KryoFlux
Documentation Fichier de Flux

Jean Louis-Guerin

Revision 1.1- 01/12/2013

Traduction v1.1a 07/01/2020

Gi@nts

Table des matierres

Presentation

Ce document fournit une *description* des ***fichiers Flux*** utilisés par le programme ***DTC*** (DiskTool Console) eventuellement connecté à la carte *KryoFlux*.
Un fichier de Flux est soit produit par le programme *DTC* en mode **lecture (imaging mode)** ou absorber par le programme *DTC* en mode **écriture (backup mode).**
Ce document fournit également de l'aide aux programmeurs sur le *Decodage* *des fichiers Flux*.
Il est basé sur mon interprétation des documentations publiées par *Software Preservation Society* et *KryoFlux Product & Services Limited* (voir *references*) mais il comprend également des informations basées sur des expérimentations. À noter que certaines sections de ce texte sont tirées, presque directement, des documents originaux référencés par *SPS*.

Je tiens à remercier **István Fabián** de SPS qui m'a fourni beaucoup d'informations détaillées sur le fonctionnement de la carte *KryoFlux* et le code pour décoder un fichier de flux.

> *Note: Les utilisateurs standards de la carte KryoFlux ne devraient pas être concernés par les informations présentées ici qui sont principalement d'intérêt pour les programmeurs qui veulent écrire des outils autour des fichiers de flux*

Les data du fichier de flux sont stockées en binaire et ne peuvent donc pas être directement affichées ou éditées avec un éditeur de texte. Il est intéressant de noter que le contenu du fichier de flux n'a pas été conçu comme un format de fichier en soi, car il s'agit en fait d'une copie exacte du **protocole de flux** d'octets utilisé entre la carte KryoFlux et le système hôte lors d'une communication sur un lien USB.

Le protocole de flux d'octets est **optimisé** pour un minimum de communication et d'utilisation CPU.
Le SoC de la KryoFlux est doté d'un matériel dédié à l'échantillonnage, à la communication, etc.
Chacun d'eux est desservi par des interruptions pour un fonctionnement asynchrone et efficace.
Le firmware comporte donc de nombreux sous-systèmes asynchrones communiquant avec le matériel, avec des exigences, des contraintes et un consommation CPU très spécifiques.
Si le firmware ne parvient pas à les desservir, le système devient inutilisable.
Il faut savoir qu'une seule piste peut avoir environ 500 000 inversions de flux par seconde pour un disque Haute Densité qui est transmis à l'ordinateur hôte via une liaison USB avec une bande passante limitée.

Par conséquent, le fichier/protocole de flux est défini pour une efficacité du transfert et du traitement, forcément, une certaine complexité en découle lors de son décodage.

Notez que les fichiers Flux sont spécifiques au matériel (au dispositif KryoFlux) et ne sont donc pas destinés à une préservation à long terme.

> *AVERTISSEMENT : Le format de fichier Flux peut être modifié selon les besoins. La version décrite dans le présent document a été réalisée à l'aide de la version 2.2 de DTC.*

Créer une image disque avec la carte KryoFlux

Afin de tout capturer sur une disquette, il est nécessaire d'échantillonner tous les '***flux reversals'*** entre plusieurs ***signaux d'index***. La carte KryoFlux commence à échantillonner les données avant le premier ***signal d'index*** et peut échantillonner les données après le dernier ***signal d'index***. Ceci est important pour s'assurer que toutes les informations présentes sur une disquette sont capturées. En dehors des ***signaux d'index***, il est impossible d'effectuer un décodage significatif.

Pour diverses raisons, en particulier pour les jeux, de multiples révolutions de données doivent être capturées en un flux constant. Cela signifie qu'un fichier de flux devrait généralement contenir plus de ***deux signaux d'index***. Afin d'analyser correctement les protections utilisées sur les disquettes, il est généralement nécessaire d'enregistrer plusieurs révolutions. Pour que SPS puisse produire correctement des fichiers IPF avec l'analyseur CTA, le minimum requis est de cinq révolutions (6 index).

KryoFlux Clocks & Counters

La carte KryoFlux Device est pilotée à partir d'un ***'Master Clock'*** (**mck**).
De ce 'master clock' sont derives deux horloges synchrones :

- Le ***'Sample Clock'*** (**sck**) utilisé par le ***Sample Counter*** pour échantillonner le 'Flux reversals'.
- Le ***'Index Clock'*** (**ick**) utilisé par l'***Index Counter*** pour échantillonner les Signaux d'Index.

Les fréquences d'horloge sont définies par le H/W de la KryoFlux, et peuvent être interrogées à l'aide d'une commande de périphérique.
Les ***valeurs par défaut*** sont stockées en virgule flottante de 64 bits:

Abbreviation	Nom	Valeur d'Horloge
mck	Master Clock	((18432000*73)/14)/2=48054857,14285714
sck	Sample Clock	mck / 2 = 24027428,57142857
ick	Index Clock	mck / 16 = 3003428,571428571

Depuis le firmware 2.0+ de la carte KryoFlux, la carte transmet des informations matérielles qui comprennent les valeurs de ces deux horloges (voir section KryoFlux Hardware Information).
Il est recommandé d'utiliser ces valeurs car le Hardware de la KryoFlux peut éventuellement changer ces fréquences dans le futur.

Sample Counter

Le ***Comteur d'échantillon*** sert à mesurer le temps écoulé entre deux inversions de flux, ou entre une inversion de flux et un signal d'index. Ce compteur a une largeur de 16 bits et les éventuels débordements sont enregistrés. Ce compteur est remis à zéro après chaque enregistrement d'inversion de flux.

Index Counter

Le ***Compteur d'Index*** est un 'compteur libre' (non réinitialisé). La valeur de ce compteur est enregistrée à chaque fois qu'un signal d'index est détecté.

Data Format

Les 'Data' dans un ***fichier de Flux*** sont alignées par octet pour une efficacité de traitement. Cela signifie qu'aucune information n'est codée au niveau du bit et qu'il n'est donc pas nécessaire de décomposer un octet en bits pour l'interpréter.

Les 'Data' stockées dans des mots de 16 ou 32 bits utilisent le systeme '***little-endian***' (l'octet le moins significatif en premier, et l'octet le plus significatif en dernier).
Ceci ne s'applique pas aux **Flux Blocks** qui utilisent un codage spécifique.

Description des fichiers de Flux

Les 'Data' dans un ***fichier de Flux*** sont organisés en ***Blocks*** qui ont une longueur variable allant de un à plusieurs octets. Le premier octet d'un '***bloc'*** de fichier de flux est appelé le ***Block Header***. Il indique comment interpréter le '***Bloc'***.
Un fichier de Flux contient deux types de blocs :

- Les blocs ***ISB*** (In Stream Buffer) sont utilisés pour communiquer la valeur temporelle des inversions de flux échantillonnées..
- Les blocs ***OOB*** (Out Of stream Buffer) sont utilisés pour aider à l'interprétation/vérification du fichier de flux ainsi que pour transmettre d'autres informations critiques comme la synchronisation des signaux d'index ou les informations matérielles de la carte KryoFlux.

Pour une explication des terminologies ISB/OOB veuillez vous référer à la section KryoFlux Device Behavior

L'information la plus importante à extraire d'un fichier de flux est :

- Timing des '**Flux Reversals'**: Toutes les inversions de flux de données détectées par la carte KryoFlux sont stockées dans les blocs *ISB*.

- Timing des '**Index Signals'**: Tous les signaux d'index détectés par la carte KryoFlux sont transmis dans les blocs speciaux OOB : Les Blocs d'*Index*. Les informations d'index fournies permettent de calculer précisément l'***index Time***' (temps entre deux signaux d'index) ainsi que de trouver l'***Index Position*** en référence aux inversions de flux de données actuelles.

Block Header

L'interprétation des informations contenues dans un ***bloc*** de données dépend du ***Block Header***. Cet en-tête peut prendre les valeurs suivantes (triées par ordre croissant) :

Header	Nom	Longueur	Description
0x00-0x07	Flux2	2	Flux block: Compteur de 'flux reversal' codé sur 2 Octets
0x08	Nop1	1	NOP block: Continu le décodage à la position courante + 1
0x09	Nop2	2	NOP block: Continu le décodage à la position courante + 2
0x0A	Nop3	3	NOP block: Continu le décodage à la position courante + 3
0x0B	Ovl16	1	Flux block: Prochain compteur de ' flux reversal' devra être augmenté par 0x10000.
0x0C	Flux3	3	Flux block: Compteur 'flux reversal' codé en 3 Octets
0x0D	OOB	variable	Premier octet d'un bloc tampon 'Out Of Flux'
0x0E-0xFF	Flux1	1	Flux block: Compteur 'flux reversal' codé sur 1 Octet

ISB (In Stream Buffer) Blocks

Un ***Bloc ISB*** est soit un ***Bloc*** de ***Flux*** ou un ***Bloc NOP*** *(i.e. pas un* ***Bloc OOB*** *)*.

Flux blocks

Un Bloc de **Flux** est utilisé pour stocker la valeur du ***Sample Counter***. Cela correspond au nombre de ***Sample Clock Cycles*** (sck) entre deux inversions de flux.

Les valeurs absolues des temps d'inversion de flux peuvent être calculées en divisant la valeur du ***sample counter*** par le ***Sample Clock*** (sck):

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

Flux1 block

Ce bloc permet de stocker trés efficacement le timing échantillonné du ***Reversal Flux*** car il est codé sur un seul octet. Le *Block Header* a une valeur comprise entre **0x0E-0xFF**.

0x0E-0xFF

Dans ce cas, la valeur échantillonnée du ***Reversal Flux*** est directement égale à la valeur du *Block Header*. En pratique, vous constaterez que la plupart des valeurs du ***Reversal Flux*** se situent dans cette plage (0x0E-0x0FF) et cela contribue donc à un codage très efficace du fichier de flux.

```
FluxValue = Header_value;
```

Flux2 block

Ce bloc permet de stocker le timing du ***Reversal Flux.*** Il est codée sur deux Octets.
Le *Block Header* a une valeur comprise entre **0x00-0x07.**

0x00-0x07	Value1

Dans ce cas la valeur échantillonée du ***Reversal Flux*** est calculé comme suit :

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

Flux3 block

Ce bloc permet de stocker le timing d'un échantillon d'un ***Reversal Flux*** codé sur trois Octets.
Le *Block Header* a une valeur égale à **0x0C**.

0x0C	Value1	Value2

Dans ce cas la valeur échantillonée du ***Reversal Flux*** est calculé comme suit :

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

Ovl16 block

Ce bloc indique que le **prochain** *FluxBlock* aura une valeur supérieure à la valeur maximale d'un nombre 16bits (0xFFFF). Le *Block Header* a une valeur égale à 0x0B.

0x0B

Dans ce cas, la valeur **suivante** est incrémentée de 0x10000.

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

Ce bloc est inséré à chaque fois que le ***SampleCounter*** déborde. Il n'y a pas de limite au nombre de blocs Ovl16 présents dans un flux, et donc la valeur maximale pour un ***Reversal Flux*** est pratiquement illimitée, bien que le décodeur du logiciel hôte de la carte KryoFlux utilise une valeur de 32 bits. Les valeurs du ***Reversal Flux*** qui ne s'inscrivent pas dans les 16 bits sont assez inhabituelles, mais ont été trouvées dans des jeux qui tentent de duper l'AGC (Automatic Gain Control).

Optimisation du codage du Data Flux

Un bloc Flux2 peut être utilisé pour coder des données dans la plage 0x0000-0x07FF.
Mais en pratique, il est plus efficace d'utiliser un bloc Flux1 (un seul octet) pour coder des données dans la plage **0x000E- 0x00FF**. Par conséquent, le Flux2 est uniquement utilisé pour encoder des données dans la plage 0x0000-0x000D ou des données dans la plage 0x0100-0x07FF.
Pour des raisons similaires (meilleur rendement), un bloc Flux3 n'est utilisé que pour coder des données dans la plage 0x0800-0xFFFF.

Si la valeur du ***Reversal Flux*** à transmettre est supérieure à 0xFFFF alors un ou plusieurs bloc(s) ovl16 est (sont) utilisé(s) pour ajouter 0x10000 à la prochaine valeur du ***Reversal Flux.***

NOP Blocks

Un bloc NOP (No-operation) est utilisé pour sauter un ou plusieurs octets dans le tampon de flux. Cela permet au firmware de créer des données dans son tampon circulaire sans avoir besoin de briser une seule séquence de code lorsque le remplissage du tampon circulaire se termine. Un bloc NOP commence avec un ***Block Header*** dans les plages **0x08-0x0A.**

NOP1 block

Le Bloc NOP1 est utilisé pour sauter un octet dans le tampon. Le ***Block Header*** est égal à **0x08**.

0x08

Saute juste cet octet pendant le décodage.

NOP2 block

Le Bloc NOP2 est utilisé pour sauter deux octets dans le tampon. Le ***Block Header*** est égal à **0x09**.

0x09	0xXX

Saute juste ces deux octets pendant le décodage.

NOP3 block

Le Bloc NOP3 est utilisé pour sauter trois octets dans le tampon. Le ***Block Header*** est égal à **0x0A**.

0x0A	0xXX	0xYY

Saute juste ces trois octets pendant le décodage.

Les blocs OOB (Out Of stream Buffer)

Un bloc ***OOB Block*** est soit utilisé pour aider à l'interprétation/vérification du fichier de flux, soit il contient des informations spécifiques (signal d'index, info KryoFlux HW). Notez que les blocs OOB sont envoyés de façon complètement **asynchrone** par rapport aux blocs ***ISB*** (voir section KryoFlux Device Behavior).

Un bloc ***OOB Block*** est composé d'un ***OOB Header Block*** (toujours de 4 octets) suivi d'un bloc ***OOB Data*** optionnel.

Le bloc *OOB Header* contient trois champs :

- Le premier champ (un octet) contient le ***Block Header*** et est toujours égal à **0x0D**.
- Le second champ (un octet) contient le ***Type*** de l'OOB (voir ci-dessous).
- Le troisième champ (deux octets) indique la **taille** du bloc ***OOB Data***.

0x0D	Type	Size	OOB Data Block

Le prochain bloc de donnée optionnel d'***OOB Data*** contient des informations spécifiques à chaque type de bloc OOB. Le tableau suivant énumère les différents types de bloc ***OOB***

Type	Nom	Signification
0x00	Invalid	OOB invalide
0x01	StreamInfo	Information Flux (multiples par track)
0x02	Index	Index signal data
0x03	StreamEnd	Plus de flux à transférer (un par track)
0x04	KFInfo	Information HW venant de la carte KryoFlux
0x0D	EOF	Fin du fichier (plus de données traiter)

Invalid block

Ce n'est pas clair quand ce bloc OOB est utilisé mais il indique clairement et définitivement un problème 😵.

0x0D	0x00	0x0000

Un bloc invalide contient les champs suivants :

- Type = 0x00
- Size = 0x0000?

StreamInfo block

Un bloc StreamInfo fournis des information sur le déroulement du transfert de données. Il est envoyé chaque fois que la communication et le niveau de consommation CPU de la KryoFlux le permettent, naturellement la commande des blocs StreamInfo est garantie. Il est possible d'avoir plusieurs blocs StreamInfo à la fois. Il sert principalement à vérifier qu'aucun octet n'a été perdu lors de la transmission, mais il peut également être utilisé pour calculer la vitesse de transfert de la liaison USB entre l'hôte et la carte KryoFlux.

0x0D	0x01	0x0008	Stream Position	Transfer Time

Un bloc StreamInfo contient les champs suivants :

- *Type* = 0x01
- *Size* = 0x0008 (Taille du data block suivant)
- *Stream Position* (4 octets) indique la position (en nombre d'octets) de l'OOB Header dans la mémoire tampon du flux.
- *Transfer Time* (4 octets) donne le temps écoulé (en millisecondes) depuis le dernier bloc StreamInfo. Il est donc possible de calculer la vitesse de transfert entre l'hôte et la carte ainsi que le distortion du transfert.

Index block

Ce bloc est utilisé pour fournir des informations de timing sur un Index detecté.

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

Un bloc d'Index contient les champs suivants :

- *Type* = 0x02
- *Size* = 0x000C (12 decimal – taille du prochain Data block))
- *Stream Position* field (4 octets) indique la position (en nombre d'octets) dans le buffer du Flux du **prochain** flux reversal juste après que l'index ait été détecté.

- *SampleCounter* (4 octets) donne la valeur du ***SampleCounter*** quand l'index a été détecté . Ceci est utilisé pour obtenir un timing précis de l'Index par rapport au flux reversal précédent. Ce timing est donnée en nombre de ***Sample Clock*** (**sck**).
  Notez qu'il est possible qu'un ou plusieurs débordements du compteur d'échantillons se produisent avant que l'index ne soit détecté.

- *IndexCounter* (4 octets) stocke la valeur de l'***IndexCounter*** quand l'Index est detecté. La valeur est donnée en nombre de d'***Index Clock*** (ick). Pour obtenir des valeurs de timing absolues à partir des valeurs du Index counter, vous devez diviser ces nombres par l'Index clock (ick)

Pour plus d'information sur l'interprétation du timing, voir les sections Index Timing Consideration et Analysis of Index Information.

StreamEnd block

Un bloc StreamEnd block indique que tous les Flux blocks ont été transmis.Il fournit également un code d'état Kryoflux qui indique si le transfert de flux a été effectué correctement par le hardware.

0x0D	0x03	0x0008	Stream Position	Result Code

Un bloc de StreamEnd contient les champs suivants :

- *Type* = 0x03
- *Size* = 0x0008 (taille des data block)
- *Stream Position* field (4 octets) indique la position (en nombre d'octet) du bloc OOB Header dans le fichier de flux.
- *Hardware Status Code* (4 octets) renvoie une valeur telle que définie ci-dessous. Une valeur de 0 indique que le transfert de flux en continu a réussi ; toute autre valeur indique divers problèmes.

Code d'état du Hardware :

Valeur	Nom	Signification
0×00	Ok	Succès du transfert (ne signifie pas que les données sont bonnes, mais que le transfert de flux a réussi.)
0×01	Buffer	Transmission des données vers l'hôte n'ont pas pu suivre le rythme de lecture du disque.
0×02	No Index	Aucun signal d'index détécté

KFInfo block

Un bloc KFInfo est utilisé pour transmettre des informations de la carte KryoFlux à l'hôte.

0x0D	0x04	Size	Info Data (ASCII)

Un bloc KFInfo contient les champs suivants :

- *Type* = 0x04
- *Size* (2 octets) = nombre d'octets du bloc de données KFInfo (y compris le zéro de terminaison)
- *Info Data* – une chaîne d'information ASCII à terminaison nulle.

Plus de détails sur les informations hardwares transmises sont disponibles dans la section KryoFlux Hardware Information

EOF block

Un bloc EOF est utilisé pour indiquer la fin du fichier de flux. Aucun traitement ne doit être effectué au-delà de ce bloc.

0x0D	0x0D	0x0D0D

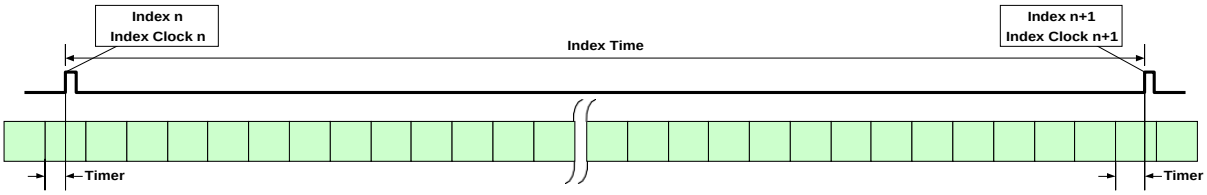
Un bloc EOF contient les champs suivants :

- *Type* = 0x0D
- *Size* = 0x0D0D (non significatif)

Consideration du timing d'Index

Les valeurs de timing du ***Flux Reversal*** enregistrées dans un fichier de flux n'ont de sens que lorsque les positions des ***Index Signals*** sont connues. Une fois que toutes les données du fichier de flux ont été traitées, plusieurs calculs sont nécessaires sur les Index data afin de déterminer :

- ***Index Position***: La position exactes ou un Index Signal s'est produit en références aux Flux Reversals. Il peut être déterminé pendant le décodage en stockant la position de toutes les Reversals flux et la position où chaque signal d'index se produit.
- ***Index Time***: C'est le temps nécessaire pour un tour complet du disque. Il est égal au nombre de cycles d'horloge d'index depuis que le dernier index s'est produit.
  Il peut également être calculé en additionnant toutes les flux reversal que nous avons enregistrées depuis l'index précédent, en ajoutant la valeur du **Sample Counter** à laquelle l'index a été détecté (voir le champ *SampleCounter* dans l'Index Block) et en soustrayant la valeur du **Sample Counter** de l'index précédent.
  Les détails du calcul sont expliqués dans la section Analysis of Index Information.



L'***Index Time*** permet de calculer la valeur exacte de la vitesse de rotation FD pour une révolution . Par exemple, pour un lecteur qui tourne à 300 tr/min, le temps entre deux index doit être de 200 ms. Nous savons par expérience que la valeur réelle diffère et il est donc important de surveiller le RPM pour chaque tour échantillonné. Notez que jusqu'au premier index, un temps d'index ne peut pas être généré car il s'agira toujours d'une révolution partielle.

L'***Index Position*** est également importante, car c'est le seul marqueur sur un disque qui peut être utilisé pour aligner parfaitement les données en écriture, ou pour décider de la position exacte des données en lecture.

RPM Interpolation

Pour augmenter la fiabilité, le logiciel de décodage peut effectuer une interpolation des RPM lors de la conversion du timing en valeurs absolues. Si le régime d'un index est significativement différent de celui de l'index suivant, il se peut que le lecteur de disque effectuant la lecture ne soit pas fiable et que la vitesse du lecteur d'un index à l'autre ne soit pas constante. Mais même si le RPM est très stable, il peut avoir été réglé incorrectement, comme par exemple 301 RPM au lieu de 300 RPM. Cela affecterait toutes les flux reversals sur la piste. Comme il y a des centaines de milliers d'échantillons, les différences finiront par s'additionner. Nous pouvons modérer ces variations en convertissant chaque valeur de flux en utilisant une valeur interpolée. Différents algorithmes d'interpolation sont possibles pour faire cela. Par exemple, le temps mesuré pour les inversions de flux peut être corrigé en utilisant un facteur qui prend en compte la vitesse réelle de l'entraînement (par exemple 301 tr/min), en fonction de la vitesse attendue de l'entraînement (par exemple 300 tr/min) avec la formule suivante :

```
CorrectedValue = OriginalValue * Expected_RPM/Actual_RPM;
```

Décodage des fichiers de flux

Il est recommandé de décoder un fichier de flux KryoFlux en deux passes :

- La première passe est utilisée pour analyser le fichier de flux afin de récupérer et de stocker toutes les informations importantes (timing et positionnement des flux/index).
- La deuxième passe est utilisée pour analyser les données stockées afin de calculer le positionnement exact des signaux d'index par rapport aux inversions de flux ainsi que les temps d'index.

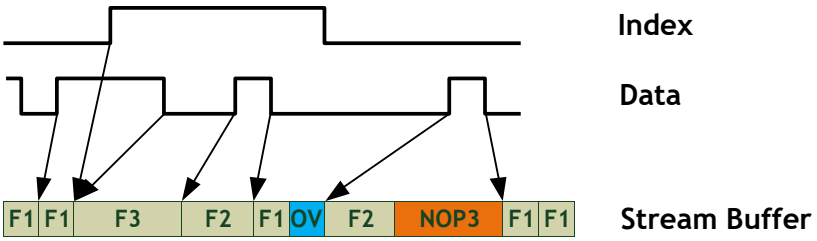
Il est également recommandé de vérifier si les informations hardware de la carte KryoFlux concernant SCK et ICK ont été transmises. Si c'est le cas, ces valeurs devraient être utilisées plutôt que les valeurs d'horloge par défaut.

Comportement de la carte KryoFlux

Afin de traiter correctement les données stockées dans un fichier de flux, il est utile d'avoir une compréhension de base de la façon dont la carte KryoFlux fonctionne. (merci à **IstvánFabián**).

Lors de la création de l'image à partir des signaux provenant d'un lecteur de disquette qui lit une disquette, il y a des processus principaux qui s'exécutent indépendamment dans la carte KryoFlux

- Le 1er processus est appelé **sampling process**. Comme son nom l'indique, il est chargé de capturer les données du lecteur de disquette et de stocker ces informations dans un tampon appelé ***Stream Buffer***. Ce tampon ne contient donc que les blocs de flux. (y compris les blocs Ovl16) et éventuellement quelques blocs NOP (considérés comme des données sans valeur).



Dans cet exemple hypothétique (non strictement réaliste) on peut voir que chaque valeur d'inversion de flux est stockée par la carte KryoFlux dans le ***Stream Buffer*** en tant que blocs Flux1 ou Flux2 ou Flux3 (voir section Flux Data Encoding). Cette valeur correspond à la valeur du Sample Counter au moment de l'inversion de flux et une fois enregistrée, le compteur est remis à **zéro**.
Chaque fois que le Sample Counter déborde, un bloc Ovl16 est stocké dans la *Stream buffer*.
Le firmware peut aussi ajouter des blocs NOP dans le tampon du flux le cas échéant (voir les blocs NOP). Lorsqu'un signal d'index est détecté, l'information n'est pas placée dans le Stream Buffer mais on enregistré la position de la prochaine inversion de flux dans le tampon de flux ainsi que la valeur du Sample Counter (temps depuis l'inversion de flux précédente) et de l'Index Counter.

- Le deuxième processus est appelé **processus de transfert**. Il est chargé de transférer les données de la carte KryoFlux à l'hôte par le biais de la liaison USB. La première priorité du processus de transfert : transmettre les données stockées dans la Steam Buffer. Mais, lorsque la communication et le temps CPU KryoFlux le permettent, ce processus transmet également des " informations supplémentaires " : les blocs **OOB**.
Ils sont utilisés soit pour transférer des informations d'index, soit pour aider au décodage du transfert. Ces blocs ne font pas partie du Stream Buffer et sont "insérés à la volée", par le processus de transfert, entre les blocs ISB à des moments imprévisibles. Cela implique que les informations dans les blocs OOB sont complètement asynchrones par rapport aux informations dans le bloc ISB. Par exemple, il est possible de transmettre des informations sur un Index qui se réfèrent à une inversion de flux **non** encore transmise !

L'échantillonnage peut être arrêté après un certain nombre de signaux d'index de manière automatique ou programmée par une commande ; DTC peut utiliser les deux. Actuellement, la diffusion en continu est demandée pour s'arrêter après un nombre spécifié d'Index, mais si le DTC détecte certaines erreurs, il peut envoyer un ordre d'arrêt à tout moment, ce qui arrêterait la diffusion en continu dès que possible (i.e. à un endroit aléatoire sur la piste). Même si l'échantillonnage doit être arrêté à un signal d'index, puisqu'il est indépendant du streaming, il peut ou non s'arrêter immédiatement, tout dépend du hasard.

Le processus de transfert renvoie toujours toutes les données qui ont été échantillonnées avant de signaler la fin du transfert à l'hôte. En d'autres termes, il peut y avoir un ou plusieurs échantillons après le dernier signal d'index (si le mode d'arrêt d'index est utilisé) ou il peut n'y en avoir aucun.

Information sur le Hardware KryoFlux

A partir du firmware 2.0 de la carte KryoFlux, la carte transmet des informations dans un ou plusieurs blocs KFInfo. La plupart des données transmises sont des informations sur la version du micrologiciel, du matériel, etc.

Plusieurs chaînes (habituellement deux) peuvent être transmises de la carte KryoFlux à l'hôte. Chaque chaîne est terminée par un zéro.
Le champ de taille dans le KFInfo indique la longueur de la chaîne, l'espace de fin y compris.

Les informations à l'intérieur des chaînes sont transmises sous forme de paires “name” “value” séparées par une virgule (","). Par example “host_date=2012.01.22, host_time=17:44:47”.

Parmi les informations transmises, deux chaînes sont particulièrement importantes:
Le sample clock (**sck**) et l’ index clock (**ick**). Vous devriez utiliser ces valeurs au lieu des valeurs par défaut spécifiées dans la section KryoFlux Clocks & Counters .

Voici un exemple des informations transmises :

```
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
```

Analyse syntaxique du Fichier de Flux

Il est recommandé de stocker les informations significatives dans des tableaux de structures (Flux, Index, et Info) qui peuvent être interrogés par l'application cible.Les tableaux peuvent être alloués à partir du pool de mémoire et libérés à la fin du programme ; cependant, la gestion de la mémoire des différents tableaux n'est pas décrite ici. L'analyse est pilotée par le ***Block Header*** définit la nature et la longueur des blocs.

Tous les blocs sont décodés dans une boucle qui va balayer complètement le ***fichier de flux*** jusqu'à ce qu'un bloc EOF soit trouvé. Chaque *bloc* est traité en trois étapes :

- Nous calculons d'abord la longueur du bloc en fonction du type d'en-tête.
  Cette information est utilisée pour déplacer le pointeur vers le bloc suivant :
  - Pour les blocs Flux1, Nop1, et Ovl16 la longueur est de 1
  - Pour les blocs Flux2, Nop2 la longueur est de 2
  - Pour les blocs Flux3, Nop3 la longueur est de 3
  - Pour les blocs OO, la longueur est égale à la longueur du bloc OOB Header (4 octets) plus la longueur du bloc OOB Data indiqué dans le champ ***Size***. (voir OOB Blocks). La seule exception est le bloc EOF dont la taille n'est pas significative.
- Nous calculons alors la valeur réelle de l'inversion de flux lorsque le bloc est de type Flux1, ou Flux2, ou Flux3, ou Ovl16.
- La dernière étape consiste à traiter réellement le bloc :
  - Si le bloc de données est de type Flux1, Flux2 ou Flux3, nous créons une nouvelle entrée dans un tableau de flux et nous enregistrons la ***valeur du flux*** ainsi que la ***position du flux***.
  - Si le bloc est un bloc StreamInfo, nous utilisons les informations ***Stream Position*** pour vérifier qu'aucun octet n'a été perdu pendant la transmission. Nous pouvons également utiliser le ***TransferTime*** pour l'analyse statistique de la vitesse de transfert.
  - Si le bloc est un bloc d'index. Nous créons une nouvelle entrée dans un tableau d'index et nous stockons la ***StreamPosition***, les ***SampleCounter*** et l'***IndexClock***.
  - Si le bloc est un bloc KFInfo, nous copions les informations dans un Chaîne.
  - Si le bloc est un bloc StreamEnd, nous utilisons les informations de Stream Position pour vérifier qu'aucun octet n'a été perdu pendant la transmission et nous vérifions le ***Result Code*** pour nous assurer qu'aucune erreur n'a été trouvée pendant le traitement.
  - Si le bloc est un bloc EOF, nous arrêtons l'analyse du fichier.

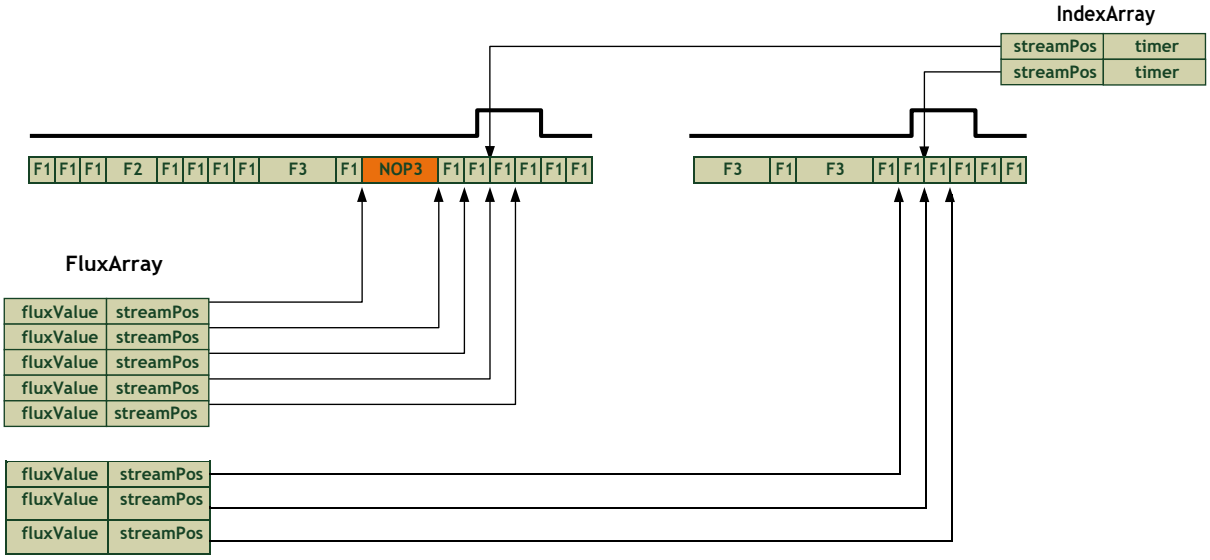
Lorsque l'analyse du fichier de flux est terminée, nous avons toutes les informations de données dans trois tableaux (Flux, Index, et KFInfo) mais nous devons encore analyser les informations d'Index comme expliqué dans la section suivante.

Analyse de l'information 'Index'

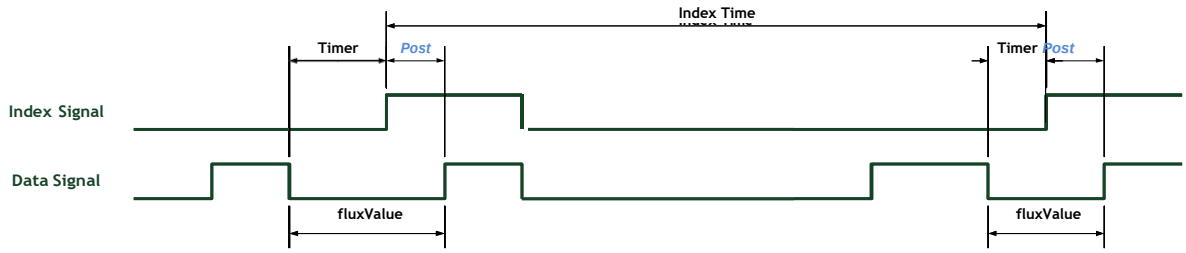
Il est extrêmement important de pouvoir positionner les différents signaux d'index par rapport aux inversions de flux (et vice versa) et il est également important de mesurer le temps exact écoulé entre deux signaux d'index.

Pour cela, nous devons effectuer des analyses sur les données stockées. Les images suivantes montrent un exemple de tampon contenant des blocs Flux et NOP tels que stockés dans le tampon de flux de la KryoFlux (nous pourrions également avoir quelques blocs Ovl16 non montrés ici).

Comme nous l'avons vu, pour chaque inversion de flux, nous stockons la valeur ainsi que la **stream position** et pour chaque signal d'index, nous stockons les valeurs **Sample Counter** et ***d'Index Clock*** ainsi que la **stream position** de l'inversion de flux suivante lorsque l'index a été détecté, comme indiqué dans l'image suivante.



Si nous regardons plus précisément les informations de synchronisation à proximité de deux signaux d'index adjacents, nous avons quelque chose comme ceci:



Pour chaque signal d'index

- La **Stream Position** indique la position de la **prochaine** inversion de flux dans la stream buffer.
- La valeur du **Sample Counter** indique à quelle distance du début du **précédent** flux reversal l'Index est détecté.

Donc, si nous voulons calculer l'**Index Time** (temps entre deux index), nous devons additionner toutes les valeurs d'inversion de flux entre les deux signaux d'index, puis soustraire la valeur du **Timer** du premier signal d'index et ajouter la valeur du **Timer** du second signal d'index.
Notez que tous ces timing sont donnés en nombre sample clocks.

Une autre solution pour calculer **Index Time** consiste à prendre la valeur de l'Index Clock du second Index et à soustraire la valeur de l'Index Clock du premier Index. Cela donne le nombre d'**Index clock** entre les deux signaux d'index.

Il y a plusieurs conditions marginales pour les signaux d'Index que vous devriez considérer.

Débordement de Sample Counter avant un Index

Une certaine complexité survient si ce qui a été écrit en dernier dans le tampon du flux est un débordement. Le décodeur de flux et d'index doit prendre en charge les cas suivants ; le décodeur de flux doit trouver la position "réelle" du flux pendant le décodage des données et le décodeur d'index utilisé doit trouver l'index correct référencé. Cela est quelque peu délicat car à ce stade, les inversions de flux sont déjà décodées et ne sont donc jamais représentées par une seule valeur, de sorte que le décodeur d'index vérifie la plage des positions de flux écoulées entre deux cellules.

Pointage de l'Index après le dernier Flux

Le Firmware de la KryoFlux pointe toujours vers la prochaine position qui doit être écrite par l'échantillonneur. Le décodeur de flux devrait ajouter un flux vide supplémentaire à la fin du flux mais ce flux ne fait pas partie du flux décodé à ce moment puisque nous ne savons pas si cela s'est produit ou non, sans décoder les données d'index. Si l'analyseur d'index détecte que l'index pointait vers un flux inexistant, il doit "activer" le flux vide ajouté ci-dessus.

Index détecté avant n'inporte quel Flux

Un autre cas existe : Lorsqu'un signal d'index est détecté mais qu'il n'y a pas d'inversion de flux antérieure.

Terminologie

- Flux Reversal : Une inversion de flux ou Transition de flux sous la tête du lecteur de disquette. Dans la documentation originale SPS , on parle de cellule.
- ISB Blocks: Tous les blocs qui ne sont pas des blocs OSB (i.e avec un Bloc Header différent de ***0x0D***). Les blocs ***In Stream Buffer*** contiennent des informations d'inversion de flux placées dans le tampon de flux par le processus d'échantillonnage KryoFlux. Ces données sont appelées Stream Data dans la documentation originale de SPS.
- OOB Blocks: Les blocs ***Out Of stream Buffer*** sont utilisés pour transmettre des informations d'index/matériel ou pour aider au décodage du fichier de flux.Les blocs OOB ont un Bloc Header égal à 0x0D. Ils contiennent des informations supplémentaires (pas dans la Stream Buffer) transférées à l'hôte par le processus de transfert KryoFlux. On parle de " Out Of Band " dans la documentation originale de SPS.
- Stream Position: Position dans la Stream Buffer originale du flux KryoFlux (i.e la mémoire tampon avant l'insertion des blocs OOB)

References

- SPS KryoFlux Project Presentation
- SPS Stream Protocol

Historiques des Documents

- V1.1a – 07/01/2020 – French Traduction by Gi@nts
- 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