1 TABLE DES MATIERES

2 DES FAUX BITS DANS LE CHILI CON CARNE OU L'HISTOIRE MOUVEMENTEE D'UNE RESTAURATION DE FICHIER DAI

Voici l'histoire complète de la **restauration d'un programme DAI** présent sur une cassette audio « **DaiNamic** » enregistrée il y a une quarantaine d'années.

Une telle restauration se traite pratiquement comme une enquête policière. Je dois pourtant vous avouer que j'ai passé beaucoup plus de temps que si j'avais d'abord réfléchi posément comme raconté ci-dessous… Que voulez-vous, on a beau le savoir, on veut toujours aller plus vite en tentant ceci ou cela au petit bonheur la chance… Mais à ma décharge, je dois aussi avouer que la méthode aléatoire fonctionne bien de temps en temps…

Quoiqu'il en soit, mon degré de contentement augmente de manière exponentielle lorsque l'on traite cela « *à la Columbo* » !

Tout commence par mon programme **WAV2DAI** qui prend en entrée le code numérique d'un fichier « wav » représentant le signal analogique d'une bande magnétique, puis le transforme en code numérique spécifique au DAI. Dans environ 75% des cas, ce programme me délivre en sortie un beau fichier « .DAI » qu'il suffit ensuite de charger dans sa « petite » mémoire.

3 UNE AFFAIRE DE FAUX BITS

Pour le fichier « **20.wav** » la situation se présentait assez mal, les messages d'erreurs qui s'affichaient sur la sortie standard d'erreurs (**stderr**) étaient les suivants :

```
Echantillon N° : 2376125
        ***
        Pb dans fonction RBYTE
        Echantillon N° : 2376125
        b=-1 signifie que les deux états hauts précédents ont la
même durée c'est probablement dû à une corruption du signal un peu
avant...
        tenter une correction visuelle sous Audacity...
        Fonction RBLK : fin anormale Nb octets déjà lus : 105F
        Fonction RBLK : Checksum précédent : 83
        Fonction RBLK : Checksum en cours : B9
        Pb de lecture dans RBLK
Echantillon N° : 2376125
        Fin de fonction LireBloc, position N° 2376125
        ***
        Fonction TraiterFichierDAI : Problème lors de la lecture du
bloc N°4.
```

4 QUE FAIRE ?

4.1 D'ABORD REGARDER DANS LES LOGS DU PROGRAMME

Voici ce qu'on peut y lire :

Fin d'un bout de leader en : 153302

L'octet **55H** a été trouvé.

Type fichier : 30

Fin du header en : 154359 (début du dernier état bas de l'octet donnant le type de fichier.)

BLOC N°1 : lecture de la longueur du nom de fichier et de son checksum

LgNom = 14

CheckSum LgNom lu = 71

CheckSum LgNom calculé = 71 **(OK)**

N° d'échantillon après lecture du checksum de la longueur du nom de fichier : 155940

BLOC N°2 : lecture du nom de fichier et de son checksum

Début de fonction LireBloc, position N° 155940

Nb octets à lire = 14

Fonction RBLK : Nb octets lus : 14

Fonction LireBloc :

checksum calculé 87

checksum lu 87 **(OK)**

Fin de fonction LireBloc, position N° 167035

Nom fichier = **COMPUBOGGLE [2 fin]**

CheckSum lu = 87 **(OK)**

N° d'échantillon après lecture du nom de fichier : 167035

BLOC N°3 : lecture de la longueur de bloc et de son checksum

```
LgBlock = 1944

CheckSum lu = B5 (OK)

Echantillon N° : 168619
```

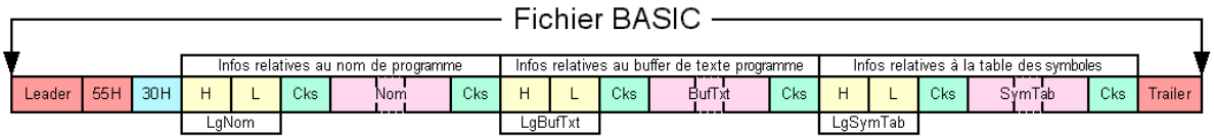
```
BLOC N°4 : lecture du bloc et de son checksum

Début de fonction LireBloc, position N° 168619

Nb octets à lire = 1944
```

Puis les logs s'arrêtent suite à la découverte du bit incohérent disposant de 2 niveaux hauts ayant la même longueur…

4.2 QUE FAIRE ENCORE ?

En analysant les données ci-dessus, on voit que c'est un fichier BASIC (type 30).

En consultant mon site WEB (http://bruno.vivien.pagesperso-orange.fr/DAI/doctechnique/FormatFichiersCassette/FormatK7index.htm) on peut visualiser la forme que doit avoir un tel type de fichier :

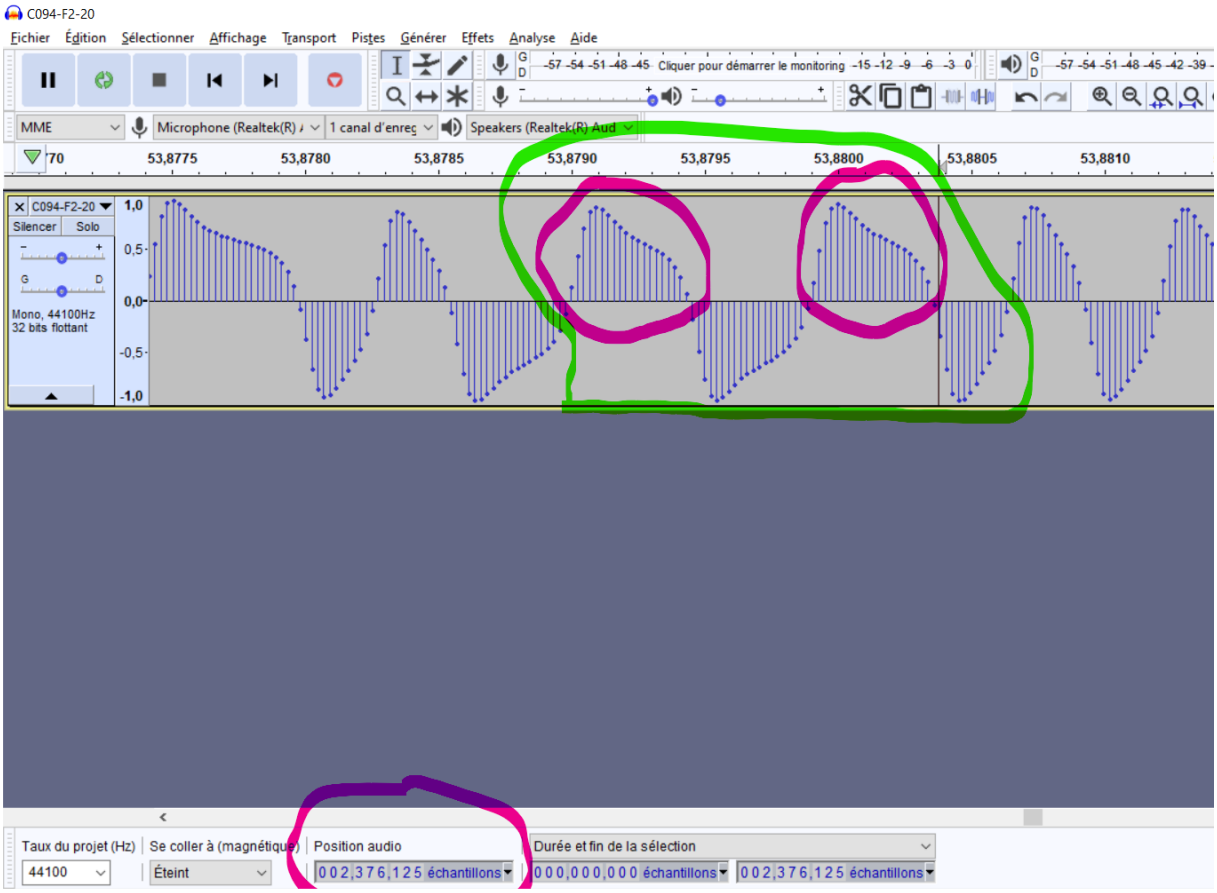


Le bloc N°4 dans lequel l'incohérence a été trouvée est le ***buffer text*** comportant le code semi-compilé du programme BASIC (on compte les blocs en comptant les checksums « Cks » en vert ci-dessus).

La situation est grave, ce n'est pas une erreur de nom de fichier, là j'aurais pû tricher en lui en donnant un autre nom. Ici nous sommes sur une erreur intervenue en plein dans le bloc représentant le corps du programme… Il va falloir essayer de comprendre ce qui s'est passé en utilisant tous les indices à notre disposition et essayer de voir si l'on peut reconstituer les informations détériorées…

4.3 QUESTIONNONS LE FICHIER WAV EN QUESTION

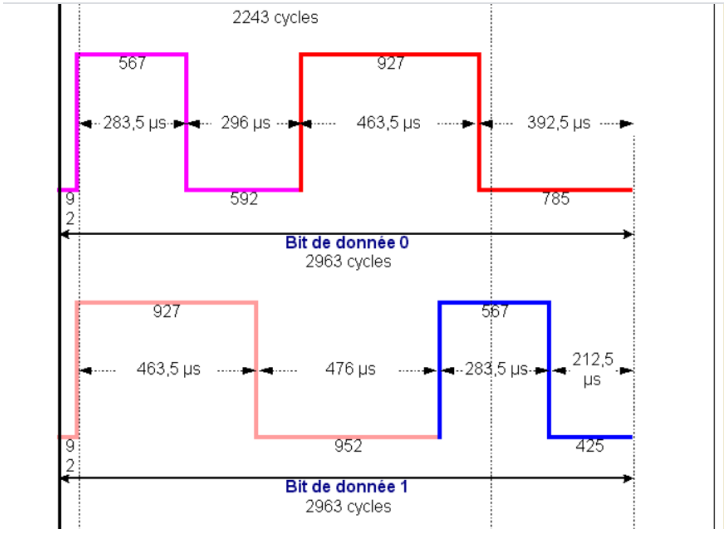
- ⇨ Ouvrons-le dans l'excellent logiciel Audacity.
- ⇨ J'enfile mon imperméable, je prends ma vielle 403 peugeot et je me rends sur les lieux : l'échantillon N° 2376125 signalé sur la sortie stderr, voici ce que l'on y voit :



Hou… Ce n'est pas beau ! Quand je vais raconter ça à ma femme elle n'en reviendra pas.

Le programme DAI2WAV s'est arrêté sur **l'échantillon 2 376 125** en nous indiquant qu'un bit dispose de deux états hauts ayant la même durée. Les deux yeux entourés de mauve ont la même taille et la tête entourée de vert représente ce que mon programme interprète comme étant un bit incohérent.

Or, un bit « zéro » ou un bit « un » doit respecter cette forme (http://bruno.vivien.pagesperso-orange.fr/DAI/doctechnique/AudioCassetteInterface/formatSignaux.htm):



On voit sur ce schéma :

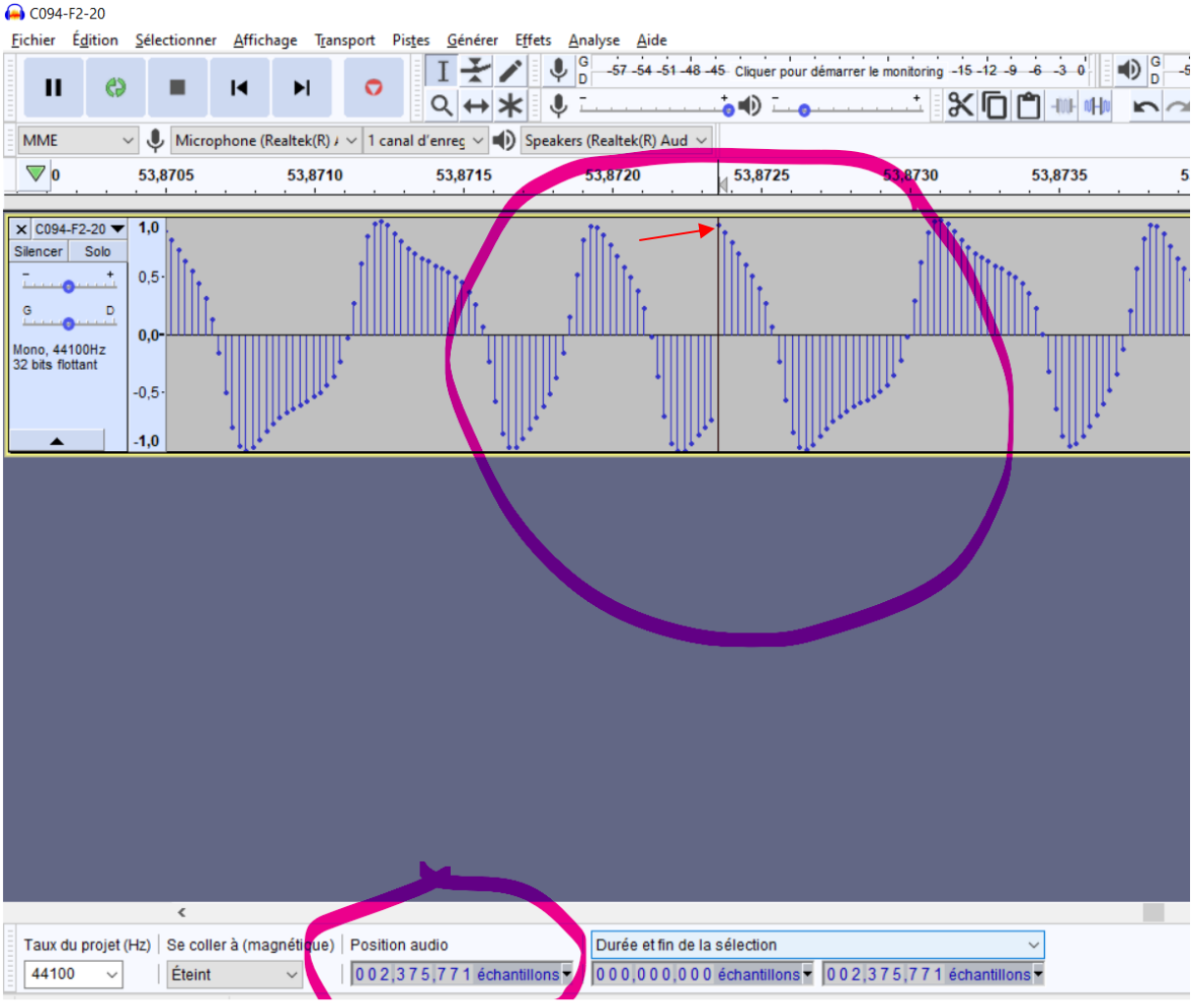
- qu'un bit « zéro » ordinaire à deux niveaux hauts, un de 567 µs et un de 927µs
- qu'un bit « un » ordinaire à deux niveaux hauts, un de 927 µs et un de 567µs

Or dans notre cas de figure le programme nous indique qu'il a trouvé un bit avec deux niveaux hauts égaux… C'est qu'il y a un problème !

Le problème pourrait être là où le programme trouve l'incohérence, cela arrive, mais il se peut aussi que cela vienne d'un décalage de lecture suite à un incident en amont dans l'enregistrement (donc en amont sur la bande magnétique).

⇨ Visualisons ce qui se passe un peu avant ce soit-disant bit incohérent. Faisons défiler le signal vers la gauche. Avec une certaine expérience, on se rend compte qu'il y a une bizarrerie au niveau de la flèche (photo ci-dessous) <u>on voit la présence d'un flanc plus raide</u> que les autres.

⇨ Nous sommes maintenant à la position **2376771**

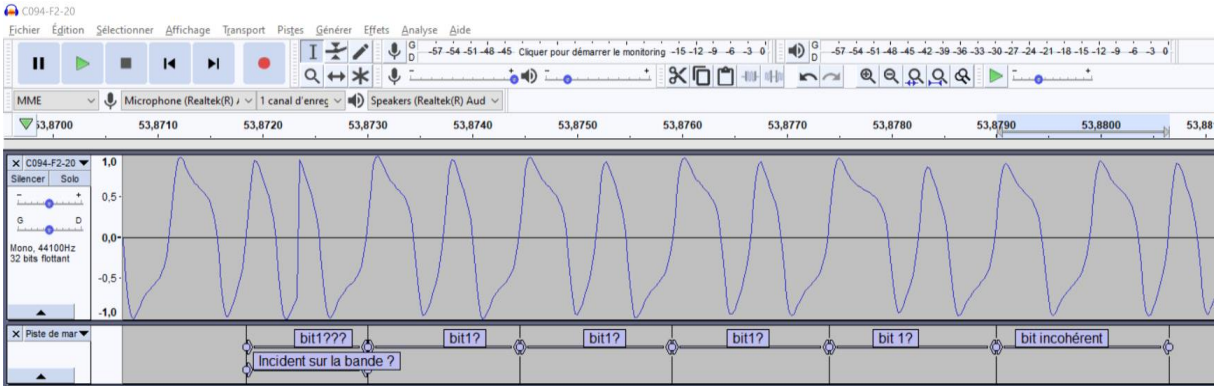


« To bit or not to bit, that is the question. » William Checksum.

⇨ Nous allons donc remonter la piste en faisant le chemin inverse réalisé par le programme depuis l'échantillon où l'incohérence est supposée. Pour cela, plaçons-nous sur le bit

considéré comme incohérent et interprétons tous les bits rencontrés précédemment par le programme jusqu'à la zone suspecte au flanc raide située en amont…

On peut voir ci-dessous en « dézoomé », les 6 bits interprétés par le programme entre la zone suspectée et le bit incohérent. Il faut garder à l'esprit que si le bit signalé incohérent est peut-être un faux bit dû à un décalage de lecture, et si c' est le cas alors tous les bits interprétés depuis l'incident sont également de faux bits !



Vous pouvez voir que j'ai mis un point d'interrogation derrière chaque bit entre le défaut constaté visuellement et le bit signalé comme incohérent car rien ne prouve que ces bits ont bien été interprétés. Ceci illustre le fait que ce sont probablement de faux bits et qu'il faut s'en souvenir !

Dans ce cas particulier, mon expérience m'indique qu'il y a eu un sérieux problème à l'endroit signalé « Incident sur bande ».

Avant de nous précipiter sur des manpulations correctives, il faut encore aller à la recherche d'indices.

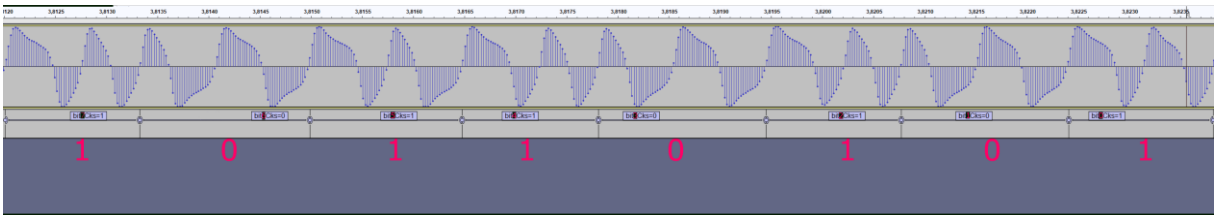
4.4 QUE POUVONS-NOUS ENCORE APPRENDRE DE PLUS ?

Essayons de déterminer le nombre approximatif d'échantillons que doit faire un bit « un » ou « zéro » sur cet enregistrement de cassette… Le plus simple est de regarder dans les logs l'endroit où se situe le précédent checksum correct, c'est celui du bloc N°3 et les logs nous apprennent qu'il était en position 168619

```
--------------------------------------------------------------------------------

BLOC N°3 : lecture de la longueur de bloc et de son checksum

        LgBlock = 1944

        CheckSum lu = B5  (OK)

        Echantillon N° : 168619

--------------------------------------------------------------------------------
```

Le message nous indique que le checksum lu était égal à B5 et le « (OK)» nous indique que le checksum calculé par le programme WAV2DAI est identique à celui relu. C'est donc que nous avons un endroit fiable pour aller identifier la forme des signaux 0 et 1 sur cet enregistrement de cassette.

Positionnons-nous en 168619 et séparons les bits du checksum en remontant dans le fichier. Quand le programme indique 168119 c'est la position juste après le deuxième niveau haut du dernier bit du checksum lu. Ceci nous permet de repérer les 8 bits du checksum :



On trouve bien 10110101 = B5 en hexadécimal. Tout va bien…

Dans cet octet :

- un bit 0 mesure entre 72 et 73 échantillons ;
- un bit 1 mesure entre 58 et 66 échantillons ;

Si l'on mesure la longueur du bit identifié « **Bit incohérent** » on constate qu'il mesure 73 échantillons ce qui pourrait le rendre compatible avec un bit à zéro, par contre le bit précédent identifié « **bit1 ?** » mesure 70 échantillons ce qui nous laisse penser que ce n'est pas un bit à 1, je pense donc qu'un décalage s'est produit entre l'incident sur bande et le bit incohérent… Cet indice me conforte dans la piste suivie. Je progresse dans mon enquête, mais je dois téléphoner à ma femme pour lui annoncer que je mangerai pas avec elle ce soir…

Continuons l'enquête : la longueur du bit intitulé « **Bit1 ???** » /« **Incident sur la bande** » mesure seulement **51** échantillons… C'est peu pour un bit 1.

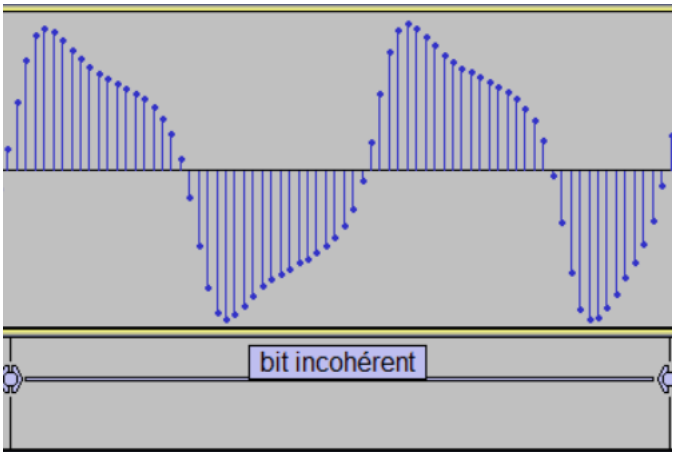
Est-ce que la bande a été ralentie pour une raison ou une autre lors de l'enregistrement il y a 40 ans ? Est-ce une autre raison ?

Quoi qu'il en soit, notre diagnostic se confirme, c'est bien à l'endroit indiqué « **Incident sur la bande** » qu'il se passe quelque chose d'anormal.

4.5 UNE HYPOTHESE SUPPLEMENTAIRE EST NECESSAIRE

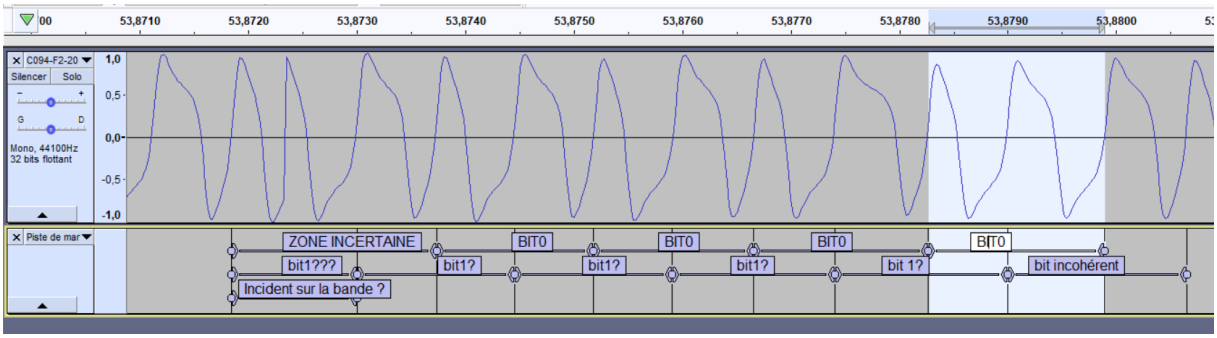
Faisons alors une deuxième hypothèse : supposons qu'au lieu-dit **« bit incohérent** » tout aille bien, mais que la lecture se retrouve décalée du fait d'une catastrophe au lieu-dit **« Incident sur bande** »…

Considérons donc avec notre nouvelle hypothèse, il n'y a pas d'erreur ici :



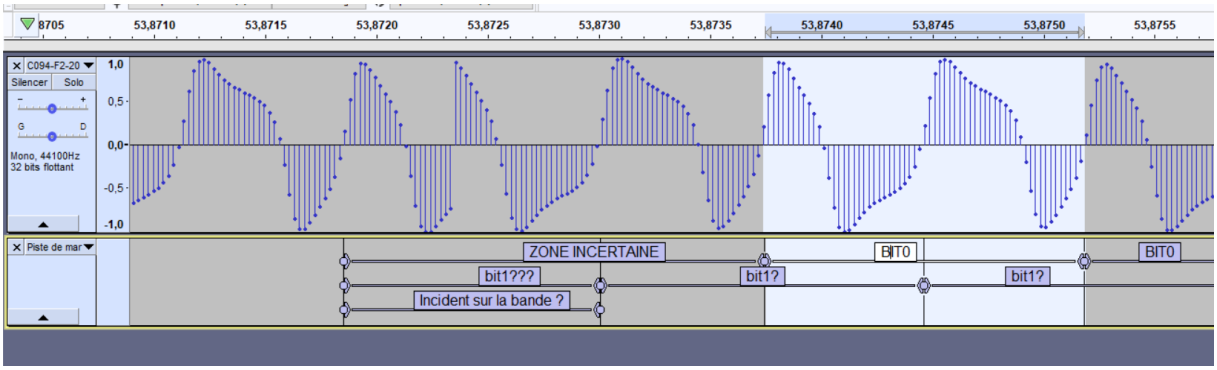
La zone « **bit incohérent** » ne peut se produire que lorsqu'un bit 0 est immédiatement suivi d'un bit 1, nous sommes donc en train de visualiser la moitié d'un bit 0 suivi de la moitié d'un bit 1… Nous allons

donc nous servir de ces informations pour **visualiser avec une nouvelle série de marqueurs** la séquence de bits entre le lieu de la supposée catastrophe et le bit incohérent :



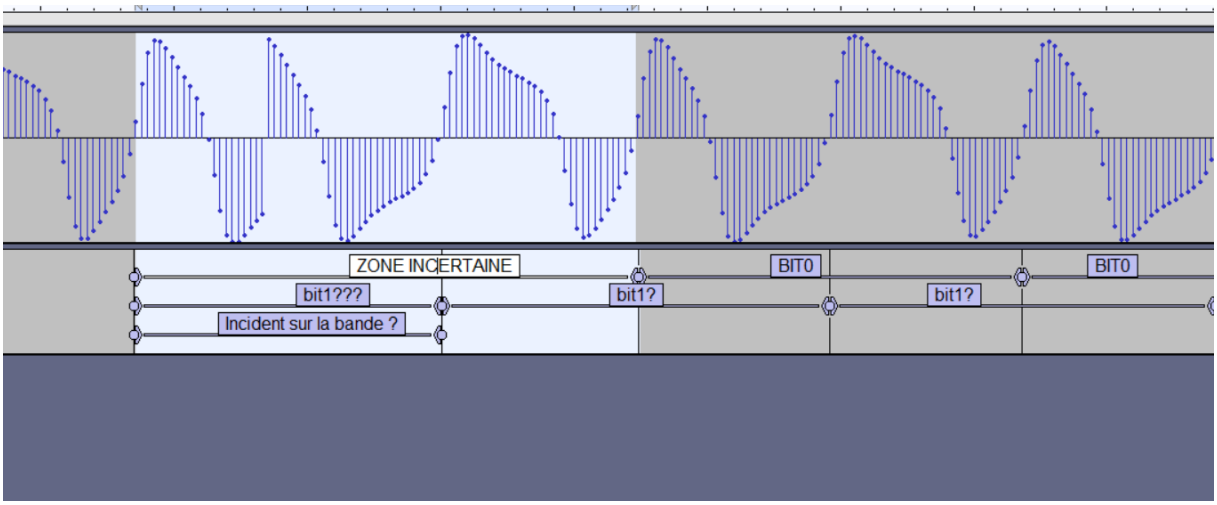
On voit qu'on arrive à interpréter au **maximum QUATRE BIT 0 (que j'ai indiqués en majuscules pour mieux les différencier)**.

En zoomant sur le dernier BIT0 sur la gauche (en surbrillance ci-dessous) on voit que l'interprétation d'un cinquième bit nous ramènerait en plein dans la zone incertaine de la supposée catastrophe !!!



Oui mais ce nouvel exercice nous permet de comprendre que la zone d'incertitude est plus importante qu'initialement supposé.

Mettons en surbrillance la zone qui va du début de l'incident sur bande jusqu'au dernier BIT0 que nous venons d'interpréter :



On voit grâce à cette enquête que la zone de catastrophe est assez large pour accueillir 2 bits !!!

5 VERS LA SOLUTION

Nous allons donc tenter une opération chirurgicale, à savoir supprimer la zone de catastrophe et introduire à sa place un signal bien formé correspondant à 2 bits, **cela nous fait 4 cas à analyser** :

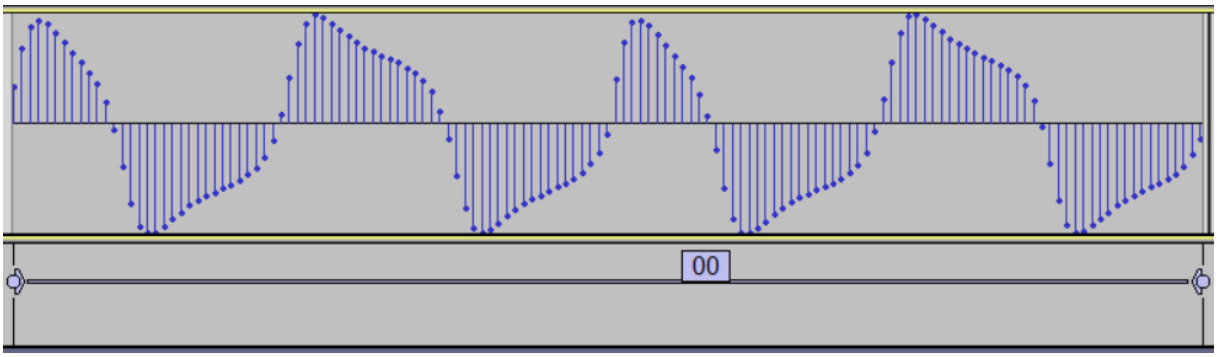
00

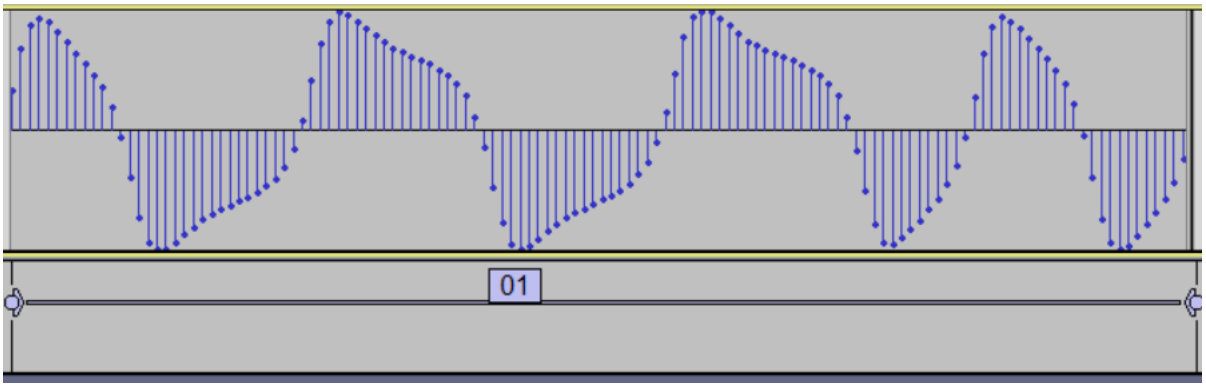
01

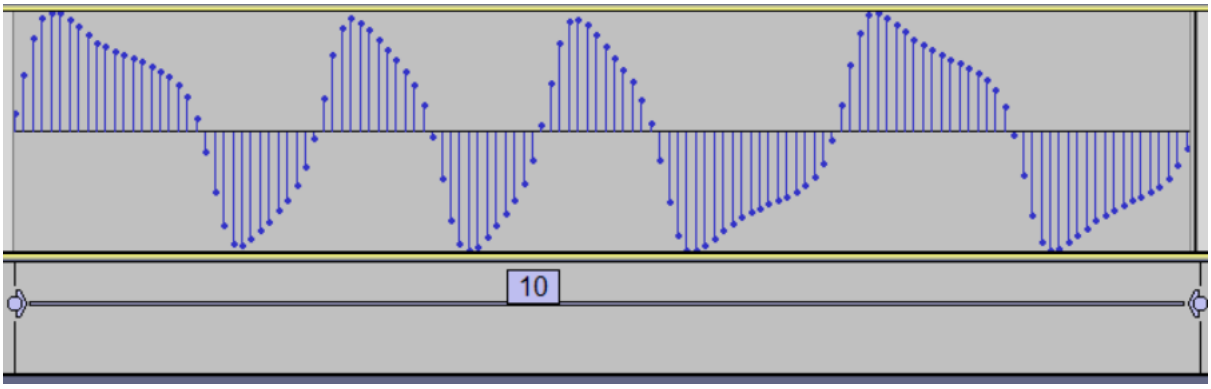
10

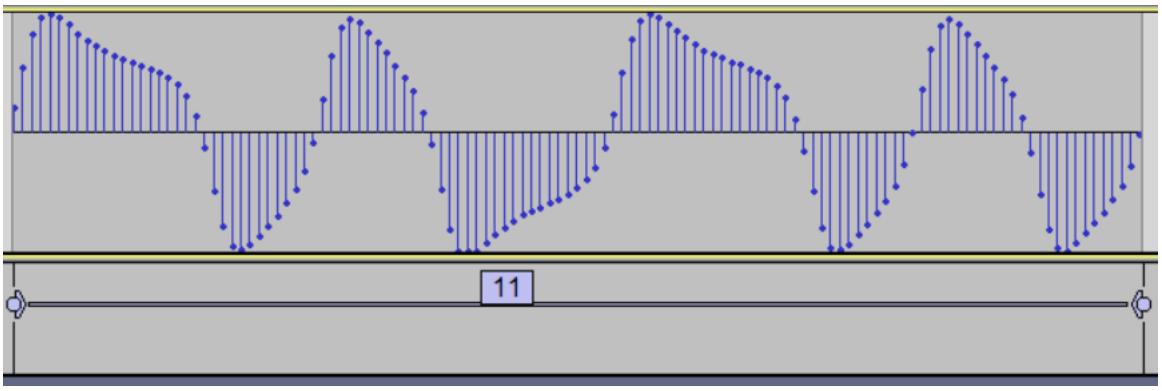
11

On se prépare 4 formes d'ondes correspondant à ces 4 cas, pour cela, retournons dans un checksum correct, celui du bloc N°3 en position **168619** où nous pourrons nous servir en bits 0 et 1 corrects.









Il nous reste à supprimer la « ZONE INCERTAINE » d'une largeur de 83 échantillons pour la remplacer par l'un de 4 morceaux 00, 01, 10, 11 qui font respectivement 142, 131, 131, 124 échantillons.

En visualisant la forme de la zone incertaine, il me semble que c'est plutôt 00 qui conviendrait, nous allons donc commencer par cette combinaison.

BINGO !!! Et voici la preuve dans le fichier de logs, les seules erreurs signalées sur la sortie stderr sont celles liées à la recherche du leader en début de fichier, erreurs qu'il est facile de faire totalement disparaitre en enlevant le début du leader qui est toujours chaotique, ceci étant dû à l'interface cassette peu sophistiquée du DAI. Voici les logs sans intervention chirurgicale sur le leader:

```
RIFF Header : RIFF
Taille suite du Fichier : 8260428
Type RIFF : WAVEChunk ID : fmt
Taille Chunk fmt : 16
wFormatTag : 1  => Microsoft Pulse Code Modulation (PCM)
1 canal (mono)
44100 frames par seconde (Freq échant.)
2 octets par sample frame
88200 octets par seconde
16 bits par échantillon
Taille du CHUNK DATA = 8260290
Durée = 93.654082 secondes
Début des échantilons en : 44
Nb de frames lues = 4130192
Somme = 5779109
Moyenne = 1.399235
Minimum = -32768
Maximum = 32767
```

C094-F2-20.dai

Lecture du leader

Début des données WAVE en 44

Fin d'un bout de leader en : 32056

Fin d'un bout de leader en : 38533

Fin d'un bout de leader en : 53397

Fin d'un bout de leader en : 153302

L'octet 55H a été trouvé.

Type fichier : 30

Fin du header en : 154359 (début du dernier état bas de l'octet donnant le type de fichier.)

BLOC N°1 : lecture de la longueur du nom de fichier et de son checksum

LgNom = 14

CheckSum LgNom lu = 71

CheckSum LgNom calculé = 71 (OK)

N° d'échantillon après lecture du checksum de la longueur du nom de fichier : 155940

BLOC N°2 : lecture du nom de fichier et de son checksum

Début de fonction LireBloc, position N° 155940

Nb octets à lire = 14

Fonction RBLK : Nb octets lus : 14

Fonction LireBloc :

checksum calculé 87

checksum lu 87 (OK)

Fin de fonction LireBloc, position N° 167035

Nom fichier = COMPUBOGGLE [2 fin]

CheckSum lu = 87 (OK)

N° d'échantillon après lecture du nom de fichier : 167035

BLOC N°3 : lecture de la longueur de bloc et de son checksum

LgBlock = 1944

CheckSum lu = B5 (OK)

Echantillon N° : 168619

BLOC N°4 : lecture du bloc et de son checksum

Début de fonction LireBloc, position N° 168619

Nb octets à lire = 1944

Fonction RBLK : Nb octets lus : 1944

Fonction LireBloc :

checksum calculé ED

checksum lu ED (OK)

Fin de fonction LireBloc, position N° 3573539

CheckSum lu = ED (OK)

Echantillon N° : 3573539

BLOC N°5 : lecture de la longueur de block et de son checksum

LgBlock = 32D

CheckSum lu = F (OK)

Echantillon N° : 3575116

BLOC N°6 : lecture du bloc et de son checksum

Début de fonction LireBloc, position N° 3575116

Nb octets à lire = 32D

Fonction RBLK : Nb octets lus : 32D

Fonction LireBloc :

checksum calculé 14

checksum lu 14 (OK)

Fin de fonction LireBloc, position N° 4002756

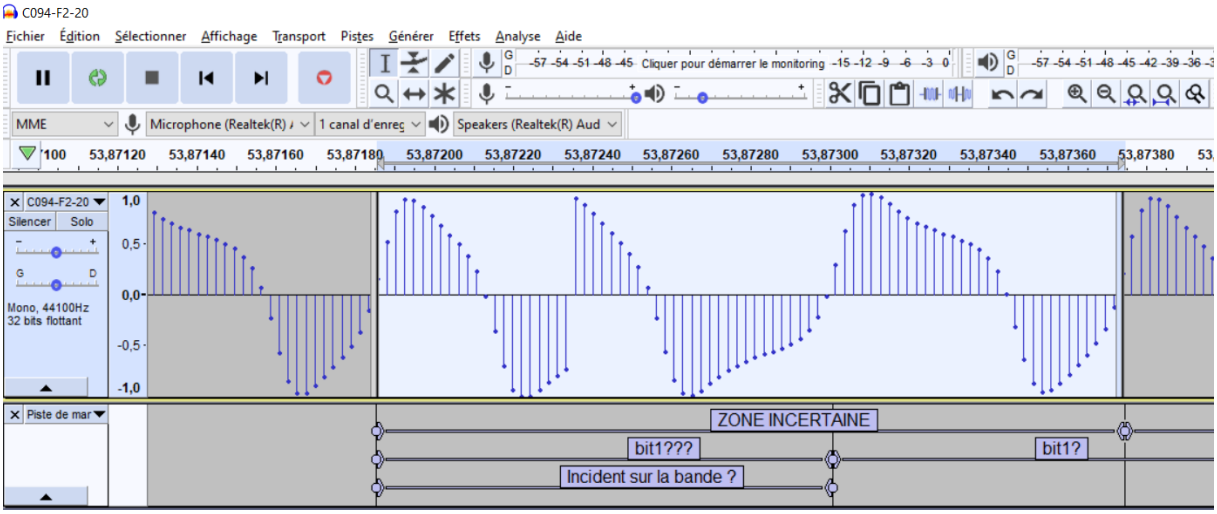
CheckSum lu = 14 (OK)

Echantillon N° : 4002756

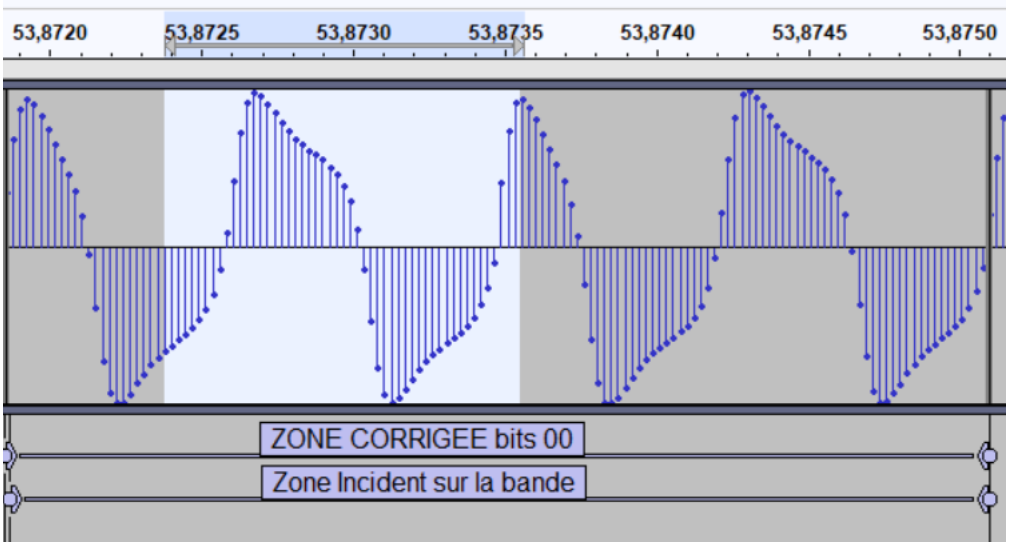
En enlevant le début du fichier jusqu'à la moitié du leader, les erreurs disparaissent totalement sur la sortie stderr, ainsi que dans les logs (les lignes signalées en orange).

6 CONCLUSION DE L'AFFAIRE

On peut supposer que lors de l'enregistrement de ce fichier, il y a une quarantaine d'années, un incident est survenu lors du défilement de la cassette. Petit récapitulatif visuel :



On voit maintenant clairement que 2 bits étaient affectés par cette erreur mécanique, il manquait **50 échantillons à la fin du premier bit « zéro »** et **3 échantillons au début du deuxième bit « zéro »**, c'est ce que j'ai mis en surbrillance ci-dessous :



Comme vous avez pu le voir sur cet exemple assez tordu, la restauration d'un fichier sur cassette nécessite beaucoup d'observation et de patience.

Le programme **COMPUBOGGLE** est sauvé, je saute dans ma 403 pour aller manger un excellent Chili Con Carné préparé par Claudia « *Rançon pour un homme mort » 1971*.