

Atari Floppy Disk Copy Protection

By Jean Louis-Guérin (DrCoolZic)
Revision 1.4 – June 24, 2015

Atari Floppy Disk Copy Protection

Table of Contents

Table of Contents	2
Chapter 1. Presentation	4
Chapter 2. Copy protections detail description	5
2.1 Protections based on data	5
2.1.1 Number of tracks (NOT)	6
2.1.2 Shifted tracks (SFT)	7
2.1.3 Track Layout Pattern (TLP)	9
2.1.4 Number of Sectors (NOS)	9
2.1.5 Sector Sizes (SSZ)	10
2.1.6 Invalid ID Field (IIF)	10
2.1.7 Duplicate Sector Number (DSN)	12
2.1.8 Sector within sector (SWS)	13
2.1.9 Non Standard DAM (NSD)	13
2.1.10 Sector with No ID (SNI)	14
2.1.11 Sector with No Data (SND)	14
2.1.12 Data CRC Error (DCE)	14
2.1.13 Data Track (DTT)	15
2.1.14 Hidden Data into GAP (HDG)	15
2.1.15 Hidden data into nonstandard tracks (HDT)	15
2.1.16 Invalid Data in Gap (IDG)	16
2.1.17 Invalid Sync-mark Sequence (ISS)	16
2.1.18 Partially formatted track (PUT)	16
2.1.19 Fuzzy Sector (FZS)	17
2.1.20 Fuzzy Track (FZT)	17
2.2 Protections based on timing	18
2.2.1 Long / Short Sector (LGS & SHS)	18
2.2.2 Long/Short Track (LGT & SHT)	19
2.2.3 Sector Bit-rate Variation (SBV)	19
2.2.4 No Flux Area (NFA)	20
Chapter 3. Preservation of Atari floppy disks	21
3.1 Cleaning a floppy disk to create correct image	21
3.2 Why do we need several revolutions for preservation?	21
3.3 Kryoflux short presentation	23
3.4 Supercard Pro short presentation	23
Chapter 4. Technical Information	24
4.1 Atari Low-Level Formats	24
4.1.1 Format for 9/10/11 Sectors of 512 Bytes	25
4.1.2 "Standard" 128-256-512-1024 Bytes / Sector Format	26
4.2 WD1772 DPLL Input Circuitry	27
4.2.1 Description	27
4.2.2 WD1772 Detection of Fuzzy Border Bits	29
4.3 WD1772 MFM track language	30
4.4 WD1772 Synchronization (sync marks detection)	31
4.5 False sync mark detection	32
4.6 Overlapping Sync Mark	32
4.6.1 Overlapping \$4489-\$4489 (\$A1-\$A1)	32
4.6.2 Overlapping \$5224-\$4489 (\$C2-\$A1)	33
4.6.3 Overlapping \$4489-\$5224 (\$A1-\$C2)	33
4.6.4 Overlapping \$5224-\$5224 (\$C2-\$C2)	33
4.6.5 Invalid Sync sequence	33
4.7 WD1772 Bug in Read/Write Track commands	34
4.8 WD1772 CRC Information	35
4.8.1 CRC Computation	35
4.8.2 Playing with the CRC	35
4.9 No Flux Area on Disk	37
4.9.1 Checking NFA with the WD1772	37

Atari Floppy Disk Copy Protection

4.9.2	Special case of No Flux Area over index	38
4.10	Unformatted Diskette / Track / Sector.....	41
4.10.1	Presentation	41
4.10.2	Partially unformatted track	42
4.10.3	Partially formatted Track	44
4.10.4	Unformatted track detection	44
4.10.5	How to reproduce unformatted areas on Floppy Disks?	44
4.11	Fuzzy Bits	46
4.11.1	Flux Reversals in Ambiguous Area	46
4.11.2	MFM Flux Timing Violation.....	46
4.11.3	Weak Bit	47
4.12	Write Splices.....	48
4.12.1	Sector write splices	48
4.12.2	Track write splices.....	49
4.13	Hidden data.....	50
4.13.1	Union Demo / Dragon Flight hidden sequence	50
4.13.2	Jupiter Masterdrive hidden sequence	50
4.13.3	Realm of the Troll.....	51
Chapter 5.	Analysis of Games/Programs.....	52
5.1	<i>Barbarian (from Psygnosis).....</i>	<i>53</i>
5.2	<i>Bob Morane</i>	<i>54</i>
5.3	<i>Colorado</i>	<i>54</i>
5.4	<i>Computer Hits Volume 2 (Beau-Jolly).....</i>	<i>55</i>
5.5	<i>D50 Editor V2 (Dr.T)</i>	<i>57</i>
5.6	<i>Dragon flight.....</i>	<i>58</i>
5.7	<i>Dungeon Master (FTL Inc.).....</i>	<i>59</i>
5.8	<i>Eco by Ocean</i>	<i>60</i>
5.9	<i>Golden Axe</i>	<i>61</i>
5.10	<i>Jupiter Masterdrive.....</i>	<i>62</i>
5.11	<i>Kick Off 2 (Anco Software 1990).....</i>	<i>63</i>
5.12	<i>Maupiti Island</i>	<i>64</i>
5.13	<i>Night Shift (US Gold)</i>	<i>64</i>
5.14	<i>Obitus.....</i>	<i>65</i>
5.15	<i>Operation Neptune.....</i>	<i>65</i>
5.16	<i>Populous (Electronic Arts).....</i>	<i>65</i>
5.17	<i>Power Drift.....</i>	<i>66</i>
5.18	<i>Sherman M4.....</i>	<i>67</i>
5.19	<i>Star Glider 2.....</i>	<i>67</i>
5.20	<i>Theme Park Mystery (Image Works)</i>	<i>68</i>
5.21	<i>Time of lore.....</i>	<i>69</i>
5.22	<i>Turrican.....</i>	<i>70</i>
5.23	<i>Vroom.....</i>	<i>71</i>
5.24	<i>Wizball, Ocean.....</i>	<i>72</i>
5.25	<i>Z-out.....</i>	<i>72</i>
Chapter 6.	References.....	73
6.1	<i>Documents / Articles</i>	<i>73</i>
6.2	<i>Forums Threads</i>	<i>73</i>
6.3	<i>Related Patents</i>	<i>74</i>
6.4	<i>Web Sites</i>	<i>74</i>
6.5	<i>FDC & Related Information</i>	<i>74</i>
6.6	<i>Game References.....</i>	<i>74</i>
Chapter 7.	Document history.....	76

Atari Floppy Disk Copy Protection

Chapter 1. Presentation

This document describes **floppy disk protection mechanisms** used on the Atari platform. This type of **copy protection** is very old and, with many years of development and the usage of sophisticated floppy disk hardware, it has conducted to numerous protection methods frequently referred as **key disk protection**. The key disk protection method has at least two obvious qualities: first, a *key disk* can be simultaneously used as *protection* and *distribution* disk and second, this type of protection is very cheap but nevertheless hard to tamper with. So, key disk protections have been widely used to protect Atari programs and games. To fully understand the key disk based protections, you need to have some basic knowledge about FD/FDC data and operation.

Some of the FD protection mechanisms are generic to many platforms while some are directly related to a specific Floppy Disk Controller used on a specific platform. Therefore, in order to get a general understanding, I have reviewed the FD protections mechanism used on several platforms: Amiga, Commodore C64, PC, Tandy, Atari 8 bits and Atari ST 16 bits (see the [references section](#)). Information about the different copy protection mechanisms presented here is the result of experimentations and reading from the Web. Links to the original information on Web sites can be found at the end of this document in the [references section](#).

In order to validate this document, I have analyzed the protections of many original floppy disks with several programs that I have developed over time:

- For detailed analysis of *timing information*, the first program that I have created is called **Analyze**. It runs on Atari and PC. This program reads the flux reversals stream files produced on Atari by the **Discovery Cartridge** and performs a detailed analysis. This program takes its root in experiments I have done back in the 80s! The program is now obsolete and replaced by the **AUFIT** program presented below.
- For basic protection analysis I have created a program running on Atari called **Panzer** (Protection **AN**alyze**ZER**) that automatically detects and reports many protections. This program also provides the capability to directly run several FDC commands and analyze the sectors and tracks information (including **timing** for track and sector) read.
- **KFAnalyze** program reads input **Stream** files generated by the **KryoFlux** board. A Stream file contains Atari FD information at the flux reversals level, it is therefore possible to provide very accurate detections of protections especially those related to bit cell timing variation. The heart of this program is a precise emulation of the Western Digital WD1772 Floppy Disk Controller. The emulation mimics a full DPLL data separator and provides functions equivalent to the **read track**, **read address**, and **read sector** commands reading data directly from the *Stream* files. Therefore it is possible to process the Stream information as if we were read by an Atari WD1772 FDC but with a lot of extra information especially timing information. This is the ancestor of the Aufit program presented below.
- My latest program for analyzing Atari floppy disk content is called **AUFIT** (Atari Universal FD Image Tool). It provides many features to analyze and display FD content at the flux transition level (as provided by KryoFlux and Supercard Pro) using a nice Graphical User Interface. Beyond FD content analysis, the programs also provides the capability to convert the information in several Atari images formats (Pasti, ST, MSA) for emulation.

I want to thanks to many people on [Atari forum](#) for taking time to discuss some of the protections presented here (See [HERE](#) and [HERE](#)).

Chapter 2. Copy protections detail description

In this section I provide a detailed description of the different protection's mechanisms used in Atari Key disks. The protections have been grouped into two categories:

- ★ [Protections based on data](#)
- ★ [Protections based on timing](#)

2.1 Protections based on data

This category contains protections based on using non-standard or impossible to write (on Atari) data content in the tracks and/or sectors of a diskette.

A "normal diskette" has one or two sides (i.e. single or double sided) each having 80 tracks numbered from 0 to 79. A more detailed description of formats can be found in the [Atari Low-Level Formats](#) section.

A "standard track" on an Atari is composed of 9 sectors each with 512 bytes of data sequentially numbered from sector 1 until sector 9.

However it is not uncommon to use diskettes with up to 11 sectors and more than 80 tracks as it allows packing more data. A good duplication/imaging program should be able to detect and reproduce all these alternatives and therefore they are not really considered as protection.

But beyond these basic variations of a diskette's data content we will see that some protections uses mechanism **difficult to detect** (so that a copy program would not easily find them) and some that **cannot be reproduced** without special hardware.

Atari Floppy Disk Copy Protection

2.1.1 Number of tracks (NOT)

A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. Some simplistic protections are based on extra or missing tracks.

2.1.1.1 Extra tracks (EXT)

- **Description:** A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. It is possible to write up to 82 or even 83 tracks on one side of a diskette. It is also possible to “hide” one or several tracks on the second side of an “officially” (as specified in the boot sector) single sided diskette.
- **Creation:** Easy to create on Atari. Note that some early Atari drives are single sided, and some cannot position the head past track 79. Beware that using tracks over 82 has been reported to damage some floppy drives.
- **Detection:** You have to probe the diskette using FDC commands to check if some extra tracks exist (probing 82 tracks is usually sufficient). For Single Sided diskette, you also need to probe for hidden track on second side.
- **Duplication:** Easy by software.
- **Emulation:** Just need to store information about extra tracks.
- **Example:** Passengers on the Wind (Infogrames) uses tracks 80 & 81.

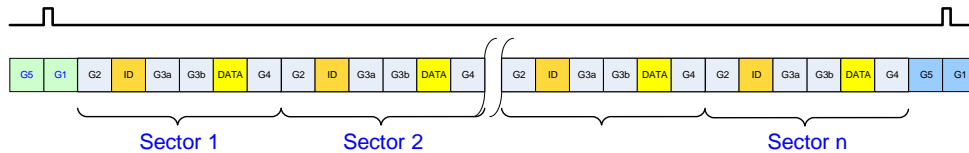
2.1.1.2 Missing tracks (TNF)

- **Description:** A “normal Atari diskette” has 80 tracks numbered 0 through 79 on each side. It is possible that not all of these tracks are formatted. For detail description of unformatted track please refer to [Unformatted Diskette / Track / Sector](#). Note that it is possible to hide data in a track that seems unformatted. Hiding data in what looks like an unformatted track is usually difficult to detect (for example see [Power Drift](#)).
- **Creation:** On a non-preformatted diskette you only need to format the “non-missing” tracks. On a preformatted diskette (usually diskettes are sold DOS pre-formatted) you need to mimic unformatted tracks by writing, for example, some random data to those tracks without sync but the results is really not the same.
- **Detection:** Using WD1772 commands: i.e. a **seek** command with the *verify option* should fail on unformatted track, or a **read address** should not find any sector.
- **Duplication:** If only the absence of sector is tested then it is easy to reproduce by software.
- **Emulation:** The preservation file needs to flag missing tracks (e.g. indicating 0 sector).
- **Examples:** [Barbarian](#) (Track 74 – 79 missing), Run the Gauntlet, [Kick Off 2](#)

Atari Floppy Disk Copy Protection

2.1.2 Shifted tracks (SFT)

Normally the first sector of a track starts shortly after the index pulse and the last sector of the track end-up before the next index pulse. On a normal track, the post-index GAP (at beginning of a track) is about 60 bytes and the pre-index GAP (at the end of a track) is about 600 bytes. In this case the track **write splice** (location where the floppy drive write gate is turned on/off) is located at the index.



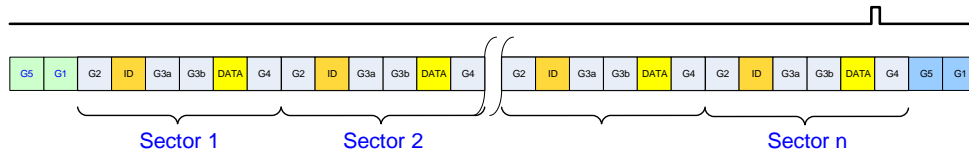
Sector positions relative to the index pulse for a normal track

Several protections shift the position of the track relative to the index. Note that in this case the [track write splice](#) is no more located at the index. The shifted track protections can be further sub-classified as explained thereafter but usually this is irrelevant for emulation.

*This type of protection is challenging for hardware copier. The copy should **not** be done from index to index as this will results in a track write splice in middle of a data segment. The copy should start from the first sector until the last sector using the correct shifted starting position with respect to the index.*

2.1.2.1 Data over index (DOI)

- **Description:** A sector where the *Data Field* span “over the index”. Normally all sectors of a track should end up before the index pulse. Yet it is possible to create a track with a total length that is slightly more than what a normal track can hold. This results in the last sector “wrapping around” the beginning of the track.



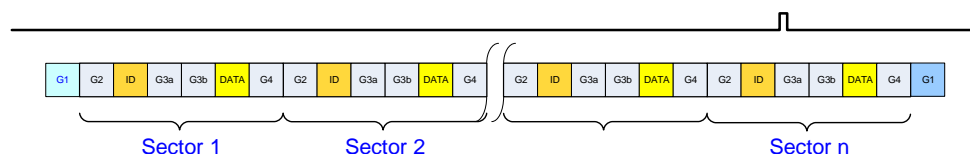
Sector positions relative to the index pulse for a track with Data over Index

- **Creation:**
 - ★ On Atari: it is possible to create a “long track” with a total length that is slightly more than what a normal track can hold (usually about 10 to 20 bytes). This is done by placing the header of the last sector close to the end of the track. The **write-track** command starts at the index pulse and continues until the next index pulse. Therefore the last sector will be **truncated** during the format (i.e. write track) operation. However the **write-sector** command on this truncated sector will execute normally and this will result in data being written over and beyond the index pulse.
 - ★ On Mastering machine: Normally writing a track is triggered by the index pulse. It is possible to shift the start of the write operation by some amount (for example time of 20 bytes) and of course to shift by the same amount the stop of the write operation.
- **Detection:** The last sector spread over the index pulse but it is read as a normal sector by a **read-sector** command. It is therefore necessary to use a **read-track** command to find out that the last sector actually wrap over the beginning of the track or to somehow measure the start position (timing) of the last sector.
- **Duplication:** Once detected the duplication of such sector can be done by formatting correctly the track.
- **Emulation:** Requires to store the track and/or sector position in the preservation file.
- **Example:** [Kick Off 2](#) places almost all the data of one sector at the beginning of a track.

Atari Floppy Disk Copy Protection


2.1.2.2 Data beyond index-pulse (DBI)

- **Description:** This is an extreme variation of the [Data over index](#) protection. Normally all sectors of a track should end up before the index pulse but it is possible to create a track where the *ID Field* for the last sector is placed at the very end of the track with the corresponding *Data Field* placed at the very beginning of the track. You have to remember that the Data Address Mark of the *Data Field* is to be found within 43 bytes from the last *ID Field* CRC byte and therefore placement of the *ID Field* and corresponding *Data Field* in the track is needs to be very accurate. The last sector “wraps around” the beginning of the track. See [Computer Hits Volume 2](#) for an example.



Sector positions relative to the index pulse for a track with data field beyond index

- **Creation:** It is almost impossible to position correctly such ID field on an Atari. Therefore this protection was usually created with mastering machines. The track is shifted so that the index pulse occur just at the end of the last ID field and of course the corresponding data field is located at the beginning of the track.
- **Detection:** This type of sector is read normally by the **read-sector** command. It is therefore necessary to use a **read-track** command to find out that the last sector actually spread over the beginning of the track or to measure the position of the last sector.

 **Note:** The DMA can only transmit multiple of 16 bytes from the FDC. Therefore during a **read-track** command, one or several of the last bytes (always less than 16) may **not** be transferred by the DMA. Consequently it is possible that a **read-track** do **not** transfer the *ID Field* (or transfer it partially) when it is placed at the very end of a track. However the FDC **read-address** and **read-sector** commands read the ID field for this sector correctly.

- **Duplication:** It is almost impossible to **reliably** place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to be reproduced correctly.
- **Emulation:** Requires to store the track content and/or sector position.
- **Example:** [Computer Hits Volume 2 \(Beau-Jolly\)](#)

2.1.2.3 ID over index (IOI)

- **Description:** A sector where the *ID Field* span “over the index”. This is a variation of the Data Over the Index-pulse protection. But in that case the index pulse happen **inside** an ID field. Please refer to the [Data Over Index-pulse](#) protection for more details.
- **Creation:** It is almost impossible to position an ID over the index on an Atari. Therefore this protection could only be created on mastering machines.
- **Detection:** It is usually not possible to read this ID using a **read track** command because the ID segment is at the very end of the track and usually some data read get stuck in the DMA buffer (see above). Even though this ID can’t be seen using a **read track** it can be read normally using **read address** and **read sector** commands.
- **Duplication:** It is almost impossible to **reliably** place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to reproduce the key disk.
- **Emulation:** Requires to store the track content and/or the sector position.
- **Example:** [Colorado](#), [Computer Hits Volume 2](#) disk 2.

Atari Floppy Disk Copy Protection

2.1.2.4 ID beyond index (IBI)

- **Description:** This is an extreme variation of the [ID over index](#) protection. In this case only the sync marks that belong to the last ID field are located before the index pulse but the rest of the ID fields and the corresponding data field wrap around the track's beginning.
- **Creation:** It is impossible to position an ID beyond the index on an Atari. Therefore this protection could only be created on mastering machines.
- **Detection:** It is usually not possible to read this ID correctly using a **read track** command because the sync of the ID segment are located at the end of the track and therefore not seen by the **read track** command. Even though this ID can't be seen using a **read track** it can be read normally using **read address** and **read sector** commands.
- **Duplication:** It is impossible to place an ID field at the very end of the track on an Atari due to floppy drives rotation speed variation. Therefore this protection requires specific hardware to reproduce the key disk.
- **Emulation:** Requires to store the track content and/or sector position.
- **Example:** [Computer Hits Volume 2](#) second disk.

2.1.3 Track Layout Pattern (TLP)

- **Description:** With the WD1772 FDC it is possible to slightly modify the layout of a track by varying the number of characters in the gaps in different position of the track (e.g. vary the length of the GAP4 placed between the different sectors). It is therefore possible to create a track with a specific layout pattern different from the standard pattern. This is a sort of floppy disk **water-marking** technique.
- **Creation:** It is quite easy to format a track with specific values for each GAPs by sending the appropriate information to the FDC during the **write-track** command.
- **Detection:** Measure the layout of the different fields of the track using the **read-track** command and look for a specific pattern. Note that some tolerance needs to be taken in account as the number of bytes reported for a specific gap may vary from read to read.
- **Duplication:** Once detected it is easy to duplicate by software.
- **Emulation:** Requires storing the track information in the preservation file.
- **Example:** Does not seems to be used on Atari?

2.1.4 Number of Sectors (NOS)

- **Description:** The standard Atari FD format uses tracks with 9 sectors of 512 data bytes. However many games use 10 or even 11 sectors per track just to pack more data on the diskette. However alone number different from 9 should not be considered as a protection. The following values are often used:
 - ★ Tracks with less than 9 sectors often use sectors with 1024 data bytes.
 - ★ Tracks with 11 sectors push several of the parameters that can be handled by the WD1772 FDC close to their limits. This is especially true considering that the IBM Floppy Drive standard allows a 3% rotation's speed variation. These tracks are therefore often referred as "**read only**" because once written they can't be modified. This is due to very low number of bytes used in the GAP fields that does not allow for the write sector command to work correctly.
 - ★ Tracks with 12 or more sectors (e.g. 70!) clearly indicate that some "tricks" have been used as 12 real sectors **won't fit** on a track.
- **Creation:** Up to 11 is possible in software, but remember that with 11 sectors it is almost impossible to write data **consistently** without using special hardware.
- **Detection:** Easy with multiple **read-address** command.
- **Duplication:** Easy in software for a number of sectors per track up to 10. Duplicating track with 11 sectors is possible but more challenging.
- **Emulation:** Requires nothing special the preservation file just needs to store the data information for all the sectors of the track using **read-sector** commands.
- **Examples:** [Computer Hits Volume 2](#): 11 sectors / track, [Theme Park Mystery](#): 12 sectors / track, [Sherman M4](#): 70 sectors / track.

Atari Floppy Disk Copy Protection

2.1.5 Sector Sizes (SSZ)

- **Description:** Normally the tracks have sectors with 512 bytes long *Data Field*. But it is possible to create a track with different data field size (usually a mixture of 512 and 1024)¹. This is a more reliable approach to increase the overall capacity of a track rather than using 11 sectors of 512 bytes. Non-standard sector size are not be considered as a protection. Two common examples of format with different sector size are:
 - ★ 9 sectors of 512 bytes plus 1 sector with 1024 bytes, and
 - ★ 5 sectors of 1024 bytes plus 1 sector with 512 bytes.
- **Creation:** Easy on Atari.
- **Detection:** Easy with multiple *read-address* command.
- **Duplication:** Easy on Atari.
- **Emulation:** Requires nothing special the preservation file just needs to store the data information for all the sectors of the track using *read-sector* commands.
- **Examples:** [Kick Off 2](#), [Turrican](#) uses tracks with a mixture of 1024 and 512 bytes sectors.

2.1.6 Invalid ID Field (IIF)

An *ID Field* contains the following information after the ID Address Mark: the **Track Number**, the **Side/Head Number**, the **Sector Number**, the **Sector Length**, and two **CRC** bytes. To understand these protections you need to know that during a *read-sector* command when an *ID Field* is located on the disk, the WD1772 compares the Track Number of the *ID Field* to its internal Track Register. If there is no a match, the next *ID Field* is read and a comparison is made again. If there is a match, the Sector Number of the *ID Field* is compared with its internal Sector Register. If there is no Sector match, the next encountered *ID Field* is read off the disk and a comparison is made again. If both matches and if the *ID Field* CRC is correct, the sector is located and an internal register is loaded with the Sector Length. Invalid ID field can further be decomposed:

2.1.6.1 Non-standard IDAM (NSI)

- **Description:** The normal IDAM (ID Address Mark) used by the WD1772 is the character **\$FE** which is sent after a sequence of 3 **\$A1** sync marks. An undocumented feature of the WD1772 is that it accepts any character in the range **\$FC-\$FF** as an IDAM².
- **Creation:** During a *write-track* command it is possible to use any value in the range **\$FC-\$FF** instead of the normal **\$FE** IDAM character.
- **Detection:** As the *read-address* command and the *read-sector* command execute normally it is easy to hide the fact that a non-standard IDAM has been used. Detection can be done using a *read-address* command.
- **Duplication:** Once detected this protection is easy to duplicate.
- **Emulation:** Requires to store the track information as well as the address information.
- **Example:** [Z-out](#)

¹ Note that several of the BIOS calls will **not** work for sectors with size different than 512.

² Note that, in MFM, for the marks characters between **\$F8** and **\$FF** the least significant bit is always ignored by the WD1772 and therefore : **\$F8 = \$F9**, ..., **\$FE = \$FF**

Atari Floppy Disk Copy Protection

2.1.6.2 Invalid track number (ITN)

- **Description:** A sector with an *ID Fields* that contains a track number different from the actual track number (in FDC register). In order for the *type I commands* (e.g. **seek**) to succeed, on such a track, the verify bit has to be reset. Otherwise the FDC check that at least one sector has the correct track number. The **read-sector** command using “standard” parameters will also fail.
- **Creation:** Use **write-track** command with incorrect track number in *ID Field*.
- **Detection:** The **read-sector** command compares the track number of the *ID Field* with the track register if this matches it then compares the sector number of the *ID Field* with the sector register. If any compare operation fails the FDC retry 5 times then terminate the command with a record not found (RNF) error. Reading this kind of sector is possible but requires playing with the FDC registers (i.e. loading the track register with invalid value).
- **Duplication:** Easy by software
- **Emulation:** The preservation file should store the exact ID block.
- **Example:** [Star Glider 2](#), [Dragonflight](#)

2.1.6.3 Invalid head number (IHN)

- **Description:** An *ID field* with an invalid Side/Head Number (i.e. not equal to 0 or 1). Normally this field is supposed to be equal to the side you are reading however it should be noted that the WD1772 does not use this information so any value can be used.
- **Creation:** It is possible to write invalid values for the Side Number of an *ID Field* by sending the appropriate data to the FDC during a **write-track** command.
- **Detection:** Use a **read-address** command and compare the side value.
- **Duplication:** Can easily be done by software
- **Emulation:** The exact content of the ID field need to be saved in the preservation file.
- **Example:** [Star Glider 2](#), [Dragonflight](#)

2.1.6.4 Invalid sector number (ISN)

- **Description:** During the format command the character loaded into the data register of the WD1772 is written to the disk. However the characters \$F5 and \$F6 are used to write respectively the *Sync Characters* \$A1 and \$C2 with a missing clock transition and the character \$F7 is used to generate two CRC bytes. This implies that it is not possible to create a sector with an ID ranging from 245 through 247 (\$F5-\$F7). In fact the WD1772 documentation indicates that the sector number should be kept in the range 1 to 240.
- **Creation:** It is **not** possible to create a sector with an ID in the range of 245-247 with the WD1772 FDC and therefore creating such *ID Field* requires specific hardware.
- **Detection:** Can easily be done with a **read-address** command.
- **Duplication:** Requires special hardware.
- **Emulation:** The sector with an invalid ID number is read as a normal sector by a **read-sector** command and stored in the preservation file like any other standard sector.
- **Example:** [Dungeon Master](#) (FTL Inc.) use a sector number of **247** (\$F7) on track 0

It is actually possible to write a byte between \$F5-\$F7 inside an ID field using the escaping capability of the WD1772 see [WD1772 MFM track language](#).

Atari Floppy Disk Copy Protection

2.1.6.5 Invalid sector length (ISL)

- **Description:** An *ID field* with an invalid Sector Length (i.e. not in range 0-3). Normally this field is supposed to take the value 0, 1, 2, 3 corresponding to respectively 128, 256, 512, 1024 data bytes size. As the WD1772 only uses the last three bits of the sector length information it is possible to write sector length value larger than 3. For example 0x03 and 0xFF are equivalent.
- **Creation:** It is possible to write invalid values for the Sector Length of an *ID Field* by sending the appropriate data to the FDC during a **write-track** command.
- **Detection:** Use a **read-address** command to get all the fields.
- **Duplication:** Can easily be done by software.
- **Emulation:** The exact content of the ID field need to be saved in the preservation file.
- **Example:** [Star Glider 2 Z-Out](#).

2.1.6.6 ID CRC Error (ICE)

- **Description:** A sector that has a CRC error in the *ID Field*. This results in a sector that cannot be read by the **read-sector** command.
- **Creation:** Easy with the **write-track** command. For example by sending 2 normal bytes (e.g. \$00, \$00) at the end of the field instead of one "Write CRC" character (\$F7).
- **Detection:** It is possible to read this kind of sector ID field with a **read-address** command and to verify that it has a wrong CRC. But it is not possible to read the sector with a **read-sector** command.
- **Duplication:** Can easily be done by software
- **Emulation:** Requires to store the complete track and address information in the preservation file.
- **Example:** xxx

2.1.7 Duplicate Sector Number (DSN)

- **Description:** A track where, two (or more) sectors use the same sector's number. Using blindly a **read-sector** command, for this duplicated sectors, result in reading randomly one of the two sectors based on current head position. In order to read a specific one, it is necessary to issue a **read-sector** command delayed by a specific amount of time from the *index pulse*. Usually, to facilitate the detection, these two sectors are placed well apart (e.g. at the beginning and the end of the track). Sometimes the second ID field is not followed by a corresponding data field ([no sector block](#) protections).
- **Creation:** Easy in software.
- **Detection:** Easy by using **read-address** and/or **read-track** commands.
- **Duplication:** Easy in software.
- **Emulation:** The information for all sectors including the duplicate sector needs to be saved. It is also necessary to store the position of the sector in the track.
- **Example:** [Night Shift](#) uses a duplicated sector numbered 66 (the duplicated sectors also use the [no data block](#) protections).

Atari Floppy Disk Copy Protection

2.1.8 Sector within sector (SWS)

- **Description:** During formatting it is possible to place a new sector that overlap with a previous one. Therefore when reading these sectors we have the impression that the second sector is located within the first one. The layout of a first sector contains the fields: GAP2-**ID Field**-GAP3-**Data Field**-GAP4. The included sector has its own GAP2-**ID Field**-GAP3-**Data Field** placed inside the **Data Field** of the including sector. This is possible because during a **read-sector** command the sync mark detector of the WD1772 is turned off and therefore the included field are treated as normal data (sync sequence not recognized). A detailed explanation of this protection can be found in the [Theme Park Mystery](#) example. An even more complex variant is to have a sector within another sector which is itself located within another sector (SWSWS). Even with such a complex layout it is possible to read correctly an “included sector”! For an example of SWS-WS-WS look at [Computer Hits Volume 2](#). It is also possible to shift by one bit-cell the included sector in respect to the including sector. This trick allows to read data bits as well as clock bits of the overlapped data field as in [Turrican](#) to check presence of [NFA](#).
- **Creation:** Only possible in specific cases on Atari and therefore usually requires usage of specific hardware.
- **Detection:** The **read-address** command allows to read the ID fields of the including and included sectors. The **read-sector** command reads the including sector beyond the start of the included sector because during a **read-sector** command the sync mark detector of the WD1772 is turned off. The included sector is read normally as if no including sector was placed before. Usually look for this protection when a track has a number of sector equal or exceeding 12. To confirm this protection you can use a **read-track** command. Another alternative is to check the data inside the including sector's *Data Field* and look for GAP2 followed by an *ID Field* etc. However beware that this will not always work due to the way the FDC works. For example it is not possible to find the ID and DATA field of sector 16 inside sector 0 of track 2 of [Computer Hits Volume 2](#) because it is shifted.
- **Duplication:** Require special hardware. Often combined with other protections like [NFA](#).
- **Emulation:** Once the protection is detected the preservation program should store the track layout and the information about the including and following sectors.
- **Example:** [Theme Park Mystery](#), [Computer Hits Volume 2](#), [Turrican](#), Nitro Boost Challenge

2.1.9 Non Standard DAM (NSD)

- **Description:** The normal DAM (DATA Address Mark) used by the WD1772 is either the character \$FB for *normal data* and \$F8 for *deleted data* which is sent after a sync sequence of 3 \$A1 sync marks. An undocumented feature of the WD1772 is to accept the any character \$F8-\$FB as a DAM (see also [Non Standard IDAM](#)).
- **Creation:** During a write-track command it is possible to use \$FC or \$F9 instead of the normal \$FB or \$F8 DAM character.
- **Detection:** As the **read sector** command execute normally it is easy to hide the fact that a non-standard DAM has been used. Detection can be done through a **read track** command where you have to look for a \$FC/F9 character instead of \$FB/F8 in the header of the *DATA field*. Note that when an alternate DAM is used, the DATA Field still reads without a CRC error.
- **Duplication:** Once detected this protection is easy to duplicate.
- **Emulation:** Requires storing the complete track in the preservation file.
- **Example:** No example found

Atari Floppy Disk Copy Protection

2.1.10 Sector with No ID (SNI)

- **Description:** A sector with a *Data Field* but not preceded by an *ID Field*.
- **Creation:** on Atari it is quite easy to format a sector of a track with a *DATA field* not preceded by an *ID Field* using a ***write-track*** command.
- **Detection:** There is no way to read this kind of sector with a ***read sector*** command. Therefore the only way to detect the presence of such data field is by using a read track command. Therefore this kind of sector it is very rarely used.
- **Duplication:** Can easily be done by software.
- **Emulation:** Requires storing the track information in the preservation file.
- **Example:** Gunship (D1 from Air Supremacy Compilation), [Vroom](#) after sector 106 has a fuzzy SNI (see [Fuzzy Track \(FZT\)](#))

2.1.11 Sector with No Data (SND)

- **Description:** A sector with an *ID Field* but not followed by a *Data Field*.
- **Creation:** on Atari it is quite easy to format a sector of a track with an *ID field* not followed by a *Data Field* using a ***write-track*** command.
- **Detection:** This kind of sector is found using a ***read-address*** command, but is not found using a ***read-sector*** command. This is because during the ***read-sector*** command the FDC expects to find a DAM/DDAM within 43 bytes from last *ID Field* CRC byte, if not the sector data is searched again for 5 revolutions and the command is terminated with the Record Not Found (**RNF**) Status bit set.
- **Duplication:** Can easily be done by software.
- **Emulation:** Requires storing the track information in the preservation file.
- **Example:** [Night Shift](#) uses [duplicate sectors](#) 66 both of them having No Data fields

2.1.12 Data CRC Error (DCE)

- **Description:** A sector that has a CRC error in its *Data Field*.
- **Creation:** Easy during ***write-track*** command by using the same mechanism as described in [Invalid ID CRC](#).
- **Detection:** Can easily be done using a ***read-sector*** command. The data sector is *read normally* but the CRC error status bit is set at the end of the command.
- **Duplication:** Can sometimes be done in software.
- **Emulation:** The content of the sector should be stored as normal but the CRC error indicator must be added to the preservation file.
- **Example:** [Populous](#)

Atari Floppy Disk Copy Protection

2.1.13 Data Track (DTT)

- **Description:** This kind of track does not contain the Atari standard ID / Data / Gap fields. The track is usually composed of a special Header field followed by a Single Data field. In order to be read correctly the Header needs to be preceded by 3 \$A1 sync marks. The only way to read the Single data field is to use a **Read Track** command. Remember that during a **Read Track** command the sync detector of the WD1772 is active at all time and therefore any MFM sequence of bits that contains 0x000101001 will cause the FDC to resynchronize and consequently the data are not read correctly after that. To avoid a resynchronization an escape character (often 0x07 or 0x0F) is inserted whenever the input data contains this sequence. When the track is read the escape characters are removed to get back the original data.
- **Creation:** As the Data record can contain "invalid code" (i.e. code like 0xF5-0xF7) it can't be written using a **Write Track** command. It is therefore mandatory to use special hardware to write this kind of track.
- **Detection:** A **Read Track** command is used. The software looks for at least three 0xA1 then decodes the rest of the Header and then reads the data record according to parameter passed in the header. A checksum is often added to the data field and can be used to verify that the data record has been read correctly.
- **Duplication:** Not possible in software requires special hardware.
- **Emulation:** For emulation it is necessary to save the complete content of the track as read by the **Read Track** command.
- Example: [Maupiti Island](#) (escape character 0x07), [Golden Axe](#), Hot Rod, International Soccer (escape character 0x0F), Albedo

It is even possible to split the track into several "pseudo-sectors". For example in Albedo the track is split into 5 pseudo-sectors

2.1.14 Hidden Data into GAP (HDG)

- **Description:** It is possible to write hidden data into any gap. However hidden data are usually placed in the post DATA Gap (Gap of 40 bytes) as well as in the pre and post index GAP (respectively 664 and 60 bytes on standard diskettes). See "[copy me I want to travel](#)" from [Claus Brod](#) for a complete explanation and some interesting examples. There are some known sequences described in [Hidden data using spurious sync sequence](#).
- **Creation:** Extra data can be written into Gap only during the **write-track** command. It is recommended to use **Sync Marks** in front of the data to be able to read them correctly.
- **Detection:** You need to use a **read-track** command to be able to read the inter-sector information. But it is hard to find this information if you do not know what and where to look for. Therefore some heuristic needs to be used (e.g. presence of sync marks into GAP).
- **Duplication:** Although it is difficult to detect, it is easy to reproduce with the **write-track** command.
- **Emulation:** Requires storing the track information in the preservation file.
- Example: [Jupiter Masterdrive](#), Dragonflight, Union Demo

2.1.15 Hidden data into nonstandard tracks (HDT)

- **Description:** It is possible to hide data into a nonstandard track.
- **Creation:** Only possible on an Atari if no invalid bytes are used.
- **Detection:** Use the **read track** command.
- **Duplication:** Not possible if invalid bytes are used.
- **Emulation:** Requires storing the track information in the preservation file.
- Example: [Realm of the Troll](#) track 79.0

Atari Floppy Disk Copy Protection

2.1.16 Invalid Data in Gap (IDG)

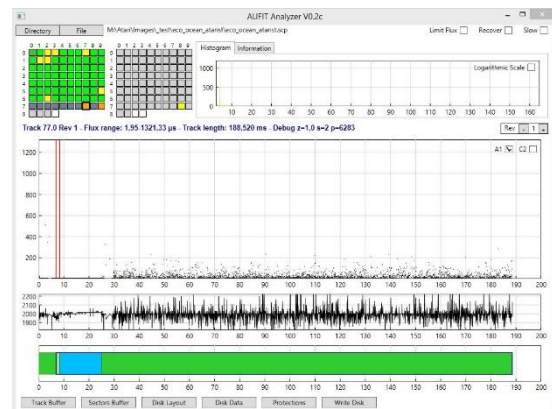
- **Description:** During the format command character loaded into the data register of the WD1772 is written to the disk. However the characters \$F5 and \$F6 are used to write the Sync Marks and the character \$F7 is used to generate of two CRC bytes. This implies that it is not possible to have a character ranging from 245 through 247 (\$F5-\$F7) inside any of the GAPS³. Reading these characters into GAPS requires using a **read-track** command. In order for these invalid characters to be read correctly with a **read-track** command they are usually preceded by one or several **sync** character. Be aware that the byte \$F7 can be used to escape special character (see [WD1772 MFM track format language](#)).
- **Creation:** It is **not** possible with the WD1772 to write a character within the range 245-247 into any GAP. Therefore writing invalid character into GAPS requires mastering machines.
- **Detection:** Can easily be done with a **read-track** command.
- **Duplication:** Require special hardware.
- **Emulation:** It is necessary to save the complete content of the track.
- **Example:** [Operation Neptune](#) & [Bob Morane](#) uses 0xF7 as gap bytes

2.1.17 Invalid Sync-mark Sequence (ISS)

- **Description:** A normal Sync mark sequence is composed of 3 Sync Marks (3 x \$A1 or 3 x \$C2) followed by an Address Mark (IAM = \$FC, IDAM = \$FE, DAM = \$FB, or DDAM = \$F8). Any other sync sequence is considered as invalid. Note that an invalid sequence is usually used to sync up the data separator in order to read [data into gap](#) or for the [Data track](#) protection. But it is also abnormal to have less than 2 or more than 3 Sync Marks in sequence. See also [Invalid Sync sequence](#).
- **Creation:** It is quite easy to create an invalid sync mark sequence during format by sending appropriate information to the FDC using the **write-track** command.
- **Detection:** Only possible with the **read-track** command as the **read-sector** command just ignore invalid sync mark sequences.
- **Emulation:** Requires storing the track information in the preservation file.
- **Duplication:** Easy by software.
- **Example:** [Barbarian](#) (one Sync alone on Track 0, series of Sync on Track 48 & 62)

2.1.18 Partially formatted track (PUT)

- **Description:** Inside what looks like an unformatted track it is possible to hide a sector.
- **Creation:** This kind of track can only be created using special hardware.
- **Detection:** The program verify that it can only reads the known sector and that no other sector exist.
- **Emulation:** Requires to store the content of the **read track** command in the preservation file.
- **Duplication:** Requires special hardware.
- **Example:** [Eco](#) tracks 77 & 79

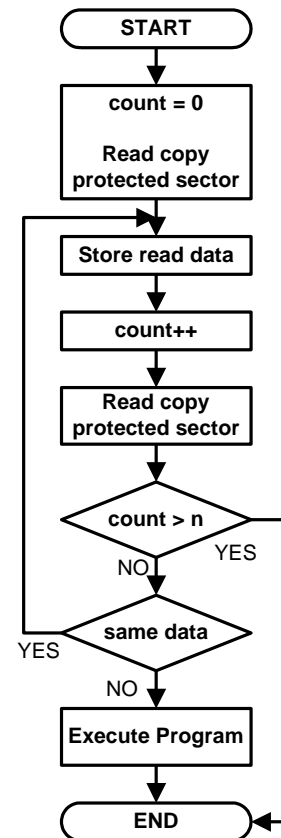


³ Note that it is not possible to modify the GAP2 or GAP3b (\$00). Therefore writing hidden bytes must be done in GAP1 and/or GAP3a and/or GAP4

Atari Floppy Disk Copy Protection

2.1.19 Fuzzy Sector (FZS)

- **Description:** A sector that contains [fuzzy bits](#). Reading this sector several times returns different data.
- **Creation:** Cannot be created on Atari, requires mastering machines. Please refer to the [fuzzy bits](#) section.
- **Detection:** The flowchart on the right describes a copy recognition routine that tests for fuzzy bytes in the data field (patent 4,849,836). The protected sector that contains fuzzy bytes is read several times and randomness of the returned data is checked. If the same data is read several times on the protected sector the program is not executed. Very often, as in *Dungeon Master*, the protection is verified several times during execution of the game/program.
- **Duplication:** Difficult and requires special hardware.
- **Emulation:** The preservation file should have an indicator to record the fact that a sector has Fuzzy bytes. Usually the first and last 32 bytes of a fuzzy sector do not contain fuzzy bytes. It is also good to store information about bits that have changed in the different read operations.
- **Example:** TODO



2.1.20 Fuzzy Track (FZT)

- **Description:** This is somewhat similar to [Fuzzy Sector](#): the protected track that contains [fuzzy bits](#) is read several times and randomness of the returned data is checked. This is usually done in specific areas as explained below.
- **Creation:** Cannot be created on Atari, requires special hardware. Please refer to the [fuzzy bits](#) section.
- **Detection:** If you know the location of the fuzzy bytes, it is easy to read the same data several times and to check that returned data are different. However detecting fuzziness in a read track without specific information is difficult because there are many reason why a read track returns random data in several places. For example the beginning of a track reads differently until the first sync because the position where the read track starts vary.
- **Duplication:** Difficult and requires special hardware.
- **Emulation:** The preservation file should have an indicator to record the fact that a track has a Fuzzy data track. Note that Pasti STX does not support this kind of protection.
- **Examples:** [Power Drift](#) (track 1 side A of floppy disk 2). [Vroom](#).

Atari Floppy Disk Copy Protection

2.2 *Protections based on timing*

This section describes the protections based on variations of the standard 4 µs cell bit-rate. Although different techniques are used, the result of using bit-rate variation is always the same (with the exception of NFA): the overall time-length of a byte read from the drive, is different from a “normal 32 µs byte”. Therefore detection of this protection requires to be able to measure timing information when reading the block of bytes that compose a sector.

2.2.1 Long / Short Sector (LGS & SHS)

- **Description:** This kind of sector can be created by writing a sector of a track with an apparent rotation speed of the drive slightly above or below the normal speed. In practice this is obviously not done by varying the rotation speed of the drive but by changing the bit-cell clock. This results in a reading time for this sector above or below the reading time of a “normal sector”. The IBM standard specifies that the FDC circuitry should handle a variation of the drive’s rotation speed within $\pm 2\%$ range. Therefore the DPLL of a FDC is supposed to accept at least a 4% variation. But in practice the WD1772 DPLL (See [WD1772 DPLL Input Circuitry](#)) can handle at least 10% variation for MFM encoding (as described in this DPLL [Patent](#)). It is therefore possible to write sectors with bit cells at frequencies between 225 and 275 KHz (corresponding respectively to 3.6 to 4.4 µs bit width) and to still be guaranteed to read the data correctly. However the resulting sector will be longer or shorter than a normal sector. The most famous usage of this protection was done by Rob Northen in the **Copylock** (RNC) protection mechanism⁴ (see [an interview with Rob Northen](#)): in this case the bit width is changed to approximately 4.2µs (about 4 to 5% variation) to result in a shorter sector. The beginning of the sector (for about 32 bytes) is written at normal speed so that we are sure that the data in this section are always read correctly.
- **Creation:** Cannot be done on an Atari. It requires mastering machines with the capability to vary the bit cell width on the fly.
- **Detection:** can’t be done with standard TOS call. It requires to use specific routines to measure the time it takes to read the bytes in the short/long sector.
- **Duplication:** Difficult and requires special hardware.
- **Emulation:** The preservation file should store timing information about the sector.
- **Example:** [Populous](#) - Track 0 Sector 6, Back to the Future (T0-S6)

⁴ According to vauvillf: there has been 2 RNC. The old one used for example on Arkanoid2, and Thundercats... It was possible to copy RNC-1 with the **acopy** program (only 2 to 3 times). Then there was a big evolution of the RNC protection sometime in 1988: with this one it was no more possible to copy the protection by software, and it was also using the famous trace decoding loop. Apparently the description provided here refers to the RNC-2 protection.

Atari Floppy Disk Copy Protection

2.2.2 Long/Short Track (LGT & SHT)

- **Description:** This kind of track can be created by writing all bytes of a track with an apparent rotation speed of the drive slightly above or below the normal speed. This results in a track that contains more or less bytes than a normal 6240 bytes track. In practice this is obviously not done by varying the rotation speed of the drive but by changing the bit-cell width. The IBM standard specifies that the FDC circuitry should handle a variation of the drive's rotation speed within $\pm 2\%$ range. Therefore the DPLL of a FDC is supposed to accept at least a 4% variation. But in practice the WD1772 DPLL (See [WD1772 DPLL Input Circuitry](#)) can handle a 10% variation for MFM encoding (as described in the DPLL [Patent](#)). It is therefore possible to write sectors with bit cells at frequencies between 225 and 275 KHz (corresponding respectively to 3.6 to 4.4 μ s bit width) and to still read the data correctly.
- **Creation:** It requires special mastering machines that can vary the bit cell width on the fly.
- **Detection:** You can use a **read track** command. The normal track length is around 6240 bytes and it is sufficient to check that the track has more (or less) than a 5% above the nominal value (e.g. less 6027 in Arkanoid).
- **Duplication:** Difficult and requires special hardware.
- **Emulation:** The preservation file should store timing information about the track as well as the number of bytes of the track.
- **Example:** Arkanoid , Indiana Jones Last Crusade, Guntlet II, Garfield, Speedball
Awesome (T79 < 6000 bytes)

2.2.3 Sector Bit-rate Variation (SBV)

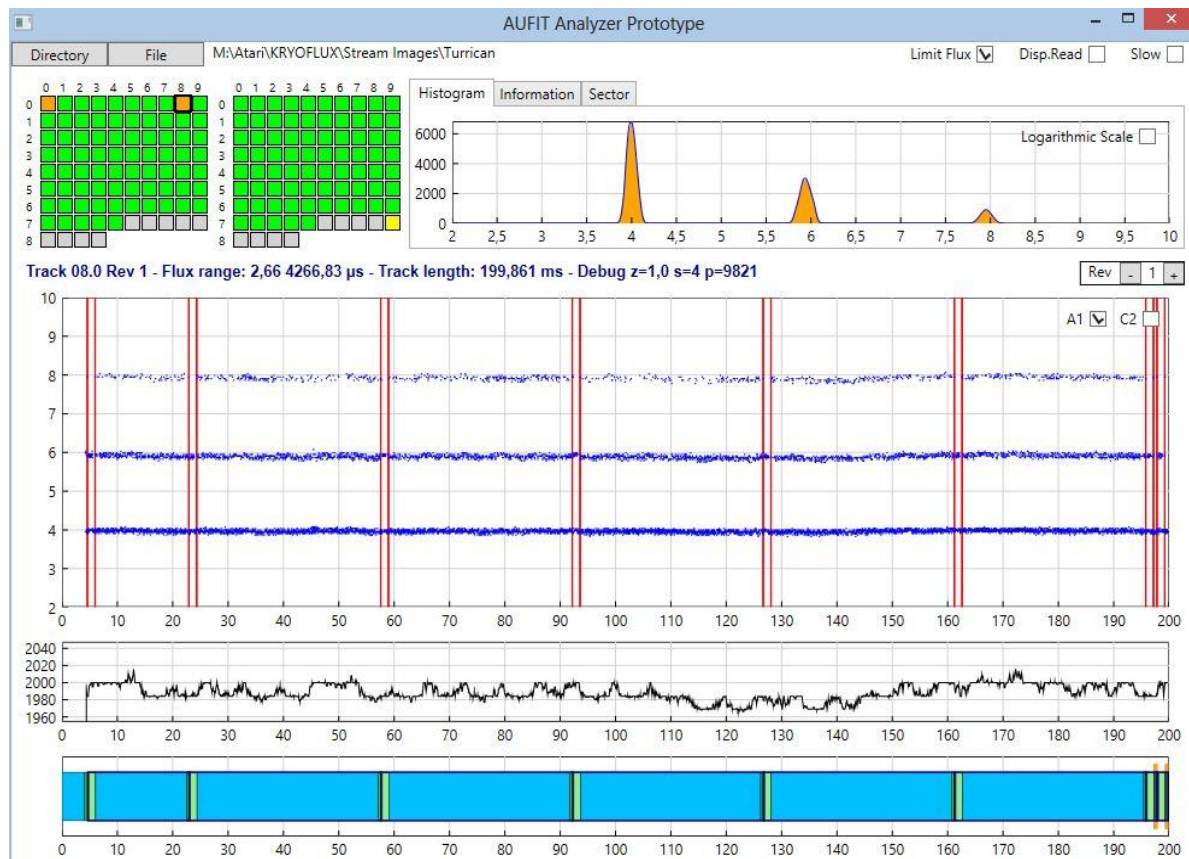
- **Description:** This is a more difficult to detect bit-rate variation. A sector is divided into several segments. Each of them uses a "drive rotation speed" slightly above or below the normal speed. By using faster and slower segments in the same sector it is possible to have the timing of these segments to compensate resulting in a sector with a normal overall timing. For example the **MacroDOS** protection from *Speedlock Associates* divides a sector into 4 segments with normal-faster-slower-normal rotation speed resulting in an overall standard time length.
- **Creation:** Requires special hardware that have capability to vary the bit width.
- **Detection:** It is quite difficult to detect this protection because the overall sector length is the "normal" length. It is therefore necessary to measure the timing of blocks of characters (usually multiple of 16 using DMA transfer) that compose a sector and to compare them to standard block length to check for specific above or below patterns.
- **Duplication:** Require specific hardware
- **Emulation:** The preservation file should store detail timing information about the sector. On Atari it is only possible to store timing information about reading a 16 bytes block.
- **Example:** [Golden Axe](#), [Colorado](#), Starblade, Treasure Trap, Damocles

Atari Floppy Disk Copy Protection

2.2.4 No Flux Area (NFA)

- **Description:** A track that contains a *very long area without reading flux transitions*. Note that this is quite different from an unformatted area (no flux transitions recorded) because reading an unformatted area return many random flux transitions due to the fact that the gain of the amplifier (ACG) on the read channel is pushed to its maximum resulting on picking up noise on the head. In order to produce such area some tricks needs to be used as explained in the [No Flux Area on Disk](#) section. This is difficult to produce even with specialized hardware.
- **Creation:** Requires specific hardware.
- **Detection:** No Flux Area result in reading 0x0000 MFM word in the FDC shift register (no clock transition and no data transition). However the WD1772 FDC only allow to read the data bytes of the MFM word but not the clock bytes. It is therefore not possible to directly check that the clock bytes in an NFA are also null. This is why the NFA protection places the no flux area in a sector within another sector, where the included sector is shifted by a half-cell. The including sector allows to read the “data part” of the NFA and the included sector allows to read the “clock part” of the NFA. For more information refer to [Checking NFA with the WD1772](#) section.
- **Duplication:** Difficult and requires special hardware.
- **Emulation:** The preservation file need to save the track data and also needs to save the two sectors that allow to read the data and the clock.
- **Example:** [Turrican](#).

Here is an example of a NO Flux Area that is located over the index. As indicated the NFA is 4.27ms long, starts before the end of the track, and wrap around the index.



Atari Floppy Disk Copy Protection

Chapter 3. Preservation of Atari floppy disks

Information presented in this document about **protection mechanisms** can help in the design of techniques/programs for **duplication** or **preservation** of original Atari diskettes with the following philosophy:

-
- ✍ A preservation technique should always do the most to ensure the integrity of the preserved data. The preserved data should operate just like the original and not remove any protection, or modify the program being preserved in any way. The preservation technique must do the up most to check that the preserved data is identical to the original.
-

Specially designed programs can duplicate key disks for many of the “simple” protections presented here. But duplication of key disks using more advanced protections requires using specially designed hardware like the vintage **Discovery Cartridge** or the recently released **KryoFlux** and **SuperCard Pro** devices. Analog hardware copiers, like the **Blitz** cable and associated software, can sometime create a working copy of a protected diskette but they **do not fulfill** the above requirements of producing a copy identical to the original.

Preservation has different meanings for different people but it can be classified into two categories:

- A “real preservation” is intended to save all the required information from a floppy disk so that it is not only possible to emulate the original FD but it is also possible to physically duplicate the original FD. For example the files produced by the **Discovery Cartridge**, the **KryoFlux**, and the **SuperCard Pro** devices allow to emulate or to backup protected disks.
- An “emulation preservation” is intended to save enough information from a floppy disk so that it is possible to emulate the behavior of the original FD in a software or hardware emulator. For example the files produced by the **Pasti** imager allow to emulate protected disks. However it is not possible to recreate a FD from Pasti files.

It is interesting to note than most emulation / duplication programs do *not care* about (and sometimes can't detect) the detailed underlying protection mechanisms used. They just store enough information to replicate the effect of a specific protection. For example they detect [fuzzy bytes](#) but they do not care if they result from [bits in Ambiguous areas](#), or from [bits rate violation](#).

In the following sections we are going to explain how to correctly use several devices specially designed to preserve Atari floppy disks.

3.1 Cleaning a floppy disk to create correct image

Here are some basic rules to follow to create the best possible image:

- Use a known good original: Always use original disk that has not been modified.
- Write protect your original: In order to keep an unmodified disk always make sure that the original have the protect notch in the correct position at all time you use the disk including during the imaging operation.
- Clean your original: Atari floppy disks games are getting very old. They are prone to be dirty even if not used too much because of the environment. This results in deteriorated magnetic signal picked up by the read head. Carefully clean your disks with rubbing alcohol and cotton swabs. Rotate the disk in its jacket, cleaning the surface until no more residue is found on a clean cotton swab.
- Clean your floppy drive head: After reading several disks the head will have accumulated a lot debris. Clean the drive's head with a commercial head cleaner or by using the same rubbing alcohol and cotton swab technique used to clean your disks.

3.2 Why do we need several revolutions for preservation?

You might be tempted to sample flux transition for only one revolution in order to save space on hard disk. However for **preservation** this **won't work** for the following reasons:

Atari Floppy Disk Copy Protection

- For duplication you can usually sample the flux transitions of only one revolution. You should get a perfect backup of the original floppy if
 - ★ your original is in **perfect condition**,
 - ★ you have set all parameters of your imaging device correctly, and
 - ★ If the floppy you use for the backup is also in perfect condition.
- For preservation you **must** sample the flux transitions of **at least three revolutions**. But it is recommended to sample **five revolutions** in order to be able to verify the integrity of the sampled data as explained below.

The rationale for using five revolutions is the following:

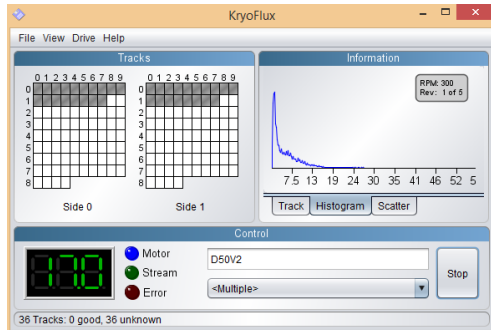
- ★ By **definitions** fuzzy bytes are detected by reading several times the same bytes and comparing if the values are different. Therefore this kind of protection implies to sample at least two revolutions but three or more is preferred (majority rule).
- ★ Many Atari games use protections based on [shifted tracks](#). In such a case the region “under” the index belongs to an ID or a DATA field and therefore it is not possible to start reading or writing data at the position of the index (this must be done at the location of the [track write splice](#)). Therefore this kind of protection implies to sample at least two revolutions. The combination of the last two requirements result in the necessity to sample at least three revolutions.
- ★ Because of the age of the floppy disks, the magnetic signal picked up by the read head is often distorted. A program like AUFIT uses an advanced DPLL algorithm that allows to recover many imperfections on the read flux transitions but unfortunately this is not always sufficient. By sampling extra revolutions it is possible to combine data from multiple revolutions to recover the original information. For example AUFIT is able to select and use a correct sector (one with a good CRC) among multiple revolutions. The more revolutions you have imaged the more chances you have to recover!

Therefore based on the above if you want to reliably preserve information from a floppy disk it is recommended that you use 5 revolutions.

Atari Floppy Disk Copy Protection

3.3 Kryoflux short presentation

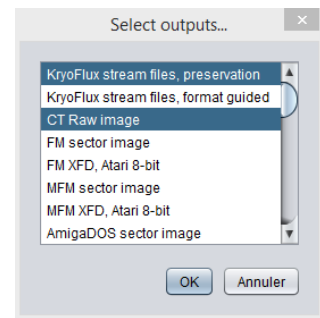
Using [Kryoflux](#) for preservation is pretty simple. Start the DTC GUI (kryoflux-ui.jar) and in the **select output** field of the control section select **multiple**. This open a new window and here select **Kryoflux stream files, preservation** and **CT Raw image**. This will



save the stream RAW files in a separate directory as well as the CTA raw file.

By default Kryoflux device settings are set to preserve 5 revolutions and sample up to the maximum track number.

Therefore you just need to specify the image path (in file settings), the output name and start imaging.

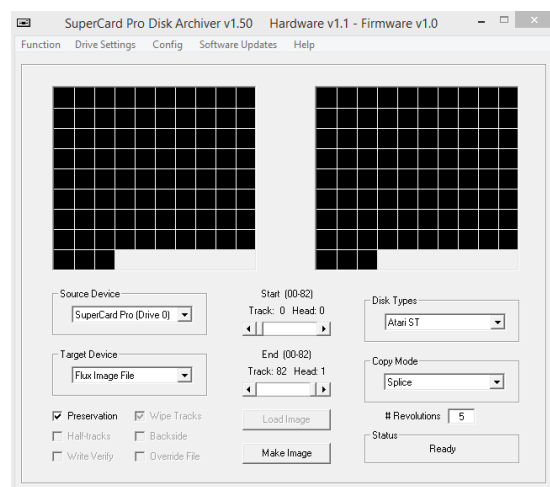


3.4 Supercard Pro short presentation

[Supercard Pro](#) can be used to just backup (duplicate) Atari floppy disks or it can be used for real preservation. These two usages have different requirements:

- For duplication you can usually sample the flux transitions of only one revolution. If your original is in perfect condition, if you have selected the correct mode, and if the floppy you use for the backup is also in perfect condition then you should get a perfect backup.
- For preservation you must sample the flux transitions of **five revolutions** in splice mode.

For Supercard Pro you must change the **# Revolutions** value to 5 and the end track number to 82 **each time** you want to preserve a floppy as these values are not automatically saved (even using the **save configuration** command).



Atari Floppy Disk Copy Protection

The table below indicates the values of the different gaps usually used for standard Atari diskette with 9 sectors of 512 user data bytes. It also indicates the minimum acceptable values (as specified in the WD1772 datasheet) of these gaps when formatting nonstandard diskettes.

NAME	STANDARD VALUES (9 SECTORS)	MINIMUM VALUES (DATASHEET)
Gap 1 Index postamble	60 x \$4E	32 x \$4E
Gap 2 ID preamble	12 x \$00 + 3 x \$A1	8 x 00 + 3 x \$A1
Gap 3a ID postamble	22 x \$4E	22 x \$4E
Gap 3b Data preamble	12 x \$00 + 3 x \$A1	12 x \$00 + 3 x \$A1
Gap 4 Data postamble	40 x \$4E	24 x \$4E
Gap 5 Index preamble	~ 664 x \$4E	16 x \$4E

Standard Sector Gaps Value (Gap 2 + Gap 3a + Gap 3b + Gap 4) = 92 Bytes / Sector

Minimum Sector Gaps Value (Gap 2 + Gap 3a + Gap 3b + Gap 4) = 72 Bytes / Sector

Standard Sector Length (Sector Gaps + ID + DATA) = 92 + 7 + 515 = 614 bytes

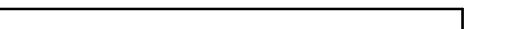
Note that the minimum values as specified in the WD1772 datasheet are not respected in the case of a track formatted with 11 sectors:

Minimum Sector Length (Sector Gaps + ID + DATA) = 72 + 7 + 515 = 594

The ID and DATA preamble are used to lock the PLL and should normally be kept as 12 \$00 bytes. The FD format do not reserve a write splice byte (where the head write current is switched on or off) and therefore it should be considered as part of the data preamble field for format and write operations, and as part of the ID postamble for read operations.

One complete ID/DATA segment looks like this

ID Segment										Data Segment						
ID preamble		ID Field							ID postamble	Data preamble		Data Field				Data postamble
12 x 00	3 x A1	IDAM FE	Track #	Side #	Sect #	Size	CRC1	CRC 2	22 x 4E	12 x 00	3 x A1	DAM FB or DDAM FB	User Data 512 Bytes	CRC1	CRC 2	40 x 4E

Write Gate 

As this format does not define any precise location *write splice* field, it should be included as part of the DATA preamble field for **format** and **write** operations and as part of the ID postamble for read operations.

4.1.1 Format for 9/10/11 Sectors of 512 Bytes

Note that the 3 1/2 FD are spinning at 300 RPM which implies a 200 ms total track time. As the MFM cells have a length of 4 µsec this gives a total of about 50000 cells and therefore about 6250 bytes per track. The table below indicates possible values of the gaps for tracks with 9, 10, and 11 sectors.

Name	9 Sectors: # bytes	10 Sectors: # bytes	11 Sectors: # bytes
Gap 1 Index postamble	60	60	10
Gap 2 ID preamble	12+3	12+3	3+3
Gap 3a ID postamble	22	22	22
Gap 3b Data preamble	12+3	12+3	12+3
Gap 4 Data postamble	40	40	1
Total Gap 2-4	92	92	44
Record Length	614	614	566
Gap 5 Index preamble	664	50	20
Total Track	6250	6250	6250

Respecting all the minimum value on an 11 sectors / track gives a length of:

$$L = \text{Min Gap 1} + (11 \times \text{Min Record Length}) + \text{Min Gap 5} = 32 + 6534 + 16 = 6582$$

Atari Floppy Disk Copy Protection

(which is about 332 bytes above max track length). Therefore we need to decrease each sector by about 32 bytes in order to be able to write such a track. For example the last column of the table above shows values as used by Superformat v2.2 program for 11 sectors/track (values analyzed with a Discovery Cartridge).

As you can see the track is formatted with a Gap 2 reduced to 6 and Gap 4 reduced to 1! These values do not respect the minimum specified by the WD1772 datasheet but they make sense as it is mandatory to let enough time to the FDC between the ID block and the corresponding DATA block which implies that Gap 3a & 3b should not be shortened. The reduction of Gap 4 & 2 to only 7 bytes between a Data Field and the next ID Field does not let enough time to the FDC to read the next sector on the fly but this is acceptable as this sector can be read on the next rotation of the FD.

This has an obviously impact on performance that can be minimized by using sectors interleaving. But it is somewhat dangerous to have such a short gap between the data and the next ID because the writing of a Data Field need to be perfectly calibrated or it will collide with the next ID block. This is why such a track is usually reported as “read only” (as in DC documentation) and is sometimes used as a protection mechanism.

Of course you have more chance to successfully write 11 sectors on the first track (the outer one) than on the last track (the inner one) as the bit density gets higher in the latter case. It is also important to have a floppy drive that have a stable and minimum rotation speed deviation (i.e. RPM should not be more than 1% above).

4.1.2 “Standard” 128-256-512-1024 Bytes / Sector Format

The table below indicates standard (i.e. classical) gaps values for tracks with sectors of size of 128, 256, 512, and 1024.

Name	29 sectors of 128 bytes	18 sectors of 256 bytes	9 Sectors of 512 bytes	5 Sectors of 1024 bytes
Gap 1 Index postamble	40	42	60	60
Gap 2 ID preamble	10+3	11+3	12+3	40+3
Gap 3a ID postamble	22	22	22	22
Gap 3b Data preamble	12+3	12+3	12+3	12+3
Gap 4 Data postamble	25	26	40	40
Total Gap 2-4	75	77	92	120
Record Length	213	343	614	1154
Gap 5 preamble	73	76	664	480
Total Track	6250	6250	6250	6250

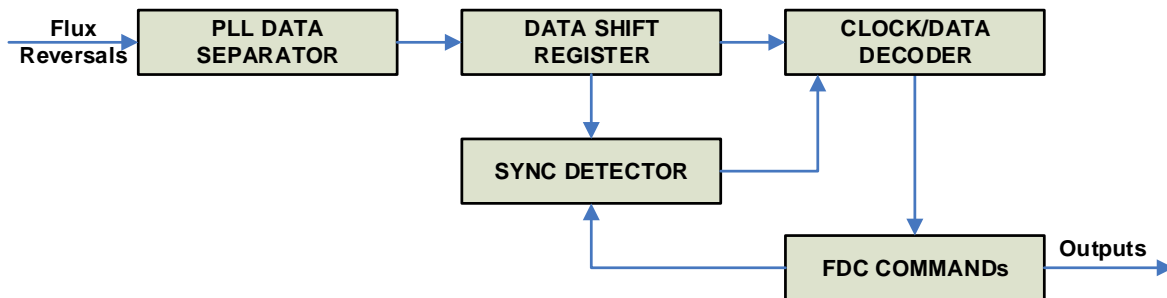
Atari Floppy Disk Copy Protection

4.2 WD1772 DPLL Input Circuitry

4.2.1 Description

This section provides basic information on the DPLL of the WD1772 and how the decoded bits are entered into the FDC shift register. It does not describe the *data separator* which is based on usage of an AM (Address Marks) detector to find a specific pattern in the shift register (usually during gaps) described latter in this document.

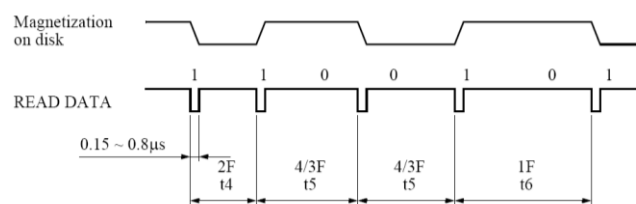
This is a simplified block diagram of the input circuitry of the FDC:



The WD1772 uses a digital phase lock loop (DPLL) circuit for reading the input data transmitted from FD media. Inspection windows are established that have duration proportional to the frequency of arrival of the data, and **start/stop times** that can be adjusted so that subsequent data bits will be received in the **middle** of the inspection windows. To achieve this, the DPLL circuitry applies **frequency** and **phase** corrections that compensate the input data frequency drift. This drifts are usually due to unsteadiness of the motor drive speed (the frequency drift), and the migrations of the magnetic reversals area (the phase drift). The DPLL used inside the WD1772, as well as many other FDC build in the 80s, implements an algorithm described in the public US patent 4,870,844. The patent is rather complex and in this section I will only highlight some of the most important aspects of the DPLL algorithm that are useful to understand the behavior in the context of fuzzy bits, long/short track, etc.

If you want to fully understand the behavior of the DPLL please refer to the patent. Note that in order to provide precise results my **Aufit**, **Analyze**, **KFAnalyze**, and **KFPanzer** programs **fully implement the DPLL algorithm as described in the patent**.

Typical MFM encoding



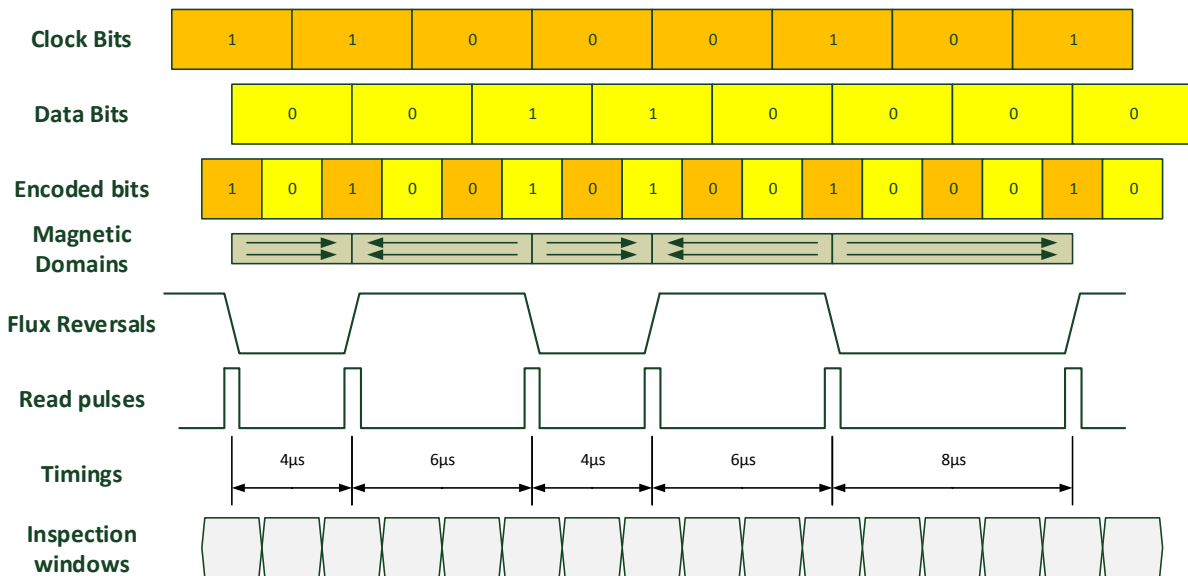
Note : READ DATA pulse will be detected within t7 from is nominal position. (When PLL separator is used with recommended write pre-compensation.)

Density mode	rpm	t4	t5	t6	t7
2MB mode	300	2µs, Nom.	3µs, Nom.	4µs, Nom.	±350ns
1MB mode	300	4µs, Nom.	6µs, Nom.	8µs, Nom.	±700ns

As we can see the **nominal values** for the possible reversals spacing in **DD MFM** (1MB mode) are: 4µs, 6µs, or 8µs.

Atari Floppy Disk Copy Protection

Let's first review a typical Double Density MFM data encoding:

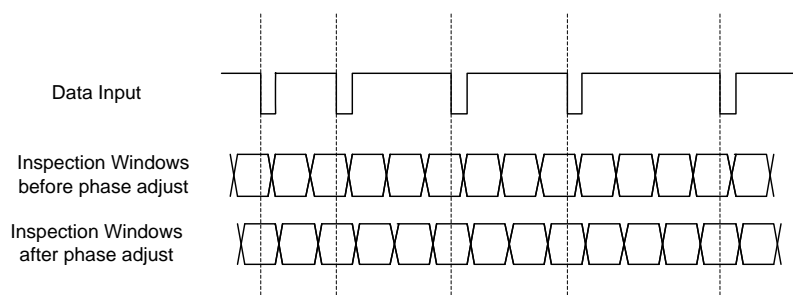


The data input circuit of the FDC ensures that the data pulses received are converted into data bits and stored in the **data shift register (DSR)**. For that matter the digital phase lock loop defines *inspection windows* that repeat every 2μs (a half cell size). A one is input to the shift register if a data pulse is received at any time during one inspection windows; otherwise a zero is stored in the shift register as the value for the current bit.

The period of the inspection windows is gradually adjusted (expanded or shortened) to compensate an eventual frequency shift affecting the input data transfer. This frequency correction is computed based on the **history** of the location (relative to the inspection window) of the **last three** flux reversals.

Ideally, individual pulses should be located in the middle of the inspection windows. To achieve this, the start and stop times of the inspection windows are adjusted to compensate for deviation (from ideal) in time of arrival of the most recently detected data

pulse. This phase correction is done proportionally to the distance of the reversals with the middle of the inspection window.



The proper ratio of phase and frequency correction provided in the loop is carefully balanced so that the DPLL is **fast settling** but **stable**. A large amount of phase correction cause the loop to settle faster but also make it more sensible to noise. On the other hand if too much frequency correction is used, the loop can become unstable.

It is interesting to note that the DPLL as defined in the patent allow an input frequency variation of up to **9%**. This corroborates the actual measurement made with a WD1772 that correctly interprets bits with a variation of at least 9 to 10 % for DD MFM (and about 100% for SD FM!). Note that these values are well above the variation used by the **Copylock** and **MacroDOS** protection mechanisms (usually less than 5%) and therefore the data within this kind of sector should be read correctly.

Atari Floppy Disk Copy Protection

4.2.2 WD1772 Detection of Fuzzy Border Bits

With the above information it is now easy to understand that if a bit reversal happens close to the border of an inspection window (also called Ambiguous area) it will be detected into the first or the next inspection window based on small variation of the drive rotation speed between two **read-sector** commands and this will therefore result in pseudo random values returned (fuzzy bits).

For example having a reversal 5 μ s apart from the previous one can be interpreted as a reversal after 4 μ s or a reversal after 6 μ s based on small frequency fluctuation of the rotation speed between two reads. One might argue that it is not possible to make sure that these “marginal reversals” will be positioned correctly due to the fact that the rotation's speeds of different drives are somewhat different and therefore precise reversals timing on a floppy diskette cannot be guaranteed. But in practice this is where the frequency and phase correction of the WD1772 DPLL comes into play. As explained above the inspection window will have its size (i.e. frequency) and position **corrected** based on the input reversals stream after reception of only a few reversals. Therefore the DPLL of the FDC automatically adjusts the frequency of inspection windows for any acceptable (about 10%) variation of drive speed and adjusts the phase so that a “normal reversal” will be perfectly in the middle of the inspection window and a “marginal reversal” will be perfectly at the border of the inspection window.

This also explains why, in most cases, “fuzzy bits” are used in “compensating pair”: for every two subsequent fuzzy bits the first reversal is placed at one extreme (e.g. at the beginning) of the inspection window and the “compensating reversals” of the next fuzzy bit at the other extreme (e.g. at the end) of the inspection window. By using this kind of “compensating bits” we guarantee that the frequency and the phase of the inspection windows are (almost) not affected.

Atari Floppy Disk Copy Protection

4.3 WD1772 MFM track language

During the **write track** command (format) the WD1772 needs to be told to perform specific actions as:

- write special sync marks with invalid MFM encoding,
- write special address marks that identifies the beginning of ID/Data fields, and
- write the content of the CRC register.

Therefore a range of values from \$F5 to \$FF has been defined as having special meaning (what I call WD1772 track language) for the FDC:

\$00-\$F4	Are not interpreted by the WD1772. This means that during all read or write commands (including read/write track) nothing special is done on these MFM bytes and are therefore transferred directly.
\$F5	<ul style="list-style-type: none">■ During read track, read sector, read address, write sector commands this byte as no special meaning.■ During a write track command this byte (unless escaped by a \$F7 byte) is written as an \$A1 sync byte with partially missing clock bit (\$4489) and the FDC internal CRC register is preset to the value \$CDB4.
\$F6	<ul style="list-style-type: none">■ During read track, read sector, read address, and write sector commands this byte as no special meaning.■ During a write track command this byte (unless escaped by a \$F7 byte) is written as a \$C2 sync byte with partially missing clock bit (\$5224).
\$F7	<ul style="list-style-type: none">■ During read track, read sector, read address, and write sector commands this byte as no special meaning.■ During a write track command this byte (unless escaped by a \$F7 byte) forces the FDC to write the content of the CRC register. Any byte placed after a \$F7 byte is not interpreted (escaped). In other word bytes \$F5 through \$F7 are treated as normal bytes when placed after a \$F7 byte.
\$F8, \$F9	Deleted Data Address Mark (DDAM) – Normally \$F8 <ul style="list-style-type: none">■ During a read track, read address, write track, and write sector command this byte as no special meaning.■ During a read sector command if this byte is located after three \$A1 sync marks it indicates the start of the sector “deleted data field”. The FDC sync mark detector is switched off after reception of this byte.
\$FA, \$FB	Data Address Mark (DAM) – Normally \$FB <ul style="list-style-type: none">■ During a read track, read address, write track, and write sector command this byte as no special meaning.■ During a read sector command if this byte is located after three \$A1 sync marks it indicates the start of the sector “data field”. The FDC sync mark detector is switched off after reception of this byte.
\$FC-\$FF	ID Address Mark (IAM) – Normally \$FE <ul style="list-style-type: none">■ During a read track, read sector, write track, and write sector command this byte as no special meaning.■ During a read address command if this byte is located after three \$A1 sync marks it indicates the start of the sector “ID field”. The FDC sync mark detector is switched off after reception of this byte.

Atari Floppy Disk Copy Protection

4.4 WD1772 Synchronization (sync marks detection)

With MFM encoding a clock is only added for two consecutive 0 data bits and therefore it is not possible to directly differentiate between clock and data bits on arbitrary sequence of bits. At the beginning of a track the controller don't know where the byte boundaries are located and so usually begins in the middle of a byte to read. The content of the track appears to be shifted by some bits and actually the first few bytes (usually two) have nothing to do with the true content of the track. This is also due to the fact that the DPLL is not yet synchronized.

There is a long string of zero's sequence encoded at the beginning of each ID and DATA field. This sequence provides to the DPLL enough time to adjust the frequency and center the inspection window. This is especially important for the DATA field because a Write splice occurs when the read/ write head re-write a data field. The slight variations in the rotational speeds cause the first flux change to occur in different positions for each write operation and therefore the DPLL needs to adjust to this new frequency/position. But this sequence is not really part of the synchronization.

It is only after receiving a synchronization mark \$A1 or \$C2 with partially missing clock bit that the controller reads the bytes with the correct byte boundary. These 2 special bytes with partially missing clock bits are called **Sync Marks**. In practice a sector ID or DATA field starts with a sequence of 3 consecutive Sync Marks followed by an Address Mark⁵ (IAM, DAM, or DDAM) as described in the [track format language](#).

It is interesting to note that the first synchronization byte in a sequence of three \$A1 sync marks is **always** read incorrectly by the WD1772. The first synchronization byte is inaccurately decoded as a \$C2, a \$14, or even sometimes a \$0A byte. If the controller is incorrectly shifted by a half bit (indicated by the fact that a sequence of \$00 bytes is read as \$FF bytes) the data and clock pulses are swapped.

The following table detail the usage of the Sync marks by the WD1772

\$A1	<p>Sync Mark with missing clock bit between bit 4 and 5⁶ (\$4489)</p> <ul style="list-style-type: none">■ This byte is used to synchronize (differentiate clock & data bit) the FDC on bytes boundary.■ \$A1 sync mark detection is active at all time during a read track command.■ \$A1 sync mark detection is deactivated after reception of 3x\$A1 followed by an IAM during a read address command.■ \$A1 sync mark detection is deactivated after reception of 3x\$A1 followed by a DAM/DDAM during a read sector command.
\$C2	<p>Sync Mark with missing clock bit between bit 3 and 4 (\$5224)</p> <ul style="list-style-type: none">■ This byte is used to synchronize the FDC on byte boundary.■ \$C2 sync mark detection is active at all-time only during a read track command but not active during a read address or read sector command.

Note that neither the \$4489 nor the \$5224 encoding **violates the 1,3 RLL rules** (sequence of 4 consecutive 0) and therefore this is why it is possible to find in a bit stream some sequences that are similar to the \$C2 synch mark. This is known as false sync mark detection problem during a read track command and it is detailed in the next section.

⁵ For the address marks characters between \$F8 and \$FF the least significant bit is always ignored by the WD1772 and therefore : \$F8=\$F9, FA=FB, FC=FD, \$FE = \$FF

⁶ The bits order is from MSB to LSB (the way they are sent) with first bit being numbered 0.

Atari Floppy Disk Copy Protection

4.5 False sync mark detection

As mentioned above the sync mark detection is enabled at all-time during the read track command. The biggest problem is that synchronization is done not only on the \$A1 and \$C2 sync marks (with partially missing clock bit), but also on specific sequence of bits.

If you remember the normal id/data field preamble is a sequence of \$00 (usually 12) followed by 3 x \$A1 sync marks. When a \$00 byte (1010101010101010) is placed in front of an \$A1 sync mark (0100010010001001) it results in a false \$C2 sync mark (0101001000100100) detection:

10101010101010101001001	\$00+\$A1 (SM)
0101001000100100	\$C2 (SM)

This explain why the id/data preamble is detected as \$14 \$A1 \$A1 or \$C2 \$A1 \$A1

More generally the WD1772 is resynchronized whenever the following combination of 9 bits "000101001" appears in a bit stream. This can happen anywhere as the with the following byte combinations:

- \$29 and previous byte even (i.e. LSB set to 0)
- \$52 or \$53 and previous byte divisible by 4 (i.e. the two LSB set to 0)
- \$A4-\$A7 and previous bytes divisible by 8 (i.e. the three LSB set to 0)
- \$14 and the following byte >= 128 (i.e. MSB set to 1)
- \$0A, \$8A and following byte with bit 7 cleared and bit 6 set (e.g. \$43)
- \$05,\$45, \$85, \$C5 and following byte with bits 7, 6 cleared and bit 5 set (i.e. \$21)

Non only the controller synchronizes to the presented sequence (i.e. \$29), but it stays incorrectly synchronized and therefore all the following bytes are shifted by multiple of "half bit", which results in mix-up of data and clock pulses, and so the decoded bytes are totally unrecognizable.

This error occurs everywhere on track 41. The value 41 is \$29 in hexadecimal and therefore all the address fields of these tracks are read incorrectly as well as the bytes following this incorrectly decoded header.

False sync marks detection problem can be used for protection as explained in the document [Copy me, I want to travel](#) by Claus Brod.

4.6 Overlapping Sync Mark

It is possible to find in the input stream some sequences of bits that contains overlapping sync marks. We have 4 possible combinations of overlapping sync mark:

- ★ \$A1-\$A1
- ★ \$C2-\$A1
- ★ \$A1-\$C2
- ★ \$C2-\$C2

4.6.1 Overlapping \$4489-\$4489 (\$A1-\$A1)

We take the \$4489 pattern and we try to find how it can overlap another \$4489 pattern. We find two following two cases:

```
0100010010001001
      0100010010001001

and

0100010010001001
      0100010010001001
```


Atari Floppy Disk Copy Protection

4.6.2 Overlapping \$5224-\$4489 (\$C2-\$A1)

We take the \$4489 pattern and we try to find how it can overlap a \$5224 pattern. We find the following two cases:

```
0101001000100100
    0100010010001001
```

and

```
0101001000100100
    0100010010001001
```

4.6.3 Overlapping \$4489-\$5224 (\$A1-\$C2)

We take the \$5224 pattern and we try to find how it can overlap a \$4489 pattern. We find only one possible case:

```
0100010010001001
    0101001000100100
```

4.6.4 Overlapping \$5224-\$5224 (\$C2-\$C2)

If we take two \$5224 pattern and we try to find how they can overlap. We can see that this is not possible.

4.6.5 Invalid Sync sequence

A “normal” sync sequence is composed of three \$4489 sync mark character (\$A1 with missing clock) used in front of an IAM. Any other sequence of sync marks should be considered as a non-normal sequence that I refer as an invalid sync sequence. You will find in this document several places where the sync marks \$4489 or \$5224 characters are used for special usage.

It is interesting to note that in order to be read correctly an ID field or a DATA field must be preceded by **exactly** by 3 x \$4489 sync marks. For example if the sync sequence is composed of two or four sync marks the ID field is **not** detected by the WD 1772.

However there is a special sequence that can be used instead of the normal 3 x \$4489 sync mark sequence: 7 x \$4489.

This works because the following happen in the WD1772:

- After reception of the first 3 x \$4489 the FDC is ready waiting to get an IAM
- At reception of the \$4489 character the FDC detect a false sync sequence because it is not an IAM. Therefore the fourth \$4489 is discarded and the FDC return in sync sequence search with the sync detector kept active.
- The next 3 sync marks are detected correctly as if they were a new sequence of 3 sync mark (even though they are sync marks 5, 6, and 7).
- At the reception of the IAM the sync detector in the FDC is de-activated and the sector is read normally.

Note that should also work for a sync sequence where we add a multiple of 4 x \$4489 sync character in front of the normal sync sequence (i.e. 11, 15, 19...).

Atari Floppy Disk Copy Protection

4.7 WD1772 Bug in Read/Write Track commands

When you read a normal track you expect to get a number of bytes around 6250 bytes and on a slow track you may get may be up to 6600 byte. But under certain circumstances you get much more, in fact you might even get an almost infinite number of bytes.

How is this possible and when does it happen?

Apparently this happen when reading a track that have sync mark placed “over the index pulse”. Here is the explanation that I am aware of:

- ★ Normally during read track command the FDC start on a first index pulse signal and stops when it receive the next index pulse signal but if the FDC is busy processing a sync byte then the index pulse is no longer recognized.

So a **read track** on this kind of track (i.e. with sync mark - \$4489 or \$5224 - over index) sometimes the FDC does not properly detect the index pulse and therefore lots of extra bytes are send to the DMA until it overflows. It seems that this also happen during the **write track** command when you provide a sequence of \$F5 or \$F6 when reaching the index.

Therefore a program that reads track should be able to handle this problem for example in **Panzer** during a read track command I set the number of DMA count to 40 ($40 \times 512 = 20480$) and reserved a buffer large enough to accommodate all these bytes.

Note that this does not happen systematically (probably due to rotation speed variation) so you can read the same track correctly several times and get this problem at other times.

Atari Floppy Disk Copy Protection

4.8 WD1772 CRC Information

4.8.1 CRC Computation

The WD1772 documentation indicates that the CRC uses the CCITT CRC16 polynomial and that the CRC register is preset to all ones (\$FFFF) during the **write track** command when the first \$F7 byte is received (see [WD1772 MFM track format language](#)). This results to a CRC value of \$CDB4 at the end of a normal sync sequence of 3 x \$4489.

In practice, probably for practical reasons, the CRC register is preset to \$CDB4 each time a \$F5 character is received. This can be verified by writing a sequence with more or less sync bytes than the normal three sync marks sequence (remember that you won't be able to read the corresponding sectors) and looking at the CRC result. Whatever is the number of \$F5 sync marks written the CRC is always reset to \$CDB4 by the last sync.

For example if you use the sequence \$F5 \$F7 you will see that the WD1772 writes the two bytes \$CDB4 (content of the CRC register) after the sync character.

No other character (including \$C2 sync mark) presets the CRC to a predefined value.

It is interesting to note that any byte placed after a \$F7 is transmitted unchanged (escaped) by the WD1772. For example with the sequence \$F7 \$F5 the FDC will write two CRC bytes followed by the \$F5 byte. A sequence of repeating \$F7 is sometime used in protection.

4.8.2 Playing with the CRC

We have seen that some protections are based on writing on purpose bad CRC in the ID or DATA fields. Usually the checksum of an address or data field is calculated by the controller but you can bypass this behavior and writes your own checksums to create errors. This is often not detected by copy programs.

Let's first create a broken checksum in an address field. This is relatively simple, because address fields are written by using a write track command sequence like the following:

```
F5 F5 F5 FE 00 00 01 02 F7
```

This sequence of bytes writes an address field with track and head number 0 sector number 1 and size 2. The byte \$F7 forces the controller to write the calculated checksum on the floppy disk.

If we replace the \$F7 byte by two \$00 bytes with the following sequence:

```
F5 F5 F5 FE 00 00 01 02 00 00
```

This address field is read with a checksum error and the sector is unreadable⁷.

You can try the following sequence:

```
F5 F5 F5 FE F5 00 01 02 F7
```

The Sync byte \$F5 within the address field generates a checksum error in the header to read; because when writing the WD1772 calculates the checksum of the byte sequence

```
FE F5 00 01 02
```

But it is read:

```
FE A1 00 01 02
```

The checksums of these two bytes sequences are of course different and therefore you get a CRC error which implies that you no longer can read the data field.

⁷ Remember that it is possible to read an id field with a wrong CRC using a read address command, but it is not possible to read a sector with a CRC error in its header using a read sector command.

Atari Floppy Disk Copy Protection

However it is simple to compute the checksum of the FE A1 00 01 02 sequence (\$56AD) and replace the \$F7 byte by these bytes during format.

```
F5 F5 F5 FE F5 00 01 02 56 AD
```

Another interesting sequence is to write a \$F7 sync byte in an address field and not get a checksum error. You know that when writing consecutively two \$F7 bytes on the disk the second one is escaped. The first is interpreted as a request to write the checksum register content and the one is explicitly written as \$F7. For example with the following sequence:

```
F5 F5 F5 FE F7 F7 02 F7
```

The address here is specified with only three bytes (\$F7 \$F7 \$02) because the first \$F7 bytes when writing is translated into two CRC bytes. The resulting checksum is correct.

The address field is read as:

```
14 A1 A1 FE B2 30 F7 02 AA 14
```

The track number is read \$B2 (178), the head is \$30 (48), and the sector number is \$F7 (247) than you cannot usually write on a floppy disk with the WD1772. A copy program trying to write this header just as read will be generated a sector header like this:

```
14 A1 A1 FE B2 30 00 00
```

And that is of course quite different from the original.

By the way, you can see the very nice reproductive properties of the CRC Checksum. The sequence starting with the byte, \$FE, creates the checksum of \$B230 (with the CRC register initialized to \$CDB4 after the last \$A1) and if you send this checksum right back to the CRC register the result is 0. This is how the controller works when it reads a data or address fields and the associated checksum. If the CRC register contains zero the data or address field is correct.

Similarly, you can also write a \$F5 or a \$F6 byte in an address field. Because after a \$F7 byte is written the following byte is unchanged (escaped). The sequence

```
F5 F5 F5 FE F7 F5 02 F7
```

is read

```
14 A1 A1 FE B2 30 F5.02
```

In a DATA field, a deliberate checksum error is more difficult to produce. If the data field of the sector to write contains no bytes over \$F4, we can directly write these bytes during formatting (write track command) and like for the ID field write a fake checksum at the end of the sector (for example 00 00) to generate the CRC error (instead of using the normal \$F7 that would generate the correct two bytes checksum).

But if we need to write the data using a write sector command (so that any byte including bytes over \$F4 can be correctly written) the correct CRC will be written automatically at the end of the command.

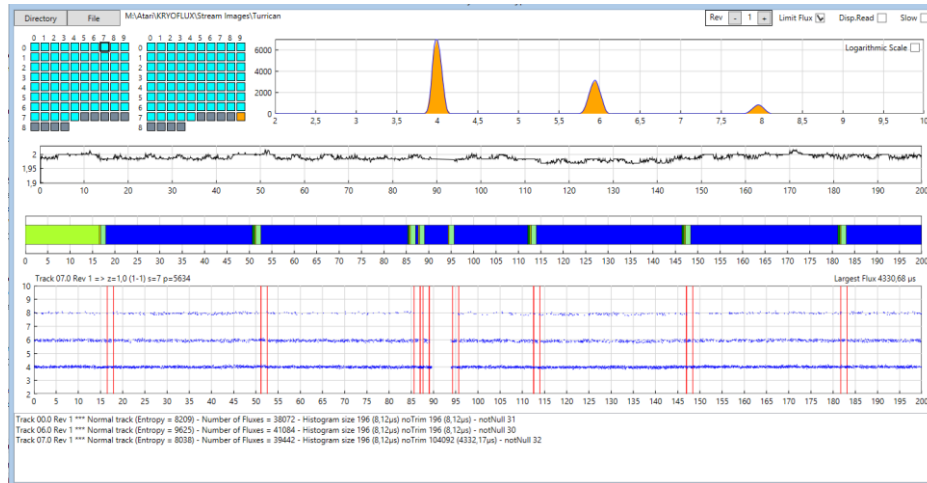
In that case, to generate a CRC error, it is possible to use the following procedure:

- First step: format the track with correct checksum and empty data in the sector concerned.
- Second step: write the sector and stop the FDC command execution just before it writes the checksum. The result is therefore a sector with new data but old checksum. Stopping precisely the command before writing the checksum is difficult.

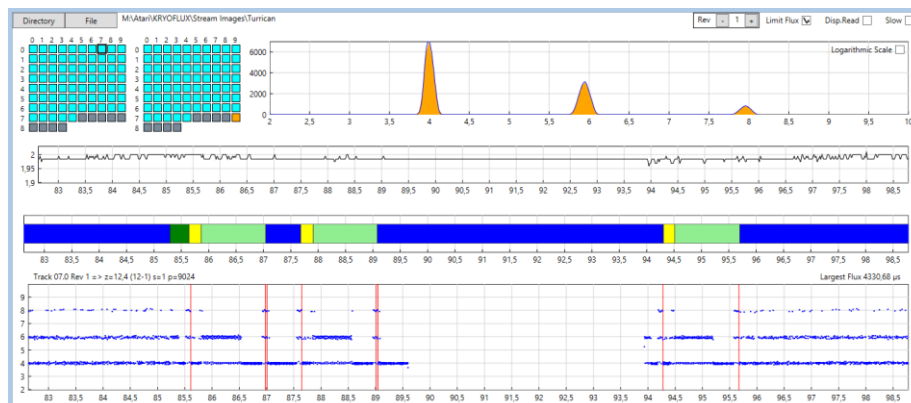
Atari Floppy Disk Copy Protection

We can see inside data block (starting with 0xFB DAM) the presence of 3 sync character followed by the ID block for sector 16 (sector within sector). However if we look further down we do not see the sync marks for the corresponding data block. Instead we see the presence of 3 bytes with value \$14 followed by a byte \$00 and several bytes 0xFF. But if we look at the line below (that contains the clock bytes) we can see that the 3 x \$A1 sync bytes are in fact located in the “clock” bytes. During the read command the sync mark detector of the WD1772 will take care of shifting the input stream by a half cell to correctly read the sector 16 data.

The end result is that you can read the “data” bytes of the NFA by reading sector 0, and you can read the “clock” bytes of the NFA by reading sector 16 (sector within sector).



As you can see around 90 ms inside the track we have a region without any transition for a period of 4330μs.



If we zoom close to the NFA we see that we have a first sector, and inside this sector we have a second sector (sws) and that the data segments of these sectors both includes the NFA.

Note: Inter-GAP in Green, ID in yellow, Intra-GAP in light green, Data in blue.

4.9.2 Special case of No Flux Area over index

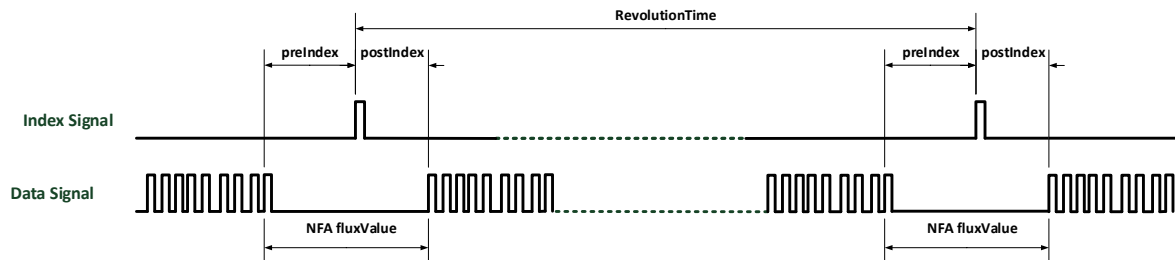
It is possible to have the No Flux Area located over the Index pulse. This is a hard to handle case for programs that reads the flux transitions produced by devices like Kryoflux and SuperCard Pro.

It is interesting to note that, for obvious reasons, in (almost) all cases the index pulse and the data pulse are not synchronized. In order to correctly interpret the information sampled, it is therefore necessary to know the position of the index pulses relative to the data pulse.

In “normal” cases (i.e. for data pulse in range 4 to 8 μs) it is acceptable to ignore the position of the index relative to the current flux transition, but in case where a no flux area is located over the index it is mandatory to get and interpret correctly this information.

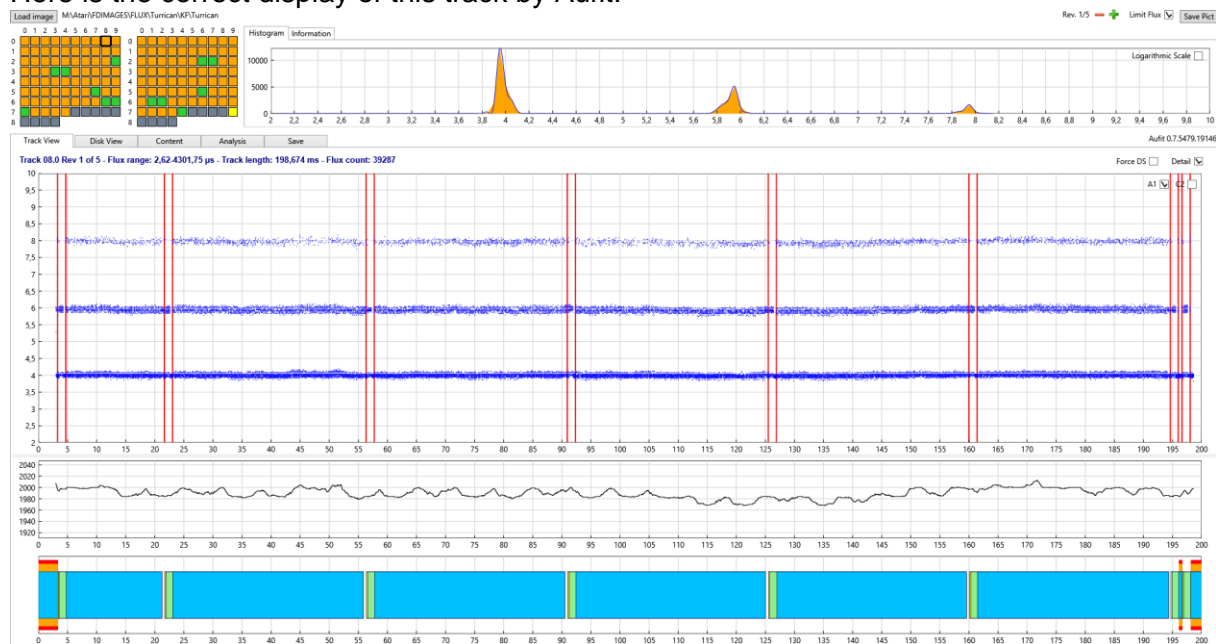
Atari Floppy Disk Copy Protection

Here is a typical case of an NFA over index. As we can see we have a huge area without flux transition located just above the index. In the figure we show three important values: one is the “NFA flux value” (typically around 4 to 5 ms.), the pre-Index time value, and the post-Index time value (only 2 of these three values are required as the third can be easily computed from the other two).



For a practical example I use the Turrican game. On my version track 8 has a NFA of about 4.3 ms located on top of the index. The pre-index value is about 3 ms and therefore the post-index is about 1.3 ms.

Here is the correct display of this track by Aufit.



The Kryoflux raw stream format provides the NFA value as well as the pre-index timing (see my documentation [KryoFlux Stream File Documentation](#)). The only way to be able to provide this kind information is to start to sample flux before the index as the Kryoflux device does.

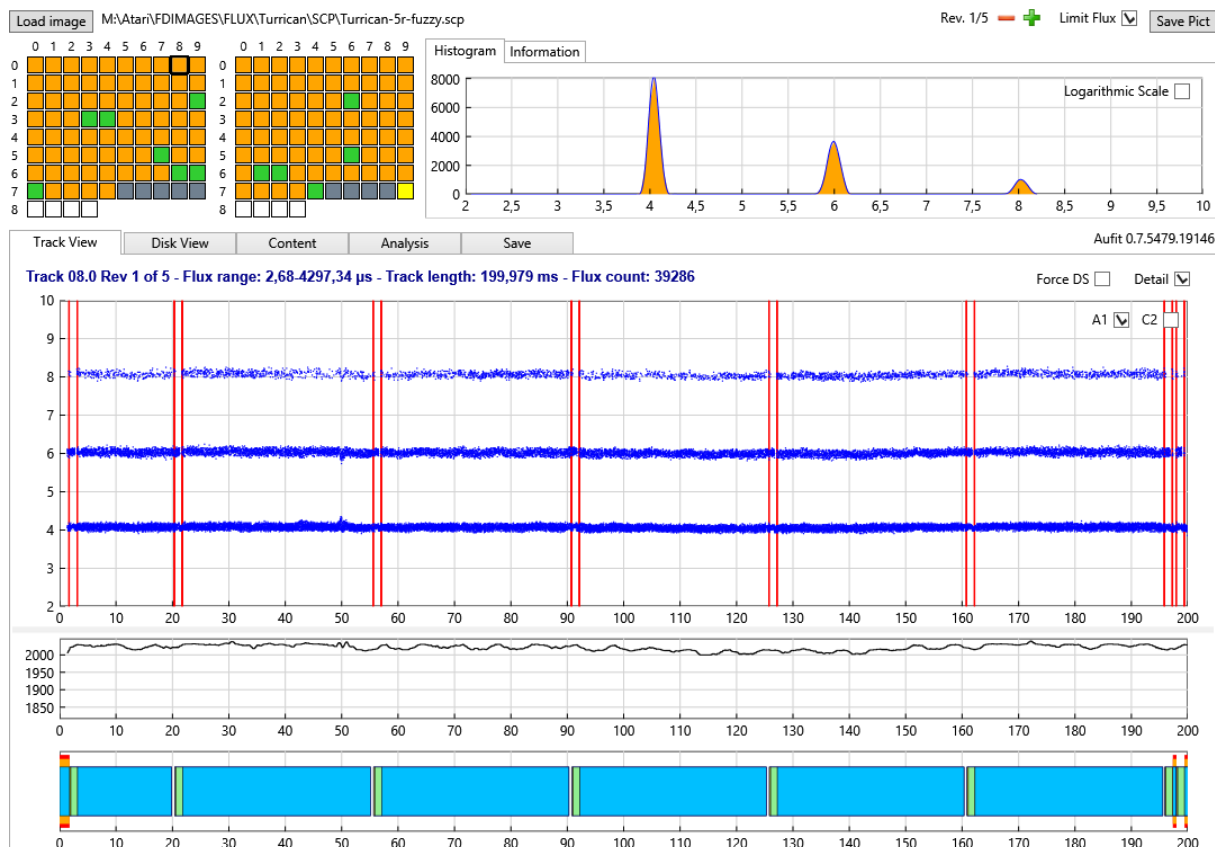
Unfortunately the SCP format does not provide any information about the positioning of the index pulse relative to data pulse. I have requested this feature several times on Atari-Forum as well as on the SCP forum without success. In SCP device:

- The sampling of transitions always starts at the index pulse. It is therefore not possible to detect the index position for the transition happening before the index.
- The result is the No Flux Area is just not transmitted on the first rotation. On the second revolution the NFA is passed as if it was the first transition.

Atari Floppy Disk Copy Protection

- Normally it is expected that the sum of the length of all the transitions in one revolution is equal to the revolution time given as the time between two indexes. But this does not work in SCP format because the sum does not include the post-index part of the NFA and therefore the value in that case is much smaller than the expected value. This is to compare to the exact value transmitted by raw stream as explained page 14 of [KryoFlux Stream File Documentation](#).

Because of all the reasons explained above it is impossible to get correct information from the SCP file in this case. However AUFIT take advantage of the last reported problem to compute the post-value that is added in front of the transitions. But as mentioned the first revolution is nevertheless not displayed correctly as the NFA is not transmitted in this revolution. In the subsequent revolutions the display is not correct either because the complete NFA is transmitted incorrectly as the first transition of the revolution.



As you can see both the start and the end of the NFA are incorrectly displayed. If you select revolution 2 the complete NFA is now placed entirely as the first transition.

Fortunately for the user it seems that in all the games, using NFA, that I have tested the relative position of the NFA (in respect to the index pulse) is not relevant. Therefore even though the information displayed and saved in the image file is incorrect writing the protection in Pasti file seems to work in all cases.

Atari Floppy Disk Copy Protection

4.10 Unformatted Diskette / Track / Sector

4.10.1 Presentation

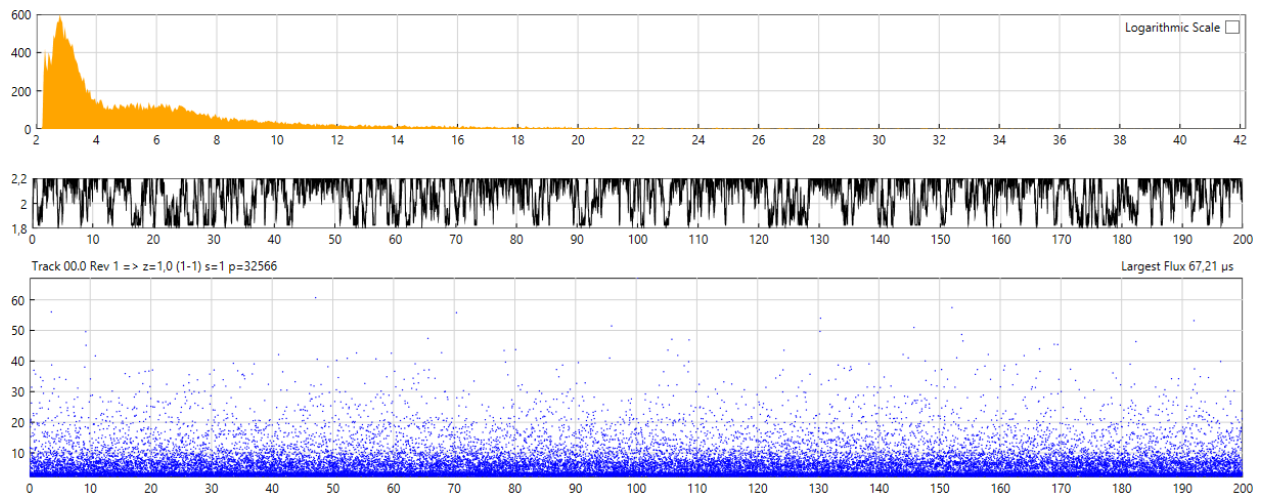
By definition an unformatted diskette would be a diskette that has never been formatted. During formatting, the particles are aligned forming a pattern of magnetized tracks, each broken up into sectors, enabling the controller to properly read and write data. Here is a definition from [Wikipedia](#): “A blank “unformatted” diskette has a coating of magnetic oxide with no magnetic order to the particles”.

The magnetization on the surface should be relatively uniform and in an ideal world the head should not peak up any flux reversal and therefore the read circuitry should not return any data pulse. But in practice many pseudo random transitions are detected. Two things explain this behavior:

- As we have no regular flux transitions, the drive's automatic gain control (ACG) is pushed to its maximum possible gain because it presumes a weak signal coming from the drive's read head.
- Some random flux variations exist naturally on the magnetic surface and due to the fact that the ACG is turned to its maximum they may be detected as flux transitions. This can be compared to “hearing” noise from a blank audio tape when the volume pushed very high.

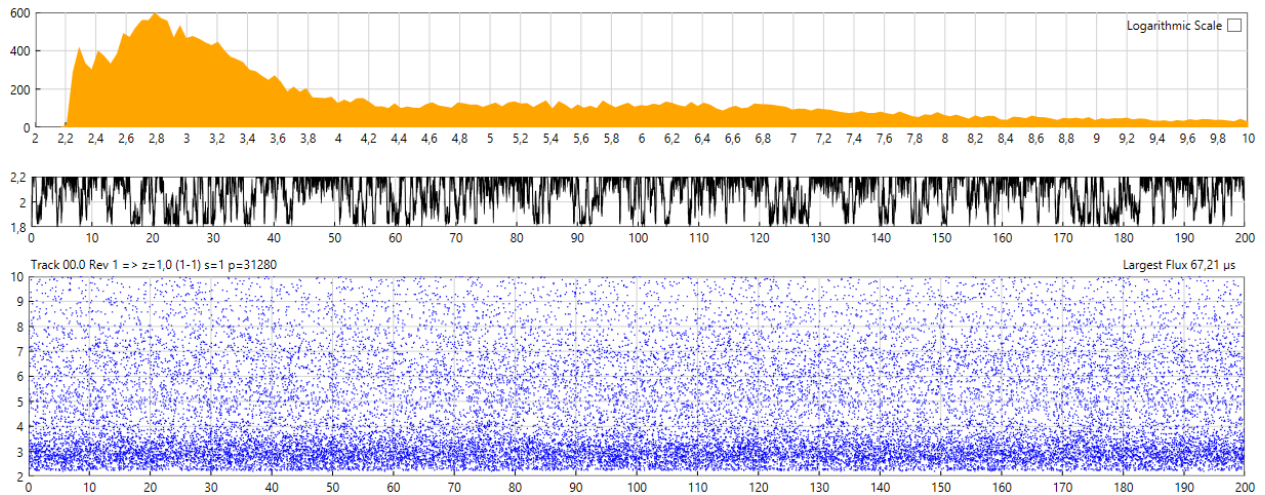
The detected data seems “random” in the sense that the data is never the same twice. But it seems that the “repartition” of these transitions is related to the drive speed (and humidity, temperature) fluctuations. Note that compared to a normal track the number of flux transitions detected on an unformatted track is obviously much lower.

We can see that most of the transitions are typically located between 2μs and 7μs. The image below show a typical histogram for an unformatted track:



If we limit the scale of the displayed transitions to 10μs we have the following image:

Atari Floppy Disk Copy Protection

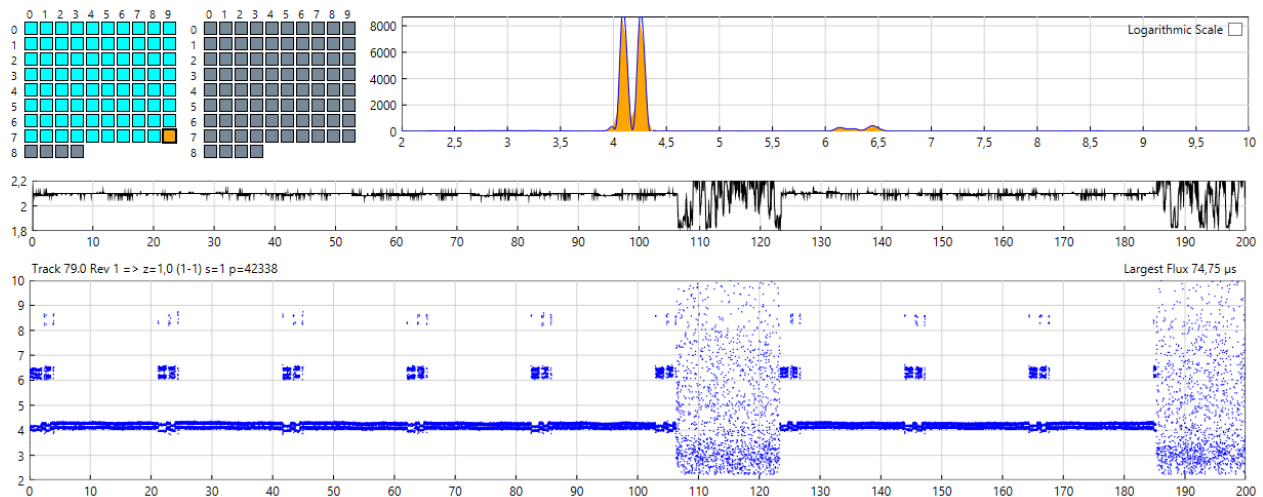


Few things I have noticed:

- Usually all the unformatted tracks of one diskette looks very similar (i.e. they have equivalent histograms) but they look different from one diskette to another. This “signature” is probably dependent of the diskette as well as of the drive used.
- When you record the flux transitions of a track over several revolutions, usually you do not see much difference between one revolution and the next except for unformatted track. Due to the random nature of unformatted track the data information recorded differ widely from one revolution to the next even though the histogram stay relatively the same.

4.10.2 Partially unformatted track

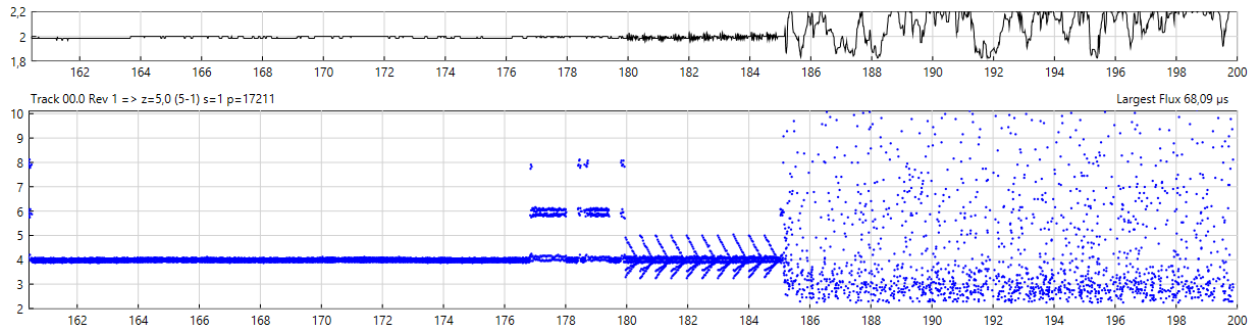
Some protections use partially unformatted tracks. For example:



The example above shows the track 79.0 of Night Shift game. We can see that we have two unformatted sections in this track. The clock, decoded by the DPLL, in these regions vary a lot and you can easily imagine that the data read from these sections contains fuzzy bits.

Atari Floppy Disk Copy Protection

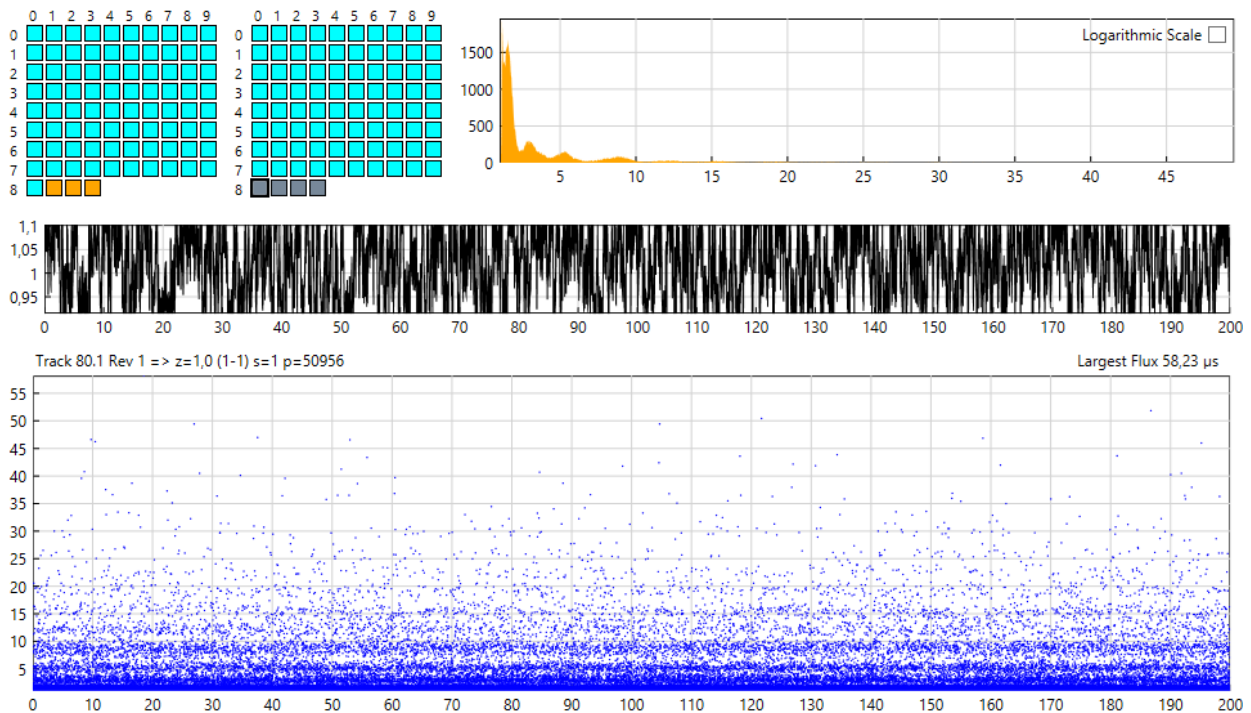
Another example taken from a protected diskette from [DrT a D50 Patch Editor](#). In the following chart I have zoomed on an unformatted area near the end of track 00.0:



We see that the end of the track is unformatted and have therefore random transitions, but just before that area we also see a strange “fish bone” pattern. We have some flux transitions placed closed to 5µs and 3µs which are exactly at the border of the inspection window and this will certainly results in fuzzy bits. Note that with this fish bone pattern the transitions close to 5µs are “compensated” by transitions close to the 3µs and this results (thanks to the DPLL) in a relatively stable 2µs clock.

All the previous examples were taken from Double Density Floppy disks (the one used on Atari). But I have also experimented with High Density floppy disks (PC diskettes).

I have noticed that the unformatted tracks from a HD floppy look different! They seem to exhibit a much more pronounced “banding effect”:



This is probably due to usage of different magnetic coating that have different coercivity.

Atari Floppy Disk Copy Protection

4.10.3 Partially formatted Track

Another trick used for protection is partially formatted track: At first glance the track seems to be unformatted but in fact it contains some information (sometimes this information is used by duplicators and/or by manufacturer). Some protections may use as little as just a few bytes of data and leave the rest of the track unformatted. Most copier won't be smart enough to detect this kind of hidden information.

However in order to use this trick for protection you not only need to write the hidden bytes but you also need to be able to read and check them correctly with the FDC of the platform. In the case of a MFM DD diskettes used with Atari ST, the only reliable way to decode and check hidden bytes in an unformatted track is to have one or several SYNC marks in front of the data. Even with sync mark it is difficult to detect this kind of information using a read track command unless you know exactly what you are looking for.

4.10.4 Unformatted track detection

It is usually easy to visually detect that a track is unformatted on a scatter chart but it is more difficult to detect this by software. The detection should provide reliable information and ideally detect partially unformatted track as well as partially formatted track. There are several possible proposed indicators:

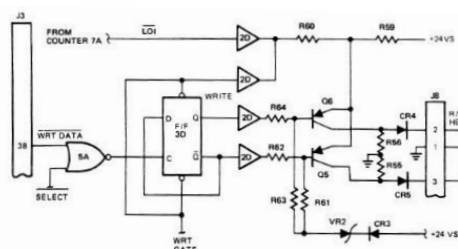
- If you cut a track in small chunk of data you should be able to match each of these blocks from one revolution to the next. Of course you need to allow for some slack in position from revolution to revolution.
- You can check for legitimate encoding on the track. For example in the case of MFM encoding this could be as simple as detecting SYNC marks sequence.
- I have successfully used a modified computation of the Shannon Entropy on the flux transition's histogram. A "normal track" has a coherent repartition and this result in a high entropy.
- The number of transition is lower than normal track.
- Unformatted track contains invalid MFM sequence. For example 0x0000

By combining the above indicators you should be able to detect “true” unformatted track (i.e. with zero encoded bytes).

4.10.5 How to reproduce unformatted areas on Floppy Disks?

The simplest idea that comes to mind is that we use a blank diskette and for unformatted areas we do not write anything. Unfortunately DD floppy diskettes you buy in the commerce are already preformatted (usually using the IBM format and with a DOS 2.0 boot sector). I don't know why the manufacturers do that, but I guess this must be for them to run some quality tests?

Note that **only** tracks 0 to 79 are preformatted on blank diskettes. If we sample the flux transitions of unformatted tracks beyond this limit (track 80-83) we can see that these tracks are similar but slightly different than unformatted tracks in range 0-79.



This seems to indicate that professional duplication machines might have used unformatted diskettes?

Writing unformatted track it is as simple as enabling the write gate and keeping the write data line of the drive to zero for the time of the track. This will keep the current flowing in the read/write head constant and therefore the flux will written will be constant.

The only potential problem comes from potential misalignment of the write head relative to previously written data. Hopefully the tunnel or straddle erase head should take care of the leftover of previous contents.

Atari Floppy Disk Copy Protection

Remember that writing data on a floppy drive is different from the same operation on an audio tape drive. On an audio tape the information the data is first erase by an erase head then the new data is written linearly by the read/write head. On a floppy drive there is no erase head (other than the tunnel / straddle erase head used from trimming track) and the new data are just written over the existing one with the head operating at magnetic saturation.

To summarize: un-formatting a track only requires to keep the write data line negated during the complete write operation. This is obviously not possible with a WD1772 FDC (it always pulse the write data line) but it is easy to control on a replicator (e.g. trace) or with a Kryoflux, SuperCard Pro device.

On an Atari the best you can do is to format a track using a buffer containing random bytes, but you will never get something similar to a real unformatted track because flux will always be located in the 4, 6, and 8 μ s bands.

Atari Floppy Disk Copy Protection

4.11 Fuzzy Bits

Fuzzy bits are known under many different names: *weak bits*, *wandering bits*, *flaky bits*, *flakey bits*, *phantom bits*, etc. **Weak bits** is the most commonly used term, however I find it confusing (as there is usually no “weakness” in weak bits). Therefore I prefer to use the term **Fuzzy bits** that does not indorse any underlying cause but clearly indicate the “fuzziness” nature of the returned data. Although fuzzy bits can be created by using different techniques the result is always the same: reading a byte that contains fuzzy bits will return random values (i.e. different values each time it is read). Fuzzy bytes could potentially be located at any place in a track but fuzzy bytes are often placed in the data field of a sector. To provide complete information we will describe below several ways to create fuzzy bytes: [Flux reversals in Ambiguous Area](#), [MFM Timing Violation](#), or [Weak Bit](#). However for emulation or backup purpose it is not necessary to know underlying mechanism used.

4.11.1 Flux Reversals in Ambiguous Area

- **Description:** These fuzzy bits are obtained by “placing” certain flux reversals in so called “Ambiguous areas” i.e. *at the border of the inspection window*. Please refer to [WD1772 Detection of Border Bits](#) section for more information.
- **Creation:** These fuzzy bits are obtained by placing the bit flux reversals in “Ambiguous areas”. More precisely the bit reversals are placed in locations that will confuse the DPLL (Digital Phase Lock Loop) of the data separator resulting in random values read (i.e. sometimes 0, sometimes 1). This is obtained by positioning the bit reversals at the **border of the inspection window**. In that case the data separator will return random values due to small variation of the drive rotation speed. In the [US patent](#) “Copy Protection for computer Disc 4,849,836” one of the techniques to create fuzzy bits consists in having flux reversals gradually sliding in and out of the inspection window border. Of course creating this kind of reversals requires special hardware that has capability to vary the FDC clock on the fly, or the capability to directly control the bit cell width/position (e.g. the Discovery Cartridge, KryoFlux board, SuperCard Pro device).
- **Detection:** As mentioned this protection results in [Fuzzy Sector](#). Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.
- **Duplication:** Difficult and requires special hardware (i.e. the Atari WD1772 cannot be used to copy this kind of bytes). Analog or digital copiers can be used but, as usual, digital copier should be preferred.
- **Emulation:** The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Example:** [Dungeon master](#) Track 0, sector 7

4.11.2 MFM Flux Timing Violation

- **Description:** These fuzzy bits are obtained by using flux reversals that violate the timing of the MFM rules.
- **Creation:** These fuzzy bits are obtained by placing flux reversals that contains MFM timing **violations** (data separated by less than 4 μ s or more than 8 μ s). For example a long series of zero data with missing clock bits. These bit-cell width are beyond the normal DPLL capture range and the next received reversal will be interpreted differently based on small random variation of the DPLL clock and/or the drive rotation speed. Of course this technique requires special hardware that has capability to vary the FDC clock on the fly, or the capability to directly control the bit cell width/position (e.g. the Discovery Cartridge). Note that this violations are often achieved by using an unformatted a section of the track. See [Unformatted Diskette / Track / Sector](#) section for more information.
- **Detection:** As mentioned this protection results in [Fuzzy Sector](#). Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several

Atari Floppy Disk Copy Protection

times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.

- **Duplication:** Difficult and requires special hardware (i.e. the Atari WD1772 cannot be used to copy this kind of bytes). Analog or digital copiers can be used but, as usual, digital copier should be preferred.
- **Emulation:** The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Example:** [D50 Editor](#) - Track 0 - Sector 10.

4.11.3 Weak Bit

- **Description:** We use the term **weak bits** for data bits that produce **weak flux reversals** below a certain threshold that will therefore result in ambiguous reading returning different values on different reads (see fuzzy bits for a [generic description](#)). The [SpinRight documentation](#) (from [SpinRite's Defect Detection Magnetodynamics](#) site) gives a good explanation on weak recorded reversals.

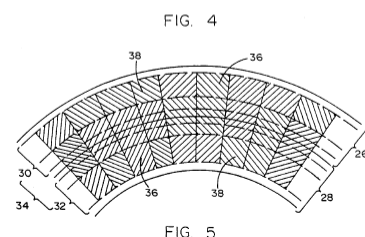
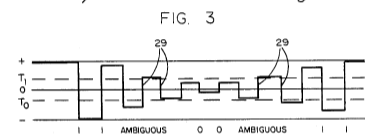
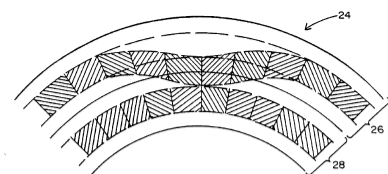
Weak bits can be created by many different means but the most popular have being described in the US Patent 4,849,836.

One method consists to move the head slightly out of alignment during write operation (see figure 3). As the Atari FD drives do not have a sophisticated track follower mechanism, this result in weak reversals during read (see figure 4).

Another method consists in writing a “protection track” in between normal tracks (see figure 5). It is obvious that this extra track will induce perturbations in the data bit flux of the adjacent tracks resulting in weak bits when there is opposition in the fluxes.

Yet another method consists in placing bits on top of *physical defects* on floppy surface. To be useful these defects have to be created precisely on specific spots of the surface layer using for example evaporation with an infrared laser.

- **Creation:** Creation of this type of weak bits requires very specialized hardware. Here we are not talking about special floppy disk controllers but about special floppy drives.
- **Detection:** As mentioned this protection results in [Fuzzy Sector](#). Therefore it can be detected by reading the same fuzzy sector (i.e. sector that contains fuzzy bits) several times and checking that returned data are random. Without specific hardware it is not possible to find the real underlying cause of the fuzzy bits but this information is of no use for an emulator or a duplicator.
- **Duplication:** It is obviously at least extremely difficult if not impossible to exactly reproduce the weak bits described in this section. However it is possible to mimic their behavior by placing [Flux Reversals in Ambiguous area](#) as this result in the same behavior and therefore should be transparent to the detection mechanism of the protected program.
- **Emulation:** The preservation file should have an indicator to record the fact that the sector is a fuzzy sector but should not care of the underlying cause of the fuzzy bits.
- **Examples:** I am not aware that this technique has been used on Atari.



Atari Floppy Disk Copy Protection

4.12 Write Splices

4.12.1 Sector write splices

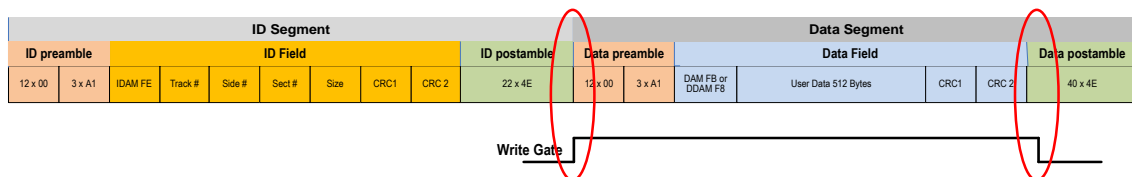
In general it is not possible with the WD1772 FDC to write an entire track in one operation. This is due to the fact that when writing a track the data in the range 0xF5 to 0xF7 are treated as special control bytes and therefore they cannot be written directly during a write track (format) operation. Therefore on an Atari machine, to use a floppy disk, the first operation consists in formatting the track using a **write track** command then the data field of each sector is written using **write sector** commands.

Here is a simplified description of the write sector command:

Upon receipt of the Write Sector Command the FDC searches on the track an ID field that has the correct track number, correct sector number, and correct CRC. If such an ID field is found the WD1772 counts 22 bytes from the CRC of the ID field and it activates the WG output. Then the following data are written on the disk:

- 12 bytes of zeroes
- Three A1 Sync
- A Data Address Mark
- The complete data field
- A two-byte CRC is computed internally and written on the disk
- One byte of logic ones.

Then the WG output is deactivated.



It is easy to understand that the activation and deactivation of the Write Gate cannot be aligned precisely with the original data written during the format operation and furthermore, as the speed of the drive fluctuates, the overall size of the data segment will most likely not be the same. This results in writing non-aligned (drifted) transitions, which most likely violates MFM rules, when the WG is activated and deactivated. These two areas are called sector write splices.

When data are written on a floppy disk with professional duplication equipment the complete track is written in one operation and therefore these sector write splice areas **do not exist**. This is one way to verify that we are working on a “master floppy disk” written on professional duplication equipment and that the sector data on the FD has not been tempered.

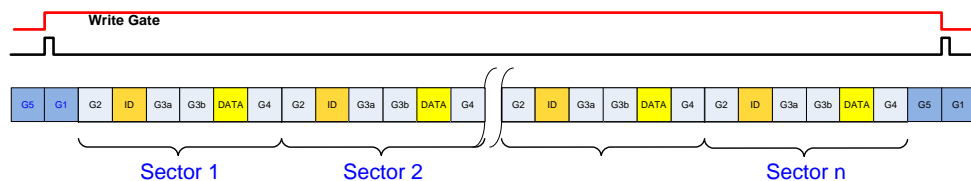
Atari Floppy Disk Copy Protection

4.12.2 Track write splices

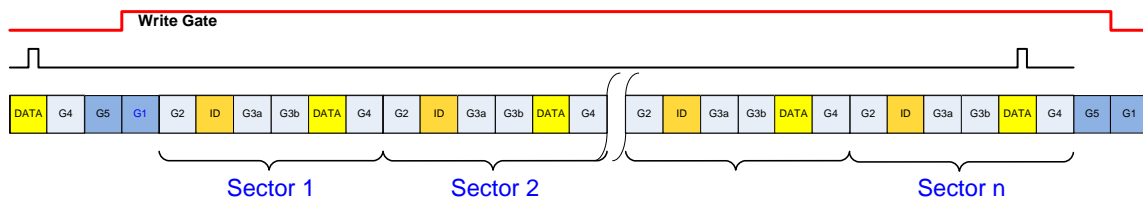
As said before when using professional duplication equipment it is possible to write a complete track in one operation and this will result in a track **without** any sector write splices. The writing of the complete track starts at a specific point of the rotation of the disk and ends up at a specific point (typically the same point). For non-protected tracks these starting and ending points (WG activation / deactivation) are aligned with the track index.

But even with professional equipment it is not possible to deactivate the write gate at precisely the same position that when it was activated. This results in writing a non-aligned (drifted) last transition, which most likely violates MFM rules, when the WG is deactivated. This area is called the track write splice.

In the case of a standard track the writing starts in the post-index gap G1 and stops in the pre-index gap G5 and therefore the track splice happens in this non-critical area.



However some protections are based on shifting the position of the complete track in respect with the index. For example the index might be positioned in the middle of the last data field (data over the index protection).



In such a case it is not possible to activate and deactivate the Write Gate at the position aligned with the index because this would result in a track write splice located in the middle of a data field and this would therefore result in corrupted data.

For this kind of protection it is therefore important for the mastering equipment to activate and deactivate the write gate in a position which is still located close to the border of the pre-index and post-index gaps. However in that case the location of the track write splice can be located anywhere with respect to the index.

Atari Floppy Disk Copy Protection

4.13 Hidden data

It is possible to hide data in many different places on a track (read [Copy me, I want to travel](#) by Claus Brod on the subject). The data can be hidden in GAPS of a track that looks normal, or they can be hidden in tracks that seems to have no data, etc.

Here we present some cases of hidden data including data hidden after spurious sync sequence. Usually sync marks are used in sequence of three in front of an address mark (IAM or IDAM). But several games like Dragonflight, Jupiter Masterdrive, or Union demo uses special sequence of false sync marks as a protection.

4.13.1 Union Demo / Dragon Flight hidden sequence

One special sequence (used in Union Demo or in Dragon flight) is read as C2 0B CD B4 F7 ... or 14 0B CD B4 F7 by the read track command of the WD1772. If you analyze this sequence at the bit stream level (using for example Kryoflux or SuperCard Pro device) you will find out that the \$C2 or \$14 bytes are in fact \$5524 sync marks, and that the following \$0B byte is a \$4489 sync mark. It is relatively hard to detect this sequence because you do not always read a sync mark value. This special sequence can be placed on track 41 to render the detection even harder (see [False sync mark detection](#)).

What is interesting is that in fact this sequence can be written on an Atari with a regular WD1772. This sound impossible as normally the byte \$F7 forces the FDC to write two CRC bytes during a write track (format) command. But as we have seen any byte can be escaped by placing a \$F7 escape byte in front of it. Therefore if we use the following sequence of bytes during a write track:

```
00 29 F5 F7 F7
```

The \$F5 byte resets the CRC register to \$CDB4, the first \$F7 forces the FDC to write the content of the CRC register (\$CDB4), and the second \$F7 is escaped (because following a previous \$F7). Therefore the sequence is written as

```
00 29 A1 CD B4 F7
```

During the read track command a first sync mark is decoded as \$C2 or \$14 (depending if the sync mark decoder has started on a clock or data bit) and the second \$4489 sync byte is decoded as \$0B followed by \$CDB4F7:

```
C2 0B CD B4 F7
```

Note that Auit is capable to detect this hidden sequence reported as the dragonflight protection.

4.13.2 Jupiter Masterdrive hidden sequence

One special sequence (used in Jupiter Master Drive) is read by the read track command (tracks 02-04) of the WD1772 as:

```
C2 00 1C 92 10 90 C2 0B CD B4 F7 00 DE AD C0 DE
```

If you analyze this sequence at the bit stream level (using for example Kryoflux or SuperCard Pro device) you will find out that the first and second \$C2 are in fact \$5524 sync marks, and that the following \$0B byte is a \$4489 sync mark. It is relatively hard to detect this sequence if you are not looking for it. As you can see DE AD C0 DE can be read as **dead code**.

What is interesting is that in fact this sequence can be written on an Atari with a regular WD1772. The following sequence can be used during a write track command

```
28 29 55 42 49 4E 4E 29 F5 F7 F7 00 DE AD C0 DE 00 00
```

First note than in this input sequence we find in hexadecimal the ASCII char UBI (55-42-49) that are obviously related to UBISOFT the maker of the game.

Atari Floppy Disk Copy Protection

The \$F5 byte resets the CRC register to \$CDB4, the first \$F7 forces the FDC to write the content of the CRC register (\$CDB4), and the second \$F7 is escaped (because following a previous \$F7). Therefore the sequence is written as

```
28 29 55 42 49 4E 4E 29 A1 CD B4 F7 00 DE AD C0 DE 00 00
```

See [False sync mark detection](#) to understand the following. During the read track command the \$28 \$29 byte are decoded as a \$C2 sync mark and the following bytes are shifted and therefore read as 00 1C 92 10 90. The \$29 \$A1 bytes are decoded as \$C2 \$0B bytes and the stream is resynchronized correctly on the second A1 so that the remaining bytes are interpreted normally as CD B4 F7 00 DE AD C0 DE 00 00.

4.13.3 Realm of the Troll

Track 79.0 seems to contain no data: it has no sector and even no sequences of 3 x \$4489 sync marks. However using a **read track** command we can see that we have a sequence of one \$4489 sync mark, followed by one \$5224 sync mark, followed by the two bytes \$45 \$EF, followed by a sequence of bytes from \$F8 to \$FF, followed by a sequence from \$00-\$F4. These different sequences repeats 23 times.

As we have 00, ..., 28, 29, 30, ... in the sequence we get a [False sync mark detection](#) as already explained: the sequence 28, 29 is interpreted as a \$5224 sync mark and all the following bytes are shifted and therefore read incorrectly (we are in fact reading clock bytes).

00224	2000	007189	01 00 00 00 0F 0E 0C 0C 09 08 08 08 03 02 00 00
00240	1969	007699	01 00 00 00 07 06 04 04 01 0B C2 45 EF F8 F9 FAÂEiøú
00256	1984	008211	FB FC FD FE FF 00 01 02 03 04 05 06 07 08 09 0A	ûüýþý.....
00272	2000	008722	0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A
00288	2000	009233	1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 14 40 !"#%&'(.@
00304	2000	009744	C0 41 C0 40 C0 47 C6 44 C4 41 C0 40 C0 43 C2 40	ÀÀÀ@ÀGÈDÀÀÀ@ÀCÂ@
00320	2000	010255	C0 41 C0 40 C0 1F 9E 1C 9C 19 98 18 98 13 92 10	ÀÀÀ@À.....
00336	1984	010766	90 11 90 10 90 07 86 04 84 01 80 00 80 03 82 00
00352	2000	011277	80 01 80 00 80 0F 8E 0C 8C 09 88 08 88 03 82 00
00368	1984	011787	80 01 80 00 80 07 86 04 84 01 80 00 80 03 82 00
00384	2000	012297	80 01 80 00 80 3F 3E 3C 3C 39 38 38 38 33 32 30?><<9888320
00400	1984	012806	30 31 30 30 30 27 26 24 24 21 20 20 20 23 22 20	01000'&\$ \$! #"
00416	1974	013316	20 21 20 20 20 0F 0E 0C 0C 09 08 08 08 03 02 00	!
00432	2000	013826	00 01 00 00 00 07 06 04 04 01 00 00 00 03 02 00
00448	1984	014335	00 01 00 00 00 1F 1E 1C 1C 19 18 18 18 13 12 10
00464	1969	014844	10 11 10 10 10 07 06 04 04 01 00 00 00 03 02 00
00480	1984	015353	00 01 00 00 00 0F 0E 0C 0C 09 08 08 08 03 02 00
00496	1974	015861	00 01 00 00 00 07 06 04 04 01 0B C2 45 EF F8 F9ÂEiøú
00512	1984	016364	FA FB FC FD FE FF 00 01 02 03 04 05 06 07 08 09	ûüýþý.....

Note that bytes \$F5 to \$F7 are carefully avoided and therefore it should be possible to write this kind of track on directly on an Atari.

Chapter 5. Analysis of Games/Programs

This section provides detailed analysis of some programs/games protections. The purpose is to illustrate the usage of some of the protections described in this document.

However it must be noted that:

- The presence of a described protection mechanism does not imply that it is actually used.
- It is possible that more protections than the one described exist for analyzed games.
- Beware that diverse releases of the same game may exist that uses different protections.
- Only Original diskettes have been used (unless specifically noted). However it is difficult to know for sure that none of these diskettes have not been modified.
- The original diskettes were created in 80's 90's and therefore may be damaged.

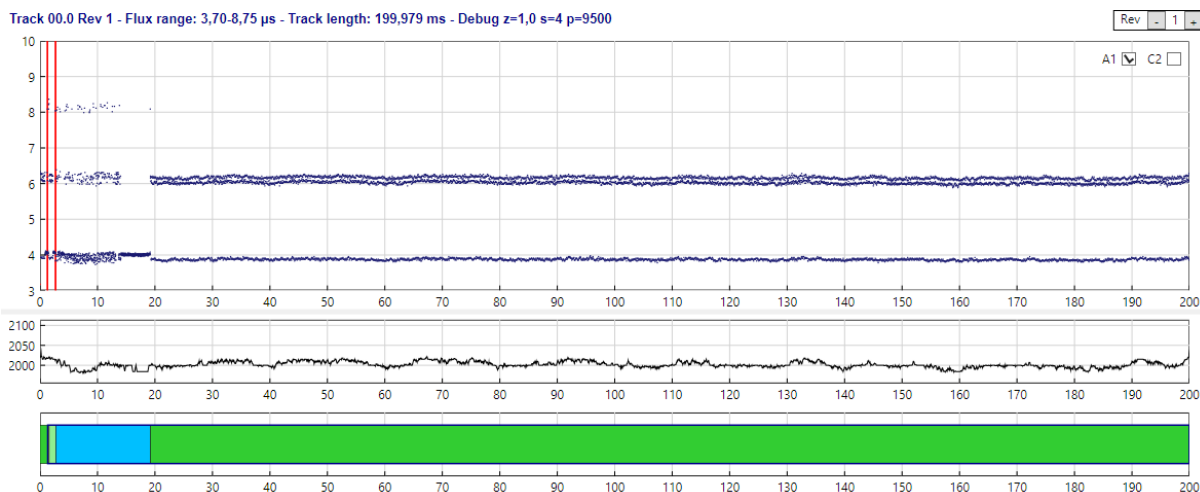
Atari Floppy Disk Copy Protection

5.1 Barbarian (from Psygnosis)

It seems like there are several versions of this game that uses **different** protections. The version I have uses the following protections:

- ★ [Non Standard Number](#) of sector: Track 0 only one sector
- ★ [Track Not Found](#): Track 74-79

If we look at track 0 of Barbarian we first see that the layout is rather unusual as the track has only **one** sector of 512 bytes! The sectors on other tracks are numbered from 10 to 18.



The track 78 seems unformatted (no sync mark, no sector) but however is MFM formatted.

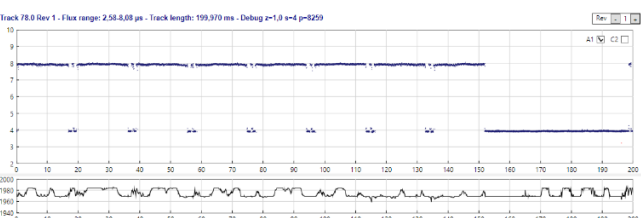
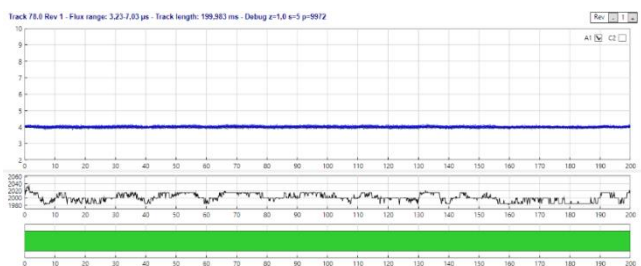
The program performs up to 10 times read track commands on track 78 and tests for value 0xFF or 0x00 at location 28 (0x1C).

On one version of Barbarian this is obtained with a special track that uses a continuous sequence of 4 μ s flux transitions. This results in a MFM string like this ...1010101010101010... Depending on which bit the sampling starts this result in values 0x00 or 0xFF. The switch from 0x00 to 0xFF that can be seen at the start of the track is due to the presence of few glitches at the beginning of the track. It is unclear if this is done on purpose.

On another version of Barbarian that also tests for 0x00 or 0xFF on track 78 but use a more difficult to produce stream of flux transitions. In this version the track uses a sequence of 8 μ s fluxes. This results in a MFM string like this ...100010001000100010001000... Depending on which bit the sampling starts this result in MFM values: 1111 2222 4444 8888 that decodes into 0x55 0x00 0xAA 0x00 respectively.

For this protection the program tests the presence of 0x00 or 0xFF, therefore Pasti imager and AUFIT always decode this kind of sequence as 0x00 in order for the protection to succeed.

Read Track for track 78.0 6232 bytes Length 199980 μ s with 0 sectors												
ID				INTER-GAP			DATA			INTRA-GAP		
SCT	POS	LEN	CRC	BYTE	LEN	BYTE	POS	LEN	CRC	FUZ	BYTE	LEN
										6229 199980		
00000	2051	000029		00	00	00	00	07	FF	FF	FF	FF
00016	2031	000545		FF	FF	FF	FF	FF	FF	FF	FF	FF
00032	2031	001064		FF	FF	FF	FF	FF	FF	FF	FF	FF
00048	2015	001583		FF	FF	FF	FF	FF	FF	FF	FF	FF
00064	2025	002100		FF	FF	FF	FF	FF	FF	FF	FF	FF
00080	2015	002618		FF	FF	FF	FF	FF	FF	FF	FF	FF
00096	2005	003134		FF	FF	FF	FF	FF	FF	FF	FF	FF
00112	2015	003650		FF	FF	FF	FF	FF	FF	FF	FF	FF
00128	2015	004164		FF	FF	FF	FF	FF	FF	FF	FF	FF
00144	2000	004678		FF	FF	FF	FF	FF	FF	FF	FF	FF



Atari Floppy Disk Copy Protection

5.2 Bob Morane

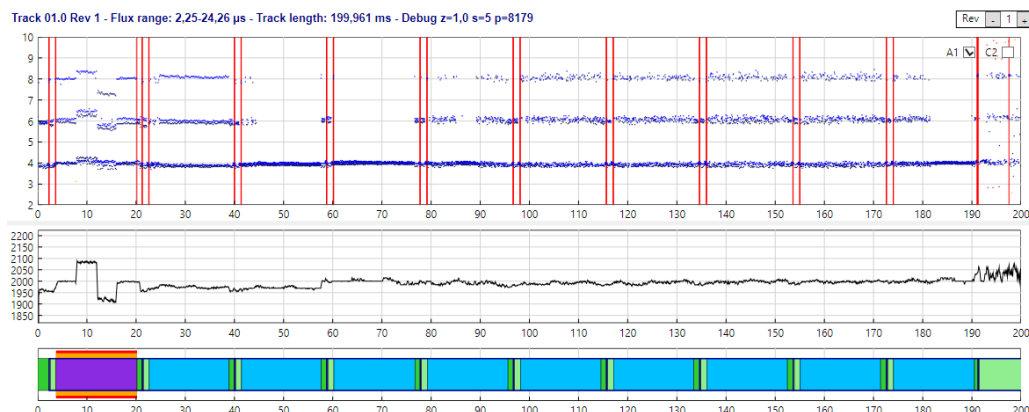
Track 50.0: All the “normal 0x4E” gap byte are replaced by the “invalid 0xF7” byte.

[illegible]

5.3 Colorado

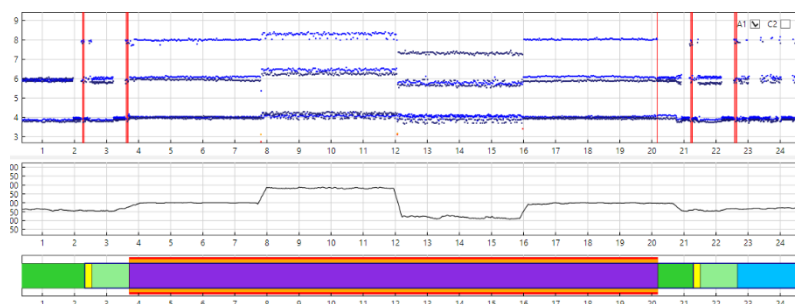
- Track 01.0 I find the following protections:
 - ★ Sector Bit-rate Variation ([SBV](#))
 - ★ Sector with Fuzzy Bits ([FZS](#)) and Data CRC Error ([DCE](#))
 - ★ Invalid ID Field ([IIF](#)) without Data Field ([SND](#))

Here is a plot of the complete track:



If we zoom in the first sector we can see some **intra-sector** clock rate variation. If we look at the intra-sector bit-rate variations we recognize a **Macrodos** protection from **Speedlock**.

Here we can see that the data field is roughly divided into four segments. In the first segment we have normal timing, in the second segment we have above normal clock values, in the third segment we have below normal clock values, followed by the last segment with



normal values. This corresponds well to the definition of [SBV](#) where we have the sector divided into 4 regions with timing: normal, above, below, and normal. Note that each segment is about 128 bytes and that the above and below clock rate compensate. This means that the overall length of this sector is 16487.46 μ s which is very close to a normal 16480 μ s sector.

Probably due to the quick shifting of the clock we have some border bits and therefore the sector also reads with fuzzy bytes and CRC error.

Now if we read the complete track and look at the end of the buffer we have some strange values:

Here we can see an abnormally long sync sequence followed by an IDAM with the following errors:

Nonstandard IDAM (IIF-NSI),
Invalid track number (IIF-ITN),
Invalid sector length (IIF-ISL) and
Invalid ID CRC (IIF-IIC)

09592	2000	189141	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyyyy
09596	2000	189653	FF FF FF FF FF FF FF FF FF FF FF 7F FF FF 9F FF	yyyyyyyyyyyð.yý.ý
09584	2013	190166	FF FF FF FF FF FF FF FF FF FF FC F8 02 12 12 12	yyyyyyyyyyúe....
06000	2023	190681	12 12 12 12 1F FF FF FF FF FF FF E1 A1 A1 A1 A1yyyyyáijii
06016	2048	191081	A1 A1 A1 FF 0E A4 96 A4 02 01 65 18 40 9C 9C	iijý.n..e...x
06032	2000	191703	9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C 9C
06048	2000	192215	9C 9C 9C 9C 9C 9C 9C 9C 9C 9D FF FF FF FFyyyyy
06064	2037	192734	FF FF FF FF FF FF FF FF 3F C9 00 EA 29 5F 2A 0E 54	yyyyyyyyyyü.e)_.T
06080	2039	193254	55 55 00 02 F0 01 32 1F 00 00 01 02 00 04 F5 96	UU..ö.2.....@.
06096	2015	193778	02 BE 80 AC 07 21 04 90 80 08 10 04 05 18 40 C1	.%.~!.....öA
06112	2048	194295	62 90 AC C1 B1 30 2C 02 22 38 20 E1 00 20 58	b..Ätö,"B ää.x
06128	2007	194810	12 0A 77 94 28 41 02 AE 05 00 00 78 0A 00 04	.w.(A.%x..x

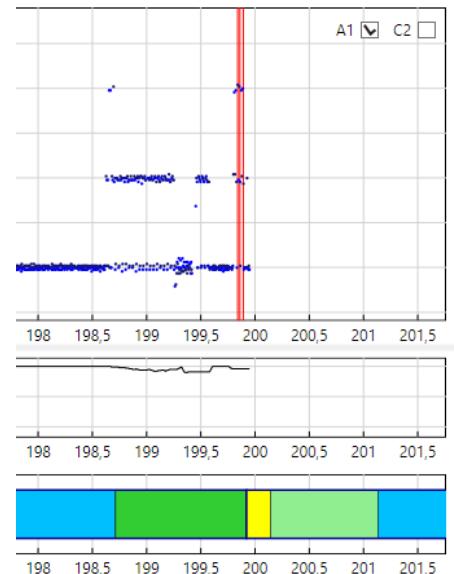
Atari Floppy Disk Copy Protection

raised at the end of the ID postamble (GAP3a) then the WD1772 write 12x\$00 3 sync bytes and a normal data field. This results in the sector to be shifted by 6 bytes but the data postamble (GAP4) is only 4 bytes and therefore we are already in the next sector!

Track 0-78 (disk 2) – Data beyond Index

The last (first?) sector of each track is pushed very close to the end of the track. If you look at the diagram you will see that in fact the sync mark and the first byte of the ID field are located before the index and the rest beyond the index including of course the data field. It is absolutely impossible to do this on an Atari and this requires very precise mastering machine. As the ID field is located “above” the index it is not possible to start/stop the writing aligned with the index (this would result in write splice inside the ID field).

If you look carefully at the transitions between 199 and 199.5 it seems that we have the track write splice at this location. This means that the mastering machine would start / stop writing about half millisecond before the index.



If we look at the end of this track buffer we find something like:

```
+ GAP2 20 bytes @199325 us length=632.73 us - TMV=0 BRD=0
186d 199325 4000 ff ff ff ff fe 01 39 39 39 38 00 00 00 00 00 00 .....9998.....
187d 199840 4000 02 a1 a1 a1 .....
= ID=0 1 bytes @199958 length=19.95 T=0 H=0 S=0 Z=512 CRC=0000 *** BAD *** TMV=0 BRD=0 BS=0
1881 199958 4031 fe
```

As you can see here we only have a sync sequence followed by an IDAM but not the rest of the ID field (remember the read track command terminates at the index). This start of the ID field (the IDAM) is therefore at the very end (only few micro seconds) of the track and therefore the rest of ID field must be at beginning of track.

Therefore if you do a **read track** command on a real Atari you have all the chance not to see this ID field. For example here is the content of the end of the track buffer as read by the **Panzer** program on a real Atari:

```
1830 3973 ff 80 00 00 00 3f ff ff ff 80 00 00 00 3f ff ff .....?.....?..
1840 3973 ff 80 00 00 00 3f ff ff ff 80 00 00 00 3f ff ff .....?.....?..
1850 4037 ff 80 00 00 00 3f ff ff ff e0 10 c8 48 48 48 48 .....?.....HHHH
1860 3973 48 48 48 48 48 48 48 48 00 00 00 00 00 00 00 00 HHHHHHHH.....
1870 4069 00 00 10 90 90 90 90 ff ff ff ff ff ff ff c2 a1 .....

```

Here you can see that we have the start of the sync sequence but not the IDAM. This is due to the Atari DMA circuit: the DMA always delivers multiples of 16 bytes due to the buffering mechanism and therefore up to 15 bytes may be “stuck” in the DMA buffer at the end of the **read-track** command.

However the WD1772 will detect this ID field without problem with a **read-address** command and will find the corresponding DATA field with the **read-sector** command.

Therefore it looks almost impossible to position this *ID Field* with this precision by software and mastering machines are required.

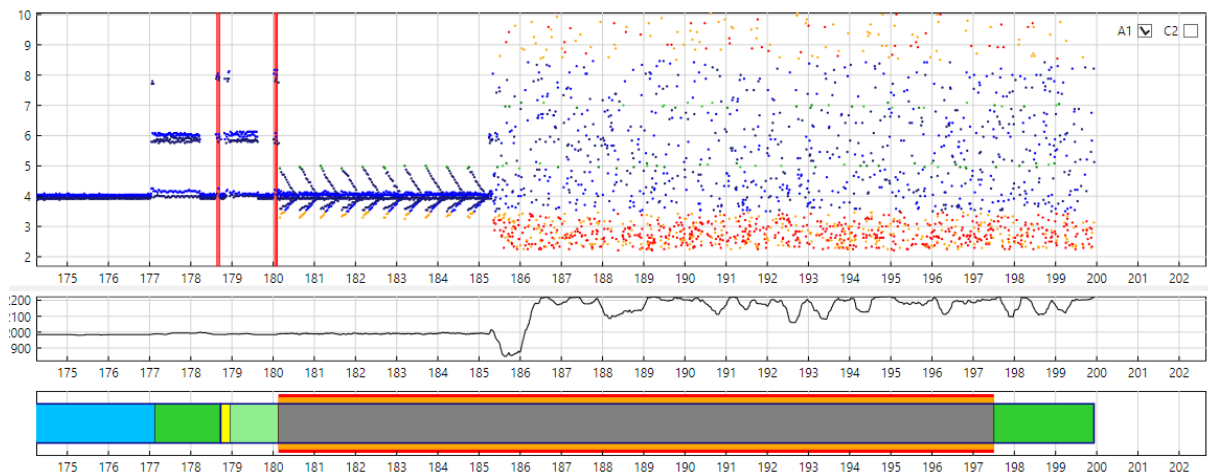
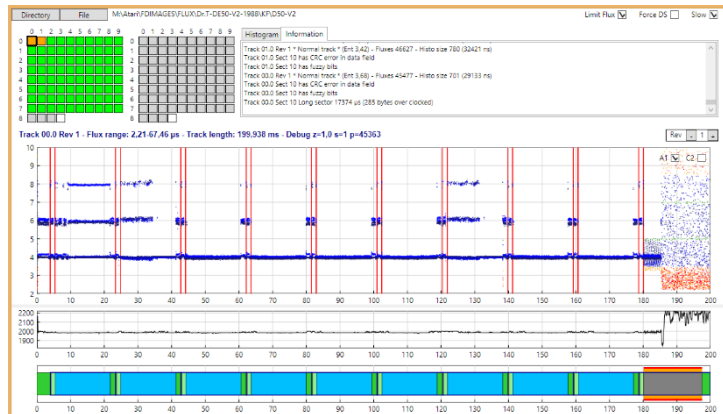
Atari Floppy Disk Copy Protection

5.5 D50 Editor V2 (Dr.T)

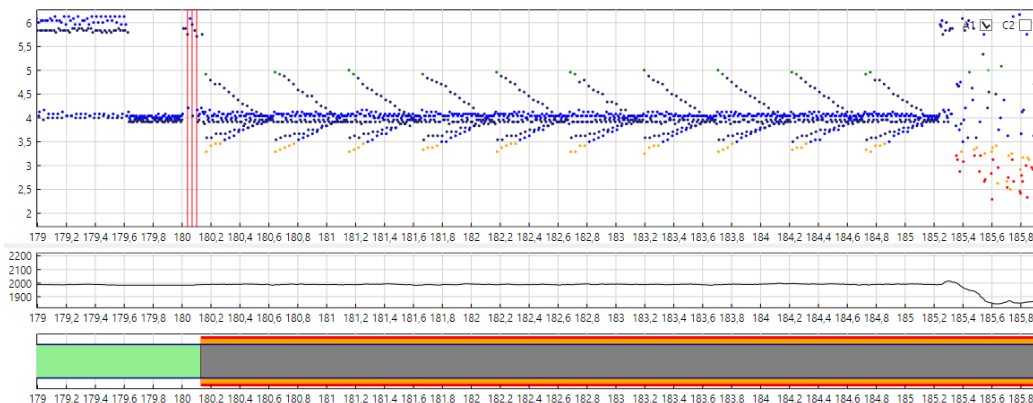
The D50 Sound Module Editor program from DrT uses the following protection mechanisms:

- ★ Track 00.0-79.0 sector 10 has fuzzy bits in the *Data Field*.

We can see that sector 10 is partially unformatted but it has also a lot of border bits. Here is a zoom on sector 10:



If we further zoom we can see that after the position 180000 we have a lot of border bits in the range from 3 to 5 μ s with a strange pattern that “compensate in pair”. This results in an average 2000 μ s cell but for sure all these border bits should generate fuzzy bits.



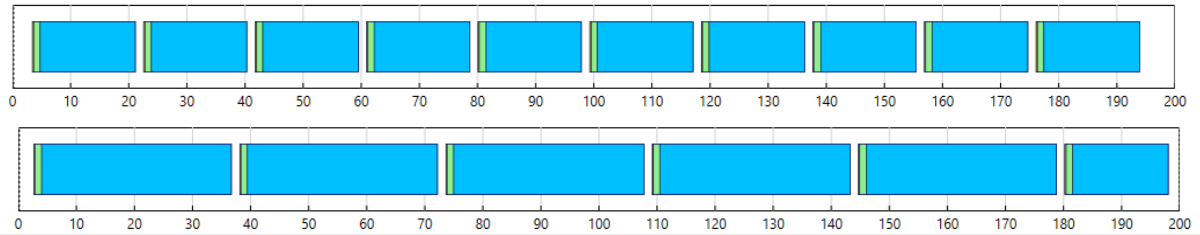
After the position 185000 we can see that we have random flux reversals. This pattern is typical of an unformatted track. Therefore we can conclude that the formatting of the track is stopped after about one third of the last sector. This is obviously not feasible with the WD1772 FDC and therefore to copy this track it is necessary to have special mastering device like Discovery Cartridge or KryoFlux board or Supercard Pro.

Note that random flux reversals result into unpredictable clock frequency (and also unpredictable inspection windows position) of the DPLL. This and the presence of border bits results in fuzzy bytes in the sector.

Atari Floppy Disk Copy Protection

5.6 Dragon flight

- Track 00.0 and 00.1 have 10 sectors of 512 bytes and all other tracks have 5 sectors of 1024 bytes plus one sector of 512 bytes.



- Most (if not all) tracks have hidden data into gap (HDG). The tracks starts with sync sequence \$5224 and \$4489 sync marks followed by \$CD-B4-F7. This sequence is read as 14-0B-CD-B4-F7 or C2--0B-CD-B4-F7
- Track 00.0 and 00.1 have a sequence of 8 invalid character \$F7 in the gap following the hidden data describe above (IDG).
- Track 00.0 and 00.1 uses an invalid head value (48 or \$30 in hex) in the ID fields of all sectors (IIF-IHN)
- All other tracks uses an invalid head value (48 or \$30 in hex) in ID fields of all sectors (IIF-IHN) as well as an invalid track number value (178 or \$B2 in hex) in ID fields of all sectors (IIF-ITN)

00000	1999	000028	24 E4 E4 E4 E0 01 72 72 72 72 72 72 72 72 72 72 72	\$äääää.rrrrrrrrrr
00016	2015	000546	72 72 72 72 72 72 72 72 72 72 72 72 72 72 72 72	rrrrrrrrrrrrrrrrrr
00032	2015	001060	72 72 72 72 72 72 72 72 72 72 72 72 72 72 72 72	rrrrrrrrrrrrrrrrrr
00048	2015	001574	72 72 72 72 72 72 72 72 71 14 0B CD B4 F7 00 41 F7	rrrrrrrq..İ÷.A÷
00064	2000	002074	F7 F7 F7 F7 F7 F7 F7 4E 4E 4E 4E 4E 4E 4E 4E 4E	+++++NNNNNNNNNN
00080	2000	002588	4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 00 00 00 00 00	NNNNNNNNNNNN.....
00096	2000	003100	00 00 00 00 00 00 00 14 A1 A1 FE 00 30 01 02 0Fijb.0...
00112	1994	003611	CA 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E	ĖNNNNNNNNNNNNNNNN
00128	1984	004121	4E 4E 4E 4E 4E 4E 00 00 00 00 00 00 00 00 00	NNNNN.....
00144	1984	004627	00 14 A1 A1 FB 60 1C 44 14 4F 0C 08 08 63 20 4F	..jjü`.D.O...c O
00160	1984	005132	3F FE 7E 7F 3F FE 73 FF FE 7F 81 3F 3F FE 7F FF	?b~.?psÿp...?p.ÿ
00176	1984	005641	3F FF FF E0 20 FF FC C8 48 40 98 03 C0 FF FE 44	?ÿÿä yüEH@..ÄÿpD
00192	1984	006152	04 49 C0 7F C0 60 20 FF FC 48 48 40 98 09 C0 7F	.IÄ.Ä` yüHH@..Ä.

Atari Floppy Disk Copy Protection

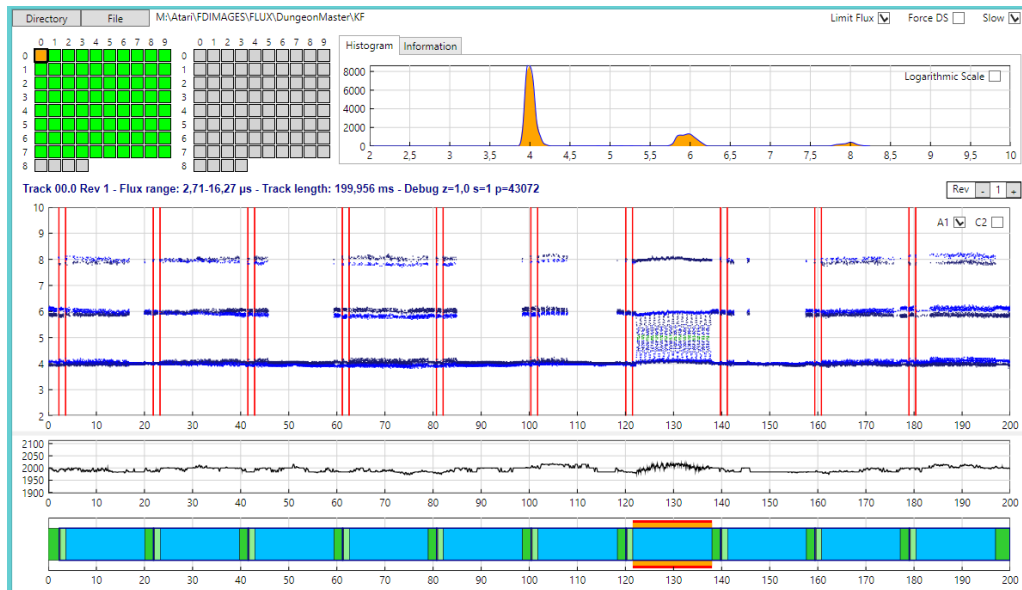
5.7 Dungeon Master (FTL Inc.)

For detail analysis of the Dungeon Master & Lost Scroll protection please refer to the [DM Protection document](#), the [detailed analysis of the Dungeon Master and Chaos Strikes Back for Atari ST Floppy Disks](#) and the [US patent "Copy Protection for computer Disc 4,849,836"](#))

The game "Dungeon Master" uses the following protection mechanisms:

- ★ [Invalid Sector Number](#): Track 0 the sector 8 is numbered 247.
- ★ [Fuzzy bits & Sector with bad Data](#): Track 0 sector 7 the *Data Field* has bits in Ambiguous areas resulting in a fuzzy sector with CRC error.

Here is the Layout of track 0

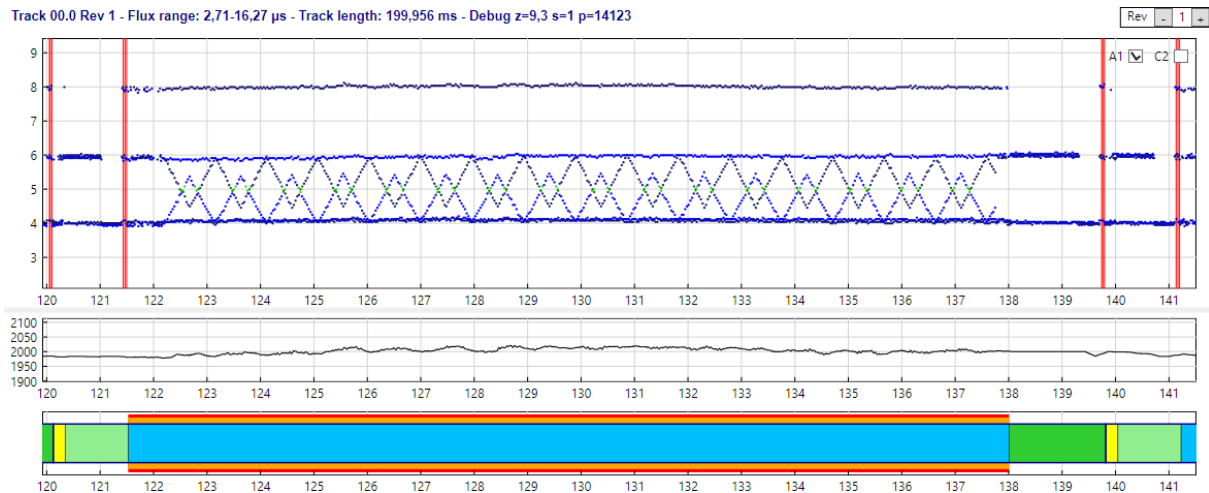


As you can see in sector 7 we have a lot of **border bits (BRD)** aka bits in Ambiguous area. Looking at the content of this sector we can see that the clock period range from 3938 ns to 4031 ns with an overall clock period of 4.01 μs

```
Detail buffer content for sector 7 with 515 bytes
= DATA ID=7 515 bytes @121545 us length=16506.79 CRC BAD CLK=4.01 TMV=0 BRD=495 DOI=0
*** Fuzzy Sector *** starting at byte position 34
0000 121545 3968 fb 07 50 41 43 45 2f 46 42 09 53 65 72 69 ca 08 ..PACE/FB.Seri..
0010 122055 3968 00 00 ef e9 01 68 68 68 68 68 68 68 68 68 68 .....hhhhhhhhhh
0020 122565 3938 68 68 68 e8 e8 e8 e8 e8 68 68 68 68 68 68 hhh.....hhhhhhh
0030 123073 3968 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhhh
0040 123583 4031 68 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 68 h.....hhhhhhhhh
0050 124092 3968 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhhhhhhh..
0060 124604 4000 e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 ...hhhhhhhhhhh
0070 125114 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhhhh....
0080 125628 3968 e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 ...hhhhhhhhhhh
0090 126141 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhh.h...h
00a0 126654 4031 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 .hhhhhhhhhhhhh
00b0 127168 4031 68 68 68 68 68 68 68 68 68 e8 e8 e8 e8 68 68 hhhhhhhh....hhh
00c0 127683 3968 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhhh
00d0 128197 3938 68 68 68 68 68 68 68 e8 e8 e8 e8 28 68 68 hhhhhh....(hhh
00e0 128710 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhh
00f0 129226 4063 68 68 68 68 68 68 e8 e8 e8 e8 68 68 68 68 hhhhhh....hhhhhhh
0100 129741 4031 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhh
0110 130257 4162 68 68 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 hh.....hhhhhhhhh
0120 130771 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhhhhhhhh.
0130 131288 3938 68 e8 e8 e8 e8 e8 68 68 68 68 68 68 68 68 h.....hhhhhhhhh
0140 131802 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhhhh..h
0150 132319 4063 e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 ...h.hhhhhhhhhh
0160 132831 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 e8 hhhhhhhhhhhh....
0170 133346 3938 e8 e8 e8 68 68 68 68 68 68 68 68 68 68 68 68 ...hhhhhhhhhhh
0180 133858 4031 68 68 68 68 68 68 68 e8 68 e8 e8 e8 e8 68 hhhhhhhh.h....h
0190 134371 4063 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhh
01a0 134882 4000 68 68 68 68 68 68 68 68 e8 e8 e8 e8 68 68 hhhhhh.hh....hh
01b0 135395 4063 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhh
01c0 135906 4063 68 68 68 68 68 68 68 e8 e8 e8 e8 68 68 68 68 hhhhhh....h..hh
01d0 136418 3968 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhhh
01e0 136931 4000 68 68 68 68 e8 68 e8 e8 e8 68 68 68 68 hhhh.h....hhhhh
01f0 137443 4000 68 68 68 68 68 68 68 68 68 68 68 68 68 ac hhhhhhhhhhhhhh.F
0200 137956 4000 42 3a f8 B:..
```

Atari Floppy Disk Copy Protection

Let's zoom to sector 7:



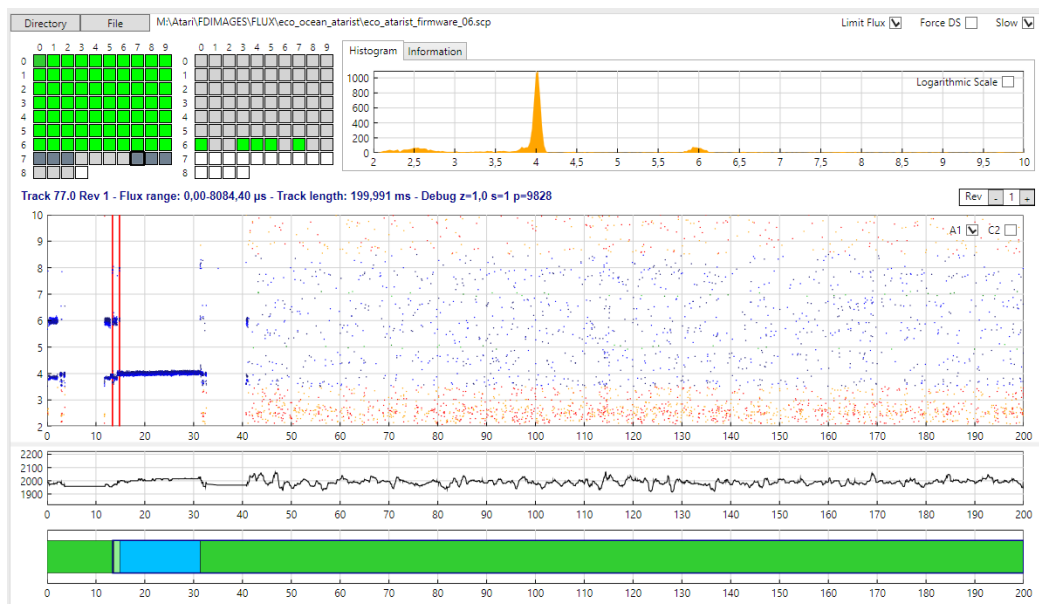
We can see that the flux reversals spacing follow a strange pattern and includes a lot of “border bits”.

Here we can see that the beginning of the sector has normal timing. But after the position 122000 we have the bit reversals gradually sliding to the border of the inspection window (close to 5000 ns). We can see that we have a pattern that looks like a sine wave and this implies that many bits are at the border of the inspection window.

As explained in the [WD1772 DPLL Input Circuitry](#), having reversals at the border of the inspection windows will result in random value latched by the DPLL data separator and therefore these bits can be considered as **Fuzzy Bits**. Reading this sector several times will results in different values returned due to the floppy disk rotation speed fluctuations.

5.8 Eco by Ocean

Tracks 77 and 79 have one sector number 2 and the rest is unformatted. The track look like this



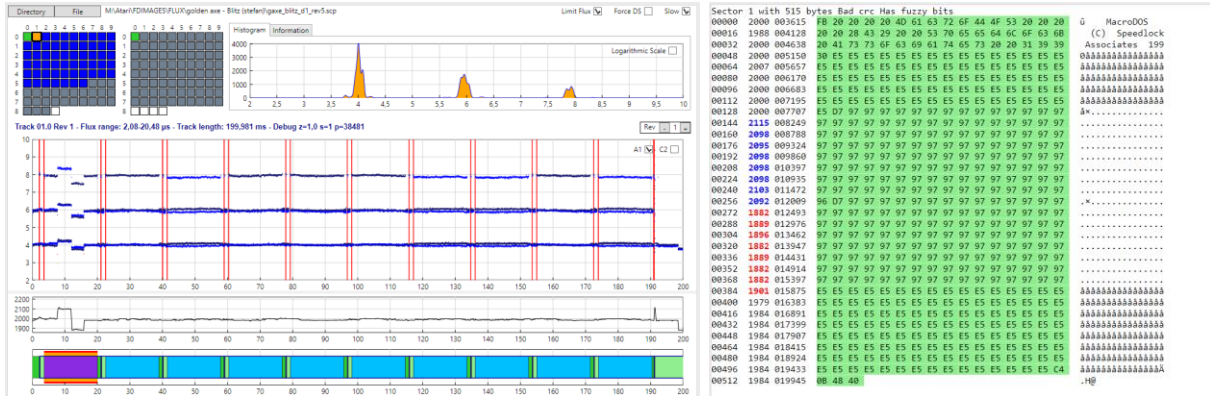
Protection checked using simple read and write calls. Checks that no sector #1 is present on track and then it tries to write on sector #2. If any of this test fails the program freezes.

Atari Floppy Disk Copy Protection

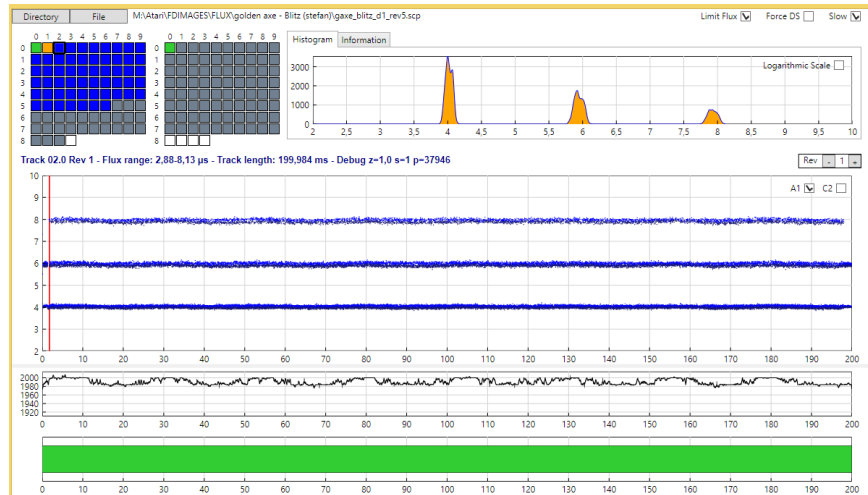
5.9 Golden Axe

On track 01.0 I find the following protections:

- ★ Intra-sector Bit-rate Variation ([IBV](#)) – MacroDOS/Speedlock
- ★ Sector with Fuzzy Bits ([FZD](#)) and Bad Data ([CRC](#))
- ★ Invalid ID Field ([IIF](#)) without Data Field ([SND](#)) (see [Colorado](#) for the [Invalid ID field](#))



Track 02.0 – 56.0
contain [Data Tracks](#).



All these tracks are read using a read track command. In this specific version we have a sequence of 3 A1 sync mark followed by the data track. The escape character used is 0x0F.

Read Track for track 02.0 6273 bytes Length 199982 µs with 0 sectors												
ID	POS	LEN	CRC	INTER-GAP	DATA	POS	LEN	CRC	FUZ	INTRA-GAP	DATA	POS
SCT				BYTE	LEN	BYTE				BYTE	LEN	
											6270	199982
00000	1971	000025	02	4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E	00	00	00	00	00	00	00	00
00016	1992	000567	4E	4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E	00	00	00	00	00	00	00	00
00032	1992	001077	4E	4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E	00	00	00	00	00	00	00	00
00048	1988	001590	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00	00	00	00	00	00	00	00
00064	2000	002099	1E	21 BE 46 DD 9F 80 47 AD EC 5C F4 0F AC 7A 09	00	00	00	00	00	00	00	00
00080	2004	002611	7F	98 5C 70 5F 3A 50 70 00 C2 21 72 8F E6 64 2E	00	00	00	00	00	00	00	00
00096	2000	003124	4E	06 A9 23 5D D4 07 5B 8B D7 C3 42 2A DC 0F 09	00	00	00	00	00	00	00	00
00112	1990	003635	5B	1A FF 21 46 70 A0 D2 7F A9 3F 77 80 61 8B 0F	00	00	00	00	00	00	00	00
00128	2000	004144	09	0F 0A 0F 93 61 24 B9 EA F3 17 E1 BE 63 A1 F8	00	00	00	00	00	00	00	00
00144	2000	004658	1A	C3 5E A4 BA C2 44 4F FA B8 90 78 FB 55 5E 1D	00	00	00	00	00	00	00	00
00160	2000	005171	BC	57 37 3F A7 E6 90 8D 30 39 84 A9 39 B9 0F 08	00	00	00	00	00	00	00	00
00176	2015	005684	68	64 3C 72 C8 E8 04 2B DE 57 67 8C 6A A7 0F 08	00	00	00	00	00	00	00	00
00192	2000	006192	21	CA 47 DA 64 FF 3A F0 85 FB 7F 4D EB 23 A7 41	00	00	00	00	00	00	00	00
00208	2000	006709	0C	CA B7 C4 F2 B3 0B 84 A2 E5 24 D8 8D BE 85 0F	00	00	00	00	00	00	00	00
00224	2004	007221	DA	E1 8A 0F BF BA A3 28 3E C2 79 57 FC 10 A0 B5	00	00	00	00	00	00	00	00
00240	2000	007732	4C	02 41 87 D4 D7 63 E5 E7 4F A6 C0 D2 E0 56 36 16	00	00	00	00	00	00	00	00
00256	2000	008244	8E	E8 17 43 0E C7 62 31 64 28 EF 6C DC B3 83 27	00	00	00	00	00	00	00	00
00272	2000	008749	DD	12 73 A3 D3 E3 2F 70 30 58 87 E7 E7 48 51 70	00	00	00	00	00	00	00	00
00288	2000	009261	3D	BB 12 63 B3 34 44 BC 35 44 7E A0 D2 81 9C 15	00	00	00	00	00	00	00	00
00304	2000	009779	A8	B8 79 77 2A 01 BF B2 B9 A6 F9 C0 62 A5 D4 8A	00	00	00	00	00	00	00	00
00320	2000	010289	0F	82 09 C6 25 5D 36 CA 4C 9C 57 B2 F2 9D 8F 45	00	00	00	00	00	00	00	00

Atari Floppy Disk Copy Protection

5.11 Kick Off 2 (Anco Software 1990)

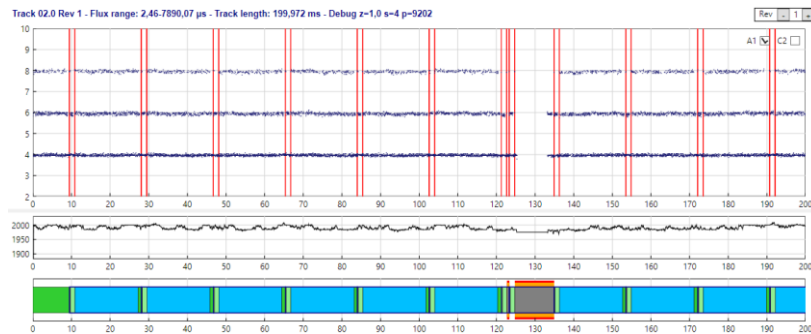
Kick Off 2 uses combinations of the following protection tracks 02.0-06.0:

- ★ [Nonstandard Sector's Number](#): 12 Sectors/Track
- ★ [Data Over Index pulse](#)
- ★ [Sector Within Sector](#) (and even Sector Within Sector Within Sector)
- ★ [Non Standard sector Size](#) (1024)
- ★ [No Flux Area](#)
- ★ [Fuzzy bytes](#)

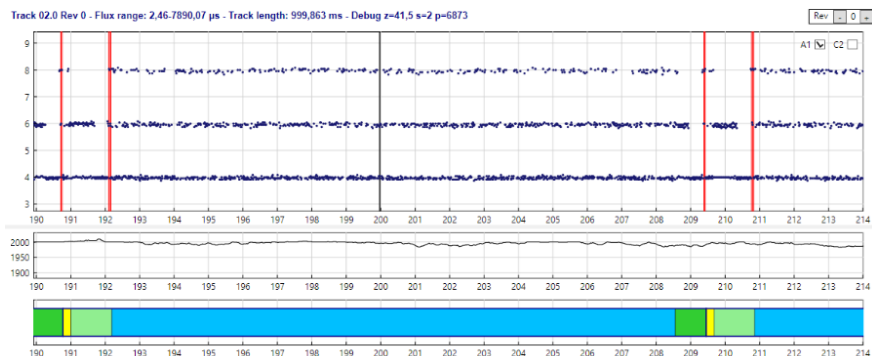
Here is the complete content of track 2

Few things to note:

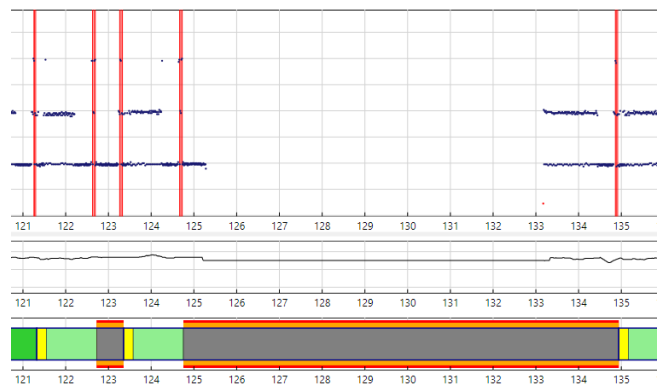
- ★ We clearly see that we have a No Flux Area (NFA) around 125ms
- ★ Just before this NFA we have several overlapping sectors.
- ★ The last sector continue past the index



If we zoom around the end of track (@ 200ms) we can see that the last sector has its ID field around 191ms and the Data field starts around 192ms and terminate at about 9ms in the next revolution.



Now if we zoom close to the NFA we can see a first sector (sector 0 with a size of 1024) and inside this sector we have a second sector (sector 16 with a size of 1024). This is the Sector within Sector protection (SWS). Both of these sectors included in their data field the NFA area (that reads with Fuzzy bits and CRC error). During mastering the flux transitions are carefully crafted so that the included sector 16 is shifted by a half cell (using a normal \$A1 sync mark) from the sector 0. Therefore the read sector command for sector 0 reads the "data bits" of the NFA (remember that during read sector the sync mark detector is disabled). The read sector command for sector 16 reads the "clock bits" of the NFA because this sector is shifted by a half cell compared to sector 0. This technique allows to check the presence of an NFA areas (both clock and data bits equal to zero) in spite of the limitation of the WD1772 that can normally only read the data bits of a sector.



See also [Checking NFA with the WD1772](#)

Atari Floppy Disk Copy Protection

5.12 Maupiti Island

Track 00.0 of disk 1 and disk 2 are normal Atari track with 9 sectors. Track 00.0 on disk 1 contains a boot loader used to load the game. Track 00.0 on disk 2 is used to save game state. All the other tracks on both disk use a non-standard [Data track](#) format.

This format contains three standard \$4489 sync marks followed by

- ★ \$FE
- ★ \$07 (escape character),
- ★ track number
- ★ \$07,
- ★ checksum high byte
- ★ \$07
- ★ checksum low byte
- ★ \$07
- ★ 5842 data bytes

The code just searches for two \$A1 at the beginning of the track, followed by \$FE. It also tests the track number (it has to match the position of the head, bit 7 is set for side 2). The checksum is an unsigned word, which is calculated by adding all 5842 (unsigned) data bytes together. The checksum is also tested.

Several things to note:

- Because the data bytes can contain any byte \$00-\$FF it can't be written with a standard FDC write track, which can't write bytes \$F5-\$F7.
- During a read track the sync detector of the FDC is active at all time. It is well known that specific sequence of bytes can be interpreted by the FDC as \$5224 sync mark. This causes all the following bytes to be read incorrectly. To prevent that happening a very simple yet efficient method is used: a \$07 escape byte is inserted in front any byte that would lead to a sync mark. [Claus Brod](#) wrote a wonderful article about this. This is also the reason to have these bytes in the track header! (See also [Obitus](#)).

5.13 Night Shift (US Gold)

- Track 0-78
 - ★ 2 Sector 66 [Sector with No Data](#)
 - ★ Sector 66 [Duplicate Sector](#)

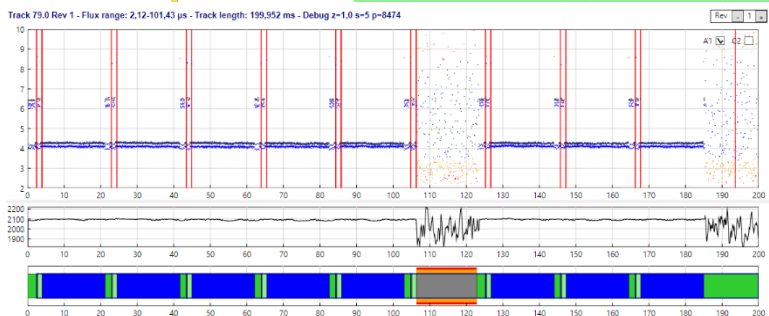
At the beginning of the track we can see that we have a sector 66 not followed by a data segment. At the end of the track we have a duplicate sector 66 also without a data segment. A read sector returns a RNF status for sector 66.

- Track 79
 - ★ Sector 6 Fuzzy bits + CRC error.
 - ★ [Short Track](#) (about 6000 bytes)

We can see that sector 6 contain an unformatted area. This result in random byte read (Fuzzy bytes) and of course a CRC error.

It seems that there is another unformatted area at the end of the track.

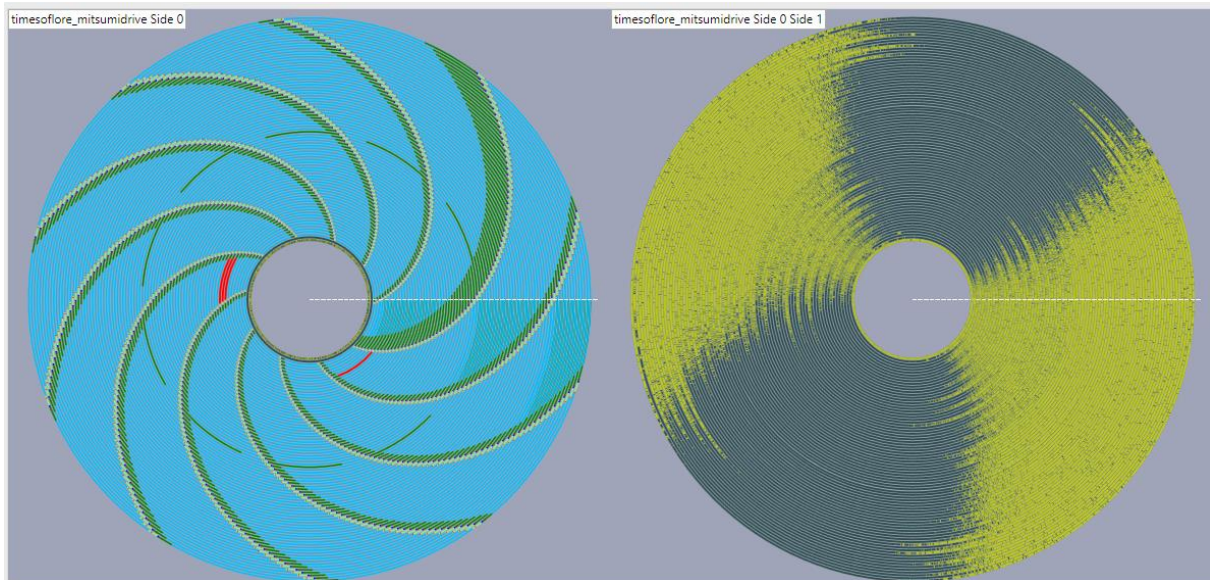
Read Track for track 00.0 6279 bytes Length 199943 µs with 11 sectors													
ID	SCT	POS	LEN	CRC	INTER-GAP	DATA	POS	LEN	CRC	FUZ	INTRA-GAP	BYTE	LEN
					BYTE	LEN					BYTE		
66	16442	222	OK		35	1197					513	16442	
1	17842	222	OK		34	1177	515	19252	16381	OK	No	50	1177
2	37308	222	OK		35	1197	515	38718	16358	OK	No	50	1197
3	56744	221	OK		34	1175	515	58150	16350	OK	No	50	1175
4	76171	222	OK		34	1175	515	77577	16373	OK	No	50	1175
5	95621	222	OK		34	1175	515	97028	16408	OK	No	50	1175
6	115111	222	OK		34	1177	515	116519	16419	OK	No	50	1177
7	134618	222	OK		34	1178	515	136028	16463	OK	No	50	1178
8	154174	223	OK		34	1179	515	155585	16447	OK	No	50	1179
9	173720	223	OK		34	1181	515	175134	16469	OK	No	50	1181
66	193288	222	OK		716	22894							
00496	1984	015839			4E	4E	4E	4E	00	00	00	00	00
00512	1984	016345			A1	A1	A1	FE	00	00	42	02	C2
00528	1984	016836			09	09	09	09	09	09	09	09	09
00544	1984	017346			0F	FF	FF	FF	FF	FF	FF	FF	A1
00560	1984	017842			FE	00	00	01	02	CA	6F	4E	4E
00576	1994	018351			4E	4E	4E	4E	4E	4E	4E	4E	4E
00592	1984	018863			00	00	00	00	00	00	00	A1	A1
00608	1984	019370			4E	4E	4E	4E	57	CB	A5	00	02
00624	1984	019880			D0	02	F8	05	00	09	00	01	00



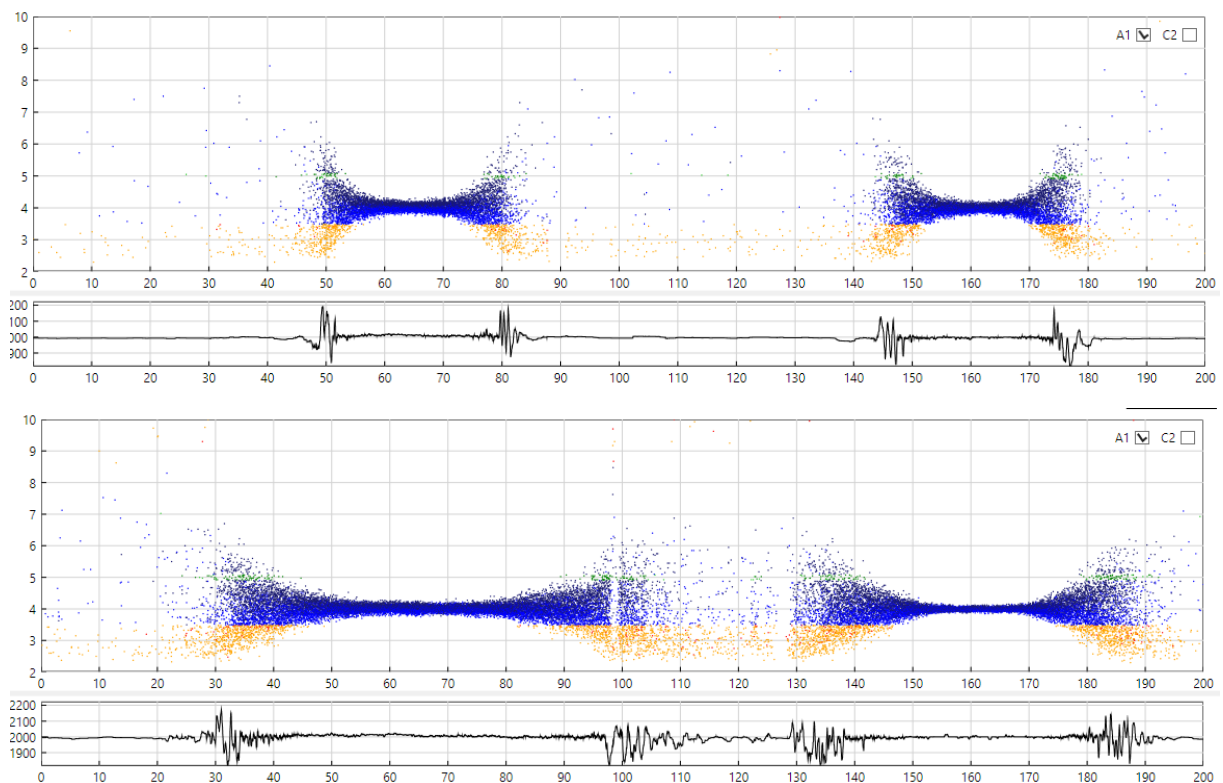
Atari Floppy Disk Copy Protection

5.21 Time of lore

Most tracks are shifted tracks. What is interesting is that the shifting of the track is proportional to the track number. We have the following pattern



Side 1 of the disk contains some strange flux transitions



But probably not used?

Atari Floppy Disk Copy Protection

5.23 Vroom

- Track 00.0 + 77.0 + 78.1 + 79.1 seems to be normal tracks.
- Tracks 01.0-76.0 + 00.1-76.1 are [Data Tracks](#). They seems to use the same format as described in [Maupiti Island](#) using a \$07 escape character (Lankhor ST format?).

- Track 78.0 is extremely strange. It contains 9 sectors alternatively formatted and unformatted. At the end of the track we find a data segment followed by another data segment. In other word a data segment not preceded by an ID segment. There is no way to read this kind of segment with a **read sector** command however it is possible to read it with

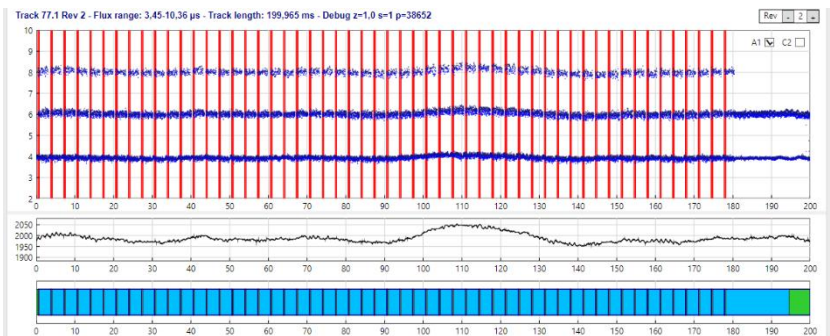
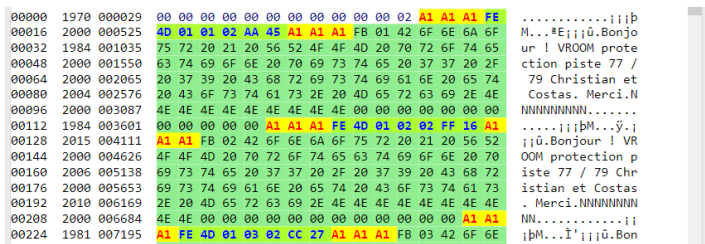
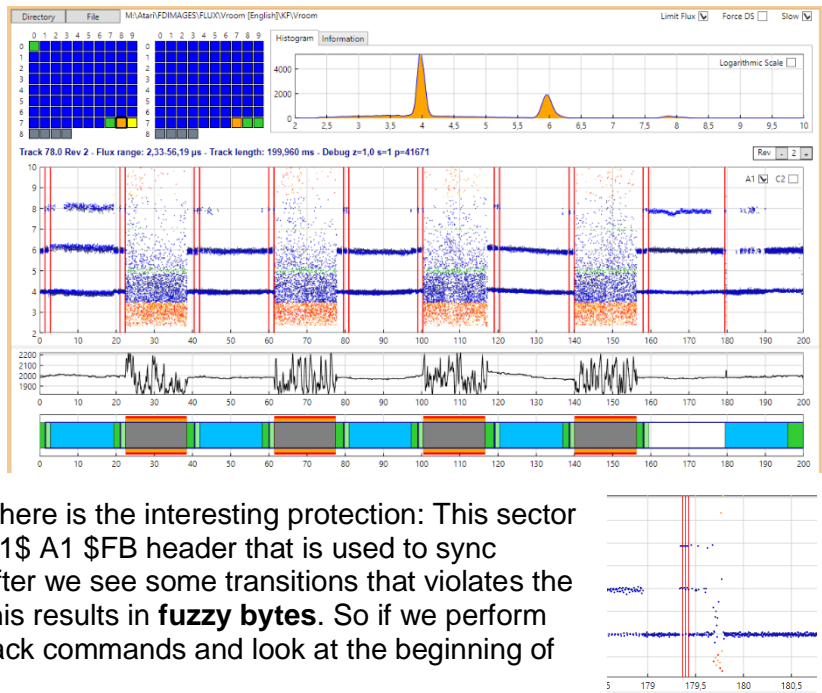
a read track command. But here is the interesting protection: This sector contains the normal \$A1 \$A1\$ A1 \$FB header that is used to sync correctly the WD1772 but after we see some transitions that violates the normal MFM coding and this results in **fuzzy bytes**. So if we perform several consecutive read track commands and look at the beginning of this segment we have:

```
rev 1 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB 7F F7 BF 7F FF FF FF FF...
rev 2 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB 7F E5 BF 7F FF FF FF FF...
rev 3 we have A1 A1 A1 FB 00 4E 80 00 E7 7F FB FF F5 00 80 00 00 00 00...
Etc.
```

Therefore this track uses the [Fuzzy Track](#) protection. Note that this kind of protection cannot be imaged with the Past format. See also [Power Drift](#).

- Track 79.0 is also strange it contains 2 \$A1 sync mark followed by a constant set of MFM flux at 4μs decoded as a continuous sequence of \$00 bytes.

- Track 77.1 contains 54 overlapping sectors. But with an extremely strange pattern I have never seen before. The data field follow the ID field immediately **with zero gap's byte!** Normally a read sector command should not work as no time is given to the WD1772 to "react" to the detection of the correct ID. Note that the sort data segment contains in French the message: "Hello.
Vroom protection track 77 / 79 Christian and Costas. Thanks."



Chapter 6. References

6.1 Documents / Articles

- Article on protection "[copy me I want to travel](#)" from [Claus Brod](#) the expert who wrote the book [Scheibenkleiste](#) covering all sort of interesting details about floppy disks, hard disks, RAM disks, CD-ROMs and other mass storage devices for the Atari (Claus web [site](#)).
- [Probing the FDC: Learn the Secrets of your Floppy - By David Small](#)
- [Atari Protected Disk Image Format & Atari Preservation Project](#)
- [Floppy disk format How can I copy my copy-protected Atari software](#)
- [An interview with Rob Northen](#)
- [Dungeon Master Copy Protection](#)
- [Disk Backup Programs: Do they really work](#)
- [Teac & Citizen](#) Micro Floppy Disk Drive Specification
- [Floppy from HP](#)
- [How to HD install Pacland \(MFM format\) using WHDLoad](#)
- [Commodore C1581-handler](#)
- [S100-Manuals - Disks and Disk Drives](#)
- [Wipe Swap File](#)
- [SpinRight Technical note](#)
- [An interview with Rob Northen](#)

6.2 Forums Threads

- [Way how SW testing copy protection](#)
- [Analysis of submitted games](#)
- [List of difficult to copy disks](#)
- [Floppy Disk Copy Protection](#)
- [Copy Protection details](#)
- [Looking for Rob Northen originals](#)
- [Rob Northen Code Found](#)
- [Weak Bits, Bit-rate var., data under index: Copy Protection](#)
- [Questions Regarding STT Images](#)
- [Protected disk images project & CAPS](#)
- [Ideas about ST floppy image make program for PC](#)
- [PASTI Project](#)
- [Copy II ST](#)
- [Looking for AntiBitos 1.4 by illegal](#)
- [Most memorable Hack/crack](#)
- [Protected Disk Image Project Seeking Beta Tester](#)
- [Ideas about ST floppy image make program for PC](#)
- [Looking for DMA file under interrupt](#)
- [Mega STE Specifics](#)
- [Copy Protected Disks](#) at AtariAge
- [Gcopy DIM file](#)
- [ST Protection routines](#)
- [Putting a second internal floppy drive in the STF](#)
- [RamDisk and ATARI-ST Disk IO](#)
- [X-out original protected](#)
- [Copy Protected Disk](#) – Turrigan NFA Protection (IFW / Mr. Vince)
- [List of Difficult to Copy Disks](#)
- [WD1772 behavior on too many syncs](#)
- [Routine to measure read sector time](#)
- [Trouble using FloImg or FdRawCmd](#)
- [WD1772 behavior on too many syncs](#)

Atari Floppy Disk Copy Protection

6.3 Related Patents

You may want to look at the following [patents](#) that describe some protection mechanisms:

- [Copy Protection for computer Disc 4,849,836](#)
- Computer Program protection method 4,462,078
- Hardware key-on-disc for copy protecting magnetic storage data 4,577,289
- Copy protecting system for software protection 4,584,641
- Techniques for preventing unauthorized copying of information recorded on a recording medium and a protected recording medium 4,734,796
- Copy protection disc format controller 5,432,647
- Data Input Circuit with Digital Phase Lock Loop

6.4 Web Sites

- [Atari ST FD \(Hardware view\)](#)
- [Atari ST FD \(Software view\)](#)
- [Atari FD Protection/Preservation](#)
- [Atari ST Copy Protections \(Markus Fritze\)](#)
- [Protections sur Atari ST/Amiga](#)
- [PASTI Project](#)
- [Software Preservation Society](#)
- [KryoFlux Products & Services Limited](#)
- [C64 Preservation Project \(Commodore\)](#)
- [Atari Disk Image FAQ](#)
- [Tim Mann's TRS-80 Pages](#)
- [The .ADF \(Amiga Disk File\) format FAQ](#)
- [Introduction to Magnetic Recording](#)
- [Funny presentation about perpendicular magnetic recording !!!](#)
- [Individual Computer Support](#)
- [The Central Point Option Board](#)
- [SpinRite's Defect Detection Magnetodynamics](#)
- [The Gentle art of Protection](#)
- [The XCOMP/2 home page](#)
- [LIBDSK library for accessing discs and disc image files](#)
- [WinUAE Amiga Emulator](#)

6.5 FDC & Related Information

- [Atari ST – FD/HD Programming – Jean Louis-Guérin](#)
- [WD1772 Floppy Disk Formatter/Controller - WD Corporation – JLG edition](#)
- [Western Digital Corporation 5.25" WD1770/1772 Floppy Disk Controller/Formatter](#)
- [8272 SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER](#)
- [Intel 82077AA FDC Datasheet](#)
- [Commodore C1581 - WD1770 FLOPPY DISK CONTROLLER](#)
- [PC87310 \(SuperI/OTM\) Dual UART with Floppy Disk Controller and Parallel Port](#)
- [Hard Disk Data Encoding / Decoding.](#)
- [Cyclic Redundancy Check](#), [CRC16-CCITT](#), [The Great CRC Mystery Terry Ritter](#)

6.6 Game References

- Arkanoid: [ST Protection M.Fritze](#)
- Au nom de l'hermine: [List of difficult to copy disks](#)
- Barbarian [Pasti/STX File Format](#) [Ways how SW testing copy protection](#) [ST Protection M.Fritze](#)
- Colorado: [List of difficult to copy disks](#) [ST Protection M.Fritze](#)
- Eco: [List of difficult to copy disks \(more more\)](#)
- Falcon: [Analysis of submitted games](#)
- Ghost Buster: [List of difficult to copy disks](#)

Atari Floppy Disk Copy Protection

- Golden Axe: [List of difficult to copy disks](#)
- Indiana jones and the last crusade: [ST Protection M.Fritze](#)
- Jupiter Masterdrive: [List of difficult to copy disks](#) (more)
- Maupiti Island: [List of difficult to copy disks](#) [ST Protection M.Fritze](#)
- Midi Maze: [ST Protection M.Fritze](#)
- Power Drift: [List of difficult to copy disks](#) [Power Drift and Pasti & patch](#) (more) [Way how SW testing copy protection](#)
- Spy Vs Spy: [ST Protection M.Fritze](#)
- Start Glider 2: [List of difficult to copy disks](#)
- Time of lore: [SCP Disk images](#) [Way how SW testing copy protection](#)
- Turrigan: [ST Protection M.Fritze](#)
- Vroom: [Pasti images that should but don't work](#)
- Wizball: [List of difficult to copy disks](#)
- Z-out: [List of difficult to copy disks](#) (more)

Chapter 7. Document history

- V1.4 June 24, 2015
Beyond many small fixes in text the following sections have been added: [Sector with No Data \(SND\)](#) protection, [WD1772 MFM track language](#), [WD1772 Synchronization \(sync marks detection\)](#), [False sync mark detection](#), [WD1772 Bug in Read/Write Track commands](#), [WD1772 CRC Information](#), [Special case of No Flux Area over index](#), [Hidden data](#). The Sync Mark in Data (SMD) protection has been removed. Analysis of [Dragon flight](#) game has been added.
- V1.3a – November 14, 2014
Fixed hyperlink not working
- V1.3 - November 12, 2014.
The categories of protection is now reduced to two (removed fuzzy and physical – not used – categories). The fuzzy bit detailed description is now moved as a section of the [Useful information](#) chapter and fuzzy sector/track added to [data category](#). [NFA](#) moved to timing category. Names for protections changed to more intuitive names. Added section of [write splices](#) and [sync marks](#). Added [Game reference section](#). Added **many** new games analysis. Added Chapter on [preserving floppy disks](#).
- V1.2 June 2014 –
Lots of information added, regrouping of related protections, new examples, etc. Most significant is [Unformatted Tracks](#), [No Sector Data Track](#), [Partially unformatted track](#), [Fuzzy Data Track](#), [No Flux Area on Disk](#), [Unformatted Diskette / Track / Sector](#) ...
- V1.0 - November 2011.
Added information on low level format, particularly about the write splice. Added description about **KFPanzer** and **KFAnalyze**. Now the analysis of games uses the output from **KFAnalyze** and especially the nice plots. Added the [Short/Long Track](#) and [No Flux reversal Area](#) protections. Remove documentation of **Analyze** program. Added more analysis of games ([Turrican](#) and others). New information about games protection based on new **KFPanzer** capabilities. Added more links to new sites. Added reference to the new **KryoFlux** board and related - After 5 years of development I consider the document mature enough to go to version 1.0
- V0.9 – September 2010.
Clean-up text based on feedback. Modified documentation to reflect the usage of the new **Panzer** (Protection ANalyZER) program. Added [ID Fuzzy Bits](#), [Invalid Data in Gap](#), and [Non Standard DAM](#) Protection. Added a *section on Preservation* for each of the protections. Added description for [Barbarian](#), [Operation Neptune](#) Game. Work with Gothmog (Christophe Fontanel) on getting more accurate information on Dungeon Master fuzzy bits protection
- V0.8 – October 2007.
Added taxonomy for the different protection categories. Rewrote of large portion of the explanations about [fuzzy bits](#). Added 5 new protections: [Invalid ID Field](#), [Non Standard IDAM](#), [Sector over Index pulse](#), [Missing Track](#) and [Sector within Sector](#). Added description for several games ([Theme Park Mystery](#), [Computer Hits Volume 2](#), [Kick Off 2](#), [Colorado](#)). Better documented [Intra-sector Bit Variation](#) with reference to [Colorado](#). For the first time lots of diskettes (over 50) have been tested and references for them have been entered in the document. And again lots of clean-up
- V0.7 – January 2007.
Several modifications based on feedback from [Ijor](#) and [Obo](#). Added a new section on [weak bits](#) based on US patent and a section on [Invalid character during format](#). Plus lots of miscellaneous cleanup.
- V0.6 – December 2006. Modifications based on feedback from [Ijor](#), I have added one section about [Double Density diskette format](#), the [Invalid sector number](#) protection, and the [intra-sector variable bit](#) rate protection – December 2006
- V0.5 – December 2006.
Incorporated feedback from [Gothmog](#) about the DM protection patent, added a section

Atari Floppy Disk Copy Protection

with several US patent about protection, modified the section on fuzzy bits, modified the [fuzzy bit detection in WD1772 DPLL](#)

- V0.4 - November 2006.
Complete documentation cleanup and links verification.
- V0.3 - October 2006.
Major Revision: Merged several related sector protections, modified extensively the description of several protections, added section on [example of protections](#), added [analyze program](#) short presentation, added [DPLL presentation](#), and added new protections: [PAT](#) and [NAT](#).
- V0.2 - June 2006.
Minor correction based on feedback.
- V0.1 - May 2006.
Initial writing.