Force interrupt command is received.

Sector length coding.

The L flag exists only on the FD1795/97. It is assumed to be 1 on the other models. For the FD1771, the L flag selects either the IBM set of sizes (128, 256, 512 or 1024 bytes) or a non-IBM set in which the sector length field is multiplied by 16 to yield the sector length (0 stands for 256, ie. 4096 bytes). The default value for the TI-99/4A controller card is 256 bytes.

L flag in command	**Sector length in ID field**						
	0	1	2	3	...n	>FF	
0: FD179x	256	512	1024	128	-	-	
	0: FD1771	4096	16	32	48	n*16	4080
1: both	128	**256**	512	1024	-	-	

Write sector

This command is used to write data into the data block of a sector on disk. The desired sector number should be placed in the Sector register. The Write sector command looks for a matching ID field just as the Read sector command does, excepts that it also checks the WRPRT* pin. If this pin is active (low) the command aborts with the "write protected" bit set in the status register.

Once a proper ID block has been found, the FDC sets the DRQ pin to request data from the CPU, counts 11 bytes after the ID block CRC (22 bytes with DD disks) and enables the WG pin. This only occurs if data has been placed in the data register, else the command terminates and sets the "lost data" bit in the status register (a way to prevent accidental write commands). The FDC then writes six >00 bytes (12 with DD disks) and a data mark. The type of data mark is determined by the "a" bit in the Command register: 0 = standard mark (>FB), 1 = deleted mark (>F8). The older model FD1771 can use 4 different data marks: >F8 through FB.

The FDC then writes data to the disk, taking bytes from the Data register and processing them through the shift register to the WD pin, after due mixing with clock pulses. Each time a byte is transfered to the shift register, the DRQ pin is activated to tell the CPU that it can place the next byte into the Data register. This operation will reset DRQ. If it does not happen in time, a >00 byte is written on disk and the "lost data" bit is set in the Status register, but the command is not terminated. The number of bytes to be written is determined by the sector lenght byte read from the ID block.

Once a sector has been written, the contents of the CRC counter is written as two bytes, followed by a >FF byte (two in DD). The WG pin is then deactivated. Note that it is not advisable to let the FDC fill the end of a sector with zeros by just stopping to send data, as writting errors won't be detected. Instead, write the number of required >00 bytes, just like any other byte value.

The Write command will go on with the next sector if the "m" bit was set in the Command register, the Sector register being automatically incremented. This goes on until a matching ID field cannot be found, or a Force interrupt command is issued.

Read ID

This command is used to read the next ID block encountered on the track. The HLD, HLT and TG43 pins are dealt with as described in the Read Sector command. The ID block is passed to the CPU via the Data register as a string of six bytes: track number, side number, sector number, sector lenght code, and CRC (2 bytes).

The track number is then copied into the Sector register, the CRC is checked and the "CRC error" bit of the Status register set accordingly.

If no ID field is found during 5 disk revolutions, the "record not found" bit is set in the Status register and the command aborts.

Read track

This command reads a whole track from disk, as raw data. It is meant for diagnostic purposes, therefore most checking mechanisms are disabled: no CRC checking is performed, ID fields are not checked for track and side number and the RG pin is not activated.

The FDC deals with the HLD, HLT and TG43 pins as described above, then check the IP* pin and waits for an index hole. The contents of the track is then passed to the Data register, via the shift register. The DRQ pin is activated once a byte is ready and must be cleared by reading the Data register before the next byte comes in. If it is not the case, the "lost data" bit is set in the Status register and reading continues.

The address mark detector is working during the whole commands. ID and data marks are thus recognized and used to synchronise the flow of bits (i.e. to detect bytes boundaries). If a mark does not occur where expected, the "lost data" bit is set. As a result, the sector ID blocks and Data blocks will always be correct, but the number of filler bytes in the gaps may vary. With the FD1771, it is possible to inactivate the address mark detector during Read Track commands (bit s*).

Write track

This command is used to write a track on disk, for formatting purposes. Some byte values (>F5-FE) are reserved to instruct the FDC to write marks or CRC values on the disk. In FM, filler bytes should be >FF, but the FD1795/97 will also accept >00. In MFM, filler bytes should be >4E. It is acceptable to use more filler bytes than specified by the format, although the IBM format does not allow that before the ID block. The number of >00 used for sync must be exact in the ID block, but may be increased in the data block.

The FDC deals with the HLD, HLT and TG43 pins as described above. It checks the WPRT* pin and aborts with the "write protected" bit set in the Status register is the pin is active. The FD1771 additionally checks the DINT* pin and issues the same error if this pin is low. Otherwise, The FDC ctivates the DRQ pin and awaits the first data byte in the Data register, for as long as it would take to write three bytes. If no data arrives, the command aborts with the "lost data" bit set.

The FDC then waits for an index hole, by checking the IP* pin. Once the hole is detected, the FDC starts writting data on disk: the first byte is transfered from the Data register to the shift register and the DRQ pin is activated. If the CPU fails to provide the next byte in time, a >00 byte is written on disk and the "lost data" bit is set in the Status register. The command terminates when the index hole is detected again.

Before to write a byte of data, the FDC checks its value. If it is in the range >F5-FE, it causes the FRC to write marks or CRC on disk: see the table below. As a consequence, those bytes cannot be part of the data written on disk. This is no problem upon formatting, since all sector data blocks contain a default value (>E5 with the TI disk manager), but it is a major pain in the butt when it comes to copy a disk track-by-track!

Byte	Meaning in FM	Meaning in MFM
>00->F4	Written normally	Written normally
>F5	Not allowed	Write >A1, missing clock between bits 4-5 Reset CRC
>F6	Not allowed	Write >C2, missing clock between bits 3-4
>F7	Generate 2 CRC bytes	Generate 2 CRC bytes

>F8->FB	Written as a mark with clock pattern >C7. Reset CRC	Written normally
>FC	Written as a track mark (clock >D7)	Written normally
>FD	Written normally. Reset CRC.	Written normally
>FE	Written as a mark with clock pattern >C7. Reset CRC	Written normally
>FF	Written normally	Written normally

Force interrupt

This command is generally used to stop a multiple Sector read or Sector write. It can also be used to place the Status register in a defined state: if a command is interrupted, the Status register reflects the current status of this command (with the "busy" bit cleared), otherwise the meaning of the status byte is that of the Force interrupt command. Finally, this command can be used to physically generate interrupts.

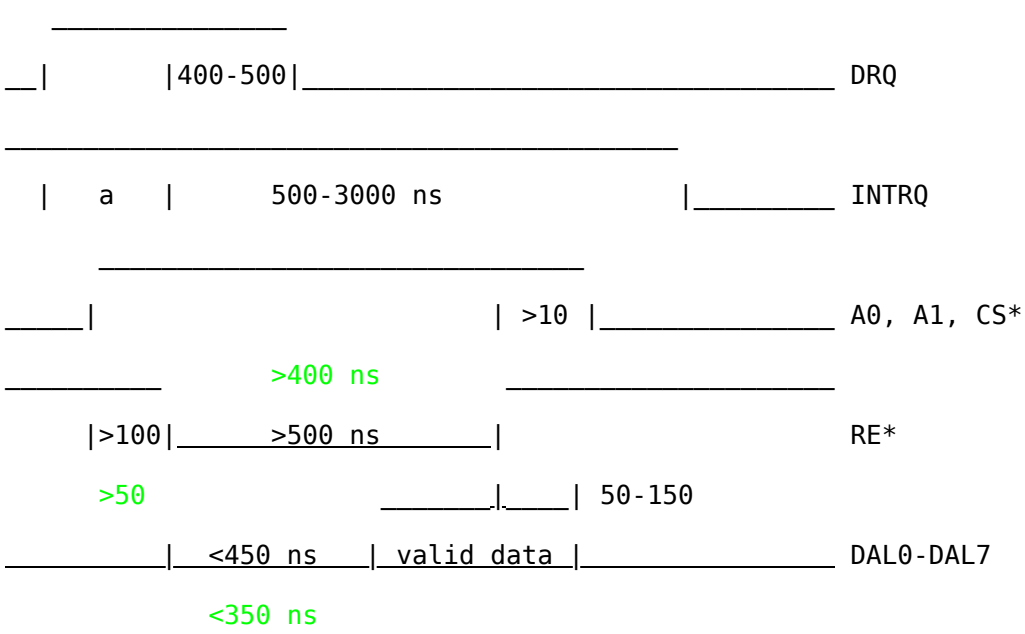
Interrupts are issued by setting one (or more) of the I0-I3 bits to 1. When one of the conditions for the interrupt is met, the INTREQ line goes active (high). I3 issues an interrupt immediately. I2 issues an interrupt when an index hole is detected. I1 and I0 issue interrupts when the READY pin changes state: I1 when it becomes not-ready, I0 when it becomes ready. I0 to I2 are reset automatically, but I3 must be reset by loading another Force interrupt command, with I3=0.

The CPU should wait 8 usec (16 usec for DD) after the command is loaded, before loading another command (else it would cancel the Force interrupt). These times doubble with a 1 MHz clock.

Timing diagrams

When diagrams are valid for both the FD1771 and the FD179x, values specific for the FD179x are indicated in green. All times are for a 2 MHz clock, you should double them for a 1MHz clock.

CPU interface: Read enable

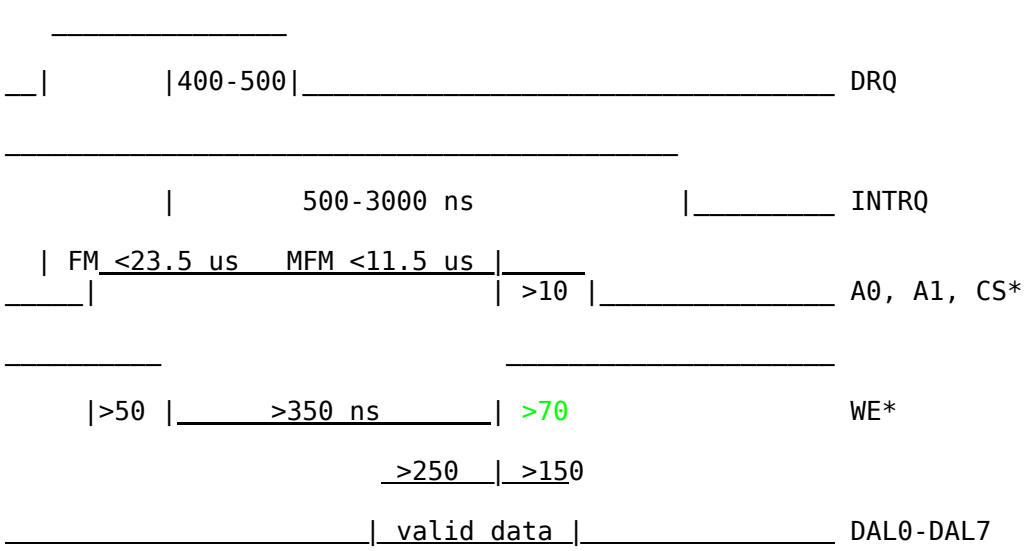


a) Tservice: 27.5 us in FM, 13.5 us in MFM

- DRQ rising edge indicates there is data in the Data register
- DRQ falling edge indicates that the Data register was read

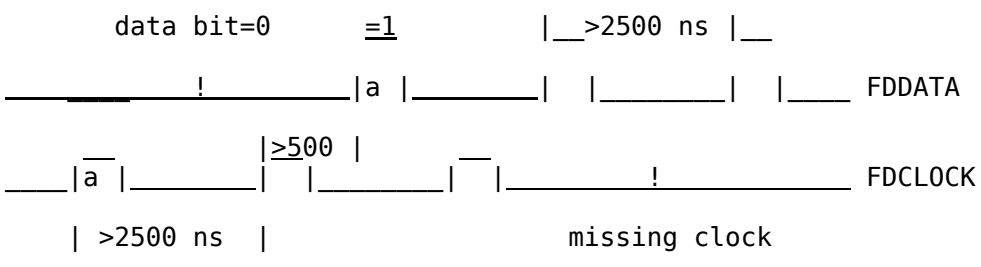
- INTRQ rising edge occurs at the end of the command
- INTRQ falling edge indicates that the status register was read

CPU interface: Write enable



- DRQ rising edge indicates that the Data register is empty
- DRQ falling edge indicates that the Data register was loaded
- INTRQ rising edge occurs at the end of the command
- INTRQ falling edge indicates that the Command register was loaded
- When writing to the Sector, Track or Data register: can't be read until 4 us after rising edge of WE*
- When writing to the Command register, the Status is not valid for 28 us in FM, 14 us in MFM.

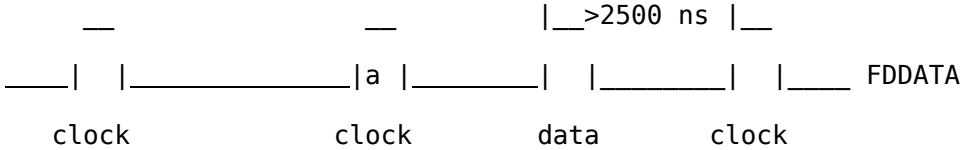
FD1771 drive interface: Input data with separator



a) 150-350 ns.

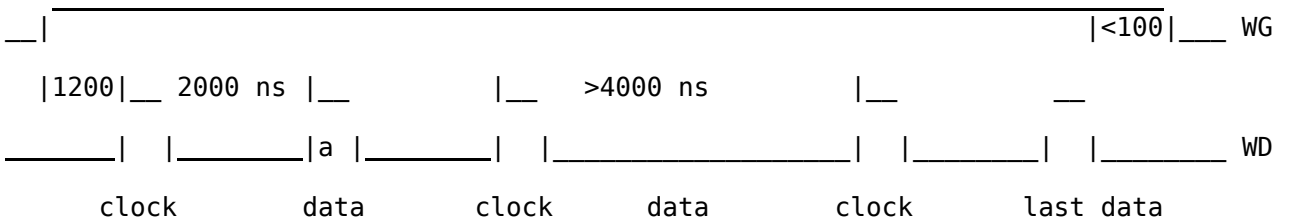
FDDATA and FDCLOCK may be reversed. The FD1771 decides which is which.

Without separator



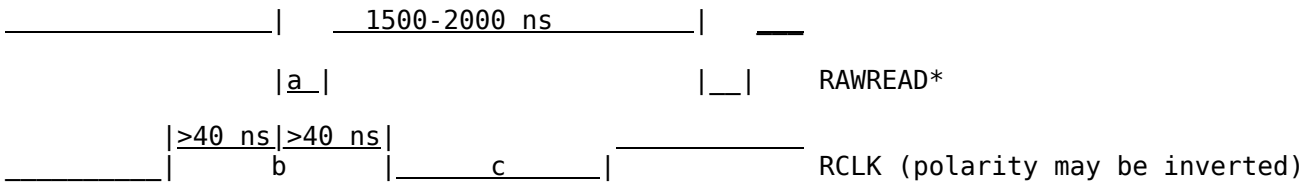
a) 150-350 ns.

FD1771 drive interface: Write data



a) 500-660 ns.

FD179x drive interface: Input data



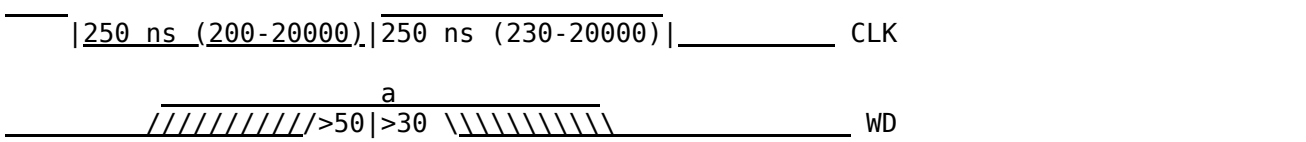
a) 100-300 ns. Can be any length if entirely whithin window.

If pulse occur in both windows, must be <300 ns in MFM, <600 ns in FM

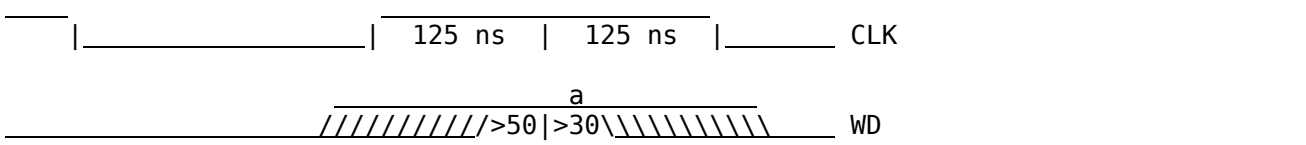
b) and c) 4 us in FM. 2 us in MFM (for 8" disks: 2 us in FM, 1 us in MFM)

FD179x drive interface: Write data

Single density (DDEN* high)



Double density (DDEN* low)

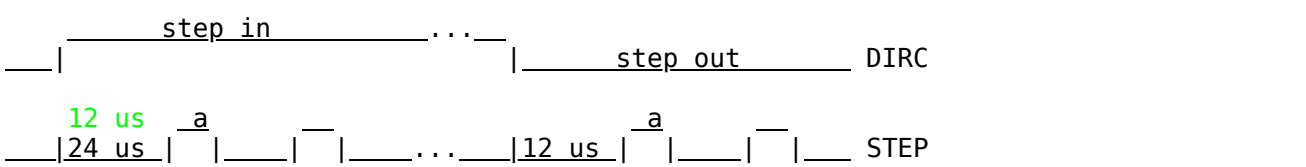


a) Pulse width 500-600 ns in FM, 200-350 ns in MFM.

Pulse edges must be in shaded area.

- Cycle time is 2,3 or 4 us (+/- clock error).
- Time from WG to WD is 2 us in FM, 1 us in MFM
- Time from WD to WG off is 2 us in FM, 1 us in MFM
- Time from EARLY or LATE to WD is at least 125 ns (in MFM)
- Time from WD to EARLY or LATE is at lesst 125 ns (in MFM)

Drive interface: Stepping

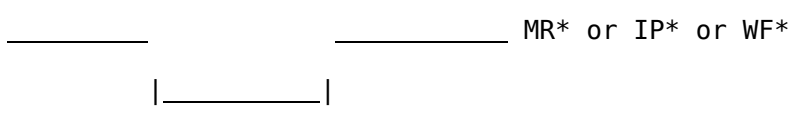


```
          |   b   |                    |   b   |
```

a) 4 us if FM 2 us in MFM

b) Step time: depends on r1,r0 bits in command

Drive interface: Other signals



- MR* (master reset) pulse width: at least 10 us for FD1771, at least 50 us for FD179x
- IP* (index pulse) pulse width: at least 10 us
- WF* (write fault) pulse width: at least 10 us

Electrical characteristics

Absolute maximum ratings

For the FD1771

Vdd to Vss.....+20V to -0.3V
Any input to Vss.....+20V to -0.3V
Icc.....30 mA nominal
Idd.....10 mA nominal
Ibb.....0.4 uA nominal
Operating temp.....0 to 70 `C
Storage temp.....-55 to +125 `C

For the FD179x

Vdd to Vss.....+15V to -0.3V
Any input to Vss.....+15V to -0.3V
Icc.....60 mA (35 mA nominal)
Idd.....15 mA (10 mA nominal)
Cin.....15 pF max
Cout.....15 pF max
Operating temp.....0 to 70 `C
Storage temp.....-55 to +125 `C

Operating characteristics

FD1771: Vdd = +12V, Vss = 0V, Vcc = +5V, Vbb=-5V, TA = 0-70 `C
FD179x: Vdd = +6V, Vss = 0V, Vcc = +5V, TA = 0-70 `C

Parameter	Test conditions	Min	Max	Unit
Input leakage	Vin=Vdd. Pins without pull-ups (not for pins 22,23,33,36,37)	.	10	uA
Output leakage	Vout=Vdd	.	10	uA
Input high voltage	.	2.6	.	V
Input low voltage	.	.	0.8	V

Output high voltage	I = -100 uA	2.8	.	V
Output low voltage	I = 1.6 mA (1.0 for 1792/94)	.	0.45	V
Power dissipation	.	.	0.6	W

Next page

Preliminary version. 3/19/99. Not for release.
Revision 1. 5/15/99. OK to release (fully disassembled card ROM).
Revision 2. 5/30/99. Tested & debugged examples.
Revision 3. 7/4/99. Added an explanation of the CRC.
Revision 4. 7/13/99. Added a paragraph on magnetic media and write precomp.
Revision 5. 9/18/99. Added data on the FD1771, a word on the CorComp card.
Revision 6. 6/16/00. Got the FD171 manual. Added section on drives. Split in two pages.
Revision 7. 4/14/05. Cluster offset in FDR description is zero-based.

Back to the TI-99/4A Tech Pages