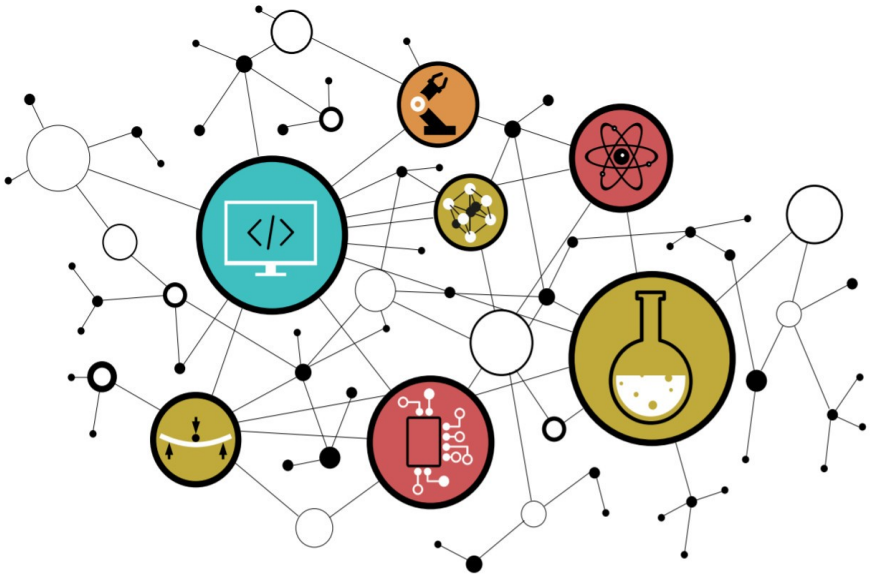
TRAVAUX PRATIQUES

PRÉLUDE



Hugo Descoubes – neku – GNU\Linux - Ubuntu 20.04 LTS – LibreOffice 6.4.6.2 – 2021

SOMMAIRE

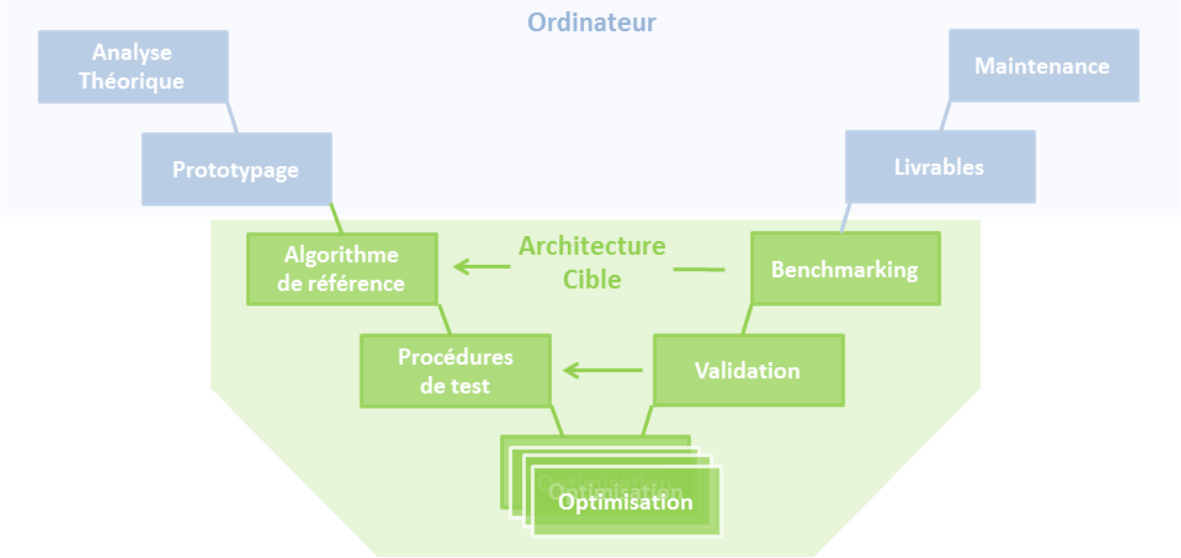
La trame de TP minimale que nous considérons comme être les compétences minimales à acquérir afin d'accéder aux métiers de base du domaine suit le séquencement suivant : chapitres 1, 2 et 3. Le reste de la trame ne sera pas évalué et représente une extension au jeu de compétences minimales relatif au domaine en cours d'étude (* devant chapitres facultatifs voire complémentaires). Libre à vous d'aller plus loin selon votre temps disponible et votre volonté de mieux comprendre et maîtriser ce domaine !

SEQUENCEMENT

Le séquencement de la trame de Travaux Pratiques correspond au chemin proposé afin d'atteindre les objectifs pédagogiques fixés. Ces objectifs ont été choisis au regard des attentes et exigences demandées par les marchés de l'industrie du logiciel et des couches basses des systèmes : systèmes embarqués, systèmes temps réel, développement de systèmes d'exploitation, développement de bibliothèques spécialisées, développement de chaînes de compilation, attaque et sécurité des systèmes, etc, tous des métiers dans divers domaines actuellement exercés par certains de nos anciens élèves. Il s'agit d'un séquencement conseillé qui n'a en aucune façon volonté à être imposé (étudiant comme enseignant encadrant). Pour se dérouler sous les meilleurs hospices, il serait cependant préférable dès le début de l'enseignement de rythmer 1 à 2 heures par semaine de travail personnel à la maison en dehors des séances en présentiel avec enseignant. Voici une proposition de séquencement (2h par créneau de TP) :

- Avant le début des TP : Lire le document de prélude, installer les outils de développement (IDE CCS 5.5, bibliothèques CSL et DSPLIB), valider l'exercice 2.1 (projet hello seulement) et faire le travail préparatoire n°1 (dsp-tp-preparations.pdf)
- Séance n°1 : Exercices 2.1 et 2.2
- Séance n°2 : Exercice 2.3 et 2.4 *(avant la séance, faire le travail préparatoire n°2)*
- Séance n°3 : Exercice 2.5
- Séance n°4 : Exercice 2.6
- Séance n°5 : Exercice 2.7
- Séance n°6 : Exercice 2.8
- Séance n°7 : Exercices 3.1 et 3.2
- Séance n°8 : Exercice 4.1 et 4.2 *(avant la séance, faire le travail préparatoire n°4)*
- Séance n°9 : Exercice 4.3
- Séance n°10 : Exercice 5.1, etc *(avant la séance, faire le travail préparatoire n°5)*

1.1. Optimisation algorithmique



Durant cette trame de travaux pratiques, nous allons nous intéresser au workflow typiquement rencontré en milieu industriel dans le cadre d'optimisations logicielles d'algorithmes embarqués sur une cible matérielle spécialisée. Ce processus de développement peut par exemple être rencontré chez les acteurs des grands domaines du traitement du signal (traitement d'antenne, traitement d'image, traitement du son, cloud computing, etc). Nous avons travaillé par le passé et travaillons encore à l'ENSICAEN avec plusieurs partenaires industriels ayant ces types de contraintes (THALES, SAFRAN, CANON, diverses entreprises en traitement d'image, etc).

Nous nous intéresserons à l'implémentation d'un algorithme simple et standard du domaine du traitement du signal, un filtre FIR ou produit scalaire (convolution discrète). Nous nous attarderons sur les stratégies d'optimisation pour une architecture matérielle spécialisée et non à la théorie mathématique associée. Tous nos développements seront guidés par le test, étape pouvant tenir une place très importante dans le temps de développement global d'une application et le Benchmarking (analyse comparative). Observons le workflow de la trame de travaux pratiques :

- Analyse mathématique théorique de l'algorithme sur ordinateur. Optimisation mathématique à cette étape afin de diminuer la complexité algorithmique (notation "grand O" ou "Omicron" de Landau) notamment en MAC (Multiply-Accumulate) pour un algorithme du TNS (Traitement Numérique du Signal)
- Modélisation, conception et validation durant les phases de prototypage sur ordinateur. L'outil logiciel de prototypage rapide le plus rencontré en milieu industriel à notre époque dans le domaine du traitement du signal est Matlab/Simulink (Scilab en version libre). Cette étape est essentielle afin de valider la structure en pseudo code des algorithmes, les procédures de test ainsi que les vecteurs d'entrée et de sortie
- Développement et validation sur cible des procédures de test. Dans le cadre de nos développements, nous nous intéresserons aux tests de conformité/validité (cohérence des valeurs de sortie par rapport à l'algorithme de référence) et de performance (mesures temporelles)
- Développement sur cible d'un algorithme de référence en C canonique ou C naturel
- Développements, tests et validations successives sur cible des stratégies d'optimisation sur architecture processeur spécialisée (vectorisation monocœur, parallélisation multicœur, gestion optimale de la hiérarchie mémoire L1/L2/L3, périphériques d'accélération, périphériques DMA, etc)

1.2. Convolution discrète ou filtre FIR

Cette trame de TP consiste à implémenter un algorithme optimisé de filtrage FIR (ou produit scalaire ou convolution discrète) sur une architecture DSP spécialisée pour du calcul numérique. Effectuons quelques rappels sur cet algorithme. Une implémentation courante de ce filtre consiste à appeler périodiquement (période d'échantillonnage) l'algorithme dans une optique de calcul en temps réel. A chaque appel, seul un échantillon de sortie est calculé puis traité :

$$y(k)=\sum_{j=0}^{N} a(j).x(k-j)$$

x() = vecteur d'échantillons d'entrée (taille égale à N)
a() = vecteur de coefficients
y(k) = échantillon courant de sortie
k = indice courant
N = ordre du filtre
N+1 = nombre de coefficients

L'implémentation vue en travaux pratiques se fera en temps différé. Nous calculerons un vecteur de sortie complet en traitant l'information depuis un vecteur d'entrée pouvant être très long (plusieurs Mo) et dans tous les cas de figure de taille supérieure ou égale au nombre de coefficients (ordre du filtre FIR + 1) :

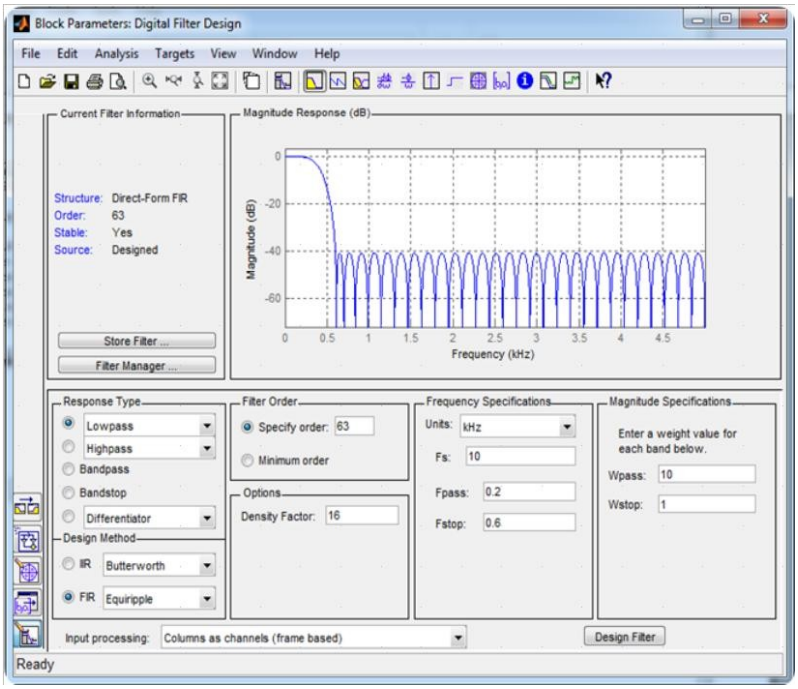
$$y(k)=\sum_{k=0}^{Y}\sum_{j=0}^{N} a(j).x(k-j)$$

x() = vecteur d'échantillons d'entrée (taille supérieure ou égale à N)
a() = vecteur de coefficients
y() = vecteur d'échantillons de sortie
k = indice courant
N = ordre du filtre
N+1 = nombre de coefficients
Y = taille du vecteur de sortie
Y+N-1 = taille du vecteur d'entrée

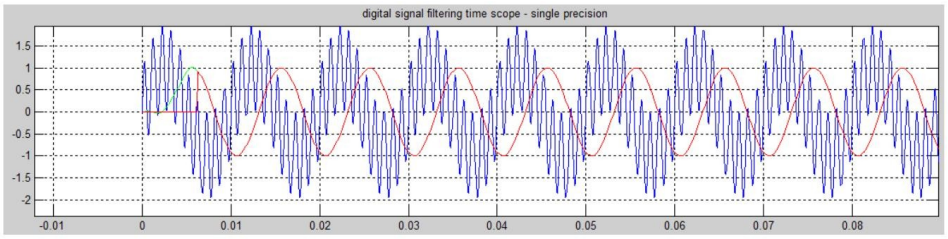
La phase de prototypage rapide sous environnement Matlab/Simulink ne sera pas à faire et vous est donnée dans le répertoire de projet /disco/matlab/. Elle est entièrement recouverte par le programme de 1ière année en Traitement Numérique du Signal. Nous nous attarderons uniquement sur les problématiques liées à l'intégration et l'optimisation sur DSP VLIW TMS320C6678. Observons l'implémentation en pseudo code Matlab au format flottant simple précision IEEE754 de l'algorithme de filtrage en temps différé précédemment présenté. Comprendre cet algorithme ...

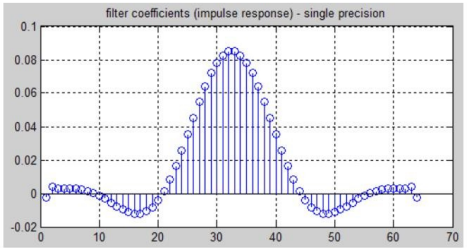
```
function yk = fir_sp(xk, coeff, coeffLength, ykLength)
    yk = single(zeros(1,ykLength));    % output array preallocation

    % output array loop
    for i=1:ykLength
        yk(i) = single(0);

        for j=1:coeffLength
            yk(i) = single(yk(i)) + single(coeff(j)) * single(xk(i+j-1));
        end
    end
end
```

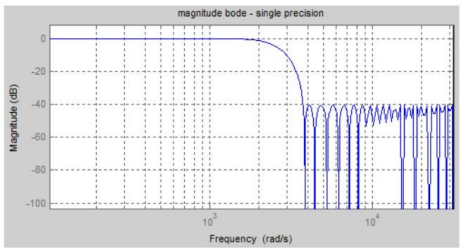
Le filtre à intégrer sur cible ainsi que les vecteurs de test d'entrée ont été générés et validés durant les phases de prototypage sous Matlab/Simulink en utilisant l'outil FDATool (cf. capture ci-dessous):



Le filtre FIR à intégrer est un filtre Passe Bas coupant à 200Hz pour une fréquence d'échantillonnage à 10KHz. Il s'agit d'un filtre ordre 63 comprenant donc 64 coefficients de symétrie paire en flottant simple précision IEEE-754 (cf. ci-dessous). Ce filtre offre un gain unitaire dans la bande passante et une atténuation de -40dB dans la bande coupée. La bande d'atténuation est comprise entre 200Hz et 600Hz. Ce filtre ne vise aucune application ou domaine spécifique et ne nous servira qu'à illustrer les concepts et stratégies d'optimisation sur processeur spécialisé. Le vecteur de test d'entrée est une simple somme de deux sinusoïdes à 100Hz et 1KHz (bleu ci-dessous). Après filtrage, seule l'harmonique à 100Hz (rouge ci-dessous) est identifiable :

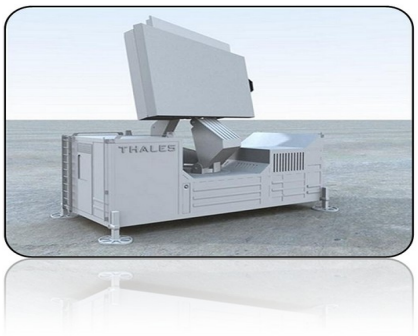






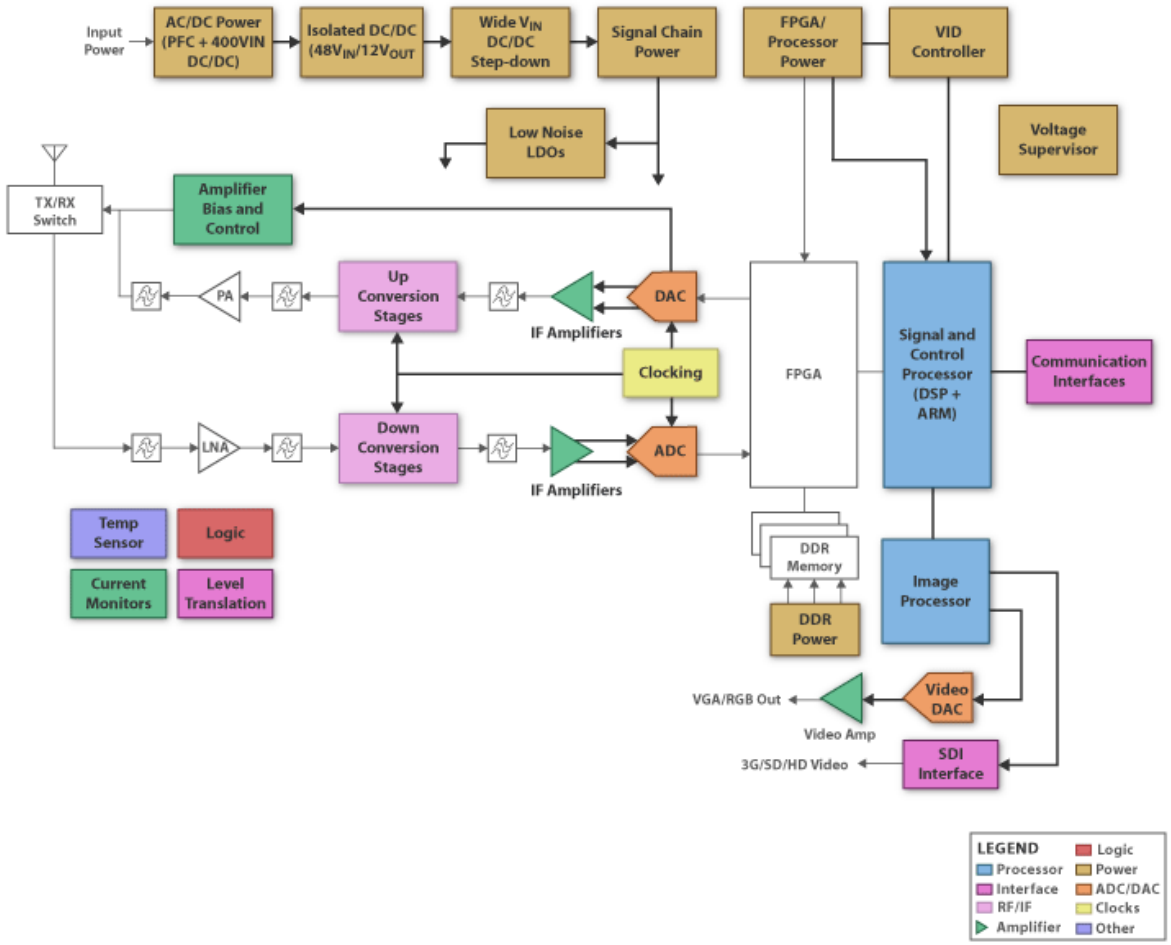

```
Fs = 10000;                     % sample frequency
Ts = 1/Fs;                      % period frequency
F1 = 100;                       % harmonic n°1 of input vector
F2 = 1000;                      % harmonic n°2 of input vector
XK_LENGTH = 2048;               % 2K/8Kb samples : length of input vector
t=0 : Ts : (XK_LENGTH-1)*Ts;    % time vector


xk = single(sin(2*pi*F1*t) + sin(2*pi*F2*t));
```

1.3. Exemple des applications Radar



Radar de défense aérienne GM 400 - THALES AIR SYSTEMS

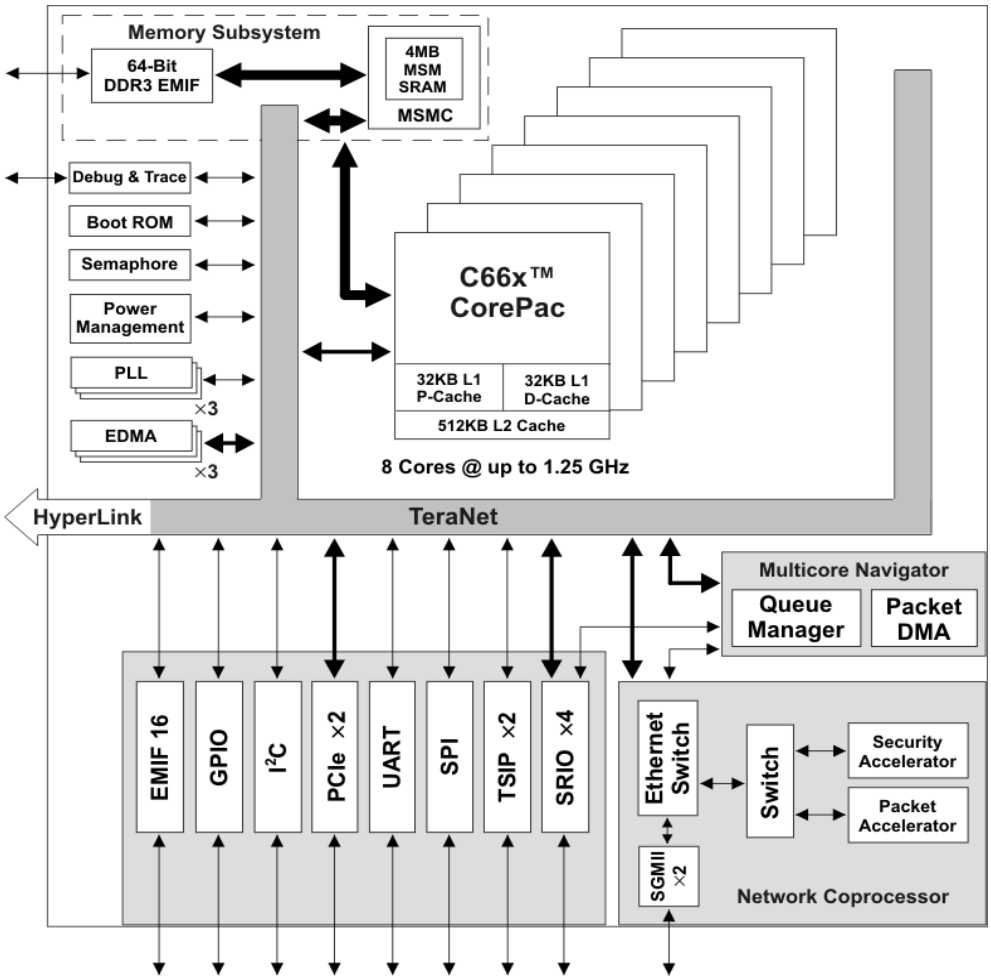
Afin de bien prendre conscience de l'intérêt de tel développement et phases d'optimisation, prenons l'exemple d'une application radar. Nous pouvons observer, ci-dessous, l'architecture matérielle typique d'un radar. Observons l'emplacement du processeur de traitement numérique du signal, dans le cas présent un DSP. Il faut savoir que d'autres familles de processeurs orientés calcul peuvent également remplir cette tâche (FPGA, MPPA, GPP voire GPU), chaque famille offrant son lot d'avantages et d'inconvénients.



Une chaîne numérique de traitement radar implémente également un produit scalaire comme celui étudié dans cette trame de travaux pratiques . A titre indicatif, sans optimisation l'algorithme du produit scalaire représente environ près de 40% du temps de traitement d'une chaîne Radar complète. Compte-tenu des contraintes temps réel excessivement lourdes imposées par ce type d'application, soit quelques Mo de données à traiter en quelques ms, nous pouvons pressentir tout l'intérêt de développer des bibliothèques de fonctions de calcul spécialisées pour l'architecture cible. Ce sera l'objectif premier de cet enseignement tout en respectant des méthodologies assurant un développement optimal.

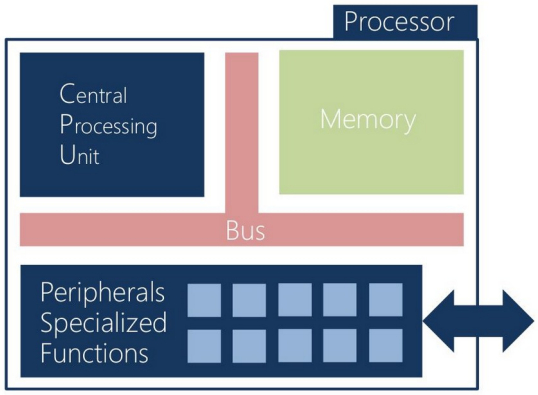
1.4. Architecture DSP VLIW C6600 de TI

Nos développements seront réalisés sur l'architecture DSP VLIW C6600 (Very Long Instruction Word) proposée par TI ou Texas Instruments (processeur utilisé par certains partenaires). Pour information, à notre époque la famille CPU C6000 de TI est l'architecture CPU leader sur le marché des processeurs DSP (Digital Signal Processor). Le processeur C6678 étudié en TP étant l'un des composants haut de gamme de la famille avec ses 8 cœurs vectoriels VLIW. Cette architecture propose quelques atouts assurant une grande flexibilité et permettant une bonne compréhension des architectures processeurs actuelles. Le processeur DSP TMS320C6678 offre les services matériels illustrés ci-dessous :



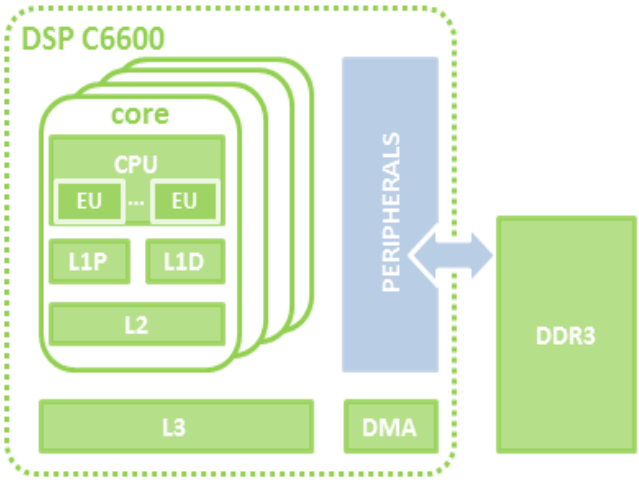
- 8 cœurs avec CPU vectoriels VLIW possédant chacun 32Ko caches L1P (L1-Program) et L1D (L1-Data), ainsi que 512Ko de cache L2 unifié (Program/Data)
- Chaque cœur peut être cadencé jusqu'à 1,4GHz
- Le niveau mémoire L3 unifié de 4Mo nommé MSM (Multi-core Shared Memory) est partagé entre cœurs
- Chaque niveau mémoire peut être configurable en cache ou en mémoire adressable SRAM permettant une architecture matérielle configurable en modèle mémoire uniforme UMA ou non-uniforme NUMA. Chose impossible sur processeur généraliste GPP Inte/AMD x86_64 par exemple.
- De la mémoire SDRAM DDR3 externe peut également être ajoutée sur circuit imprimé et est alors interconnectée via le périphérique d'interface EMIF (External Memory Interface).

En première année, nous avons découvert les bases du développement sur processeur numérique MCU (Micro Controller Unit) ou microcontrôleur. Ces processeurs étant généralistes et non spécialisés pour du calcul numérique, nous nous sommes donc assez longuement attardés sur les périphériques de communication (UART, I2C, etc) et d'interface (GPIO, ADC, etc) :



Cet enseignement est différent et doit être perçu comme une extension des compétences de première année. Cette année nous allons travailler sur machine fortement parallèle et nous nous efforcerons de comprendre puis d'exploiter au mieux les ressources matérielles proposées par notre processeur. Les périphériques d'interface ne seront donc pas vus, seule la partie traitement nous intéressera. Nous nous focaliserons sur :

- CPU vectoriel VLIW (pipelining software d'instructions et vectorisation des données)
- Mémoires L1/L2/L3 configurables en SRAM adressable ou en Cache et mémoire DDR
- Périphériques DMA (Direct Memory Access) de copie mémoire (copies sans passer par le CPU)



Même si l'exemple qui suit n'a que peu de sens, afin de bien comprendre les différences majeures entre les architectures de première année et de cette année, comparons les performances théoriques maximales des deux processeurs. Avec ces quelques chiffres, nous pouvons commencer à pressentir le potentiel pour du calcul numérique flottant de cette famille de processeur.

- Un MCU 8bits entier PIC18F27K40 de Microchip peut exécuter jusqu'à 16MIPS (soit 16 millions instructions entières 8bits par seconde)
- Un DSP VLIW 32bits flottant TMS320C6678 de Texas Instruments peut exécuter jusqu'à 22,4GFLOP/core (soit 22,4 Giga instructions flottantes 32bits en simple précision IEEE-754 par seconde et par cœur). Soit jusqu'à 179,2GFLOP pour le processeur complet grâce à ses 8 cœurs, le tout avec une horloge à 1,4GHz.

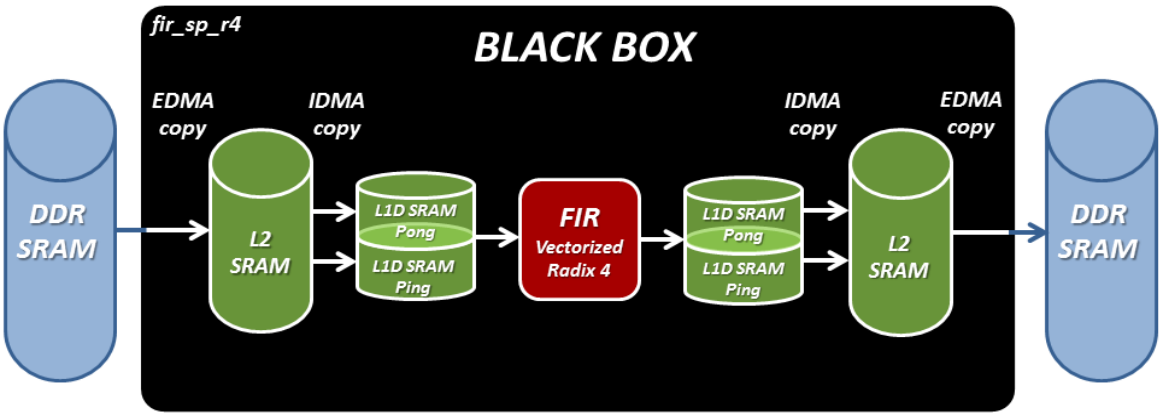
1.5. Objectifs pédagogiques

Nous venons de présenter un rapide tour d'horizon du potentiel de notre architecture processeur. Nous comprenons mieux le sens du nom de l'enseignement ''Architectures pour le calcul''. Beaucoup des **mécanismes d'accélération déterministes** (temps réel) présentés sont impossibles ou en tous cas moins performants ou moins déterministes sur architectures généralistes (GPP ou AP avec MMU/Cache). Néanmoins, une bonne compréhension des stratégies présentées précédemment vous assurera une adaptabilité forte pour des problématiques d'optimisation sur la plupart des architectures processeurs parallèles du marché (GPP, DSP, MPPA, GPU et MCU vectoriel). En milieu industriel, les ingénieurs bas niveau chargés du développement des bibliothèques spécialisées et système possèdent des compétences différentes des ingénieurs haut niveau assurant l'intégration logicielle et le développement des applications :

- **Développeur système temps réel bas niveau (système, driver ou algorithmique)** : ingénieur spécialisé dans les architectures matérielles processeurs et le développement de bibliothèques spécialisées. Maîtrise forte de l'architecture, des outils de développement, des langages C/ASM et du fonctionnement de la machine.

- **Développeur logiciel applicatif haut niveau** : concepteur et intégrateur applicatif haut niveau travaillant dans les couches hautes de l'application et utilisant les API et bibliothèques en mode boîtes noires (faible voire aucune idée de l'implémentation réelle bas niveau). Maîtrise faible de l'architecture et du fonctionnement de la machine.

```
#define XK_LENGTH 2048        /* 8kb vector*/
#define A_LENGTH 64
#define YK_LENGTH XK_LENGTH - A_LENGTH + 1

float32_t xk_sp[XK_LENGTH];
float32_t a_sp[A_LENGTH]={-0.00244444,0.00380928,...};
float32_t yk_sp[YK_LENGTH];

fir_sp_r4 (xk_sp, a_sp, yk_sp, A_LENGTH, YK_LENGTH);
```



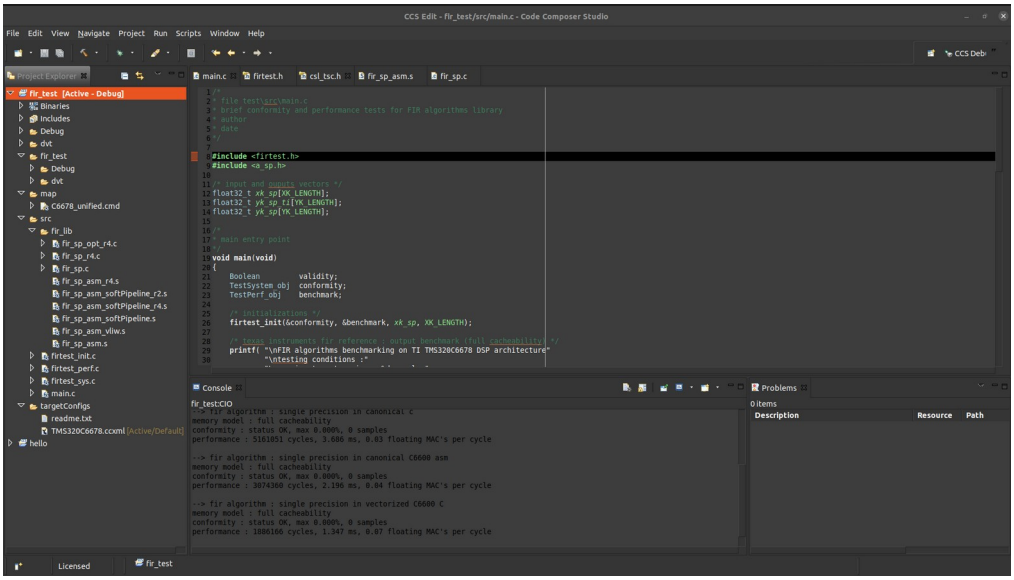
Cette trame d'enseignement vise donc à ouvrir les portes des stages et des métiers dédiés à l'optimisation et l'accélération algorithmique sur processeur spécialisé. Ces compétences, rares sur le marché, sont demandées par des entreprises ayant ce type de besoin spécifique (THALES, SAFRAN, ARIANE GROUP, sociétés en traitement d'image, cloud computing, etc). Par exemple, **un élève ingénieur réalisant la trame complète, aura développé en fin d'enseignement un seul et unique algorithme fir_sp_r4** (~30-40h de développement équivalent à une voire deux semaines entreprise). Cependant cet algorithme implémente certaines subtilités d'intégration :

- Copies montantes et descendantes des données entre DDR et L2 SRAM réalisées par DMA

- Copies montantes et descendantes des données entre L2 SRAM et L1D SRAM avec alternance des buffers de stockage (ping-pong) par IDMA interne à chaque Cœur

- Vectorisation C intrinsèque avec déroulement de boucle en base 4 sur CPU DSP VLIW

1.6. Outils de développement

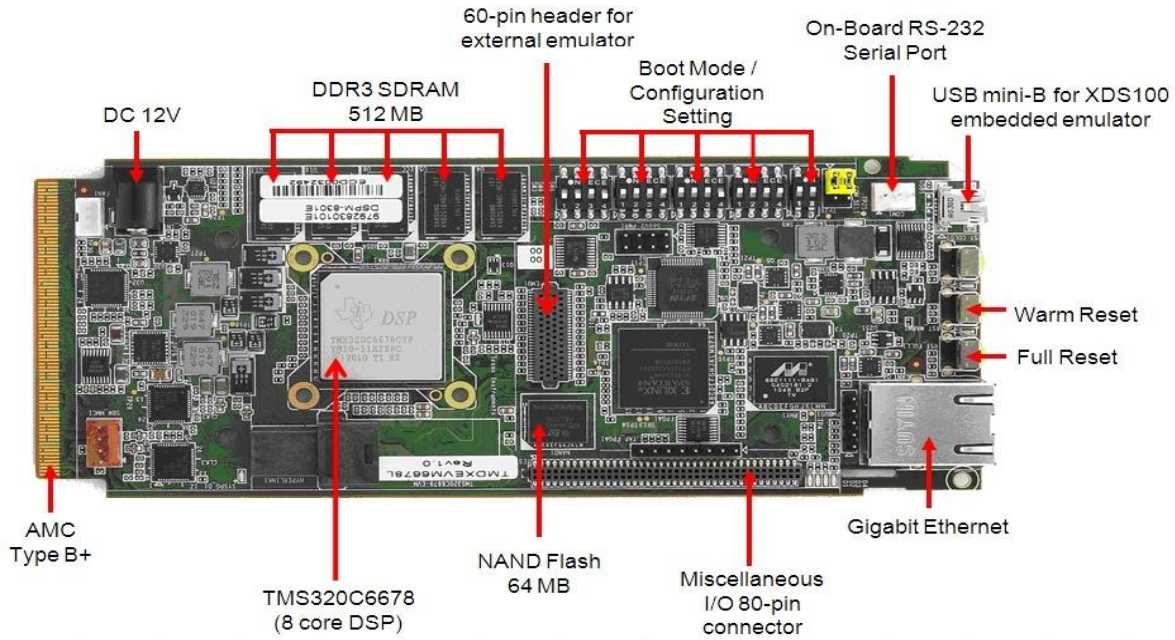


Nous développerons sous IDE CCS5.5 (Code Composer Studio) développé par Texas Instruments et basé sur un framework libre Eclipse. Les outils de développement et bibliothèques sont librement téléchargeables et installables depuis internet. Se référer à la page moodle de l'enseignement pour l'installation des outils de développement (section – OUTILS DE DEVELOPPEMENT):

https://foad.ensicaen.fr/course/view.php?id=117

La trame de travaux pratiques sera réalisée sur la plateforme de développement TMDXEVM6678L EVM (Evaluation Module) proposée par la société Advantec. Cette maquette d'évaluation embarque notamment une sonde de programmation USB XDS100 assurant la programmation et le débogage des applicatifs. Pour information, les outils de développement logiciel deviennent payants si nous souhaitons travailler avec des sondes d'émulation évoluées, telle que la sonde XDS560 également présente à l'école pour des phases de projet industriel. :

https://www.ti.com/tool/TMDSEVM6678



1.7. Benchmarking

ANALYSE COMPARATIVE

Algorithme vecteur d'entré 2K samples ou 8Ko	Architecture	Temps d'exécution ms / Cycles	Performances < 8 MACS max.	Temps de Développement heures	Observations et Limitations
DSPF_sp_fir_gen	Texas Instr. DSP VLIW C6600	0.031ms 43695cy	3.0	0	cf. documentation DSPLIB TI *nr and nh are a multiples of 4 and greaters than or equals to 4. x, h and r are double-word aligned. Interruptible code.*
	Intel GPP superscalar corei7 Haswell IA-64 A203/A201				