READ_ME

Obj 1 Programmer la DFT en simulation sur un tableau de données imposés

Nom du commit : 39fa74a965

Bref explication de l'organisation du code

Notre programme principal.c appelle la fonction CalculM qui renvoie le module mis au carré de la DFT de rang k. Cette dernière fonction est implémentée dans DFT.s où elle appelle calculReouIm la procédure permettant de calculer les imaginaires ou les réels d'après les tableaux de sinus et de cosinus.

Jeu de tests

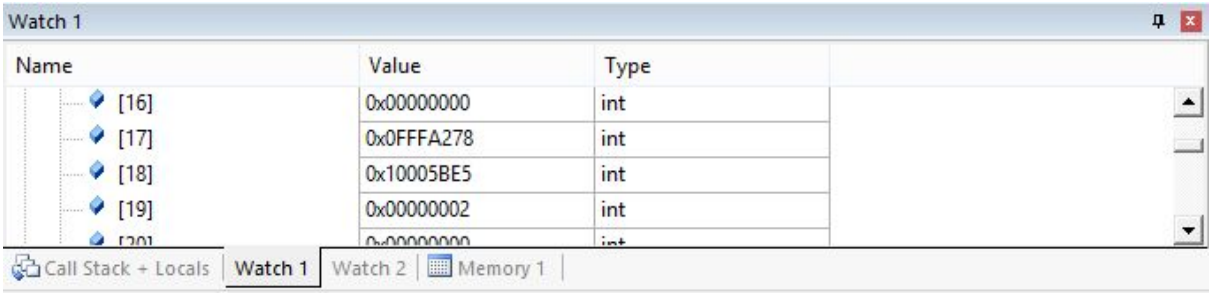
Nous avons renommé le fichier trigo.asm : TabSinCos.asm, et dans le commit effectué nous testons avec le fichier "f17p30_f18p135.asm"

Les points d'arrêts:

Positionner un point d'arrêt dans le fichier c "principal.c" au niveau de la boucle while infini.

Marche à suivre:

1. Commencez par **assembler** puis **débugger**
2. **Run**
3. Menu **View → Watch Window → Watch 1**
4. **Entrer l'expression** 'TabM' (Nous souhaitons observer le tableau contenant le module mis au carré de la DFT de rang k)



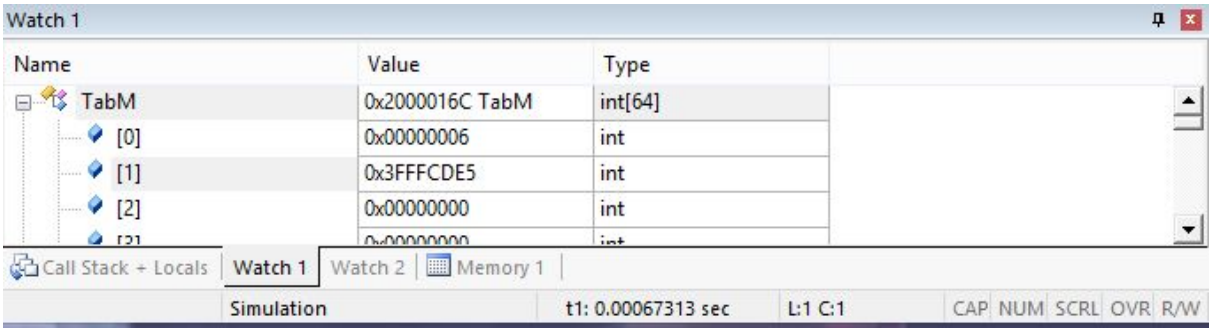
```
valeurs attendues pour k = 17 :
  Re  0x376C909D  env 0.866 * 2^30
  Im  0xE000C6D7  env -0.5  * 2^30
  M2  0x0FFFA278  env 2^28

valeurs attendues pour k = 18 :
  Re  0xD2BDF5FC  env -sqrt(0.5) * 2^30
  Im  0xD2BE8C7F  env -sqrt(0.5) * 2^30
  M2  0x10005BE5  env 2^28
```

Les résultats sont en accord avec les indications du test on retrouve bien pour k=17, M2 =0x0FFFA278 et pour k=18, M2=0x10005BE5.

En testant avec les autres fichiers on retrouve les résultats attendus aussi.

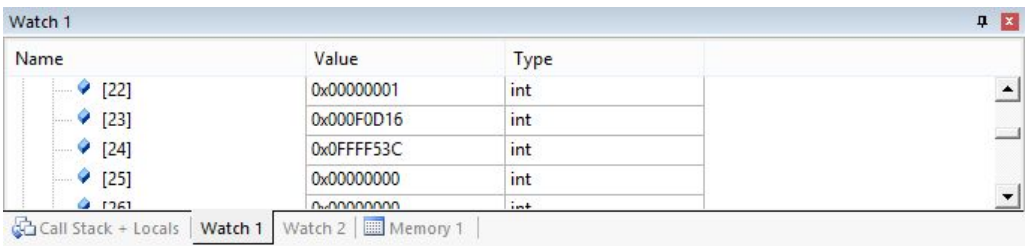
f1p-45.asm

```
valeurs attendues pour k = 1 :
  Re  0x5A82562C  env +sqrt(0.5) * 2^31
  Im  0x5A82562C  env +sqrt(0.5) * 2^31
  M2  0x3FFFCDE5  env 2^30
```



f23p-26_f24p-116.asm

```
valeurs attendues pour k = 23 :
  Re  0x0378FDBD
  Im  0x01BAD0C5  env 0.5 * Re, car tan(26.565) ~= 0.5
  M2  0x000F0D16  986390

valeurs attendues pour k = 24 :
  Re  0xE36136DD  env -0.447 * 2^30
  Im  0x393E61CA  env -2 * Re, car tan(116.565) ~= 2.0
  M2  0x0FFFF53C  env 2^28
```



Obj 2 Faire tourner la DFT « en réel » et gérer le score des 6 joueurs

Nom du commit : eeec7ac8ac

Calcul du seuil

Le CAN 12 bits peut recevoir un voltage de 0 à 3,3V. Ainsi 3,3 V sera converti en $2^{12} - 1 = 4095$ $(en\ décimal)$.

L'amplitude d'entrée du signal est de 50 mv, elle sera donc converti en $0,05 * 4095/3,3 \approx 62$.

Nous savons que le module de la DFT pour un signal sinusoïdal d'amplitude A, de fréquence kN/Te et de phase quelconque est : $|X(k)| = AN/2 = 0.5AN$. La valeur du seuil sera donc égale au module au carré :

$\left(\frac{A*N}{2}\right)^2 = 3\ 936\ 256$, avec A=62 et N=64. Cela équivaut en hexadécimal à : 3C1000.

Ce seuil calculé ne convenant pas nous l'avons ajustée pour obtenir les résultats voulus. La valeur du seuil est donc :

- en hexadécimal : 0x2A7138
- en décimal: 2781496 (correspond à la conversion de 42mv)

Observation des tirs:

Les variables que nous souhaitons observer sont:

- Les tirs des joueurs
- Le score de chaque joueur

Données

Afin d'observer differents tirs lasers, il suffit de parametrer Duree_Ech_ticks de Init_TimingADC_ActiveADC_ff().
Dans l'exemple qui suit nous avons utiliser:

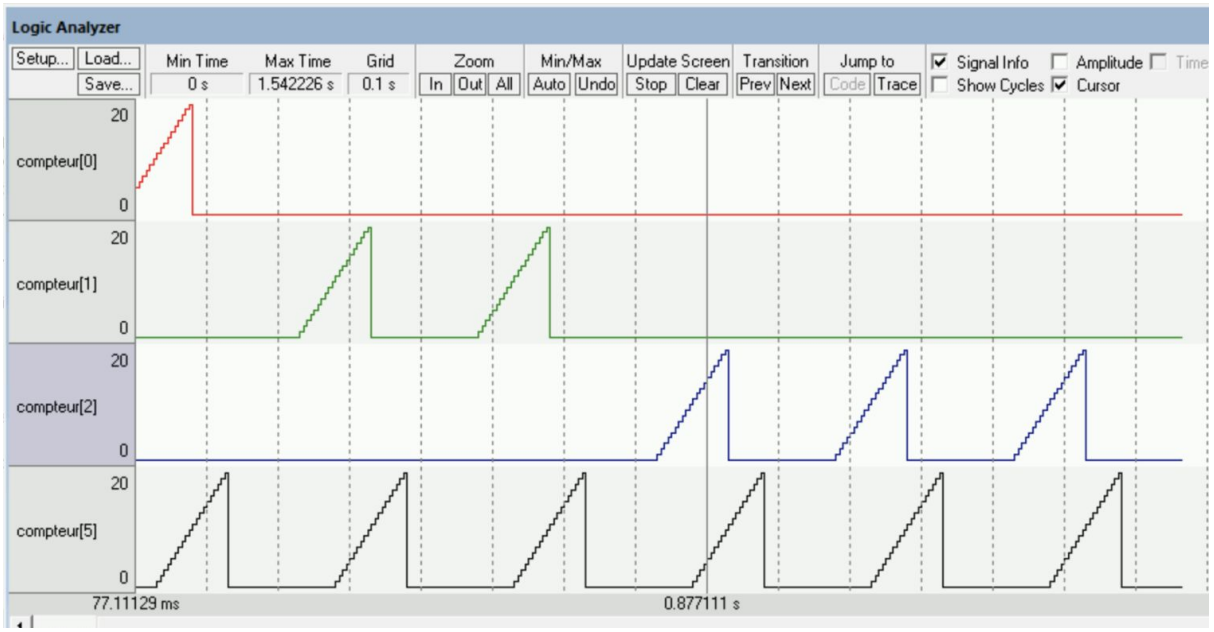
Init_TimingADC_ActiveADC_ff(ADC1, 0x52);

0x52 : sig.1 = 124, bruit = 4, sig.2 = 1000

Marche à suivre pour observer les tirs sur la cible:

1. Commencez par **assembler**  puis **exécuter**  .
2. Menu **View → Analysis Window → Logic Analyser**
3. **Setup** → Ajouter 4 nouveaux signaux : compteur[0], compteur[1], compteur[2], compteur[3], compteur[4] compteur[5](Display Type : Analog)

4. **Run**
5. **Adaptative Min / Max** en faisant un clic droit sur TIM3_CCR3
6. **Zoom All** 

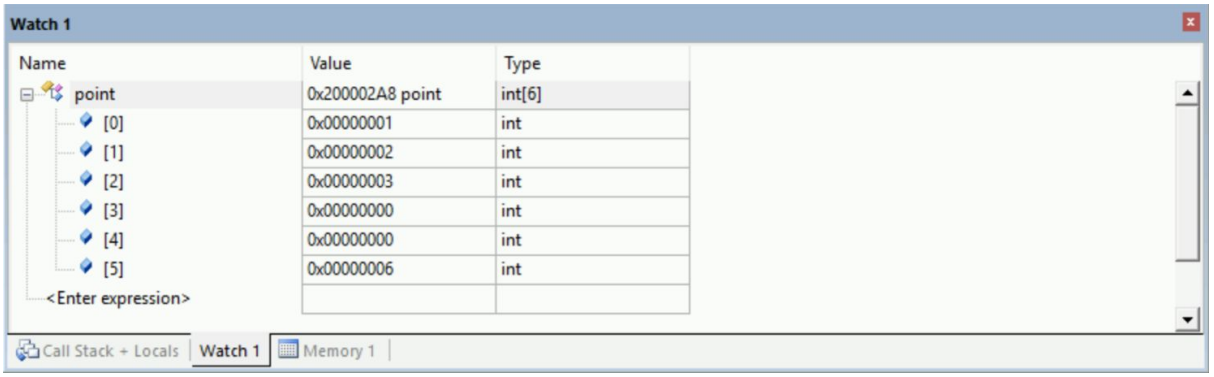


(Ici nous n'avons seulement afficher le compteur 0, 1, 2 et 5 pour une meilleure visibilité)

A chaque étape de la dft si le signal dépasse le seuil que nous avons fixé, la variable compteur est incrémenté. Lorsque compteur dépasse une certaine valeur (pour nous 3) nous incrémentons le score des joueurs.

Marche à suivre pour obtenir le score:

7. Menu **View → Watch Window → Watch 1**
8. **Entrer l'expression** 'point' (Nous souhaitons observer le tableau contenant les scores)



Le tableau de score est bien en accord avec l'affichage des tirs:

Exemple : Le signal compteur[0] n'affiche qu'un seul tir et le joueur 0 a bien 1 point (=point[0]).

Si nous laissons tourner le programme jusqu'au bout nous obtenons le tableau suivant.

Name	Value	Type
point	0x200002A8 point	int[6]
[0]	0x00000001	int
[1]	0x00000002	int
[2]	0x00000003	int
[3]	0x00000004	int
[4]	0x00000005	int
[5]	0x0000000F	int

Celui-ci est bien en accord avec ce que l'on souhaitait obtenir:

Séquence des fréquences :
signal 1 :
1 tir à 85kHz (k=17)
2 tirs à 90kHz (k=18)
3 tirs à 95kHz (k=19)
4 tirs à 100kHz (k=20)
5 tirs à 115kHz (k=23)
signal 2 : (en parallèle, décalé de 50 ms)
15 tirs à 120kHz (k=24)

Les points d'arrêts:

Afin d'observer tir après tir le son se déclencher, on peut placer un point d'arrêt à la ligne 74.

```
72                  for (int j = 0 ; j < 6 ; j++){
73                          if(compteur[j]==3){
74                                  point[j]++; // on incremente le score du joueur j
75                          }
76                  }
77          }
```

Obj 3 Gérer le son

Nom du commit : 0efe80e4d2

Lorsque votre programme déclenche l'émission d'un son *bruitverre*, nous obtenons la figure 1 dans le logic analyser. Il s'agit ici du registre spécial TIM3_CCR3. La durée entre chaque échantillon doit être de 91µs (cf figure 2).

Marche à suivre:

1. Commencez par **assembler** puis **exécuter** .
2. Menu **View → Analysis Window → Logic Analyser**
3. **Setup** → Ajouter un nouveau signal : TIM3_CCR3 (Display Type : Analog)
4. **Run**
5. **Adaptative Min / Max** en faisant un clic droit sur TIM3_CCR3
6. **Zoom All** Zoom In Out All

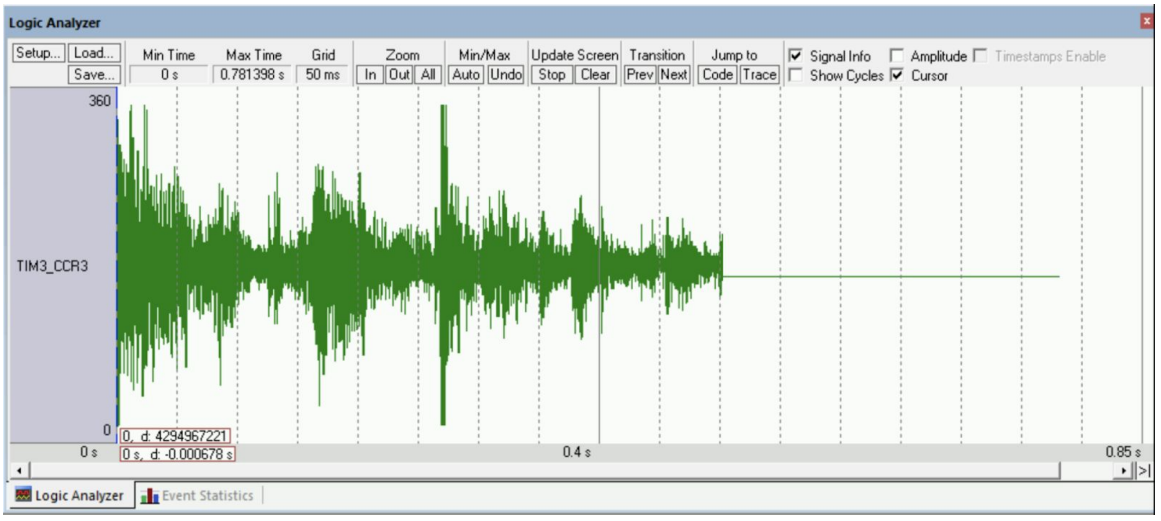


Figure 1 : Observation du signal audio bruit de verre

7. Pour observer une période : Zoom In jusqu'a bien la discerner puis déplacer le curseur pour obtenir sa valeur.

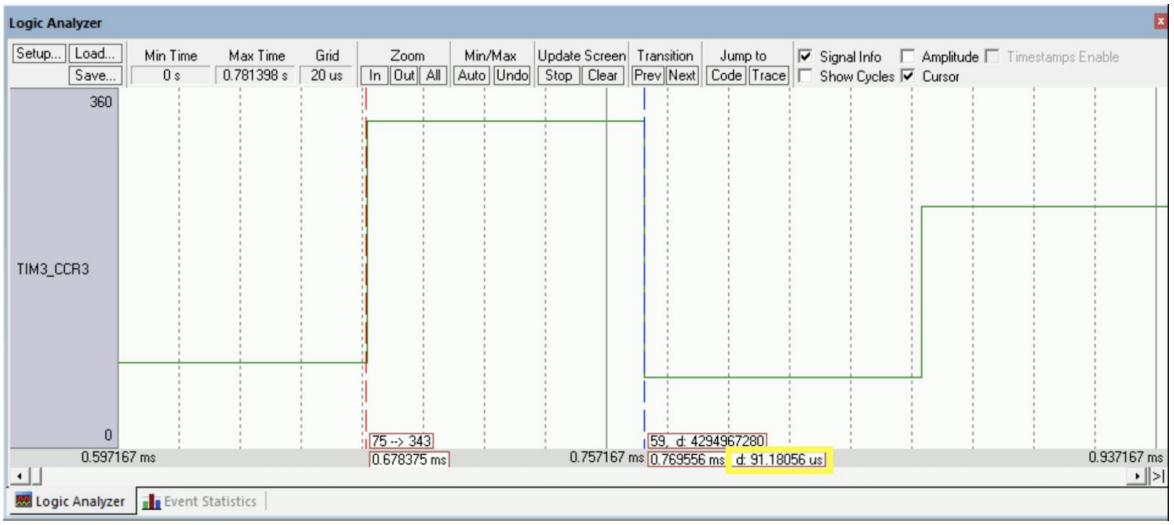


Figure 2 : Observation d'une période du signal audio bruit de verre

Obj 4 Projet final

Nom du commit : 441f31b0ca

Lorsqu'un joueur tir alors qu'un son se fait entendre, celui-ci est écourté pour laisser place à un nouveau. Nous aurions aussi pu régler ce problème en diminuant la taille du son de verre cassé à 1000 ainsi chaque son aurait eu le temps de s'émettre entre 2 salves de pistolets.

Séquence des fréquences :

signal 1 :

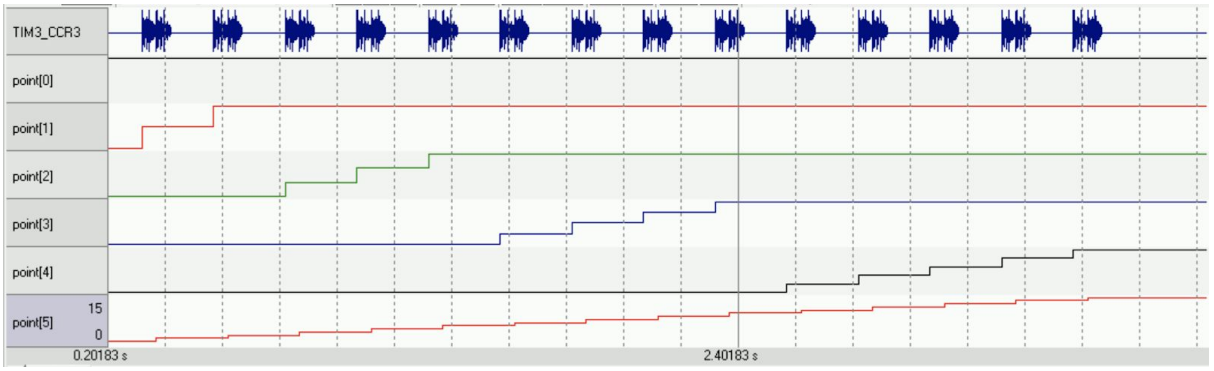
1 tir à 85kHz (k=17)
2 tirs à 90kHz (k=18)
3 tirs à 95kHz (k=19)
4 tirs à 100kHz (k=20)
5 tirs à 115kHz (k=23)

signal 2 : (en parallèle, décalé de 50 ms)

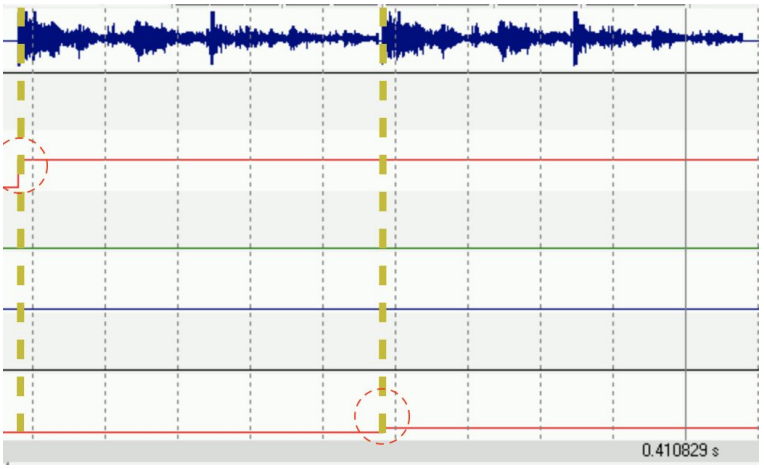
15 tirs à 120kHz (k=24)

Marche à suivre:

1. Commencez par **assembler** puis **exécuter** .
2. Menu **View → Analysis Window → Logic Analyser**
3. **Setup** → Ajouter : TIM3_CCR3 , point[0], point[1], point[2], point[3], point[4] point[5](Display Type : Analog (Display Type : Analog)
4. **Run**
5. **Adaptative Min / Max** en faisant un clic droit sur les variables
6. **Zoom All** Zoom In Out All



TIM3_CCR3 représente le son généré à chaque point gagné par un joueur (à chaque fois qu'un tir touche la cible). Chaque variable point représente un joueur, et chacune de ses incrémentations correspond à un tir sur la cible.



On observe que le son fonctionne correctement sur l'image zoomé ci dessus, à chaque tir d'un joueur, le signal démarre sur la première ligne TIM3_CCR3.

Ayant utilisé la même séquence de tir que pour l'objectif 2, afin de visualiser les résultats des joueurs référez-vous donc à la marche à suivre précédemment utilisé.

Les points d'arrêts:

Afin d'observer tir après tir le son se déclencher, on peut placer un point d'arrêt à la ligne 85.

```
81          for (int j = 0 ; j < 6 ; j++){
82              if(compteur[j]==3){
83                  point[j]++; // on incremente le score du joueur j
84
85                   etat.position = 0;
86
87
88              }
89          }
```