

KryoFlux

Stream File Documentation

Jean Louis-Guerin

Revision 1.1- 01/12/2013

Table of Content

Table of Content	2
Presentation	3
Imaging Floppy Disks with the KryoFlux device	4
<i>KryoFlux Clocks & Counters</i>	4
Sample Counter	4
Index Counter	4
<i>Data Format</i>	4
Description of Stream Files	5
<i>Block Header</i>	5
<i>ISB (In Stream Buffer) Blocks</i>	5
Flux blocks	5
Flux Data Encoding Optimization	6
NOP Blocks	7
<i>OOB (Out Of stream Buffer) Blocks</i>	7
Invalid block	8
StreamInfo block	8
Index block	8
StreamEnd block	9
KFInfo block	9
EOF block	9
<i>Index Timing Consideration</i>	10
<i>RPM Interpolation</i>	10
Decoding Stream Files	11
<i>KryoFlux Device Behaviour</i>	11
<i>KryoFlux Hardware Information</i>	12
<i>Parsing the Stream File</i>	13
<i>Analysis of Index Information</i>	13
Sample Counter Overflows before Index	14
Index pointing after last flux	15
Index detected before any flux	15
Terminology	15
References	15
Document History	15

Presentation

This document provides a [description](#) of the **Stream Files** used by the **DTC** (Disk Tool Console) program eventually connected to a [KryoFlux Device](#). A *Stream File* is either produced by the *DTC* program in **read (imaging)** mode or consumed by the *DTC* program used in **write (backup)** mode. The document also provides assistance to programmers on [Decoding Stream Files](#). It is based on my interpretation of the KryoFlux documentations published by [Software Preservation Society](#) and [KryoFlux Product & Services Limited](#) (see [references](#)) but it also includes information based on experimentations. Note that some sections of this text are taken, almost directly, from the original [SPS referenced documents](#).

I want to thank **István Fabián** from SPS who has provided to me with a lot of detail information on how the [KryoFlux device](#) operates and code to decode a Stream File.

Note: Regular users of the KryoFlux device should not be concerned by the information presented here which is mainly of interest to programmers that want to write tools around the Stream Files.

The data in the *Stream File* is stored in binary and therefore cannot be directly displayed or edited with a text editor. It is interesting to note that *Stream File* content has not been conceived per se as a file format, because it is actually an exact copy of the byte **Stream Protocol** used between the *KryoFlux* device and the host system when communicating over an USB link.

The byte *Stream Protocol* is **optimized** for a communication budget and a CPU budget. The KryoFlux SoC has dedicated sampling, communication etc. hardware. Each are served via interrupts for asynchronous and efficient operation. Therefore the firmware has many asynchronous sub-systems communicating with the hardware, with very specific requirements, constraints and CPU budget. If the firmware fails to serve these, the system will become unusable. You have to realize that a single track can have about 500,000 flux reversals per second for a High Density disks that is streamed to the host computer via an USB link with limited bandwidth.

Therefore the *Stream File/Protocol* is defined for *transfer and processing efficiency* and some complexity arises from this during decoding.

Note that Stream files are hardware specific (to the KryoFlux device) and therefore are not intended for long term preservation.

*WARNING: The **Stream file** format might be changed as needed. The version described in this document was done using Version 2.2 of DTC.*

Imaging Floppy Disks with the KryoFlux device

In order to capture everything on a floppy disk, it is necessary to sample all the *flux reversals* between several *Index Signals*. The KryoFlux device starts sampling data before the first *Index Signal*, and may sample data after the last *Index Signal*. This is important to ensure that all information on a floppy disk is captured. Outside *Index Signals* data cannot be meaningfully decoded.

For various reasons, especially for games, multiple revolutions of data should be captured in a constant stream. This means a stream file usually should contains more than two *Index Signals*. In order to correctly analyze protections used on floppy disks it is generally required to record multiple revolutions. In order for SPS to correctly produce IPF files with the CTA analyzer the minimum required is five revolutions (6 indexes).

KryoFlux Clocks & Counters

The KryoFlux Device is operated from a *Master Clock (mck)*. From this master clock two synchronous clocks are derived:

- The *Sample Clock (sck)* used by the *Sample Counter* to sample Flux reversals.
- The *Index Clock (ick)* used by the *Index Counter* to sample Index Signals.

The clock frequencies are defined by the KryoFlux hardware, and can be queried using a device command. The **default values** are stored as 64-bit floating points:

Abbreviation	Name	Clock Value
mck	Master Clock	$((18432000 * 73) / 14) / 2 = 48054857,14285714$
sck	Sample Clock	$mck / 2 = 24027428,57142857$
ick	Index Clock	$mck / 16 = 3003428,571428571$

Starting with KryoFlux Firmware 2.0+ the device transmits Hardware information that includes the values of these two clocks (please refer to the [KryoFlux Hardware Information](#) section). It is recommended to use these values as KryoFlux hardware may change these frequencies at some point in the future.

Sample Counter

The *Sample Counter* is used to measure the elapsed time between two flux reversals, or between a Flux reversal and an Index Signal. This counter has a 16 bits width and possible overflows are recorded. This counter is reset after each Flux reversal recording.

Index Counter

The *Index Counter* is a “free running” counter (not reset). The value of this counter is recorded each time an Index Signal is detected.

Data Format

Data in a *Stream File* is byte-aligned for processing efficiency. This means that no information is encoded at the bit level and therefore there is no need to break a byte down into bits in order to be interpreted further.

Data stored in 16 or 32 bits words uses the little-endian bytes ordering (the least significant byte first, and the most significant byte last). This does not apply to [Flux Blocks](#) that use a specific encoding.

Description of Stream Files

The data in a *Stream File* is organized in **Blocks** that have a variable length ranging from one to many bytes. The first byte of a stream file *Block* is called the **Block Header**. It specifies how to interpret the *Block*. A stream file contains two types of *Blocks*:

- The **ISB** (In Stream Buffer) blocks that are used to communicate the timing value of the sampled flux reversals.
- The **OOB** (Out Of stream Buffer) blocks that are used to help in the interpretation-verification of the *Stream File* as well as to transmit other critical information like Index Signals timing, or KryoFlux hardware information.

For explanation about the ISB / OOB [terminology](#) please refer to [KryoFlux Device Behavior](#)

The most important information to retrieve from a Stream File is:

- Timing of **Flux Reversals**: All data flux reversals detected by the KryoFlux device are stored in **ISB** Blocks.
- Timing of **Index Signals**: All index signals detected by the KryoFlux device are transmitted in special **OOB** blocks: the **Index** Blocks. The provided Index information allows to compute the precise **Index Time** (time between to index signals) as well as to find the **Index Position** in reference to the current data flux reversals.

Block Header

The interpretation of the information contained in a **Block** of data depends on the **Block Header**. This header can take the following values (sorted in ascending order):

Header	Name	Length	Description
0x00-0x07	Flux2	2	Flux block: flux reversal count coded on <u>two</u> bytes
0x08	Nop1	1	NOP block: Continue decoding at current position + 1
0x09	Nop2	2	NOP block: Continue decoding at current position + 2
0x0A	Nop3	3	NOP block: Continue decoding at current position + 3
0x0B	Ovl16	1	Flux block: next flux reversal count to be increased by 0x10000.
0x0C	Flux3	3	Flux block: flux reversal count coded on <u>three</u> bytes
0x0D	OOB	variable	First byte of an Out Of stream Buffer block
0x0E-0xFF	Flux1	1	Flux block: flux reversal count coded on <u>one</u> byte

ISB (In Stream Buffer) Blocks

An **ISB Block** is either a **Flux Blocks** or a **NOP Block** (*i.e.* not an **OOB Blocks**).

Flux blocks

A **Flux Block** is used to store the value of the **Sample Counter**. This correspond to the number of **Sample Clock Cycles** (sck) between two flux reversals.

The flux reversal absolute timing values can be computed by dividing the sample counter value by the Sample Clock (sck):

```
AbsoluteFluxTiming = FluxValue / sck;
```

Flux1 block

This block allows to store very efficiently the timing of a sampled flux reversal as it is coded on only **one** byte. The [Block Header](#) has a value in the range **0x0E-0xFF**.

0x0E-0xFF

In this case the value of the sampled flux reversal is directly equal to the value of the [Block Header](#). In practice you will find that most of the flux reversal values fall into this range (0x0E-0x0FF) and therefore this contribute to a very efficient coding of the stream file.

```
FluxValue = Header_value;
```

Flux2 block

This block allows storing the timing of a sampled flux reversal coded on **two** bytes. The [Block Header](#) has a value in the range **0x00-0x07**.

0x00-0x07	Value1
-----------	--------

In this case the value of the sampled flux reversal is computed as follow:

```
FluxValue = (Header_value << 8) + Value1;
```

Flux3 block

This block allows storing the timing of a sampled flux reversal coded on **three** bytes. The [Block Header](#) has a value equal to **0x0C**.

0x0C	Value1	Value2
------	--------	--------

In this case the value of the sampled flux reversal is computed as follow:

```
FluxValue = (Value1 << 8) + Value2;
```

Ovl16 block

This block indicates that the next [Flux Block](#) has a value superior to the max value of a 16 bits number (0xFFFF). The [Block Header](#) has a value equal to 0x0B.

0x0B

In this case the **next** [Flux Block](#) value is incremented by 0x10000.

```
FluxValue = 0x10000 + NextFluxValue;
```

This block is inserted whenever the [Sample Counter](#) overflows. There is no limit on the number of Ovl16 blocks present in a stream, and so the maximum value for a flux reversal is virtually unlimited, although the decoder in the KryoFlux host software uses a 32 bits value. Flux reversal values that do not fit into 16-bits are quite unusual, but have been found in games that attempt to fool the AGC (Automatic Gain Control) of the drive electronics.

Flux Data Encoding Optimization

A Flux2 block could be used to encode data in the range 0x0000-0x07FF. But in practice it is more efficient to use a Flux1 block (only one byte) for encoding data in the range **0x000E-0x00FF**. Therefore the Flux2 is only used to encode data in the range 0x0000-0x000D or data in the range 0x0100-0x07FF. For similar reasons (best efficiency) a Flux3 block is only used for encoding data in the range 0x0800-0xFFFF.

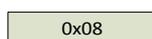
If the flux reversal value to transmit is bigger than 0xFFFF then one or several ovl16 block(s) is (are) used to add 0x10000 to the next flux reversal value.

NOP Blocks

A NOP (No-operation) Block is used to skip one or several byte(s) in the stream buffer. This makes it possible for the firmware to create data in its ring buffer without the need to break up a single code sequence when the filling of the ring buffer wraps. A NOP block starts with a [Block Header](#) in the ranges **0x08-0x0A**.

NOP1 block

NOP1 block is used to skip one byte in the buffer. The [Block Header](#) is equal to **0x08**.



Just skip this byte during decoding.

NOP2 block

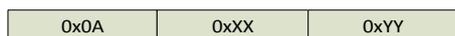
NOP2 block is used to skip two bytes in the buffer. The [Block Header](#) is equal to **0x09**.



Just skip these two bytes during decoding.

NOP3 block

NOP3 block is used to skip three bytes in the buffer. The [Block Header](#) is equal to **0x0A**.



Just skip these three bytes during decoding.

OOB (Out Of stream Buffer) Blocks

An **OOB Block** is either used to help in the interpretation/verification of the stream file or it contains specific information (index signal, KryoFlux HW info). Note that OOB blocks are sent completely **asynchronously** of the [ISB](#) blocks (please refer to [KryoFlux Device Behavior](#)).

An **OOB Block** is composed of an **OOB Header Block** (always four bytes) followed by an *optional OOB Data Block*.

The *OOB Block Header* contains three fields:

- The first field (one byte) contains the [Block Header](#) and is always equal to **0x0D**.
- The second field (one byte) describes the **Type** of the OOB (see below).
- The third field (2 bytes) indicates the **Size** of the optional **OOB Data Block**.



The next optional **OOB Data Block** contains information specific to each Type of OOB Block.

The following table lists the different types of **OOB Block**

Type	Name	Meaning
0x00	Invalid	Invalid OOB
0x01	StreamInfo	Stream Information (multiple per track)
0x02	Index	Index signal data
0x03	StreamEnd	No more flux to transfer (one per track)
0x04	KInfo	HW Information from KryoFlux device
0x0D	EOF	End of file (no more data to process)

Invalid block

It is not clear when this OOB Block is used but it definitively indicates a problem ☹.

0x0D	0x00	0x0000
------	------	--------

An Invalid block contains the following fields:

- Type = 0x00
- Size = 0x0000?

StreamInfo block

A StreamInfo block provides information on the progress of the data transfer. It is sent whenever the communication and the KryoFlux CPU budget allows it, naturally the ordering of the StreamInfo blocks is guaranteed. It is possible to have several StreamInfo blocks at once. It is used primarily to check that no bytes have been lost during transmission but it can also be used to compute the transfer speed of the USB link between the host and the KryoFlux device.

0x0D	0x01	0x0008	Stream Position	Transfer Time
------	------	--------	-----------------	---------------

A StreamInfo block contains the following fields:

- *Type* = 0x01
- *Size* = 0x0008 (size of the following data block)
- *Stream Position* (4 bytes) indicates the position (in number of bytes) of the OOB Block Header in the stream buffer.
- *Transfer Time* (4 bytes) gives the elapsed time (in milliseconds) since the last StreamInfo block. It is therefore possible to calculate the transfer speed between the host and the board as well as the transfer's jitter.

Index block

This block is used to provide timing information about a detected index.

0x0D	0x02	0x000C	Stream Position	Sample Counter	Index Counter
------	------	--------	-----------------	----------------	---------------

An Index block contains the following fields:

- *Type* = 0x02
- *Size* = 0x000C (12 decimal – size of the following data block))
- *Stream Position* field (4 bytes) indicates the position (in number of bytes) in the stream buffer of the **next** flux reversal just after the index was detected.
- *Sample Counter* (4 bytes) gives the value of the [Sample Counter](#) when the index was detected. This is used to get accurate timing of the index in respect with the **previous** flux reversal. The timing is given in number of [Sample Clock](#) (**sck**). Note that it is possible that one or several sample counter overflows happen before the index is detected.
- *Index Counter* (4 bytes) stores the value of the [Index Counter](#) when the index is detected. The value is given in number of [Index Clock](#) (**ick**). To get absolute timing values from the index counter values you need to divide these numbers by the index clock (**ick**)

For more information on index timing interpretation see [Index Timing Consideration](#) and [Analysis of Index Information](#).

StreamEnd block

A StreamEnd block indicates that all the [Flux blocks](#) have been transmitted. It also provides a KryoFlux status code that indicates if the streaming was done correctly by the hardware.

0x0D	0x03	0x0008	Stream Position	Result Code
------	------	--------	-----------------	-------------

A StreamEnd block contains the following fields:

- *Type* = 0x03
- *Size* = 0x0008 (size of the data block)
- *Stream Position* field (4 bytes) indicates the position (in number of bytes) of the OOB Block Header in the stream file.
- *Hardware Status Code* (4 bytes) returns a value as defined below. A value of 0 indicates that the streaming was successful any other value indicates various problems.

Hardware Status Code:

Value	Name	Meaning
0x00	Ok	Transfer success (does not imply data is good, just that streaming was successful)
0x01	Buffer	Buffering problem - data transfer delivery to host could not keep up with disk read
0x02	No Index	No index signal detected

KFInfo block

A KFInfo block is used to transmit information from the KryoFlux device to the host.

0x0D	0x04	Size	Info Data (ASCII)
------	------	------	-------------------

A KFInfo block contains the following fields:

- *Type* = 0x04
- *Size* (2 bytes) = number of bytes of the KFInfo data block (including the terminating null)
- *Info Data* – a null terminated ASCII String of information

More details about Hardware Information transmitted can be found in the section [KryoFlux Hardware Information](#)

EOF block

An EOF block is used to indicate the end of the stream file. No processing needs to be done beyond this block.

0x0D	0x0D	0x0D0D
------	------	--------

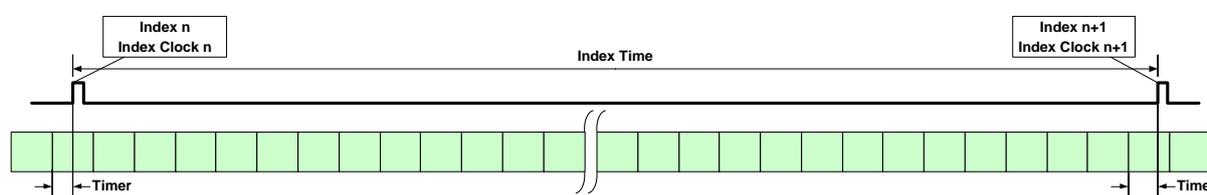
An EOF block contains the following fields:

- *Type* = 0x0D
- *Size* = 0x0D0D (not meaningful)

Index Timing Consideration

Flux Reversal timing values recorded in a *Stream File* only makes sense when the **Index Signals** positions are known. Once all of the data in the stream file has been processed, several computations are required on the index data in order to determine:

- **Index Position:** the exact position where an *Index Signal* occurred in reference to the Flux Reversals. It can be determined during decoding by storing the position of all the flux reversals and the position where each index signal occurs.
- **Index Time:** This is the time taken for one complete revolution of the disk. It is equal to the number of index clock cycles since the last index occurred. It can also be calculated by summing all the flux reversal values that we recorded since the previous index, adding the **Sample Counter** value at which the index was detected (see *Sample Counter* field in [Index Block](#)) and subtracting the **Sample Counter** value of the previous index. The computation details are explained in [Analysis of Index Information](#).



The **Index Time** allow to compute the exact FD RPM value for one revolution. For example for a drive that run at 300 RPM the time between two indexes should be 200ms. We know from experience that the actual value differs and it is therefore important to monitor the RPM for each revolution sampled. Note that up until the first index, an Index Time cannot be generated as it will always be a partial revolution.

The **Index Position** is also important, as it is the only marker on a disk that can be used to perfectly align data when writing, or deciding on the exact position of data when reading.

RPM Interpolation

To increase reliability, the decoding software can perform RPM interpolation when converting timing to absolute values. If the RPM of one index is significantly different from the following index, it may be that the disk drive doing the reading is unreliable, and the drive speed from index to index is not constant. But even if RPM is very stable, it may have been set incorrectly, like say 301 RPM instead of 300 RPM. This would affect all flux reversals across the track. Since there are hundreds of thousands of samples, the differences will add up eventually. We can moderate this variations by converting each flux value using an interpolated value. Various interpolation algorithms are possible to do this. For example the time measured for the flux reversals can be corrected using a factor that takes in account the actual speed of the drive (e.g. 301 RPM), versus the expected speed of the drive (i.e. 300 RPM) with the following formula:

$$\text{CorrectedValue} = \text{OriginalValue} * \text{Expected_RPM/Actual_RPM};$$

Decoding Stream Files

It is recommended to decode a KryoFlux stream file in two passes:

- The first pass is used to parse the Stream File in order to retrieve and store all the important information (flux/index timing and positioning).
- The second pass is used to analyze the stored data in order to compute the exact positioning of the Index Signals relative to flux reversals as well as the index times.

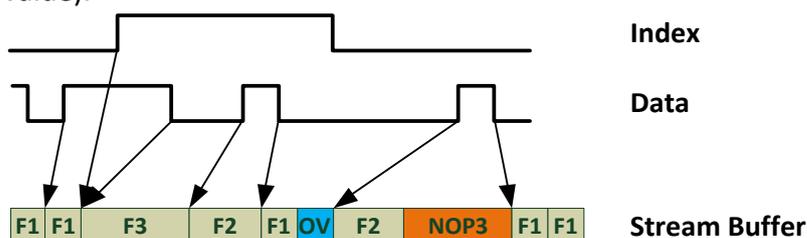
It is also recommended to check if KryoFlux hardware information about SCK and ICK has been passed. If this is the case these values should be used rather than default clock values.

KryoFlux Device Behaviour

In order to correctly process the data stored in a Stream File it is useful to have a basic understanding of the way the KryoFlux Device operates (thanks to **István Fabián**).

When imaging signals from a floppy drive reading a floppy disk there are two main processes running independently in the KryoFlux device:

- The first process is referred as the **sampling process**. As the name indicates it is responsible of capturing the data from the floppy drive and storing this information in a buffer called the **stream buffer**. Therefore this buffer only contains the Flux blocks (including Ovl16 blocks) and eventually some NOP blocks (considered as data without value).



In this hypothetical example (not strictly realistic) we can see that each flux reversal value is stored by the KryoFlux device in the **stream buffer** as a [Flux1](#) or [Flux2](#) or [Flux3](#) blocks (see [Flux Data Encoding](#)). This value corresponds to the value of the [Sample Counter](#) at the time of the flux reversal and once recorded the counter is **reset**.

Whenever the Sample Counter overflows an [Ovl16](#) block is stored in the stream buffer. The firmware can also add NOP blocks in the stream buffer when necessary (see [NOP blocks](#)). When an index signal is detected the information is **not** placed in the stream buffer but the **position** of the **next** flux reversal in the stream buffer is recorded as well as the value of the [Sample Counter](#) (time from previous flux reversal) and the [Index Counter](#).

- The second process is referred as the **transfer process**. It is responsible of transferring the data from the KryoFlux device to the host over the USB link. The first priority of the transfer process is to transmit the data stored in the stream buffer. But, whenever the communication and the KryoFlux CPU budget allow it, this process also transmits “extra information”: the **OOB** Blocks. They are used either to transfer index information or to help in the transfer decoding. These blocks are not part of the *Stream Buffer* and are “inserted on the fly”, by the transfer process, between ISB blocks at unpredictable times. This implies that the information in the OOB Blocks is completely asynchronous from the information in the ISB Block. For example it is possible to transmit information about an Index that refers to a flux reversal **not** yet transmitted!

The sampling can be stopped after a certain amount of index signals automatically or programmatically via a command; DTC may use both. Right now streaming is requested to stop after a specified number of indices, but if DTC detects certain errors it may send a stop command at any time, that would stop the streaming as soon as possible, i.e. at some random location on the track. Even if sampling is to be stopped at an index signal, since it is independent of streaming, it may or may not stop immediately, it all depends on luck.

The transfer process always sends back all the data that was sampled before signaling the transfer finished to the host. In other words; there may be one or more samples after the last index signal (if index stop mode used) or there may be none.

KryoFlux Hardware Information

Starting with version 2.0 of the firmware the KryoFlux device transmits information in one or several [KInfo block](#). Most of the data transmitted are informative information about the version of firmware, hardware etc.

Several Strings (usually two) can be passed from the KryoFlux device to the host. Each strings are null terminated. The size field in the [KInfo](#) block gives the length of the string including the trailing space.

The information inside the strings are passed as “name” “value” pairs separated by comma (“,”) character. For example “host_date=2012.01.22, host_time=17:44:47”.

Among the information transmitted two strings are particularly important: the sample clock (**sck**) and the index clock (**ick**). You should use these values instead of the default values specified in [KryoFlux Clocks & Counters](#) section.

Here is an example of the information transmitted:

```
host_date=2011.03.21
host_time=17:20:17
name=KryoFlux DiskSystem
version=2.00
date=Mar 19 2011
time=14:35:18
hwid=1
hwrv=1
sck=24027428.5714285
ick=3003428.5714285625
```

Parsing the Stream File

It is recommended to store the meaningful information in arrays of structures (Flux, Index, and Info) that can be queried by the target application. The arrays can be allocated from the memory pool and released at the end of the program however the memory management of the different arrays is not described here.

Parsing is driven by the [Block Header](#) that defines the nature and length of the Blocks.

All the blocks are decoded in a loop that will scan the complete *Stream File* until an EOF block is found. Each *Block* is processed in three steps:

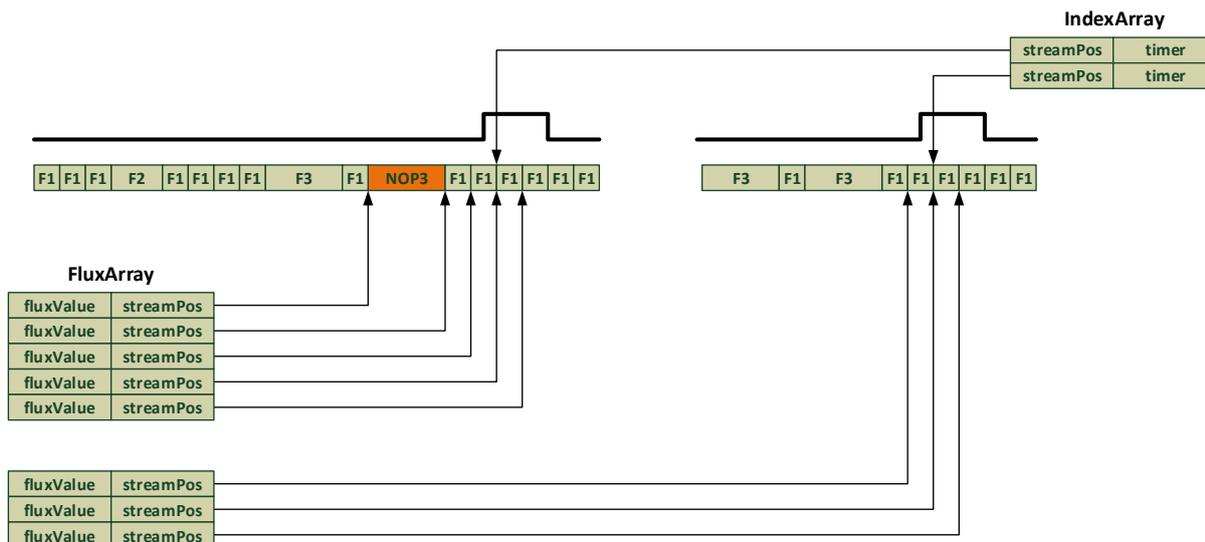
- We first compute the length of the *Block* based on the header type. This information is used to move the pointer to the next block:
 - For Flux1, Nop1, and Ovl16 blocks the length is one
 - For Flux2, Nop2 the length is two
 - For Flux3, Nop3 the length is three
 - For OOB Block the length is equal to the length of the OOB Header Block (4 bytes) plus the length of the OOB Data Block given in the **Size** field (see [OOB Blocks](#)). The only exception is the EOF block where the size is not meaningful.
- We then compute the actual value of the flux reversal when the block is of type Flux1, or Flux2, or Flux3, or Ovl16.
- The final step is to actually process the block:
 - If the data block is of type Flux1, Flux2, or Flux3 we create a new entry in a Flux array and we store the **Flux Value** as well as the **Stream Position**.
 - If the block is a StreamInfo block we use the **Stream Position** information to check that no bytes were lost during transmission. We can also use the **Transfer Time** for statistical analysis of the transfer speed.
 - If the block is an Index block. We create a new entry in an Index array and we store the **Stream Position**, the **Sample Counter** and **Index Clock** values.
 - If the block is a KFIInfo block we copy the information into a String.
 - If the block is a StreamEnd block we use the **Stream Position** information to check that no bytes were lost during transmission and we check the **Result Code** to verify that no errors were found during processing.
 - If the block is an EOF block we stop the parsing of the file.

When parsing of the stream file is finished we have all the data information in three arrays (Flux, Index, and KFIInfo) but we still need to analyze the Index information as explained in the next section.

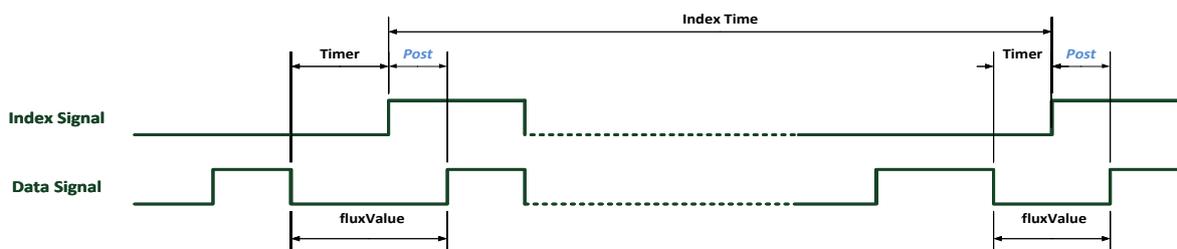
Analysis of Index Information

It is extremely important to be able to position the different **Index Signals** in respect with the flux reversals (and vice versa) and it is also important to measure the exact elapsed time between two Index Signals.

For that matter we need to perform some analysis on the stored data. The following pictures shows an example of a buffer containing Flux and NOP blocks as stored in the KryoFlux stream buffer (we could also have some Ovl16 blocks not shown here). As we have seen for each flux reversal we store the **value** as well as the **stream position** and for each Index Signal we store the **Sample Counter** and **Index Clock** values as well as the **stream position** of the next flux reversal when the index was detected as shown in the next picture.



If we look more precisely to the timing information close to two adjacent *Index Signals* we have something like this:



For each Index Signal

- The **Stream Position** points to the position of the **next** flux reversal in the stream buffer.
- The **Sample Counter** value indicates how far from the beginning of the **previous** flux reversal the index is detected.

So if we want to compute the **Index Time** (time between two indexes) we have to sum all the flux reversals values between the two Index Signals then subtract the **Timer** value of the first index signal and add the **Timer** value of the second index signal. Note that all these timing are given in number of sample clocks.

Another alternative to compute the **Index Time** is to take the **Index Clock** value of the second index and subtract the **Index Clock** value of the first index. This gives the number of **index clock** between the two index signals.

There are several marginal conditions for the Index signals that you should consider.

Sample Counter Overflows before Index

Some complexity arises if what was written last in the stream buffer is overflow. The stream and index decoder should take care of these cases; the stream decoder has to find the "real" stream position while decoding the data and the index decoder uses has to find the correct index referenced. This is somewhat tricky as at this point flux reversals are already decoded so they only ever are represented by one value, so the index decoder checks the range of stream positions elapsed between two cells.

Index pointing after last flux

The KryoFlux firmware always point to the next position to be written by the sampler. The stream decoder should add an extra empty flux at the end of the stream but this flux is not made part of the decoded stream at this point since we don't know if it happened or not, without decoding the index data. If the index analyzer detects that the index was pointing to a non-existent flux it has to “activate” the empty flux added above.

Index detected before any flux

There is another edge case when an index signal is detected but there is no previous flux reversal.

Terminology

- Flux Reversal: A flux reversal or flux transition under the floppy drive head. This is referred as a cell in the original SPS documentation.
- [ISB Blocks](#): Any Blocks that are not OSB blocks (i.e. with a Block Header different from **0x0D**). **In Stream Buffer** blocks contain flux reversal information placed in the stream buffer by the KryoFlux sampling process. This is referred as stream data in the original SPS documentation
- [OOB Blocks](#): **Out Of stream Buffer** blocks are used to transmit Index/hardware information or to help in decoding stream file. OOB Blocks have a Block Header equal to 0x0D. They contain extra information (not in the stream buffer) transferred to the host by the KryoFlux transfer process. This is referred as Out Of Band in original SPS documentation
- Stream Position: Position in the original KryoFlux stream buffer (i.e. the buffer prior to the insertion of the OOB blocks)

References

- [SPS KryoFlux Project Presentation](#)
- [SPS Stream Protocol](#)

Document History

- V1.1 – 08/11/2013 – Modified Sample/Index clock explanation. Size of OOB now clearly indicates 2 bytes length. HW information more detailed.
- V1.0 – 29/10/2011 – Added Index analysis marginal cases, KryoFlux HW information (firmware 2.0). Lots of minor fixes and terminology cleaning
- V0.3 – 10/08/2011 – Added Decoding Stream Files section with several graphics. Fixed terminology to be more consistent with SPS terminology. Update based on feedback from **István Fabián**.
- V0.2 – 08/08/2011 – Fixed errors, added information
- V0.1 – 24/07/2011 – Initial draft