# KFAnalyze
# User’s manual

By Jean Louis-Guérin (DrCoolZic)
Revision 1.3 - August, 2012

# Table of Content

# Presentation

This document describes the **KFAnalyze** program that fully analyzes and dumps the content of Atari Floppy Diskettes that have been imaged with a **KryoFlux** system from KryoFlux Products & Services Ltd and Software Preservation Society. To create **Stream** images of Atari FD you need to use a **KryoFlux** board and the **DTC** application (for more information refer to KryoFlux Documentation)

# Analysis Process overview

You first need to create an image of an Atari floppy diskette with the KryoFlux board before you can analyze this information with the KFAnalyze program. The process is the following

1. Run the **DTC** program to produce **Stream** files (one for each side of each track). For more information on how to create Stream files and on how to control the **DTC** program please refer to the KryoFlux documentation. In most cases to image an Atari FD you can use the following command:
   ***dtc -f<filename> -i0 -i2 -i4***
2. Each of the Stream Files produced can then be analyzed by running the **KFAnalyze** program. The program processes one stream image file at a time and produces an output file plus eventually several "plot files" to be processed by the **Ploticus** program. There are several flags that can be specified during invocation of the program to control the analysis.

*Note that if your focus is to analyze the copy protections of a game/program (composed of a set of FD) you should create the stream files with the **AtariDump.bat** batch file and use the **KFPanzer** program in complement with **KFAnalyze**. Please refer to the **KFPanzer** documentation for more information.*

# Running the KFAnalyze Program

The program is executed by typing:

***KFAnalyze [command_line_flags] input_file [output_file]***

Where:

- **command_line_flags** are one or several optional arguments that control several functionalities of the **KFAnalyze** program. The command line flags are described in the Command line flags and must be placed before the input and output files specification.
- **input_file** is the name of the Stream image file that you want to analyze. This parameter is **required**.
- **Output_file** is an optional name for an output file. If no output file is specified then the outputs of the analysis is sent to the standard output (usually the screen).

The program processes the input **Stream** files and executes three main phases:

- The **read track** phase: In this phase all the flux transitions of the Stream file are decoded as a WD1772 FD controller would do with a ***read track*** *command*. At the end of the analysis both the "layout" of the track as well as the content of the track buffer is outputted.
- The **read address** phase: In this phase the program decodes from the stream file all the ID blocks of the track. At the end of the analysis the "layout" of the ID as well as the content of the ID buffers are outputted for all the IDs found.
- The **read sector** phase: In this phase the program decodes from the stream file all the sectors found in the preceding phase as a WD1772 FDC would do with multiple ***read sector*** commands. Each sector is in fact decoded as many times as the number of "revolutions" analyzed by the KryoFlux DTC program. This is necessary to check if the sector contains ***fuzzy bytes***. At the end of the analysis the "layout" of the Sectors as well as the Sector buffers are outputted.

During these phases the program can also produce several plot data files to be processed by the **Ploticus** program. At the end of execution the program also checks for many copy protection mechanisms. For more information on the protections see my document Atari FD Copy Protection.

*Note that internally the program implements a PLL Data separator pipelined to data shift register and a sync mark detector as described in section KFAnalyze Inner Workings of this document. Therefore the program should produce results extremely close to the usage of an actual WD1772 FDC on a real Atari computer.*

# Command line flags

The ***KFAnalyze*** program can be invoked with zero or several flags. The flag arguments must be specified **before** the ***input_file*** and ***output_file*** names. Each argument consists of one letter preceded by a dash sign ("**-**"). When several options are used they must be separated by space(s) and each of them needs to be preceded with a dash sign. The flags can be specified in any order.

The following example:

***KFAnalyze –v –t populous00.0.raw populous.txt***

Indicates that the **KFAnalyze** program will reads the *populous00.0.raw* Stream image file and produces a *populous.txt* output file using the **-v** and **-t** options.

## -t flag

The **–t** flag is used to ask the **KFAnalyze** program to produce plot files for the **Ploticus** program. See Usage of Ploticus

## -s flag

The **–s** flag is used to disable synch mark detection in ID or DATA field during the ***read track*** phase. By default the synch mark detection is active all the time during the read track phase as the WD1772 would do during a ***read track*** commands. This sometimes leads to false sync detection for some combination of input data resulting in incorrect bits shifting of bytes in the ID or DATA fields. By using the **-s** flag the program will behave like the WD1772 would do when reading the DATA or ID block during a ***read sector*** or ***read address*** commands (i.e. the sync mark detector is turned off in ID and DATA blocks).

## -v flag

By default only most relevant information is placed in the output file. By using this flag (**verbose** mode) much more detailed information is produced (e.g. timing violation location, synch mark detection, half-cell resynchronization, etc.). This is mainly used for debugging or if you want to get many details.

## -b flag

By default the content of the buffers filled by the ***read track***, ***read address***, and ***read sector***, phases are printed to the output. When the **-b** flag is used only the layout information is printed but **not** the content of the buffers.

## -c flag

By default the printout of the buffers only include the "data value" for each byte decoded. The **-c** flag forces the program to print both the "**data** value" and the "**clock** value" for each byte decoded.

## -m flag

By default the printout of the buffers only include the "data value" for each byte decoded. The **-m** flag forces the program to print both the "**data** value" and the "**mfm** value" for each byte decoded.

## -f flag

By default during the *reading of sectors* only the content of the DATA block of the sector is displayed. Using the **-f** flag (full mode) allows to display the content of the important blocks of a sector (ID-GAP3-DATA-GAP4 blocks) during the read sector phase. This can be useful, for example, if you want to look at "hidden data" in the GAP field of a sector.

*Note that KFAnalyze places in the data sector buffer 3 extra bytes compared to what is placed in a buffer by the WD1772. The first extra byte in the buffer is the Data Address Mark (DAM) detected, followed by the actual data (usually 512 bytes), followed by the two extra CRC bytes.*

## -l flag

The -l flag is used to output the buffers in "simple mode". In this mode the location is displayed in decimal, and neither the clock value nor the byte location is displayed. This output is similar to the **Panzer** output and therefore can be used to compare files read on a real Atari**.**

## -r<rev> flag

By default the KFAnalyze program decodes the flux reversal from *the first revolution* during the read track, read address and read sector phases. The **-r flag** allows to specify the revolution to decode. This number must be in the range 1 to the number of revolution imaged minus 1 (i.e. 1..n-1). When this number is specified (e.g. **-r2**) all the outputs (layout, buffers, plots, protections) are decoded from the specified revolution.

*Note that it is not possible to analyze the last revolution. The reasons are explained in KFAnalyze Inner Workings.*

# Contents of the output File

This section describes the contents of the output file produced by the **KFAnalyze** program. One of the purposes of the program is to display in ASCII the content of a Stream binary file as well as the "layout" of a track, id, or sectors.

The content of the output file is detailed for each of the three phases of the program.

The program first display a welcome message followed by the name of the input stream file, followed by the number of revolutions used to image the FD, followed by the average rotation speed.

```
DrCoolZic - KFAnalyze V1.3a (Feb 28 2012-14:28:20) - run Tue Feb 28 17:43:13 2012
  Stream file 'ThemeParkMystery(1-1)00.0.raw' 5 revolutions - Average RPM=300.214
```

The normal speed for an Atari FD drive is 300 RPM.

*Note that the program uses the actual RPM value of a specific rotation to interpolate the flux transition values decoded in the Stream file. With the above example the transition values would be multiplied by a correction factor equal to 300.214 / 300.0.*

# Read Track Phase

## Track Layout

At the end of the read track phase the program display the "**layout**" of the track.

```
************************************************************************
Track Layout Information: 6284 Bytes - length=199.975 ms - num sectors 12
************************************************************************

Lead GAP 34 bytes length=1071.15 us
------------------+-----------+--------------------------+------------
ID                |INTRA GAP  |DATA                      |INTER GAP
Sct Pos    Lgt CRC|Bt  Lgt  BS|Bt   Lgt   CRC TMV BRD Clk |Bt  Lgt   BS
------------------+-----------+--------------------------+------------
1   1071   221 OK |37  1181  0|515  16404 ?   0   0   3.98|36  1132   0
2   20010  205 ?  |38  1195  0|515  16395 OK  0   0   3.98|35  1117   0
3   38924  220 OK |37  1178  0|515  16394 OK  0   0   3.98|35  1117   0
4   57835  221 OK |37  1183  0|515  16418 OK  0   0   3.99|35  1114   0
5   76773  220 OK |37  1181  0|515  16390 OK  0   0   3.98|35  1109   0
6   95676  219 OK |37  1175  0|515  16388 OK  0   0   3.98|35  1113   0
7   114572 220 OK |37  1177  0|515  16383 OK  0   0   3.98|35  1114   0
8   133468 220 OK |37  1174  0|515  16363 ?   0   0   3.97|36  1130   0
9   152357 221 OK |37  1178  0|515  16365 ?   0   0   3.97|37  1165   0
10  171287 222 OK |37  1185  0|515  16429 ?   0   0   3.99|35  1120   0
11  190245 222 OK |31  994   0|12   384   BAD 0   0   4.00|0   0      0
12  191847 222 OK |33  1054  0|215  6836  OK  3   0   3.97|0   0      0
------------------+-----------+--------------------------+------------
```

The first line specifies the total number of bytes of the track, and its total time length. It also indicates the number of sector found for this track.

```
Track Layout Information: 6284 Bytes - length=199.975 ms - num sectors 12
```

Number of bytes and length in micro-seconds of the leading GAP (post index) block.

```
Lead GAP 34 bytes length=1071.15 us
```

Following is a table of values with one line for each sector detected (note that since version 1.3 the *sectors within sectors* are detected in the track phase).

For each sector the following information are displayed:

- ID block: **Sct**=Sector number, **Pos**=position in the track in µs, **Lgt**=length in µs, **CRC**=OK, ?, or BAD. Note that in normal *read track mode* false sync detection can lead to false CRC indication (displayed as "?"); if you want to have the exact values for the CRC you must use the **-s** flag.
- INTRA GAP (inter-ID-DATA) block: **Bt**= number of bytes, **Lgt**=length in µs, **BS**= Bad Sync (number of bad sync bytes in the GAP)
- DATA block: **Bt**= number of bytes, **Lgt**=length in µs, **CRC**=OK, ?, or BAD, **TMV**=number of timing violation[1], **BRD**=border bits[2], **CLK**=clock average period in µs for the sector. Note that in normal *read track mode* false sync detection can lead to false CRC indication (displayed as "?"); if you want to have the exact values for the CRC you must use the **-s** flag.
- INTER GAP (post-DATA) block: **Bt**= number of bytes, **Lgt**=length in µs, **BS**= Bad Sync (number of bad sync bytes in the GAP)

Note: A Bad Sync (BS) in GAP is the one of the following:

- Sync sequence with a change in sync char (e.g. changing from C2 to A1)
- Sync sequence with more than 3 characters
- Sync sequence with less than 3 characters
- A Sync sequence is followed by a char other than an AM

## Track Buffer

The layout information is followed by the dump of the track buffer:

```
Detail buffer content 5995 (0x176b) bytes
+ LEAD GAP 31 bytes @0.000 ms length=1070.36 us - TMV=0 BRD=0 BS=0
  0000 33     4196  24 24 24 24 24 24 24 24 24 24 24 24 24 24 24  $$$$$$$$$$$$$$$
  0010 569    4196  24 24 24 24 24 24 24 24 24 24 24 24 24 24 3f  $$$$$$$$$$$$$$?

= ID=1 7 bytes @1071 us length=221.25 us I=FE T=0 H=0 S=1 Z=512 CRC=ca6f OK TMV=0 BRD=0 BS=0
  0022 1071   1966  fe 00 00 01 02 ca 6f                          ......o
+ INTRA-GAP 37 bytes @1292 us length=1181.18 us - TMV=0 BRD=0 BS=0 IDG=0
  0029 1292   1998  4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e 4e  NNNNNNNNNNNNNNN
  0039 1802   1982  4e 4e 4e 4e 4e 4e 00 00 00 00 00 00 00 00 00  NNNNNN.........
  0049 2312   1998  00 00 a1 a1 a1                                .....
= DATA ID=1 515 bytes @2473 us length=16404.35 us CRC=0020  TMV=0 BRD=0 BS=8
  004e 2473   1998  fb 60 1e 00 00 00 00 00 58 00 00 00 02 02 01  .`.......X......
  005e 2981   1982  00 02 70 00 20 03 00 05 00 0a 00 01 00 00 00  ..p. ...........
  006e 3492   1998  00 20 3c 00 00 00 07 22 3c 00 00 00 03 20 79 00  . <...."<.... y.
  007e 4003   1998  00 04 32 61 00 00 18 66 e8 e1 89 d2 81 d1 c1 72  ..2a...f.......r
  008e 4515   1998  01 61 00 00 0a 2f 39 00 00 04 32 4e 75 48 e7 7c  .a.../9...2NuH.|
  009e 5030   1998  e0 26 00 28 01 70 ff 20 03 80 fc 00 c2 4e 7e 13  .&.(.p. .....N~.
  00ae 5538   1998  ff 13 fe c2 4f c2 73 ff 87 ff ff ff e0 08 1f f0  ....O.s.........
...
```

[1] For DD MFM the cell width is 4 µs and the spacing between fluxes is 1, 1½, or 2 cell width (i.e. 4, 6, or 8 µs). A timing violation is detected if the value is less than 1 or more than 2 (less than 4 µs or more than 8 µs).
[2] Border bits are bits located on the border of the inspection window.

As you can see the dump is organized by GAP/ID/DATA/…

The content of most fields in the headers are self-explanatory. The following acronyms are used:

- TMV = number of Timing Violation
- BRD = number of Border Bits
- BS = number of Bad Sync: in GAP3/GAP4 fields this is a bad sync sequence as explained before, in ID/DATA fields this indicates sync bytes found in the field (will cause incorrect results for a read track command on the sector)
- I= Index Address Mark (IAM) value ($FE or $FF)
- T=track number in ID
- H=head number in ID,
- S=sector number in ID,
- Z=size (decoded) in ID
- IDG = number of Invalid Data into Gap

Each line of the data/clock dump contains:

- The location in hexadecimal of the byte in the buffer,
- The position on the track in µs for the first byte of the line
- The value of the PLL period in ns for the first byte of the line
- The data value in hexadecimal for the next 16 bytes
- The data value in ASCII for the next 16 bytes

If the **–c** flag is used this information is followed by:

- The clock values in hexadecimal for the same 16 bytes
- The clock value in ASCII for the same 16 bytes

For example:

```
= DATA ID=3 1027 bytes @17954 us length=32677.35 us - CRC=e7ff *** BAD *** - TMV=0 BRD=0
BSync=16
  0233 17954  4000  fb 2c 10 ff 09 0a 15 10 00 f5 f1 fb 02 f4 f5 f0  .,..............
                    00 41 e7 00 72 70 e0 67 ff 00 06 00 7c 01 00 07  .A..rp.g....|...
  0243 18465  3968  ef ff 2f 34 10 e8 e0 ff 2b 3c 20 00 f0 db eb 07  ../4....+< .....
                    00 00 40 41 e7 03 0f 00 40 41 cf ff 07 00 00 78  ..@A....@A.....x
  0253 18975  4000  27 3a 20 00 e8 e7 f7 ff 17 27 29 37 24 00 d7 e7  ': ......')7$...
                    48 40 cf ff 03 08 00 00 60 48 42 40 49 ff 00 08  H@......`HB@I...
  0263 19485  3968  13 36 34 20 f0 d7 e0 ef 13 2f 32 20 13 09 ff 0b  .64 ...../2 ....
```

It can be useful to see the clock values in cases the transitions are shifted by a half clock in overlapping sector as used in the Turrican game.

If the -**m** flag is used this information is followed by the mfm values (the values directly entered in the shift register) in hexadecimal for the same 16 bytes

For example

```
= DATA ID=1 515 bytes @2473 us length=16404.35 us CRC=0020  TMV=0 BRD=0 BS=8
  004e 2473   1998  fb 60 1e 00 00 00 00 00 58 00 00 00 02 02 01  .`.......X......
       5545 14aa a954 aaaa aaaa aaaa aaaa aaaa aaaa 914a aaaa aaaa aaaa aaa4 aaa4 aaa9
  005e 2981   1982  00 02 70 00 20 03 00 05 00 0a 00 01 00 00 00 00  ..p. ...........
       2aaa aaa4 952a aaaa a4aa aaa5 2aaa aa91 2aaa aa44 aaaa aaa9 2aaa aaaa aaaa aaaa
  006e 3492   1998  00 20 3c 00 00 00 07 22 3c 00 00 00 03 20 79 00  . <...."<.... y.
       aaaa a4aa a552 aaaa aaaa aaaa aa95 24a4 a552 aaaa aaaa aaaa aaa5 24aa 9549 2aaa
  007e 4003   1998  00 04 32 61 00 00 18 66 e8 e1 89 d2 81 d1 c1 72  ..2a...f.......r
       aaaa aa92 a524 94a9 2aaa aaaa a94a 9494 544a 54a9 4a49 5124 4aa9 5129 52a9 1524
```

The following flags have impact on the read track phase:

- -s : disable the sync detection in ID/DATA blocks
- -v : verbose mode
- -b : turn off output of buffer content
- -c : print clock bytes information
- -m : print mfm information in buffer
- -t : output files for Ploticus
- -l: output simple information in buffer
- -r<rev>: specifies the revolution to use

For more detail please refer to Command line flags

Remember that in the read track command/phase the following happen:

- The data may not be read correctly inside ID or DATA block. This is due to the fact that the sync mark detector is active at all time and therefore incorrect bit shift may happen (as a WD1772 would do).
- A well know example is that all the sector numbers are decoded incorrectly for track 41 (track 41 always causes a false sync detection)!
- The above implies that the CRC may be reported incorrectly in this phase.

However the read track phase is the only way to look at data outside of the ID/DATA blocks.

# Read Address Phase

## Address Layout

At the end of the address phase the program display the layout for all the IDs.

```
*************************************************************************
ID Layout Information: number ID 12
*************************************************************************
-----------------------------------
ID
Sct Pos     Lgt  CRC TMV BRD CLK
-----------------------------------
1   1071    221  OK  0   0   3.95
2   20010   220  OK  0   0   3.94
3   38924   220  OK  0   0   3.95
4   57835   221  OK  0   0   3.96
5   76773   220  OK  0   0   3.94
6   95676   219  OK  0   0   3.93
7   114572  220  OK  0   0   3.93
8   133468  220  OK  0   0   3.93
9   152357  221  OK  0   0   3.95
10  171287  222  OK  0   0   3.97
11  190245  222  OK  0   0   3.97
12  191847  222  OK  0   0   3.97
-----------------------------------
```

Most of the information is self-explanatory. The first line display the number of IDs found in the track:

```
ID Layout Information: # ID 12
```

For each ID found on the track the following information is provided:

- **Sct**=Sector number,
- **Pos**=position in the track in µs,
- **Lgt**=length in µs, **CRC**=OK or BAD,
- **TMV**=number of timing violation,
- **BRD**=border bits,
- **CLK**=clock average value for the sector given as the period in µs

## Address content

The layout information is followed by the complete content of each ID block:

```
= ID=1  7 bytes @1071   us length=221.25 I=FE T=0 H=0 Z=2 CRC=ca6f OK TMV=0 BRD=0
= ID=2  7 bytes @20010  us length=220.92 I=FE T=0 H=0 Z=2 CRC=9f3c OK TMV=0 BRD=0
= ID=3  7 bytes @38924  us length=221.00 I=FE T=0 H=0 Z=2 CRC=ac0d OK TMV=0 BRD=0
= ID=4  7 bytes @57835  us length=221.50 I=FE T=0 H=0 Z=2 CRC=359a OK TMV=0 BRD=0
= ID=5  7 bytes @76773  us length=220.88 I=FE T=0 H=0 Z=2 CRC=06ab OK TMV=0 BRD=0
= ID=6  7 bytes @95676  us length=219.88 I=FE T=0 H=0 Z=2 CRC=53f8 OK TMV=0 BRD=0
= ID=7  7 bytes @114572 us length=220.34 I=FE T=0 H=0 Z=2 CRC=60c9 OK TMV=0 BRD=0
= ID=8  7 bytes @133468 us length=220.17 I=FE T=0 H=0 Z=2 CRC=70f7 OK TMV=0 BRD=0
= ID=9  7 bytes @152357 us length=221.04 I=FE T=0 H=0 Z=2 CRC=43c6 OK TMV=0 BRD=0
= ID=10 7 bytes @171287 us length=222.17 I=FE T=0 H=0 Z=2 CRC=1695 OK TMV=0 BRD=0
= ID=11 7 bytes @190245 us length=222.08 I=FE T=0 H=0 Z=2 CRC=25a4 OK TMV=0 BRD=0
= ID=12 7 bytes @191847 us length=222.12 I=FE T=0 H=0 Z=2 CRC=bc33 OK TMV=0 BRD=0
```

The content of the fields are self-explanatory. Each line of the dump contains:

- The ID value of the sector
- The number of bytes
- The position in µs of the first byte of the sector header
- The length of the header
- The value of the IAM (Index address mark)
- The track number
- The head number
- The encoded size value
- The CRC value
- An indicator if the CRC is valid or not
- The number of timing violations
- The number of border bits.

The following flags have impact on the read address phase:

- -v : verbose mode
- -r<rev>: specifies the revolution to use

For more detail please refer to Command line flags

# Read Sectors Phase

In this phase the program decodes the flux transition to find the corresponding DATA sectors for each of the ID found in the read addresses phase. Furthermore each sector is read **several** times (number of revolutions -1) to potentially detect the presence of fuzzy (aka weak) bits.

## Sector Layout

At the end of the sectors phase the program display the layout for all sectors. Note that in this mode sectors within sector are detected and displayed correctly.

```
All Sectors Layout Information:
-----------------+--------+-----------------------------------------
ID               |GAP     |DATA
Sct Pos    Lgt CRC|Bt Lgt  |Bt   Pos    Lgt   CRC TMV BRD Clk  DOI  FZP
-----------------+--------+-----------------------------------------
1   1071   221 OK |37 1181 |515  2473   16420 OK  0   0   3.99 0    0
2   20010  220 OK |37 1179 |515  21410  16395 OK  0   0   3.98 0    0
3   38924  220 OK |37 1178 |515  40323  16394 OK  0   0   3.98 0    0
4   57835  221 OK |37 1183 |515  59240  16418 OK  0   0   3.99 0    0
5   76773  220 OK |37 1181 |515  78175  16390 OK  0   0   3.98 0    0
6   95676  219 OK |37 1175 |515  97071  16388 OK  0   0   3.98 0    0
7   114572 220 OK |37 1177 |515  115970 16383 OK  0   0   3.98 0    0
8   133468 220 OK |37 1174 |515  134863 16375 OK  0   0   3.97 0    0
9   152357 221 OK |37 1178 |515  153756 16409 OK  0   0   3.98 0    0
10  171287 222 OK |37 1185 |515  172695 16429 OK  0   0   3.99 0    0
11  190245 222 OK |31 994  |515  191462 16411 BAD 3   0   3.98 249  266
12  191847 222 OK |33 1054 |515  193123 16411 BAD 3   0   3.98 301  214
-----------------+--------+-----------------------------------------
```

Most of the information is self-explanatory:

The following information is provided for each sector:

- ID block: **Sct**=Sector number, **Pos**=position in the track in µs, **Lgt**=length in µs, **CRC**=OK or BAD,
- GAP (inter ID/DATA) block: **Bt**= number of bytes, **Lgt**=length in µs
- DATA block: **Bt**= number of bytes, **Pos**=position in the track in µs, **Lgt**=length in µs, **CRC**=OK or BAD, **TMV**=number of timing violation, **BRD**=border bits, **CLK**=average value of the clock for the sector (given as the period in µs), **DOI**=Data over Index contains the number of bytes for this sector which are beyond the index, **FZP**= First Fuzzy byte position – If no fuzzy bits found this value = 0, if fuzzy bits found it indicates the location in the buffer of the first byte that differs from one reading to the other.

## Sector Buffer

The layout information is followed by the dump of the DATA buffer:

```
Detail buffer content for sector 1 with 515 bytes
= DATA ID=1 515 bytes @95748 us length=16402.38 CRC OK CLK=3.98 TMV=0 BRD=0 DOI=0
  0000 95748  3968  fb 10 05 ff 06 12 17 11 06 fe ff 10 17 16 12 08  ................
  0010 96257  3968  07 0a 11 14 10 10 10 09 08 09 09 08 0d 10 0b 06  ................
  0020 96768  3968  06 08 0b 13 17 14 10 08 05 0b 11 12 10 08 07 0a  ................
  0030 97278  4000  10 12 10 08 03 05 09 0d 13 14 10 04 01 02 09 14  ................
  0040 97787  4000  17 14 10 0a 0b 10 13 10 06 00 02 06 09 0b 08 09  ................
  0050 98298  4000  0b 10 10 10 12 14 13 10 10 0b 09 0c 0b 0a 08 0a  ................
  0060 98811  4000  08 05 07 09 0a 10 10 0b 09 0c 11 17 17 10 04 ff  ................
  0070 99322  3968  04 11 20 20 10 02 fe ff 0d 11 10 05 02 0b 17 20  ..  ........... 
  0080 99832  3968  12 06 04 07 0c 12 10 07 08 10 10 0b 0a 05 05 07  ................
```

The dump is organized by GAP/ID/DATA…

The content of most fields in the headers are self-explanatory (see also Track Buffer). Each line of the dump contains:

- The location in the buffer hexadecimal,
- The position of the first byte of the line on the track in µs
- The value of the PLL clock for the first byte of the line
- The data value in hexadecimal for the next 16 bytes
- The data value in ASCII for the next 16 bytes

If the **–c** flag is used this information is followed by:

- The clock value in hexadecimal for the next 16 bytes
- The clock value in ASCII for the next 16 bytes

This can be useful to see clock when transitions are shifted by a half clock as in the Turrican game.

If the -**m** flag is used this information is followed by the mfm values in hexadecimal for the same 16 bytes

For example

```
= DATA ID=1 515 bytes @2473 us length=16404.35 us CRC=0020  TMV=0 BRD=0 BS=8
  004e 2473   1998  fb 60 1e 00 00 00 00 00 58 00 00 00 02 02 01  .`.......X......
       5545 14aa a954 aaaa aaaa aaaa aaaa aaaa aaaa 914a aaaa aaaa aaaa aaa4 aaa4 aaa9
  005e 2981   1982  00 02 70 00 20 03 00 05 00 0a 00 01 00 00 00 00  ..p. ...........
       2aaa aaa4 952a aaaa a4aa aaa5 2aaa aa91 2aaa aa44 aaaa aaa9 2aaa aaaa aaaa aaaa
  006e 3492   1998  00 20 3c 00 00 00 07 22 3c 00 00 00 03 20 79 00  . <...."<.... y.
       aaaa a4aa a552 aaaa aaaa aaaa aa95 24a4 a552 aaaa aaaa aaaa aaa5 24aa 9549 2aaa
  007e 4003   1998  00 04 32 61 00 00 18 66 e8 e1 89 d2 81 d1 c1 72  ..2a...f.......r
       aaaa aa92 a524 94a9 2aaa aaaa a94a 9494 544a 54a9 4a49 5124 4aa9 5129 52a9 1524
```

If the **-f** flag is used not only the DATA are displayed but also the GAP and ID information in the following order: ID-GAP3-DATA-GAP4. This can be useful to look for hidden data in GAP3 or GAP4. For example:

```
Detail buffer content for sector 9 with 1290 bytes
= ID=9 7 bytes @158854 length=224.98 T=50 H=0 S=9 Z=512 OK TMV=0 BRD=0 BSync=0
  0000 158854 4063  fe 32 00 09 02 82 47                          .2....G
+ GAP 37 bytes @159079 us length=1198.17 us - TMV=0 BRD=0 BSync=0 IDG=21
  0007 159079 4063  f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7  ................
  0017 159597 4031  f7 f7 f7 f7 f7 f7 00 00 00 00 00 00 00 00 00 00  ................
  0027 160117 4063  00 00 a1 a1 a1                                  .....
= DATA ID=9 515 bytes @160277 us length=16592.10 CRC OK CLK=4.03 TMV=0 BRD=0 DOI=0
  002c 160277 4031  fb 02 00 02 00 1d 7f ff 0f fc 7f fc 8f fb 80 05  ......□...□.....
  003c 160799 4063  00 93 80 00 00 ff 00 01 00 01 00 fe df ff c7 f8  ................
  004c 161312 4031  df f8 27 c7 f8 05 00 84 f8 00 03 ff 04 00 8b 03  ..'.............
  005c 161830 4031  ff f7 ff 32 c0 37 c0 ca 3f fc 05 00 84 fc 00 07  ...2.7..?.......
  006c 162348 4031  ff 04 00 8b 07 ff fb ff 09 00 0b 00 f4 ff ff 05  ................
  007c 162870 4063  00 84 ff 00 1f ff 04 00 8c 1f ff fd ff 05 00 05  ................
  008c 163388 4063  00 fa ff ff 80 04 00 8c ff 80 3f ff 3f ff 3f ff  ..........?.?.?.
```

Here we can see in GAP the presence of invalid data in gap (0xF7). In the GAP header we have a field IDG that indicates the number of Invalid Data into Gap found.

If the sector has weak bytes an indication is given

```
= DATA ID=11 515 bytes @191462 us length=16411.99 CRC BAD CLK=3.98 TMV=3 BRD=0 DOI=249 BS=7
  *** Sector has a Non Flux Area *** NFA 194377-199928
  *** Sector has Fuzzy Bytes *** starting at byte position 266
  0026 191462 1998  fb 00 00 00 00 00 00 00 00 a1 a1 a1 fe 00 00 0c  ................
  0036 191973 1998  02 bc 33 4e 4e 4e 4e 4e 4e 4e 4e d9 23 76 c5 e6  ..3NNNNNNNN.#v..
  0046 192486 1998  d3 31 b2 4e 4e 4e 4e 4e 4e 4e 4e ff ff ff ff ff  .1.NNNNNNNN.....
  0056 192997 1998  fe 14 14 14 00 ff ff ff ff ff ff ff ff ff ff ff  ................
  0066 193508 1998  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
```

In that case the sector buffer is followed by a fuzzy mask buffer. The content of each byte of the mask is an exclusive or between the different reads of a same byte.

```
+ Fuzzy Mask
  0026              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
  0126              00 00 00 00 00 00 00 00 00 a7 33 c3 c3 c3 c3
  0136              c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3
  0146              c3 c3 e4 e0 00 00 00 00 00 50 da ca e5 1f e0
  0156              00 81 75 1b 01 d3 c3 c3 c3 c3 c3 c3 c3 c3 c3
```

The following flags have impact on the read track phase:

- -v: verbose mode
- -b: turn off output of buffer content
- -c: print clock bytes information
- -m: print mfm information
- -t: output files for Ploticus
- -f: store and display ID-GAP3-DATA-GAP4 information in buffers
- -l: print buffer in minimal mode
- -r<rev>: specifies the revolution to use

For more detail please refer to Command line flags

## Protection Analysis

At the end of the analysis the program check the image to find all possible copy protection mechanisms used.

The protections are displayed like this:

```
S<sec_num> <Prot_type>(level) <detail> <(info)>
```

Where:

- sec_num is the sector number. For a track protection this field is absent.
- Prot_type is the type of protection. All the detected protections are listed below
- (level) is the level of the protection
- Detail is the detail name of the protection
- (info) is optional information displayed with certain protections. For example for a Short Track this field contains the number of bytes of the Track:
  `LGT(2) Long Track (6396)`

The protections are classified in three levels:

- Level 1: indicates that the detected mechanism is most probably used for protection.
- Level 2: indicates a high probability that the detected mechanism is used for protection
- Level 3: is used to provide information. This information may be used as a hint for hard to detect protection mechanisms.

The following protection mechanisms (with their level) are detected by the program:

- DBI: Data Beyond Index (level = 1)
  The ID field of the last sector is placed **before** the index and the corresponding DATA field of this sector is placed **after** the index. This is the extreme version of the DOI protection described below. The position of the data sector is displayed between parentheses.
- DOI: Data Over Index (level = 1)
  Indicates that the DATA field of the last sector span over the index. The number of bytes placed after the index is displayed between parentheses.
- DSN: Duplicate Sector Number (level = 1)
  Two or more sectors have the same number.
- FAA: Flux reversal in Ambiguous Area (level = 3)
  The flux reversal are placed at the border of the inspection window. This can be used to produce fuzzy bits detected as a FZD protection. This field is therefore considered as informational. The total number of transitions in ambiguous area is printed between parentheses.
- FZD: Fuzzy bytes in Data field (level = 1)
  A specific sector contains fuzzy bytes. The number of the first bytes that differ from read to read is displayed between parentheses.
- IBV: Intra-sector Bit rate variation. (level = 1)
  The same sector contains several regions with bit-rate **above** normal **and** bit-rate **below** normal (e.g. macrodos). The number of bytes that are 3% above or 3% below the normal value is printed between parentheses.
- IIF: Invalid ID field (level = 2)
  The ID field of the sector contain invalid values for the track number, and/or the side number, and/or the sector size.

- IDG: Invalid data in GAP field (level = 2)
  Some GAP fields (GAP1, GAP3, GAP4) contains data that cannot be written by a WD1772 FDC into GAP (range 245-247). The number of invalid data is displayed between parentheses.
- ISN: Invalid Sector Number (level = 1)
  ID fields contains a sector number that cannot be written by the WD1772 FDC (value in range 245-247).
- ISS: Invalid Synch Sequence (level = 3)
  A normal Synch sequence is 3 synch marks (A1 or C2) followed by an AM (address mark). Any sequence that does not follow this rule is detected and reported as an ISS. Placing a synch mark in a GAP allow to place "hidden data" in the GAP. The number of invalid synch sequence is printed between parentheses.
- ITN: Invalid Track Number (level = 2)
  Normally the track number in the ID field of the sector must be the same as the track analyzed. If the numbers are different this protection is displayed. The track value found in the ID field is displayed between parentheses.
- LGS: Long DATA Sector (level = 1)
  A normal DATA sector has a length of 16480 µs (515 * 32 µs). Long sector indicates a sector with a length 3% above normal. The length of the sector is printed between parentheses.
- LGT: Long Track (level = 1)
  A normal track contains 6250 bytes (200000 / 32µs). A long track is a track that contains about 3% more bytes. The number of bytes of the track is displayed between parentheses.
- MTV: MFM Timing Violation (level = 3)
  Normal MFM flux reversals bands for a double density diskette are 4, 6, and 8 µs. If some flux transitions are above or below these bands a MTV is detected. In some cases this corresponds to partially unformatted DATA field. This is usually used to produce fuzzy bits detected by the FZD protection. This field is therefore considered as informational. The total number of violations detected is printed between parentheses.
- NFA: No Flux reversal Area (level = 1)
  The track contains an area that do not generate any flux reversal for several milliseconds. This is a very difficult to reproduce copy protection. The position of the NFA on the track is displayed between parentheses.
- NOS: Number Of Sector (level = 3)
  The normal number of sectors per track is 9 or 10. If the actual number of sectors is above 10 or below 9 the NOS protection is detected. The number of sector found is also displayed. This is informational only
- NSD: Non Standard DAM (level = 2)
  The Address Mark placed at the beginning of the DATA field is not one of the two standard DAM. The value of the detected DAM is printed between parentheses.
- NSI: Non Standard IDAM (level = 2)
  The Address Mark placed at the beginning of the ID field is not the standard IDAM. The value of the detected IDAM is printed between parentheses.
- SBD: Sector with Bad Data (level = 1)
  The DATA field of the sector is read with a CRC error

- SBI: Sector with Bad ID (level = 1)
  The ID field of the sector is read with a CRC error
- SND: Sector with No Data (level = 1)
  A sector contains a normal ID segment which is not followed by a DATA segment.
- SHS: Short Data Sector (level = 1)
  A normal DATA sector has a length of 16480 µs (515 * 32 µs). Short sector indicates a sector with a length 3% below normal. The length of the sector is printed between parentheses.
- SHT: Short Track (level = 1)
  A normal track contains 6250 bytes (200000 / 32). A Short track is a track that contains about 3% less bytes. The number of bytes of the track is displayed between parentheses.
- SID: Synch mark in ID/DATA field (level = 3)
  The presence of synch mark inside data may indicate some tricks that can be further analyzed. This is especially true in presence of overlapping sectors (SWS) where it is possible for example to read clock flux reversals as data flux reversal by placing an appropriate synch mark. The number of bad synch mark detected is printed between parentheses.
- SSZ: Sector Size (level = 2)
  Normal Atari FD uses 512 or 1024 bytes per sector. Any other value is detected as SSZ mechanisms. The value of the size found in the ID field is displayed between parentheses.
- SWS: Sector Within Sector (level = 1)
  A sector is placed inside another sector (overlapping sectors) usually in the DATA field. This allows to have 12 or more pseudo sectors in a track.

## Write Splices

If timing violations are detected in the sector write splice areas (around 32 bytes after the ID block and few bytes after the DATA block), this may indicates that the floppy has been written on an Atari and is therefore not an original. Write splice are not related to protections but they are reported during the protection analysis as:

```
S<sector> SPL(2) *** WARNING: Possible write splice *** (num)
```

where sector is the sector number where the write splice have been detected, and num is the number of timing violation detected in the write splice areas.

Note that this is an indication only that should be further interpreted by the user.

## Usage of Ploticus

When the **–t** command line option is used the program produces several files that can be used by the **Ploticus** program. The **Ploticus** program is a free GPL, non-interactive software package for producing plots, charts, and graphics from data. You can find information about the program at http://ploticus.sourceforge.net/doc/welcome.html and the executable at http://sourceforge.net/projects/ploticus/files/ploticus/2.41/

The ASCII files produced by KFAnalyze are the following:

- **layout.dat**: This file provides data for displaying the overall layout of the track analyzed.
- **trans.dat**: This file contains the transitions and clock information for a complete track.
- For each sector:
  - **sec_data_x**.dat: This file contains the data, clock, and encoded information for a sector with the following blocks ID-GAP3-DATA-GAP4. The x value corresponds to the sector number. For example for sector 1: sec_data_1.dat
  - **sec_layout.dat**: This file provides data for displaying the overall layout (ID-GAP3-DATA-GAP4) of the sector analyzed. The x value corresponds to the sector number. For example for sector 1: sec_layout_1.dat
- **allsect.dat**: This file contains the layout for all the sectors found on the track. This is especially interesting to see sector that have data over index as well as overlapping sectors (sector within sector)

To plot the output from **KFAnalyze** you must first install the Ploticus program and run the program with the provided scripts files:

- kf_track.pl,
- kf_sector.pl,
- kf_histo.pl,
- kf_data.pl, and
- allsect.pl

The usage is the following:

**pl –xxx script**

Where xxx specifies the graphic output type. It is recommended to use either **eps** (encapsulated postscript) or **gif** (pseudo GIF image). Other format can be used (see Ploticus documentation) but have not been tested. The quality of the EPS graphic output is much better than the GIF one but requires a program than can read this type of graphics. An excellent program to read and manipulate the eps file is **Illustrator** from **Adobe.** However the GIF output provides acceptable results and can be manipulated by many programs.

The following graphics can be produced:

- To plot the overall track information you must type:
  **pl –xxx kf_track.pl** (e.g. **pl –eps kf_track.pl**)
- To plot a specific sector of a track you must type:
  **pl sec=n –xxx kf_sector.pl**
  Where **sec=n** indicates the sector to plot. For example if you want to plot the information for sector 16 you would use: **pl sec=16 –eps kf_sector.pl**
- To plot the data of a specific sector of a track you must type:
  **pl sec=n –xxx kf_data.pl**
  Where **sec=n** indicates the sector to plot. For example if you want to plot the data for sector 16 you would use: **pl sec=16 –eps kf_data.pl**
- To plot the layout of all the sectors of a track you must type:
  **pl -xxx allsect.pl** (e.g. **pl -gif allsect.pl**)
- To plot an histogram of the flux transitions of a track you must type:
  **pl -xxx kf_histo.pl (e.g. pl -gif kf_histo.pl)**

Please feel free to experiment with **Ploticus** or other plot program and let me know if you write interesting scripts.

## Track Plot

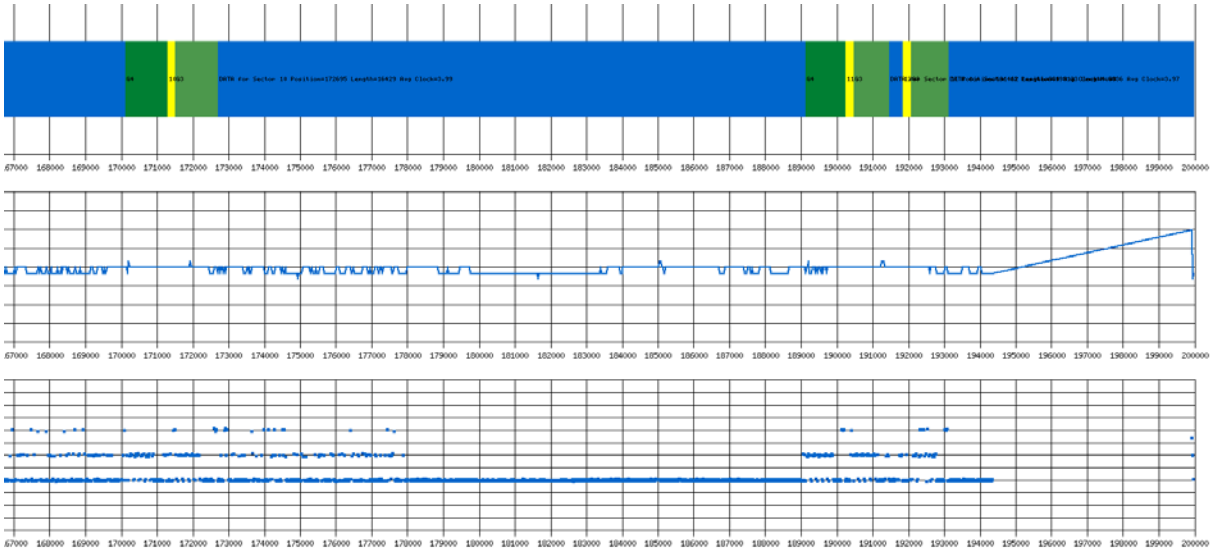
The track plot is a long skinny plot that contains the overall content of the track. It is recommended to read the output with a graphic editor so you can zoom on a specific area.

If we zoom we can see there are three "lines":

- The first line display the "block" of the track. The GAP blocks are displayed using different shades of green, the ID block is displayed in yellow, and the DATA block is displayed in Blue.
- The second line displays the value of the PLL cell clock (expressed as a period in ns). The scale ranges from 1800 ns to 2200 ns. This is useful to look at clock variation in long/short sector or track (like in Rob Northern copy protection) as well as intra sector variation (like in macrodos protection)
- The third line displays a scattered plot of the flux spacing (remember that for a track we only display one flux every five). The y value scale is from 0 to 12000 ns. Normally the dots should be centered around the 4000, 6000, and 8000 ns lines however they can appear anywhere. The "normal" transitions are displayed in **blue** but if a transition is close to the border of the inspection window it is displayed in **green**, and if the transition violates the MFM timing rules (less than 2µs or more than 8µs for Atari MFM) the transition is displayed in **red**. Large transition values are set a max value of 12000ns in order to display them correctly in the plot.

The X axis ranges from 0 to 200000 ms.

The following diagram show a zoom on the end of a plot



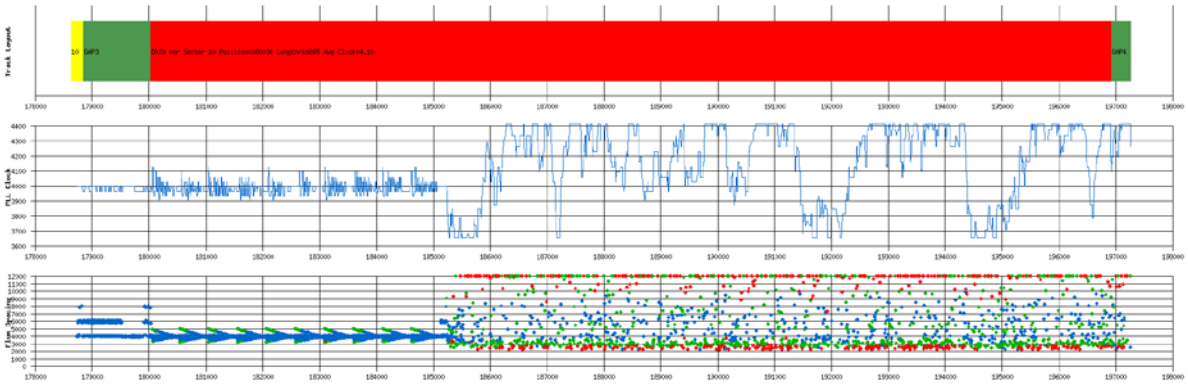
Here we can see sector within sector over a non-flux area

## Sector Plot

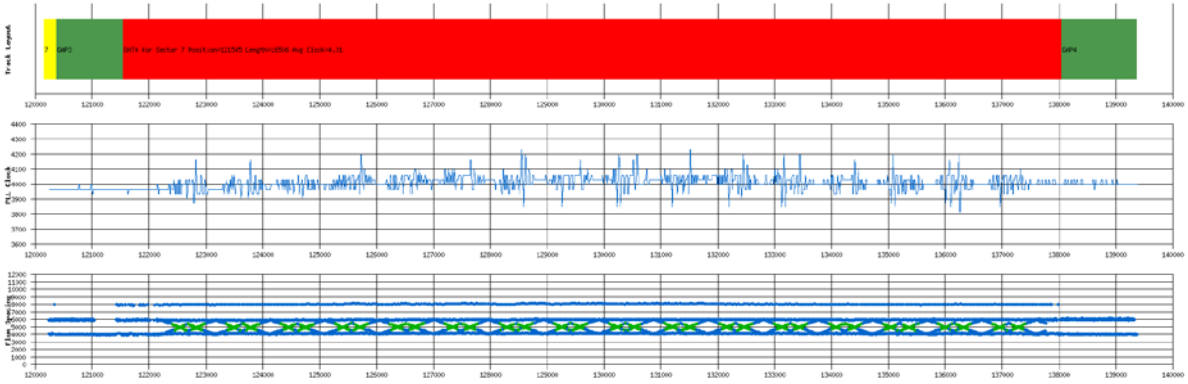
The sector plot displays the overall content of a specified sector including: ID-GAP3-DATA-GAP4 information. As only one sector is displayed this plot contains more details than the overall track plot. The information displayed is similar to the track information but all transitions are displayed (instead of one every five):

- The ID blocks are displayed in yellow,
- the GAP3 blocks are displayed in green,
- the DATA blocks are displayed using the following colors:
  - Blue if the block is read correctly (no CRC error and no Fuzzy bits)
  - Orange if the block contains some fuzzy bytes.
  - Red if the block is read with CRC errors but does not contain any fuzzy bytes.
- The GAP4 block is displayed in green.

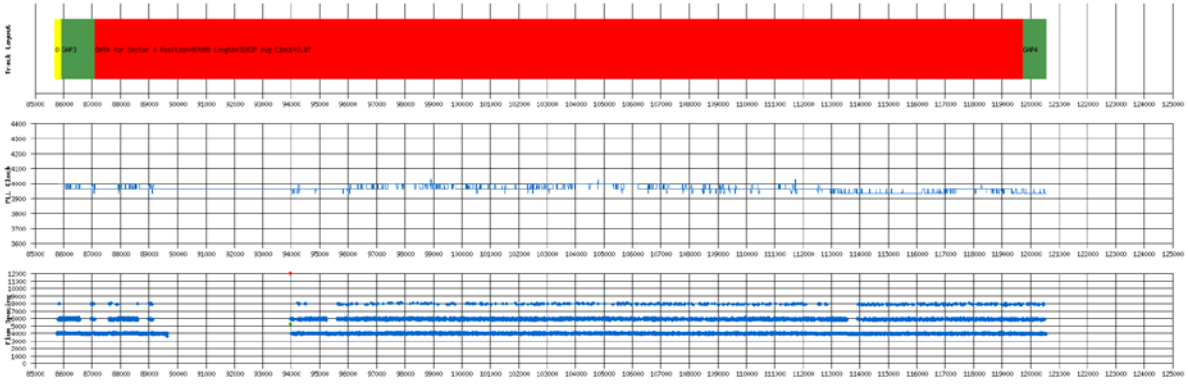
Here are some interesting examples:



Unformatted end of sector



Sliding flux close to inspection window border



Long area without transition

## Sector Data Plot

The sector data plot displays the overall content of a specified sector including: ID-GAP3-DATA-GAP4 information. This plot is very similar with the Sector Plot but it also display information about the Data, Clock, and encoded Data/Clock bytes. The bytes of the sector are displayed along flux transition, PLL clock and the sector layout therefore the plot produced is an extremely long and narrow plot. Thus it is **always** necessary to use a graphic editor to zoom in an area of interest.

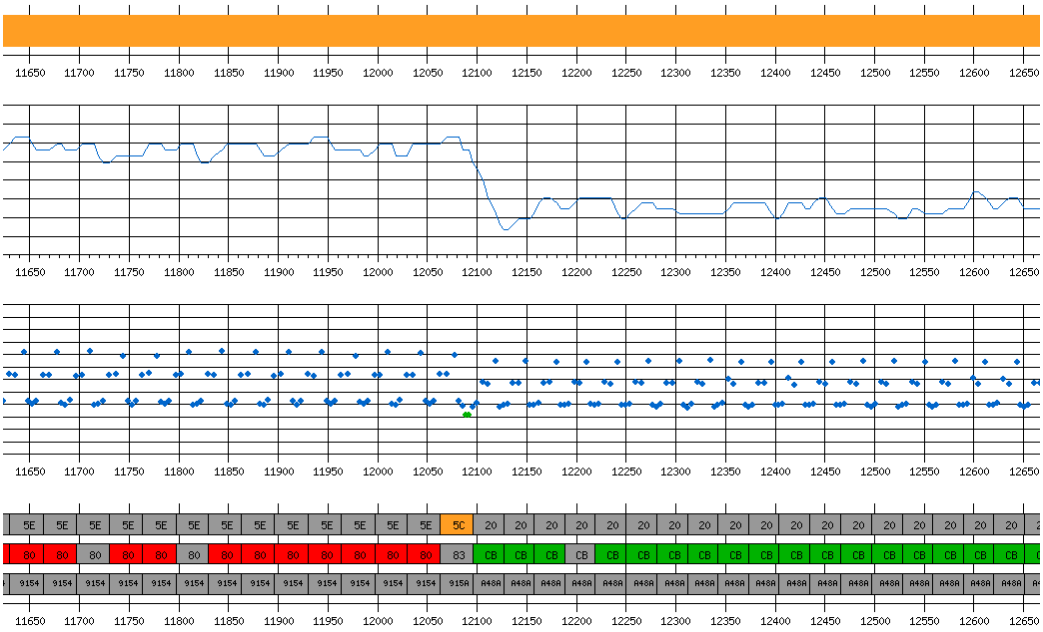
For explanation about the Sector Layout, PLL Clock and Flux Spacing please refer to Sector Plot as the information displayed is the same.

The bottom line of the plot display the **MFM Encoded Clock & Data** information as entered in the data shift register. These 16 bits word are displayed by default in grey but when a synch char is detected the display is done in yellow and a bad synch sequence is displayed in red.

The line above is used to display the extracted **Clock** information. These bytes are displayed by default in grey but when the clock value is 5% above normal the display is done in red and if the clock is 5% below normal the display is done in green.

The line above is used to display the extracted **Data** information. These bytes are displayed by default in grey. Decoded bytes that contain flux transitions with timing violations or border bits are displayed in red. If an invalid data is found inside a GAP the display is done in orange.

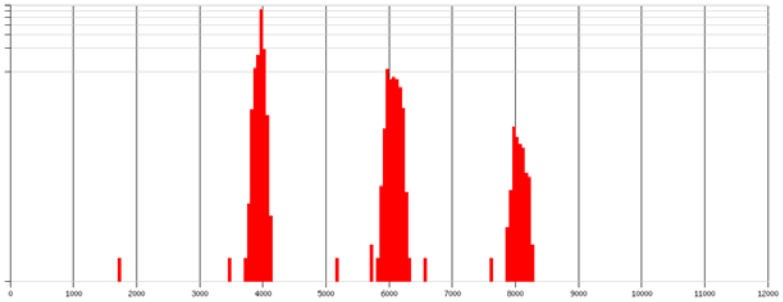
Here is an example:



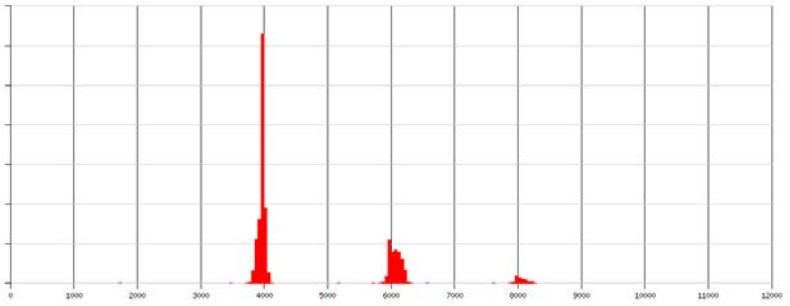
## Transition Histogram Plot

The transition histogram plot displays the distribution of the flux transitions for an overall content of the track. The histogram consists of tabular frequencies, shown as adjacent rectangles, built over discrete intervals (bins), with an area equal to the frequency of the observations in the interval. For a normal track the distribution of the flux transitions are centered around 4 , 6, and 8 µs with the maximum around 4, then smaller ones around 6 and even smaller ones around 8 µs.

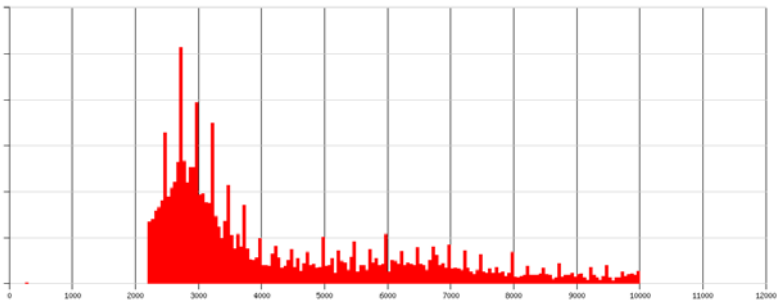
As the distribution decrease rapidly around the center frequency, by default a logarithmic scale is used. A typical output looks like:



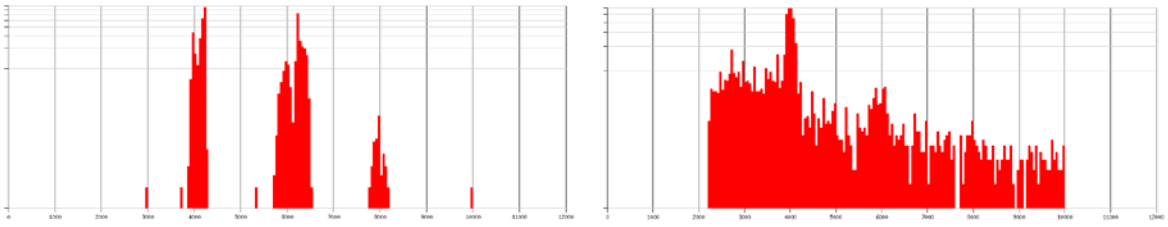
If you prefer it is possible to use a linear scale by adding the parameter **scale=linear** in the command line (e.g. pl scale=linear -gif kf_histo.pl). For example the same histogram looks like this:



An unformatted track has random transitions distribution:

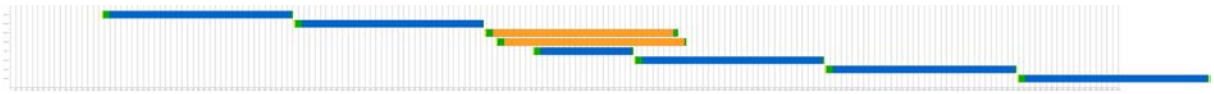


Some protection uses specific variations of the bit width on a specific sector of a track. In such a case the overall distribution is difficult to interpret. It is therefore possible to look at the transition distribution for a specific sector by using adding the parameter **sec=sec_num**. The following examples show a Copylock protection with bits shifted around 4.2 (pl -gif sec=6 kf_histo.pl) and a sector partially unformatted (pl -gif sec=10 kf_histo.pl)



## All Sectors plot

The all sector plot is a long skinny plot that contains the overall content of all the sectors found on the track. It is recommended to read the output with a graphic editor so you can zoom on a specific area.



The graphic contain one line for each sector. The following blocks are displayed for each sector:

- The ID blocks is displayed in yellow
- The GAP3 blocks is displayed in green
- The DATA block is displayed using the following colors:
  - Blue if the block is read correctly (no CRC error and no Fuzzy bits)
  - Orange if the block contains some fuzzy bits. Note that fuzzy bits imply also CRC error (in most cases).
  - Red if the block is read with CRC errors but no fuzzy bytes.
- The GAP4 block is displayed in green.

This plot is very useful to see at one glance many of the protection used in the Atari world. For example in the above example we can see that the last sector passes the index (data over index) and that therefore some room for this sector is available at the beginning of the track. The two sectors in orange indicate sectors with fuzzy bits. We can also see that we have 3 overlapping sectors (sector within sectors).

# KFAnalyze Inner Workings

This section provides details information about the inner workings of the **KFAnalyze** program. This can be useful to persons that want to better understand how the program emulates some functions of the WD1772 FDC and how the different outputs are created.

The **KFAnalyze** program uses to libraries:

- The **KFStream** Library to read and decode the input stream file and
- The **KFEmul** Library to emulate several functions of the WD1772 FDC. This library also provides facilities to write plot data for **Ploticus** and to detect floppy disk protections.

The KFAnalyze program is a member of the KFProject suite of tools that I have developed around the KryoFlux board.

This tool uses a disk image produced by a KryoFlux board (SPS Disk Imager) and produce information that enables to:

- Analyze the structure and content of a FD
- Outputs files for plotting the layout and content of a FD
- Analyze the Copy Protections of a FD
- Verify that the disk is authentic and is error free

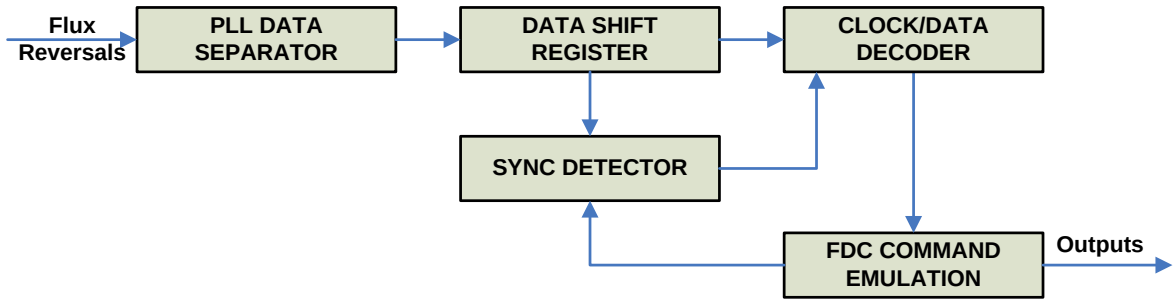
## KFStream

This library is responsible to read the Stream files. It fills mainly two array of information: one for the index and one for the actual flux transitions. The index array contains information about the actual rotation speed and the location of the flux transitions relative to the index. The flux array contains the timing information for all the flux reversals of all the revolutions imaged with the KryoFlux board.

## KFEmul

This library is the core of the program. It is responsible of the low-level decoding of the bytes as well as the high-level interpretation of these bytes.

The following diagram shows the main blocks used in the WD1772 emulation:

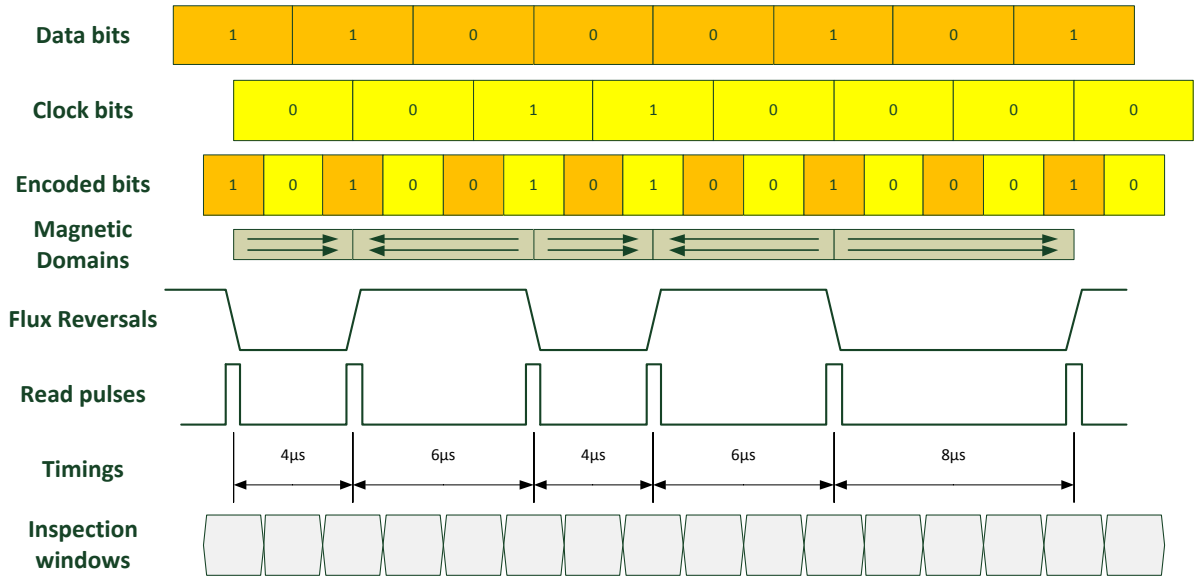



Each of these blocks is detailed below.

## PLL Data Separator & DSR

The flux reversals are read from the Stream files. This raw data consists of composite clock and data bits encoded in MFM. This encoded data has to be *synchronized* and *decoded*. These functions are performed by the Data Separator. The data separator has a *Phase Locked Loop* which attempts to lock on to the bit stream and synchronize it. Having locked-on to the bit stream the data synchronizer circuit must first determine the nominal position of clock and data bits and then generate an appropriate clock and data windows that is centered around the bit positions. For that matter the KFAnalyze program implements the PLL algorithm described in the **US patent 4,870,844**. For information the WD1772 FDC, as well as many other FDC build in the 80s, is using this algorithm (or similar ones). The patent is rather complex and this section only highlight some of the important aspects of the PLL algorithm that are of interest to understand how the clock and data are extracted from the input stream of flux reversals.

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



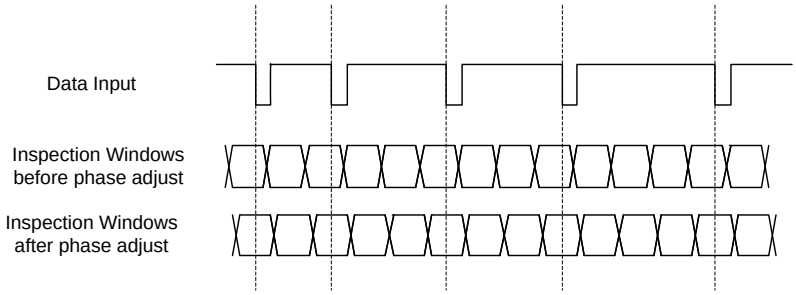

The Data and Clock bits are recorded on a floppy disk as magnetic flux reversals. The "polarity" ("direction") of the magnetic flux doesn't mean anything it is the "timing" between the flux reversals that is used to encode data.

The MFM is a 1,3 RLL (Run Length Limited) encoding. Where '1,3' means a minimum of 1, and a maximum of 3 clock periods between flux reversals. Thus '1,3' means that flux reversals occur no more often than every two clock periods but at least every 4th clock periods (3 periods without a reversal). On Atari DD MFM floppy disks, the clock period is 2µs and therefore nominal flux reversals (aka transition spacings) arise every 4, 6, or 8 microseconds.

In MFM encoding each cell contains a position for a clock pulse and a position for a data pulse. A data pulse is present if the data bit is a one. A clock pulse is present only if the data bit is a zero and the data bit in the previous bit window was a zero. As a result there is a maximum of one flux change per bit cell. Clock bits are written at the beginning of the bit cell, while data bits are written in the middle of the bit cell.

The data separator ensures that the read pulses received from the stream are converted into bits stored in a data shift register (DSR). For that matter the phase lock loop data separator defines **inspection windows** that repeat nominally every

2µs (so we sample the **data** and **clock** bits). They can be varied in *time duration* and/or *start & stop* time of the windows. A one is passed to the data shift register if a read pulse is received at any time during one inspection windows; otherwise a zero will be stored. 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 PLL apply **frequency** and **phase** corrections that compensate the input data frequency drift due to unsteadiness of the motor drive speed (frequency drift), and the migrations of the magnetic transition area (phase drift). Ideally, individual pulses should be located in the middle of the inspection windows and therefore the start/stop time of the inspection windows are adjusted to compensate for deviation in time of arrival of the most recently detected data pulse. This phase correction is done proportionally to the distance of the transition with the middle of the inspection window.




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 transitions. By sampling each bit, the phase-lock loop determines the phase error between a bit and the frequency being generated. To determine the nominal bit position around which to center the window, the data separator must track data bit frequency changes, yet ignore jitter. In this manner, even if an unpredictable bit shift occurs, the data separator can adjust the window's position to compensate for the change. Otherwise the shifted bit could be positioned outside the window.

The proper ratio of phase and frequency correction provided in the loop is carefully balanced so that the PLL 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.

Once the bit stream read from the disk has been synchronized and decoded to NRZ data, it is recorded directly by the Data Shift Register block.

*Note 1: It is interesting to know that the PLL as defined in the patent permit an input frequency variation of up to 9%. This corroborates the actual measurement made for the WD1772 that correctly interprets bits with a variation of 9 to 10 % in DD MFM. These values are above the variation used by many protection mechanisms like* ***Copylock*** *(about 5%) and* ***Macrodos*** *(+/- 5%) and therefore the data read from sectors using these protections should be delivered correctly.*

*Note 2 about* ***Border bits****. With the above information it is easy to understand that if a transition happens at the extreme border of an inspection window it will be detected into one or the next inspection window based on small variation (for example of the drive rotation speed) resulting in random values returned (fuzzy bits). For example having a transition at 5µs after the previous one can be interpreted as a transition after 4µs or a transition after 6µs based on small frequency variation of the input.*

## Synch Mark Detector & Data Decoder

The next task is to recognize the byte boundaries accurately (where the data starts). This is done by detecting a ***sync mark*** with a "missing clock", which provides a fixed reference in the bit stream to set the byte boundary.

This sync mark has two important properties: it has no runs of zeros shorter than 1 or longer than 3 (i.e. it follows the (1,3) RLL rules), and it will never occur in any bit position in any encoded data stream.

The main sync mark used in MFM is called the "A1 Sync" since the data bits form the hexadecimal value $A1 (1010 0001).

```
Data:      1 0 1 0 0 0 0 1
Clock:      0 0 0 1 1 1 0
Encoded:   100010010101001
Sync clock: 0 0 0 1 0 1 0
Sync Mark: 100010010001001
                   ^ Missing clock bit
```

Upon receipt of this sync mark a divide-by-eight counter is set, that allows acquiring the following bytes correctly.

*Note that the **synch detector** is enabled or disabled depending of the command executed by the FDC as well as the field of the track currently read. For example during a **read track** command the sync mark detector is active at all time, while during a **read sector** command it is disabled while reading an ID or DATA field.*

The WD1772 (as well as the **KFAnalyze** program) also detects a second sync mark called the "C2 Sync" since the data bits form the hexadecimal value C2 (11000010).

```
Data:      1 1 0 0 0 0 1 0
Clock:      0 0 0 1 1 1 0
Encoded:   101000010101100
Sync clock: 0 0 0 1 0 0 1
Sync Mark: 1001001000101000
                   ^ Missing clock bit
```

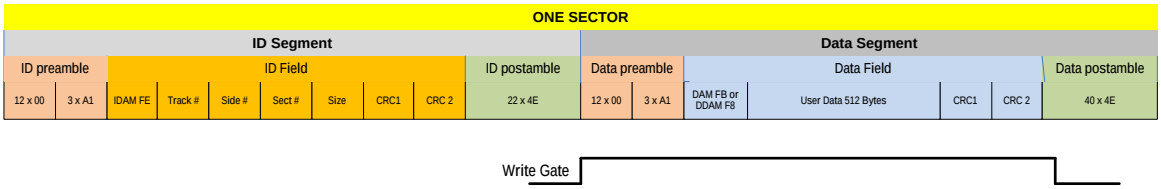
Unfortunately whenever a bit stream contains the string 000101001 it is detected by the Sync Mark Detector as a "C2 Sync" and therefore the stream is synchronized on this string and all the following bytes are shifted. This is known as the **false sync** problem/error of the WD1772. This error only occurs when reading a complete track with the read track (where the sync mark detector is enabled at all time) but not when reading sectors (where the sync mark detector is disabled while reading the ID/Data fields).

## FDC Command Emulation

Once the bit stream has been decoded into data bytes they must be interpreted based on one of the emulated FDC commands. The KFAnalyze program emulates 3 commands of the WD1772 FDC:

- The ***read track*** command
- The ***read address*** command
- The **read sector** commands

For the layout we use the following terminology:




- One sector is composed of: An ID Segment and a DATA Segment.
- An ID Segment is composed of an ID preamble (aka GAP2), an ID field, and an ID postamble (GAP3a)
- A DATA Segment is composed of a DATA preamble (GAP3b), a DATA field, and a DATA postamble (GAP4).

### *Read Track Command*

The read track command of the WD1772 is extremely simple: the reading of bytes starts with an index pulse and continues until the next index pulse. All Gap, Header, and data bytes are assembled and transferred to the buffer. This command has several characteristics: no CRC checking is performed; gap information is included in the data stream; and the Address Mark Detector is on for the duration of the command. The IDAM, ID field, ID CRC bytes, DAM, Data, and Data CRC Bytes for each sector are read correctly. The Gap Bytes may be read incorrectly during write-splice time because of synchronization.

While assembling the complete track in a buffer the KFAnalyze programs also interprets the layout of the track. Here is a simplified description of how the program interprets the different fields of each sector.

- The layout starts in a GAP1 field until
- A GAP2 field is detected as a series of $00 or $FF bytes (until a sync mark is found we may be shifted by a half bit cell). The GAP2 field is the first field of a sector. The field continues until
- An IDAM character is found after 3 sync marks. This starts an address field that is read for 7 characters (IDAM, Track, Side, Sector number, Sector length, 2 CRC bytes).
- At the end of the Address field we are in GAP3 field. This field continue until
- A DAM or DDAM character is found after 3 sync marks. This starts a data field. A data field that usually contains 515 characters (DAM, 512 Bytes, and 2 CRC bytes).
- At the end of the data field we are in GAP4 field. This field is the last field of a sector and it continues until a GAP2 is detected.
- The GAP2-ID-GAP3-DATA-GAP4 fields repeat until the end of the track.

The field interpretation of the track (called the layout of the track) performed by the KFAnalyze program is not always accurate because of some of the copy protections mechanisms used. For example sector within sector are not detected, data over index may cause in incorrect GAP2 detection, etc. The layout produce should therefore be considered as informational.

However the data entered into the buffer by the KFAnalyze program are exactly the same that would be read by a WD1772. Remember that due to the fact that the sync mark detector is enabled all the time the data inside an ID or DATA field can be read incorrectly (bit shifted) and that synchronization may be lost in GAP due to write splicing.

By default the program uses the data of the first revolution imaged by the KryoFlux board, but it is possible to use data from another revolution by using the **-r<rev>** command line argument.

### *Read Address Command*

The read address phase of the KFAnalyze program is slightly different than the WD1772 read address command: instead of reading the next encountered ID field from a track, the KFAnalyze reads **all** the ID fields of the track. The program search for 3 sync character followed by an IDAM. When found it reads into a buffer the ID field information. When an ID has been completely read the program looks for another ID field until the end of the track is found.

Much extra information is available at the end of this phase that is not available in a real WD1772. For example the location of all the ID fields in the track. In this mode sector within sector are correctly detected as a WD1772 would do.

By default the program uses the data of the first revolution imaged by the KryoFlux board, but it is possible to use data from another revolution by using the **-r<rev>** command line argument.

### *Read Sector Command*

The read sector phase of the KFAnalyze program is in fact composed of several underlying operations.

- *First* a read sector operation is performed that is equivalent to the WD1772 FDC with some particularities:
  - Normally when a read sector command is received by the FDC it searches for the next encountered sector that matches the requested ID. As some copy protection uses the capability to write several sectors with the same ID it is unpredictable to know which of the duplicated sector will be read unless special actions are taken. With KFAnalyze it is possible to clearly indicate which of the duplicated sector is read.
  - Another copy protection mechanism used is to place "data over the index". This indicates that the beginning of a sector is placed at the end of the track and that the data field extend pass the end of the track. On a physical track this means that the read continue pass the index mark. In this case the KFAnalyze program continues to read the data passed the current "revolution" with the data of the next revolution. This is the reason why the **-r<rev>** command line argument must be specified in the range 1 to n-1 revolutions as it is unpredictable to know if DOI has been used on a track and that data from the next revolution need to be used.

- *Second* multiple reads (n-1 reads) of the same sector is performed. The first read is considered as the reference and the data read are placed into the "reference buffer" displayed at the end of the program. The next n-2 read are used to read into temporary buffer the data and to compare this buffer with the "reference buffer". If in any of the buffers the values differ this clearly indicate that fuzzy bytes have been detected.
- When all the sectors including duplicate sectors and sector within sectors have been read the layout of the track and the buffers are displayed.

Internally the program searches the requested sector by looking for the correct ID field then it searches for a corresponding DATA field within the next 43 bytes (please refer to WD1772 documentation for more information).

At the end of the read sector phase the layout of all the sectors and the content of the buffer for each sector are displayed.

By default the program uses the data of the first revolution imaged by the KryoFlux board to fill the reference buffer, but it is possible to use data from another revolution by using the **-r<rev>** command line argument. Detection of fuzzy bits is always done with data of all the revolutions but the one used as reference. For example if 5 revolutions have been imaged by the KryoFlux board and **-r3** has been specified, the program will use the data in revolution 3 as the reference (buffer displayed in the output) and will compare with data from revolution 1, 2, 4 to detect fuzzy bits.

# Document / Program History

## V1.3

Code has been partially rewritten to be shared with IPFAnalyze and IPFPanzer and to fix minor bugs. The most visible changes to the user are:

- In read track phase the sectors within sector are now detected,
- for fuzzy bits a fuzzy mask buffer is now displayed,
- The program now detects write splice in GAP3 and 4.
- Extra information is output to the trans.dat file that allows displaying a histogram of the flux transitions distribution with the kf_histo.pl Ploticus script.
- It is now possible to display mfm information in the buffer with the **-m** flag.
- Removed the -p flag, the -m flag has been renamed to -l for simple dump and -m flag added to display mfm information