Systèmes embarqués et robotique : résumé cours 6

Maxime Nourry, Microtechnique BA6 2024

1.Introduction

Il n'est pas rare pour un système embarqué de recevoir et de devoir traiter des signaux. On peut par exemple citer les aides auditives qui se doivent de corriger précisément une entrée audio dans un délai acceptable pour un usage quotidien (Moins de 10 ms).
Afin de répondre à ces exigences, du hardware spécialisé a été développé.

On peut le séparer en deux catégories:

1. Le digital signal processor (DSP), un processeur dédié aux traitements de signaux
2. Une extension au processeur classique (c'est le cas dans notre Cortex™-M4).

Mais comment ces derniers se distinguent-t-ils d'un processeur classique?

2.L'opération de base MAC et ses utilisations

MAC (Multiplicate and accumulate) se décrit comme une suite de produits et leur somme.

$$a \cdot b = \sum_{i}^{n} a_i * b_i$$

Cette opération se retrouve dans:

- Le filtrage numérique (FIR Finite Impulse Response).

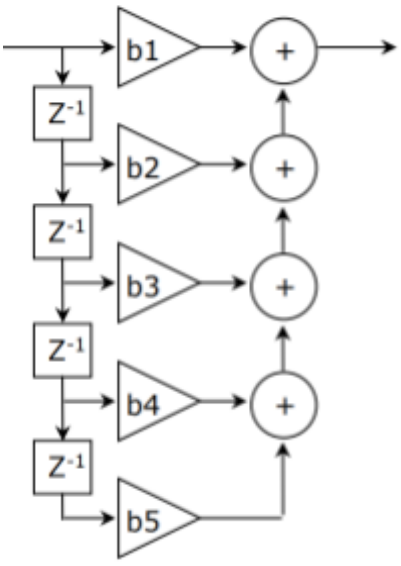

Figure1: Exemple de FIR à 5 échantillons

Dans le cas d'une moyenne, on multiplie N échantillons par 1/N avant de les additionner entre eux.

- La convolution

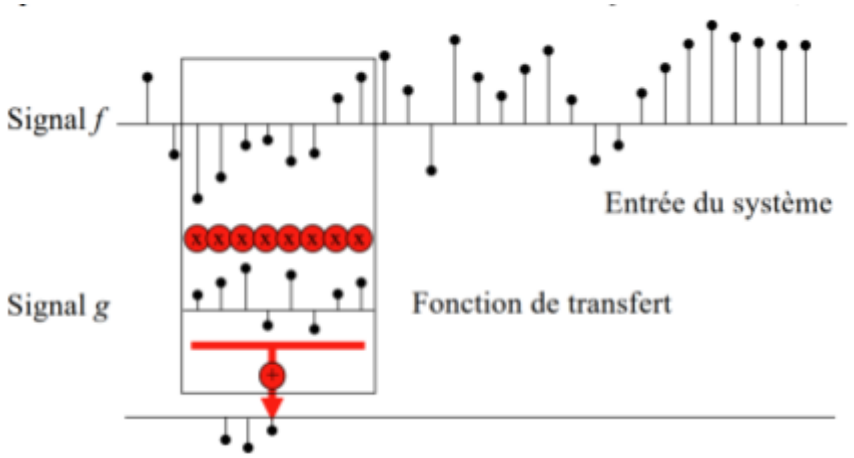

Figure 2 :représentation d'une opération de convolution. g s'obtient en observant la réponse du système à un impulsion de Dirac

- Les réseaux de Neurones

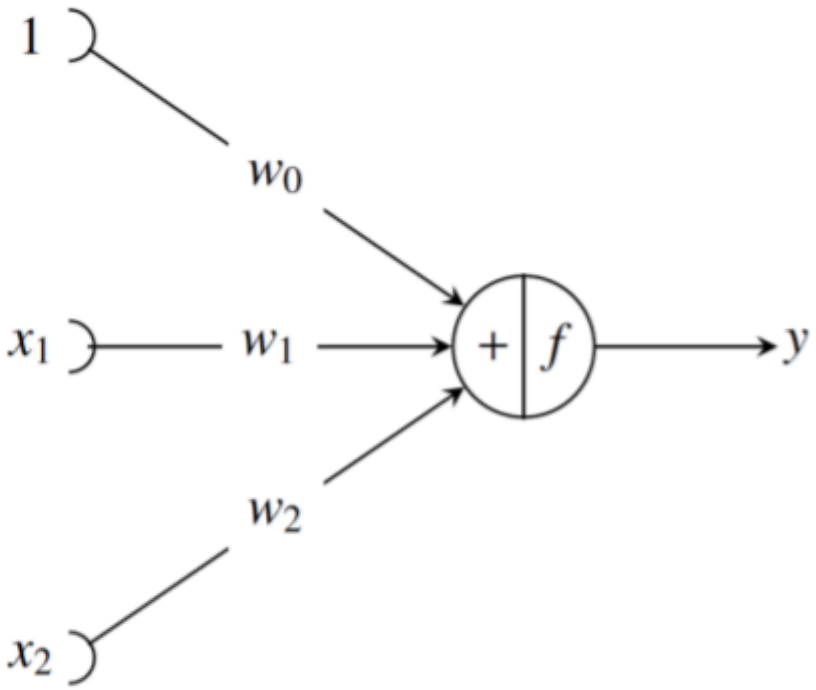

Figure 3: Représentation d'un réseau neurones. On peut y imaginer un MAC sous la forme de somme des différents poids (w0,w1,w2)

Cette opération prendrait dans un assembleur classique un nombre important d'instructions

multiplier -> additionner -> multiplier -> additionner ->.....

C'est en combinant ces instructions répétés que les assembleurs spécialisés offrent un avantage par rapport à l'assembleur classique.
Nous pouvons par exemple citer le SIMD (single instruction multiple data) qui peut aller jusqu'à réaliser la somme de deux produits en une seule et même instruction

Somme en 64 bits
A,B,C,D en 16 bits
Somme = Somme + (A*C) + (B*D)

On opère alors 4 fois plus vite que sur un processeur non spécialisé.

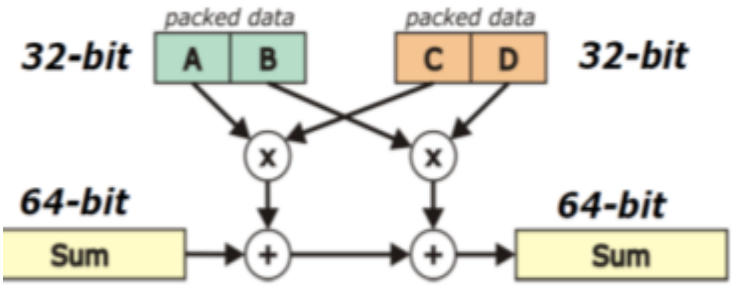

Figure 4: Représentation du fonctionnement du SIMD. A,C ainsi que B,D y sont multipliés et sommés à Sum

Notes: Cette opération est aussi faisable avec A,B,C,D en 32 bits mais au risque d'avoir une somme qui dépasse les 64 bits prévus.

Attention: Pour que le compilateur utilise les instructions SIMD on ne peut pas simplement espérer qu'il reconnaisse cela dans des additions et multiplications. Il faut lui indiquer précisément qu'il faut utiliser les instructions SIMD.

Soit par la forme:
(int32_t)(((int64_t)x*y((int65_t)acc<<32))>>32);

Soit à l'aide d'un define (voir le fichier arm_math.c).

Soit, en utilisant "*_asm*" qui permet d'écrire de l'assembleur dans du C (méthode considérée comme moins propre)

3.Buffer circulaire:

Comme vu précédemment avec l'exemple FIR. Le traitement de signal nécessite souvent plusieurs échantillons récupérés en continu.
Ceci nécessite alors un buffer optimisé afin de ne pas perdre du temps inutile.
En effet, lorsqu'un buffer normal est rempli, rajouter des informations nécessite un décalage de toutes les informations précédentes, voir figure 5.

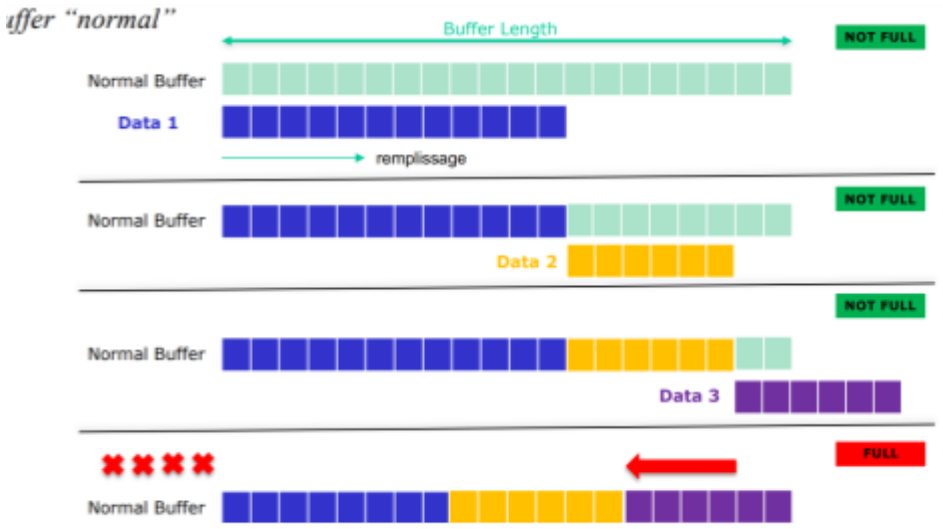

Figure 5: Comportement d'un Buffer normal

Ceci est chronophage et non désirable.

C'est ici qu'entre en jeu le buffer circulaire.
Comme son nom l'indique, à la manière d'un cercle, le data injecté revient au début quand il n'y a plus de place à la fin, voir figure 6.

On le réalise à l'aide d'un pointeur sur la dernière case de libre.

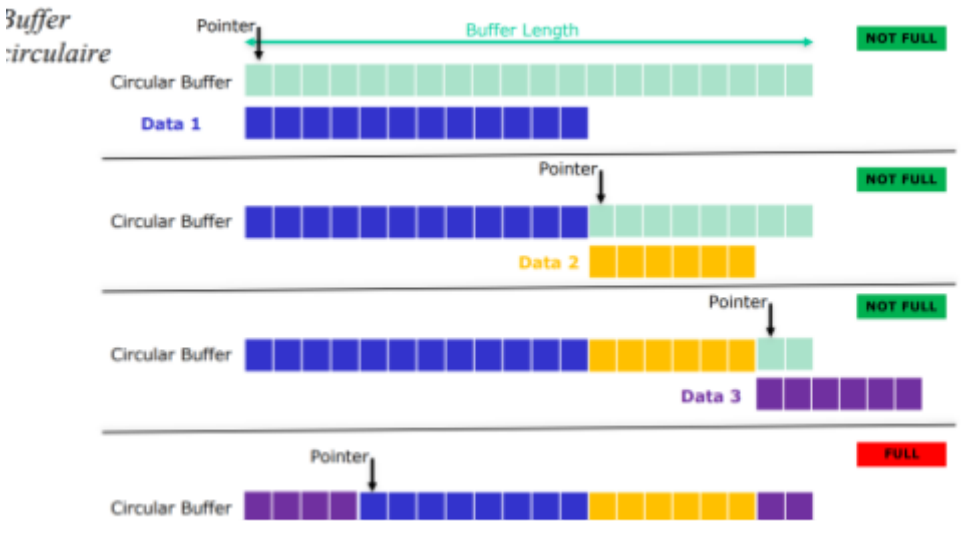

Figure 6: Comportement d'un buffer circulaire

Un tel buffer nécessite l'ajout d'une condition, afin de remettre le pointeur au début, une fois ce dernier arrivé à la fin.
Ceci n'est cependant pas toujours nécessaire à programmer, certains processeurs ayant du hardware fait pour éviter cette condition.

4.La transformée de Fourier

N'importe quel signal cyclique peut être recréé comme une somme de sinus et de cosinus.
C'est là le cœur de la transformée de fourier.
Dans le cadre du cours on parle de DFT (Discrete Fourier Transform).
Elle permet d'isoler pour traitement les différentes fréquences qui constituent un signal d'entrée.
Exemple d'utilisation: (filtrer le bruit et ne garder que la fréquence principale entendue pour un accordeur de guitare)

Cette transformée, bien que très utile, a malheureusement une complexité en $O(N^2)$.
(voir notation de Landau et P=NP pour les plus téméraires)
Il existe cependant des méthodes pour la simplifier.
C'est le cas de la FFT (fast fourier transform)
Son principe consiste en la décomposition de notre DFT en sous DFT de plus petites tailles jusqu'à obtenir une taille de 1.
On passe alors de $O(N^2)$ à $O(NlogN)$

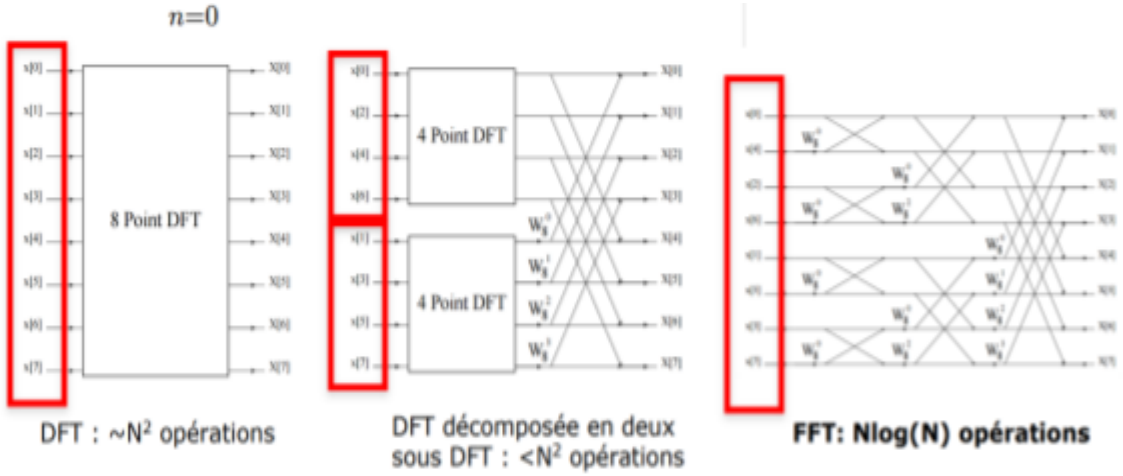

Figure 7: Décomposition de la DFT en FFT

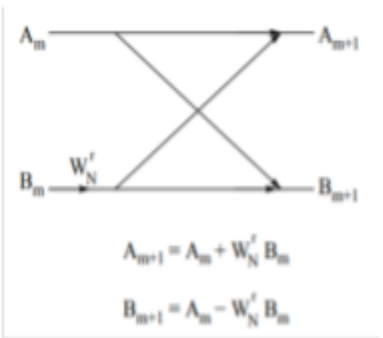

Figure 8: Une opération "Butterfly"

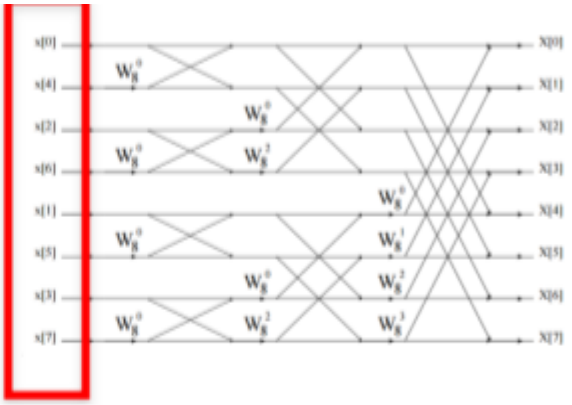

Figure 9: Représentation d'un FFT

Comme l'indique la figure 9, cette décomposition nécessite la permutation de certains bits voir ainsi que des opérations dites "*butterfly*".

Toujours dans un but d'optimisation, il n'est pas rare de trouver du hardware possédant des modes d'adressages qui suivent cette configuration.
Nous pouvons aussi la définir à l'aide de tables comme l'indique la figure 10.

```
/*
* @brief  Table for bit reversal process
*/
const uint16_t armBitRevTable[1024] = {
   0x400, 0x200, 0x600, 0x100, 0x500, 0x300, 0x700, 0x80, 0x480, 0x280,
   0x680, 0x180, 0x580, 0x380, 0x780, 0x40, 0x440, 0x240, 0x640, 0x140,
   0x540, 0x340, 0x740, 0xc0, 0x4c0, 0x2c0, 0x6c0, 0x1c0, 0x5c0, 0x3c0,
   0x7c0, 0x20, 0x420, 0x220, 0x620, 0x120, 0x520, 0x320, 0x720, 0xa0,
   0x4a0, 0x2a0, 0x6a0, 0x1a0, 0x5a0, 0x3a0, 0x7a0, 0x60, 0x460, 0x260,
   0x660, 0x160, 0x560, 0x360, 0x760, 0xe0, 0x4e0, 0x2e0, 0x6e0, 0x1e0,
   0x5e0, 0x3e0, 0x7e0, 0x10, 0x410, 0x210, 0x610, 0x110, 0x510, 0x310,
   0x710, 0x90, 0x490, 0x290, 0x690, 0x190, 0x590, 0x390, 0x790, 0x50,
   0x450, 0x250, 0x650, 0x150, 0x550, 0x350, 0x750, 0xd0, 0x4d0, 0x2d0,
   0x6d0, 0x1d0, 0x5d0, 0x3d0, 0x7d0, 0x30, 0x430, 0x230, 0x630, 0x130,
   0x530, 0x330, 0x730, 0xb0, 0x4b0, 0x2b0, 0x6b0, 0x1b0, 0x5b0, 0x3b0,
   0x7b0, 0x70, 0x470, 0x270, 0x670, 0x170, 0x570, 0x370, 0x770, 0xf0,
   0x4f0, 0x2f0, 0x6f0, 0x1f0, 0x5f0, 0x3f0, 0x7f0, 0x8, 0x408, 0x208,
   0x608, 0x108, 0x508, 0x308, 0x708, 0x88, 0x488, 0x288, 0x688, 0x188,
   0x588, 0x388, 0x788, 0x48, 0x448, 0x248, 0x648, 0x148, 0x548, 0x348,
   0x748, 0xc8, 0x4c8, 0x2c8, 0x6c8, 0x1c8, 0x5c8, 0x3c8, 0x7c8, 0x28,
   0x428, 0x228, 0x628, 0x128, 0x528, 0x328, 0x728, 0xa8, 0x4a8, 0x2a8,
   0x6a8, 0x1a8, 0x5a8, 0x3a8, 0x7a8, 0x68, 0x468, 0x268, 0x668, 0x168,
   0x568, 0x368, 0x768, 0xe8, 0x4e8, 0x2e8, 0x6e8, 0x1e8, 0x5e8, 0x3e8,
   0x7e8, 0x18, 0x418, 0x218, 0x618, 0x118, 0x518, 0x318, 0x718, 0x98,
```

Figure 10: Table d'adressage pour FFT

Notes: arm fournit des librairies permettant une utilisation simple de la FFT

Enfin, la transformée de fourier se base sur une somme de sinus et de cosinus.
On peut utiliser cette même idée de table afin d' accélérer les calculs trigonométriques.

5.Conclusion:

Le traitement de signal nécessite des opérations complexes et chronophage. C'est de ce postulat qu'émergea des méthodes spécialisées, visant à réduire le temps demandé. Que celà soit par l'usage d'algorithmes optimisés (FFT, buffer circulaire) soit par le design d'hardware spécialisé.