Famicom Disk System Disk Drive/RAM Adaptor Technical Briefing
By Brad Taylor (big_time_software@hotmail.com)
Thanks are credited to Nori and Goroh for thier existing related docs (and Ki and Sgt. Bowhack for their respective translations), and to Val Blant for assistance in cracking the CRC algorithm used by the FDS.

Note: if you are using a windows program like notepad to view this document in, I recommend changing the font over to "terminal" style, to support some DOS characters used in this document.

This document describes:
- the FDS data transfer protocol
- FDS disk magnetic encoding format
- low level disk data storage layout
- CRC calculations for files on FDS disks
- RAM adaptor pinouts & their function
- Steps to emulating the disk drive

Brief description of FDS Disk drive operation

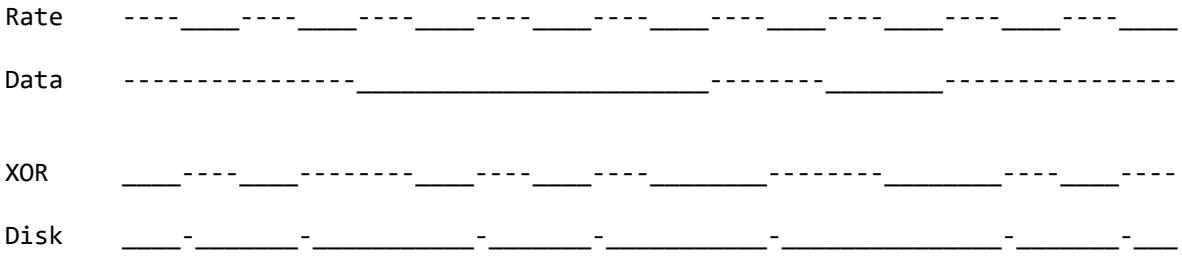
Data coming out of the FDS disk drive is a real time representation of what data the head is currently reading. This means that no formatting of the data occurs. The head is always advancing across the disk at a constant rate (sequential access) as the disk spins. When the head reaches the end of the disk (most inner track), it returns to the beginning of the disk (most outer track) and the cycle repeats, upon request from the RAM adaptor. This means that on every scan, the entire disk is read (which takes about 6 seconds). The disk drive signals the RAM adaptor when the head has been positioned to the outer most track, and is starting a new scan.

FDS data transfer protocol

In any data transfer, 2 pieces of information (signals) must be sent:

- a signal that represents the rate at which the data is being sent
- the actual data

Like most disk drive units, the FDS disk drive is sending it's data out via serial connection (1 wire, meaning 1 bit at a time). Data prerecorded on FDS disks have already had the 2 aformentioned signals "combined" using an XOR algorithm described below (note that I've used underscores (_) and hyphens (-) to respectfully represent 0 and 1 logic levels in my diagram below).



Rate is the signal that represents the intervals at which data (bits) are

transfered. A single data bit is transfered on every 0 to 1 (_-) transition of this signal. The RAM adaptor expects a transfer rate of 96.4kHz, although the tolerance it has for this rate is ñ10%. This tolerance is neccessary since, the disk drive can NOT turn the disk at a constant speed. Even though it may seem constant, there is a small varying degree in rotation speed, due to it's physical architecture.

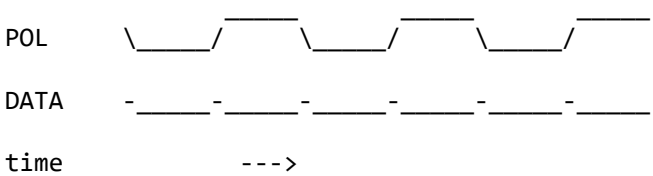
Data is the desired sequential binary data to be serialized. I specifically chose the data used here to demonstrate how it would be stored on a FDS disk. Note that if data changes, it changes syncronously with the 0 to 1 (_-) transition of the Rate signal.

XOR represents the result of a logical exclusive OR performed on the Rate and Data signals.

Disk represents what is ACTUALLY recorded on a FDS disk (and therefore, what is coming out of the disk drive, and going to the RAM adaptor). This signal is constructed of pulses that are written in sync with every 0 to 1 (_-) transition of the XOR result signal. In reality, the length of these pulses (the time it stays as 1) is about 1 microsecond. When the RAM adaptor sends data out to write to the disk, it is also in this format (although logically inverted compared to the diagram shown above). However, data sent to the RAM adaptor is not required to be in this format. The RAM adaptor responds to the 0 to 1 (_-) transitions in the serial data signal rather than the length of the pulses. This is why the RAM adaptor will interpret both serial data types (XOR or Disk) indifferently.

+--------------------------------+
|FDS disk magnetic encoding format|
+--------------------------------+

The system for magnetically storing data on FDS disks is very simple. At any instant, when the disk head is reading the disk, it has no way of telling what magnetic charge the current section of disk area has (consequently, the head produces no voltage). Because of how the physics of magnetic inductance works, the head will *only* generate voltage while a section of the disk track containing opposite charges is being read. During this time, the head produces a spike in voltage (the duration of the spike is a function of the width of the head (the area physically contacting the disk's surface), and the rotation speed of the disk). These spikes are then amplified, and sent to the data out pin on the drive. This is how data encoding on FDS disks is made. The following diagram should provide more explanation:



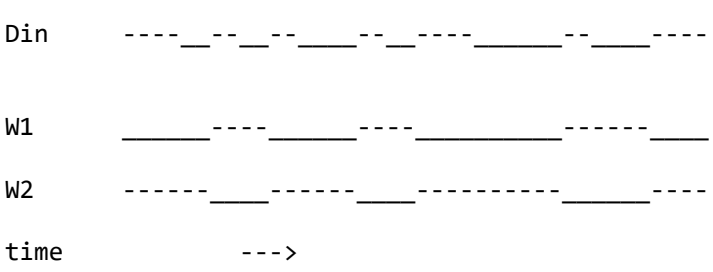
POL represents the polarity of the magnetic charge on a typical disk track.

DATA is the interpretation of the POL changes, and is also the digital signal that appears on the "data out" signal of the drive as a result.

Magnetically recording data onto disk

Data entering the disk drive intended to be written onto the disk is not written directly. Like the RAM adaptor, the data input on the disk drive responds to the positive-going edge of pulses fed into it. This is because

the data is fed into an internal positive-edge triggered toggle flip-flop (divide-by-2 counter). The output status of this flip-flop is then the signal that is directed to the drive's write head during the time disk writing is to be engaged. The 2 write head wires are connected to the flip-flop's complimentary outputs, so that they are always in opposite states during the writing phase. During writing, the current that flows through the write head (in either direction) is about 20 to 25 milliamperes. The following chart outlines the relationship between these signals:



Din is the data entering the drive, intended to be written to disk.

W1 & W2 represent the digital status of the wires connected to the disk drive's write head.

Low level disk data layout/storage

Now that we have covered the communication protocol used, let's talk about the layout of data structures on the disks. Nori's document has done a good job of providing information on the way RAW data is layed out on the disk. At this point, I recommend referring to Nori's "fds-nori.txt" and reading the first half of the document describing the disk data structures.

Here, I would like to elaborate on some items found in the aforementioned document. Note that any references to bit or byte data transfers assume the standard 96.4kHz xfer bit rate (meaning 1 bit is transfered in 1/96400 seconds).

- The disk drive unit signals the RAM adaptor when the head has moved to the beginning of the disk via the "-ready" signal it sends out (more on this later). The "-ready" signal is based on a mechanical switch inside the drive which is activated when the head is brought back to the outer most edge of the disk (the beginning). Because the switch will usually be triggered prematurely, the first 13000 bits (approx.) of data the drive will send out immediately after this switch is activated will be invalid. To compensate for this, the RAM adaptor purposely ignores the first 26100 bits (approx.) sent to it since it recieves the "-ready" signal from the disk drive.

- After this period, the RAM adaptor is going to be interpreting data read off the disk. Because of the many mechanical variables these disk drive units have, the data recorded on the disk cannot anticipate the exact point at which the RAM adaptor will start accepting data. Therefore, a padding mechanism is used, which is reffered to here as the "GAP" period.

- All GAP periods recorded on the disk are composed entirely of binary coded 0's. The RAM adaptor ignores these 0's (meaning that there is no size limit to any GAP period recorded on the disk) until a set (1) bit is recieved, which identifies the start of a file (reffered to in Nori's doc. as the "block start mark"). The data immediately following this bit makes up the file data. Size of the file is then determined by the file data read in

(consult Nori's doc). Note that this is the only way to determine the size of a file, since there really is no mechanism implemented (like there is for the start of a file) to indicate where a file's data ends.

- The order serial binary data is recorded on the disk in is the little endian format. Lesser significant bits are sent first before greater, and same for bytes.

- The length of the first GAP period present on typical FDS disks (relative to the instant the disk drive's "-ready" signal is activated) is about 40000 bits, after which the first block start mark (indicating the beginning of the first file) will appear.

- Writing to the disk does not occur in sync with the existing data on the disk. Where a write period starts and ends is bad news because it creates a gap between pulses almost always of an invalid length. This would lead to the RAM adaptor misinterpreting the data, causing errors. To overcome this, the RAM adaptor always ignores the first 488 bits (aprox.) to follow after the immediate end of any file. This period allows the RAM adaptor (or the game rather) an oppertunity to make the switch from reading from the disk to writing or vice-versa.

- After this period, the adaptor will expect another gap period, which will then lead into a block start mark bit, and the next file. This cycle repeats until there are no more files to be placed on a disk.

- The typical GAP period size used between files on FDS disks is roughly 976 bits (this includes the bits that are ignored by the RAM adaptor).

- The rest of the disk is filled with 0's after the last file is recorded (although it really shouldn't matter what exists on the disk after this).

CRC calculations

The "block end mark" is a CRC calculation appended to the immediate end of every file. It's calculation is based exclusively on that files' data. The CRC is 16-bits, and is generated with a 17 bit poly. The poly used is 10001000000100001b (the X25 standard). Right shift operations are used to calculate the CRC (this effectively reverses the bit order of the polynomial, resulting in the 16-bit poly of 8408h). A x86 example is shown below (note that ; is used to seperate multiple statements on one line, and // is used for comments). The file this algorithm is designed to work on has no block start mark in it ($80), and has 2 extra bytes at the end (where a CRC calculation would normally reside) which are 0'd. While the block start mark is actually used in the calculation of a FDS file CRC, you'll see in the algo below that the block start mark q'ty ($80) is moved directly into a register.

```
// ax is used as CRC accumulator
// si is the array element counter
// di is a temp reg

// Size is the size of the file + 2 (with the last 2 bytes as 0)
// Buf points to the file data (with the 2 appended bytes)

        mov     ax,8000h        // this is the block start mark
        sub     si,si           // zero out file byte index ptr

@1:     mov     dl,byte ptr Buf[si]
        inc     si
```

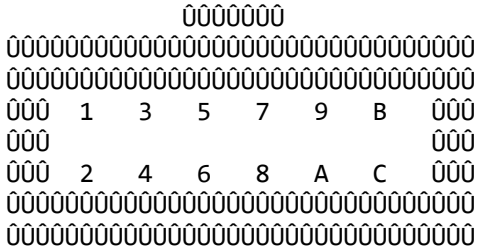
```
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di
        shr dl,1; rcr ax,1; sbb di,di; and di,8408h; xor ax,di

        cmp     si,Size
        jc      @1

// ax now contains the CRC.
```

Of course, this example is only used to show how the CRC algorithm works. Using a precalculated CRC look-up table is a much faster (~5 cc/it) method to calculate CRCs in the loop (the above method consumes over 40 clock cycles per iteration). However, if speed is not an issue, the above code uses only a fraction of the memory a table look-up implementation would consume. More memory can be saved by loading the polynomial value into a register, and even more by rolling the repeated instructions up into a second loop.

+------------------------------+
|FDS RAM adaptor control signals|
+------------------------------+



A front view of the RAM adaptor's disk drive connector.

pin #	*2C33 pin	signal description
1	50	(O) -write
2	64	(O) VCC (+5VDC)
3	49	(O) -scan media
4	32	(O) VEE (ground)
5	52	(O) write data
6	37	(I) motor on/battery good
7	47	(I) -writable media
8	-	(I) motor power (note 1)
9	51	(I) read data
A	45	(I) -media set
B	46	(I) -ready
C	48	(O) -stop motor

prefixes:

(O): signal output.
(I): signal input.
- : indicates a signal which is active on a low (0) condition.

* : The 2C33 is a custom processor inside the RAM adaptor that handles all the disk I/O. These pinouts are listed here because the 2C33 is the ultimate destination of all the disk control signals.

1: The RAM adaptor does not use this signal. An electronically controlled 5-volt power supply inside the disk drive unit generates the power that appears here. This power is also shared with the drive's internal electric motor. Therefore, the motor only comes on when there is voltage on this pin.

(O) -write

While active, this signal indicates that data appearing on the "write data" signal pin is to be written to the storage media.

(O) write data

This is the serial data the RAM adaptor issues to be written to the storage media on the "-write" condition.

(O) -scan media

While inactive, this instructs the storage media pointer to be reset (and stay reset) at the beginning of the media. When active, the media pointer is to be advanced at a constant rate, and data progressively transferred to/from the media (via the media pointer).

(O) -stop motor

Applicable mostly to the FDS disk drive unit only, the falling edge of this signal would instruct the drive to stop the current scan of the disk.

(I) motor on/battery good

Applicable mostly to the FDS disk drive unit only, after the RAM adaptor issues a "-scan media" signal, it will check the status of this input to see if the disk drive motor has turned on. If this input is found to be inactive, the RAM adaptor interprets this as the disk drive's batteries having failed. Essentially, this signal's operation is identical to the above mentioned "motor power" signal, except that this is a TTL signal version of it.

(I) -writable media

When active, this signal indicates to the RAM adaptor that the current media is not write protected.

(I) read data
---
when "-scan media" is active, data that is progressively read off the storage media (via the media pointer) is expected to appear here.

(I) -media set

When active, this signal indicates the presence of valid storage media.

(I) -ready

Applicable mostly to the FDS disk drive unit only, the falling edge of this signal would indicate to the RAM adaptor that the disk drive has acknowledged the "-scan media" signal, and the disk drive head is currently at the beginning of the disk (most outer track). While this signal remains

active, this indicates that the disk head is advancing across the disk's surface, and apropriate data can be transferred to/from the disk. This signal would then go inactive if the head advances to the end of the disk (most inner track), or the "-scan media" signal goes inactive.

+---------------------------------+
|Steps to emulating the disk drive|
+---------------------------------+

Before a data transfer between the RAM adaptor and the disk drive/storage media can commence, several control signals are evaluated and/or dispatched. The order in which events occur, is as follows.

1. "-media set" will be examined before any other signals. Activate this input when your storage media is present. Make sure this input remains active throughout the transfer (and for a short time afterwards as well), otherwise the FDS BIOS will report error #1 (disk set). If this signal is not active, and a data transfer is requested, the BIOS will wait until this signal does go active before continuing sending/examining control signals.

2. If the RAM adaptor is going to attempt writing to the media during the transfer, make sure to activate the "-writable media" input, otherwise the FDS BIOS will report error #3 (disk write protected). Note that on a real disk drive, "-writable media" will active at the same time "-media set" does (which is when a non-write protected disk is inserted into the drive). A few FDS games rely on this, therefore for writable disks, the "-write enable" flag should be activated simultaniously with "-media set".

3. The RAM adaptor will set "-scan media"=0 and "-stop motor"=1 to instruct the storage media to prepare for the transfer. This will only happen if the first 2 conditions are satisfied.

4. "motor on/battery good" will be examined next. Always keep this input active, otherwise the FDS BIOS will report error #2 (battery low).

5. Activating "-ready" will inform the RAM adaptor that the media pointer is currently positioned at the beginning of the media, and is progressing. It is during the time that this signal is active that media data transfers/exchanges are made. Make sure this input remains active during the transfer, otherwise the FDS BIOS will report an error. "-ready" shouldn't be activated until at least 14354 bits (~0.15 seconds) have elapsed relative to step #3.

6. During the transfer, the "-write" signal from the RAM adaptor indicates the data transfer direction. When inactive, data read off of the media is to appear on the "read data" signal. When active, data appearing on the "write data" signal is to be recorded onto the media.

7. The RAM adaptor terminates the data transfer at it's discretion (when it has read enough or all the files off of the media). This is done when "-scan media"=1 or "-stop motor=0". After this, it is OK to deactivate the "-ready" signal, and halt all media I/O operations.

A few final notes

- It is okay to tie "-ready" up to the "-scan media" signal, if the media needs no time to prepare for a data xfer after "-scan media" is activated. Don't try to tie "-ready" active all the time- while this will work for 95% of the disk games i've tested, some will not load unless "-ready" is disabled after a xfer.

- Some unlicenced FDS games activate the "-stop motor" signal (and possibly even "-write", even though the storage media is not intended to be written to) when a media transfer is to be discontinued, while "-scan media" is still active. While this is an unorthodoxed method of doing this, the best way to handle this situation is to give the "-stop motor" signal priority over any others, and force data transfer termination during it's activation.

- Check out my FDS loader project (which uploads *.FDS files to the RAM adaptor) for source code to my working disk drive emulator.

EOF