Architecture des DSP tms320C6x

Architecture des DSP tms320C6x

I. Introduction

Le traitement numérique du signal est au «cœur» de la plupart des technologies que nous utilisons aujourd'hui et à titre d'exemples nous citons :

- ✓ Les téléphones portables utilisent le traitement numérique du signal pour générer les multifréquences nécessaires et les envoyées aux réseaux sans fil.
- ✓ Les écouteurs anti-bruit utilisent une technologie adaptative de traitement numérique du signal pour supprimer le bruit dans l'environnement qui nous entoure.
- ✓ Les Caméras digitales utilisent le traitement numérique du signal pour compresser les images au format JPEG pour un stockage fiable afin que nous puissions stocker des centaines d'images sur une seule carte mémoire. …etc.

Le cœur réel d'une structure de traitement numérique du signal est le processeur DSP (Digital Signal Processor).
Dans cette partie du cours nous allons voir d'une manière très simplifiée l'architecture interne de ce circuit qui est soigneusement adapté et conçu sur mesure pour ce type d'applications.

II. Les algorithmes typiques du traitement numérique du signal et le temps réel :

Dans une structure de traitement numérique du signal Le signal analogique est converti en numérique via un convertisseur analogique-numérique (ADC) qui produit une série d'échantillons numériques. Ces échantillons convertis sont envoyés, à la cadence de la fréquence d'échantillonnage du signal d'entrée, au DSP pour le traitement. Ce traitement doit s'effectuer en temps réel.

Parmi les algorithmes typiques du traitement numérique du signal on peut citer :

- ❖ Filtre FIR « **F**inite **I**mpulse **R**esponse Filtre »

  $Y(n)=\sum_{i=0}^{i=N} a(i).x(n-i)$

- ❖ Filtre IIR « **I**nfinite **I**mpulse **F**ilter »

  $Y(n)=\sum_{i=0}^{i=N} a(i).x(n-i)+\sum_{i=0}^{i=M} b(i).y(n-i)$

- ❖ Convolution :

  $Y(n)=\sum_{i=0}^{i=N} x(i).h(n-i)$

- ❖ Transformation de Fourier discrète « DFT »

  $Y(n)=\sum_{i=0}^{i=N-1} x(n)\exp[-j(2\Pi/N)ni]$

La définition du temps réel dépend de l'application. Par exemple, si la fréquence la plus élevée du signal analogique entrant est 4KHz (comme la parole), le théorème de Nyquist dit que vous devez échantillonner à plus de 8 KHz (x2) pour éviter le **repliement de spectre**.

Pour une Fréquence d'échantillonnage de 10 KHz, le temps entre deux échantillons successifs est de : Te= 100μsec.

Si l'algorithme sélectionné nécessite 100 instructions par période d'échantillonnage, chaque instruction doit s'exécuter en 1μsec pour répondre à la définition du temps réel pour ce système.

Selon les normes d'aujourd'hui, 1μsec comme temps d'exécution d'une instruction est très lent - la plage de temps de cycle DSP typique est de 20 à 50 ns ou même plus rapide. Ce niveau d'augmentation des performances permet aux utilisateurs :

- L'échantillonnage des signaux d'entrée de fréquence plus élevée
- L'augmentation de la bande passante pour prendre en charge plus de fonctionnalités avec un seul processeur
- La combinaison de plusieurs applications ou plusieurs instances d'une application en un seul processeur (par exemple, au lieu d'un processeur dédié à un seul modem, plusieurs modems peuvent partager un processeur, ce qui diminue la taille et le coût.).

III. Architecture Interne du DSP tms320c6x :

La figure ci-dessous présente le schéma de principe des processeurs DSP TMS320C6000 de Texas Instruments. Ces processeurs intègrent trois parties principales :

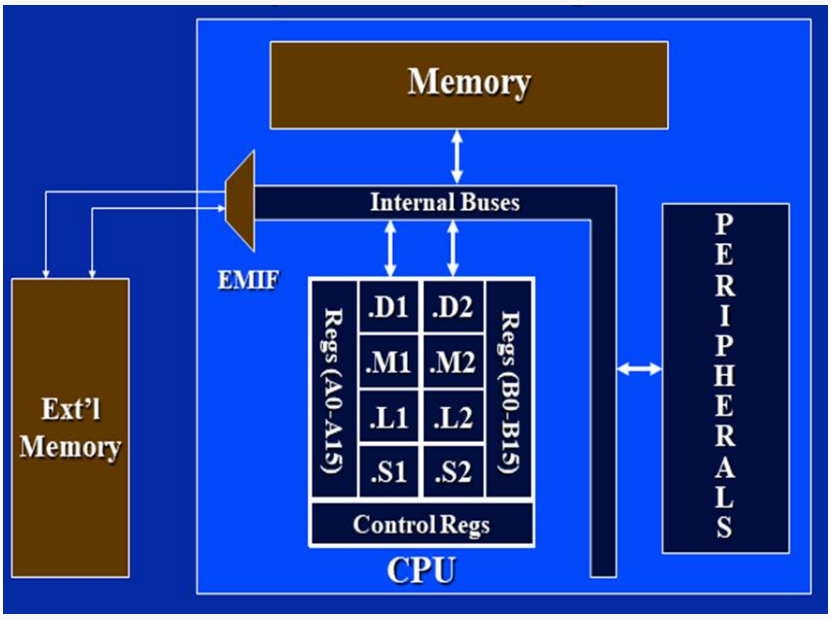


Fig1 :Schéma synoptique des DSP tms320c6x

- Une unité centrale CPU : Central Processing Unit « the processor "**core**" »
- Une mémoire cache à deux niveaux séparée en mémoire interne de donnée et mémoire interne de programme
- Des périphériques tels qu'un contrôleur d'accès direct à la mémoire (**EDMA**), une logique de mise hors tension et une interface de mémoire externe (**EMIF**)…etc.

Cette architecture qui adopte la technologie **VLIW** « Very Long Instruction Word », utilise des paquets ou des mots d'instructions de 256 bits autorisant le traitement parallèle de 8 instructions élémentaires de 32 bits. De plus une organisation Harvard est adoptée pour les mémoires et qui permet d'accéder simultanément à deux blocs distincts de mémoire, la mémoire des programmes et celle des données.

IV. Unité Centrale CPU :

La plupart de ces algorithmes (les filtres, la convolution, les FFT, etc…..), peuvent tous se résumer à une simple équation en la somme des produits comme indiqué ci-dessus.

$$Y= \sum_{i=0}^{i=n} a_i.x_i \qquad (*)$$

Les composantes fondamentales de cette équation Incluent :

- La multiplication,
- L'addition,
- Le bouclage
- L'acquisition de nouvelles valeurs de données.

Il y'a deux opérations de base « La **multiplication** et L'**addition** ». Ces opérations sont faciles à réaliser (les calculatrices les font depuis des années), mais deviennent plus difficiles pour être exécuter en temps réel. Avant les années 1970, les fonctions de multiplication et d'accumulation en temps réel nécessitent un matériel coûteux comme les processeurs matriciels et les superordinateurs avec « pipelined instructions sets » pour suivre en temps réel un signal analogique entrant. Aujourd'hui, un seul processeur peut gérer facilement ces exigences de performances.

Nous présentons dans ce qui suit l'exécution de l'algorithme de l'équation (*) par le processeur DSP toute en développant l'architecture interne des DSP de la famille des tms320c6x de Texas Instrument.

III.1.1 - La multiplication :

Commençons par écrire le code assembleur à performer par le dsp tms320c6x pour le calcul de Y:

```
MPY   a,x,prod        ;prod = a . x
```

L'unité matérielle qui effectue ce produit est bien sur un multiplieur c'est l'unité de multiplication appelée l'unité « **.M** ».

L'unité **.M** reçoit comme entrée les variables «**a**» et «**x**» comme entrées et donne le produit de la multiplication dans la variable "**prod**".

Les tms320c6x utilisent un multiplieur 16 bits qui fournit un résultat de produit sur 32 bits.

III.1.2- L 'addition :

La prochaine étape consiste à accumuler le produit « prod » dans un registre de sommation, qui sera notre dernier résultat «Y»:

```
ADD   Y, prod, Y          ; Y = Y + prod
```

L'instruction ADD ajoute Y à prod et stocke le résultat dans Y, c'est une accumulation.

En plus du multiplieur, nous aurons besoin d'une autre unité fonctionnelle qui joue le rôle de « **L'additionneur** ».Le tms320c6x dispose d'une unité arithmétique et logique qui effectue non seulement l'addition ou la soustraction sur 8, 16,32 bits, mais effectue également des opérations logiques telles que ET, OU, XOR…etc. Cette unité est appelée l'unité « **.L** ».

III.1.3- Les registre :

Pour garantir un accès facile et rapide aux variables, le tms320c6x offre un banc de registres directement connecté aux unités fonctionnelles. Ce banc de registres « Register file A » contient seize (16) registres 32 bits appelés A0-A1-….-A15. Ces registres stockent le contenu de nos variables et fournissent un accès rapide à l'intérieur et à l'extérieur des unités fonctionnelles. Nous stockons :

- Le coefficient **a** dans le registre A0;
- La variable **x** dans le registre A1,
- Le produit **prod** dans le registre A3; et le résultat Y dans le registre A4.

La syntaxe de notre programme devient alors :

```
MPY   .M1   A0, A1, A3
ADD   .L1   A4, A3, A4
```

Maintenant que nous avons terminé la multiplication et l'addition la prochaine étape selon l'équation (*) est de répéter ces opérations n fois dans une boucle bien sûr.

III.1.4- La création de boucle :

La création d'une boucle comporte quatre étapes de base. Voyons comment chacune de ces opérations est effectuée sur le tms320c6x :

1- Définir le début et la fin de la boucle par une instruction de branchement :

Le tms320c6x dispose d'une 3éme unité fonctionnelle appelée « **.S** » qui effectue le branchement. En plus de la gestion des branchements cette unité effectue également des opérations de décalage « **S**hift operations ». Le programme s'écrit alors :

```
loop:
MPY   .M1   A0, A1, A3
ADD   .L1   A4, A3, A4
B     .S1   loop
```

2- Créer un compteur de boucle (dans notre cas on a choisi n=40) :

L'étape suivante est la création d'un compteur de boucle avec une valeur initiale égale à 40, Nous avons choisi le registre **A2** comme compteur de boucle.

On utilise alors l'unité de gestion des boucles **.S1** et l'instruction MVK qui permet de charger une valeur sur 16 bits dans la moitié inférieure d'un registre à 32 bits.

```
MVK .S1  40, A2        ; A2  est défini comme compteur de boucle A2= 40
```

3- Ajoutez une instruction pour décrémenter le compteur de boucle :

On utilise l'unité **.L1** et l'instruction **SUB** pour décrémenter le registre compteur A2 à l'intérieur de la boucle.

```
SUB   .L1   A2, 1, A2
```

4- Mettre la condition de branchement sur la valeur du compteur de boucle :

L'instruction de branchement doit s'exécuter conditionnellement sur la base de la valeur de A2 (si A2 # 0, branchement).Nous pouvons accomplir cela en ajoutant [A2] avant l'instruction de branchement.

Le programme s'écrit alors :

```
            MVK   .S1   40, A2
loop:       MPY   .M1   A0, A1, A3
            ADD   .L1   A4, A3, A4
            SUB   .L1   A2, 1, A2
[A2]        B     .S1   loop
```

III.1.5- La création de pointeurs :

Les valeurs des variables a_i et x_i de l'équation(*) sont initialement stockées en mémoire interne ou externe du processeur. Pour les charger dans les registres appropriés Il faut créer un **pointeur** sur la variable. Le pointeur contient l'adresse de notre variable.

Après avoir créé un pointeur, l'étape suivante consiste à utiliser une instruction de chargement (LD) pour charger les variables «a» et "x" dans les registres appropriés et une instruction de stockage (ST) pour stocker le résultat dans Y.

Les pointeurs des variables:

```
A5 = &a
A6 = &x
A7 = &Y
```

Utilisation des pointeurs pour le chargement de registres (A0 et A1) par les données variables adressées par les pointeurs *A5 et *A6 ou le stockage de contenu de registre dans le registres pointeur pour le résultat mémoire adressée par *A7. (Voir Fig2)

```
LD      *A5, A0
LD      *A6, A1
ST      A4, *A7
```

III.1.6- Chargement/Stockage des données:

Pour faciliter les opérations de chargement**/**stockage (Load/Store operations), nous utilisons la 4ème et dernière unité fonctionnelle - l'unité « **.D »** du dsp tms320c6x.

C'est l'unité de stockage et de chargement ou de -trafic- des données mémoires internes ou externes du dsp. Cette unité est connectée d'une part aux registres et d'autre part à la mémoire du processeur dsp.

Les instructions ci-dessus s'écrivent alors comme suit en spécifiant l'unité **.D1** de chargement et stockage de données. Voir Fig 2

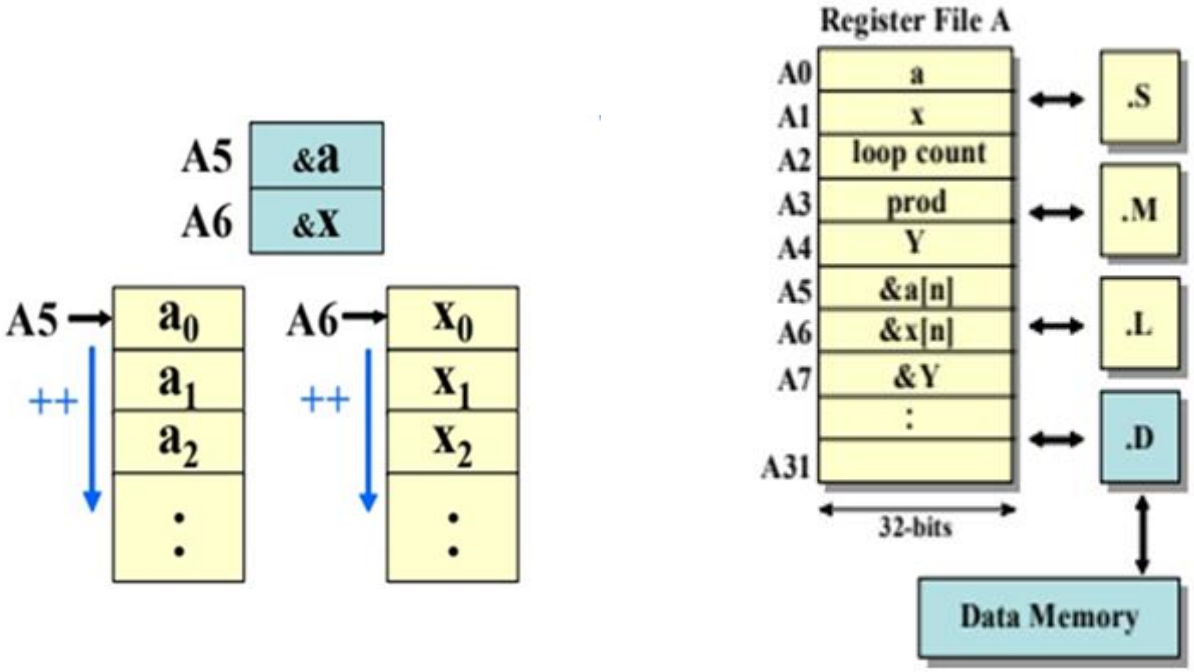


Fig2 :Pointeurs des variables a_i et x_i

Le programme :

```
        MVK     .S1     a,A5            ;move address of a
        MVKH    .S1     a,A5            ;into register A5
        MVK     .S1     x,A6            ;move address of x
        MVKH    .S1     x,A6            ;into register A6
        MVK     .S1     y,A7            ;move address of y
        MVKH    .S1     y,A7            ;into register A7
        MVK     .S1     40,A2           ;A2=40, loop counter
loop:   LDH     .D1     *A5++,A0        ;A0=an
        LDH     .D1     *A6++,A1        ;A1=xn
        MPY     .M1     A0,A1,A3        ;A3=an*xn, product
        ADD     .L1     A3,A4,A4        ;y=y+A3
        SUB     .L1     A2,1,A2         ;decrement loop counter
[A2]    B       .S1     loop            ;if A2≠0 branch to loop
        STH     .D1     A4,*A7          ;*A7=y
```

III.1.7- Organisation des unités fonctionnelles et des registres :

Pour doubler efficacement le parallélisme et augmenter la bande passante disponible, Texas Instrument, le concepteur de la famille des DSP tms320c6x, a ajouté quatre unités fonctionnelles et 16 registres. la figure ci-dessous présente l'organisation et la dénomination adoptée pour l'ensemble des unités fonctionnelles associées à leurs registres.

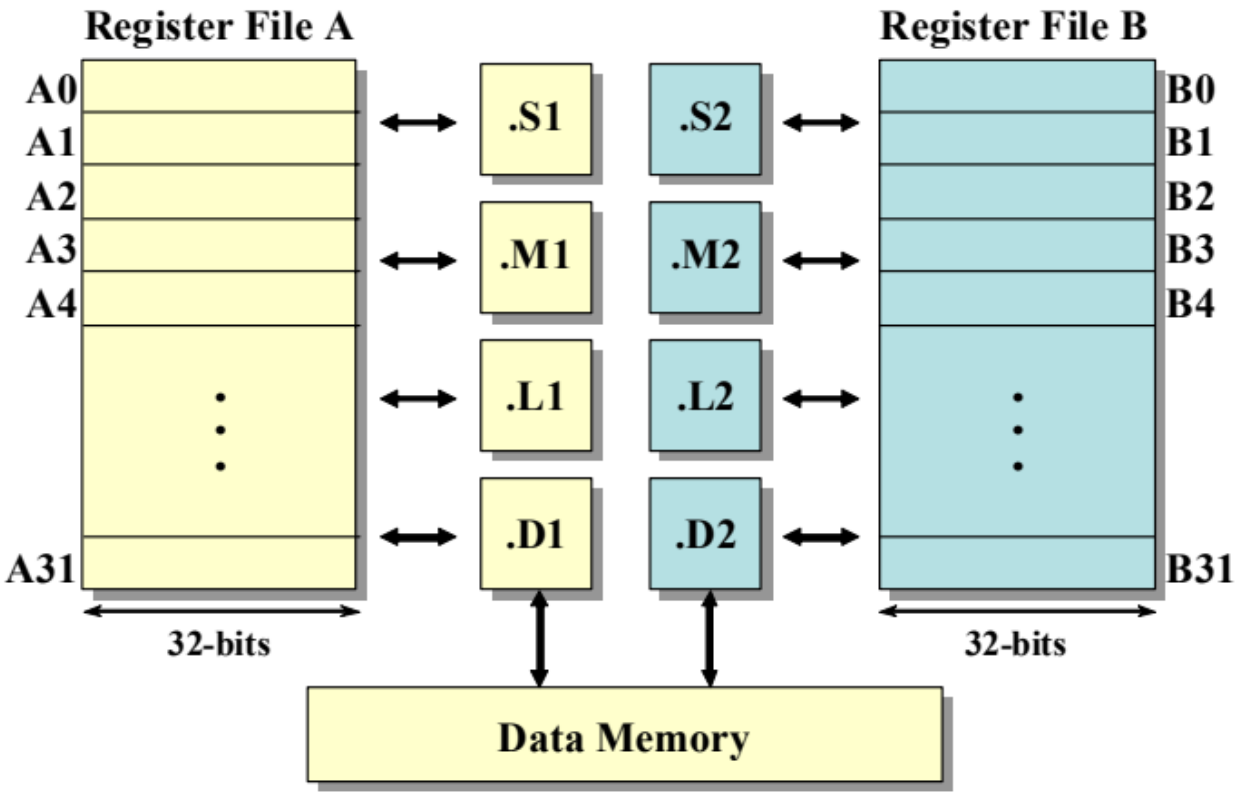


Fig3 : Unités Fonctionnelles du tms320c6x.

L'ensemble est organisé en deux chemins « **PATH** » :

PATH A Contient :

- Quatre unités fonctionnelles **.S1**, **.M1**, **.L1**, **.D1**
- Seize registres à 32 bits **A0**,**A1**,…..,**A15**

PATH B Contient :

- Quatre unités fonctionnelles **.S2, .M2, .L2 et .D2**
- Seize registres à 32 bits **B0**,**B1**,…....,**B15**

Ces processeurs sont capables d'exécuter un maximum de 8 instructions en parallèle en un seul cycle d'horloge. Le schéma fonctionnel générique des processeurs DSP de la famille des tms320c6x est présenté par la figure 5.

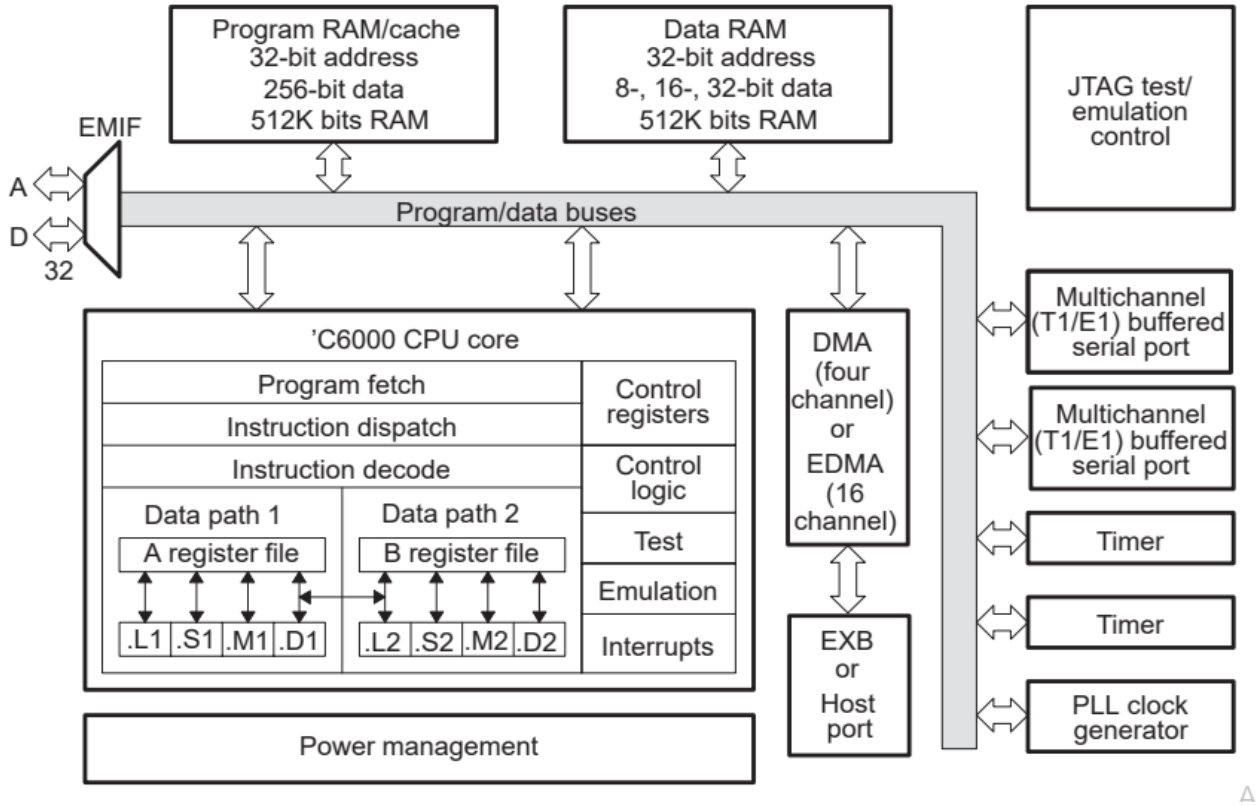


Fig1 : Schéma fonctionnel du TMS320C62x / C67x

Le tableau ci-dessous résume les différentes opérations exécutées par les unités fonctionnelles :

. Functional Unit	**Fixed-Point Operations**	**Floating-Point Operations**
.L unit (.L1,.L2)	32/40-bit arithmetic and compare operations Leftmost 1 or 0 bit counting for 32 bits Normalization count for 32 and 40 bits 32-bit logical operations	Arithmetic operations Conversion operations: DP → SP, INT → DP, INT → SP
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from the control register file (.S2 only)	Compare reciprocal and reciprocal square-root operations Absolute value operations SP to DP conversion operations
.M unit (.M1, .M2)	16 x 16 bit multiply operations	32 x 32 bit multiply operations Floating-point multiply operations
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with a 5-bit constant offset Loads and stores with a 15-bit constant offset (.D2 only)	Load double word with a 5-bit constant offset

III.1.8- L'unité « program fetch » :

La multiplicité d'unités fonctionnelles dans les dsp tms320c6x de Texas Instrument autorise l'exécution de plusieurs instructions en parallèle.

L'unité **program fetch** du CPU figure 1 récupère les instructions par paquet de huit instructions à 32 bits « Fetch Packet **FP**» à chaque cycle d'horloge du processeur dsp.

Un paquet d'exécution (**EP**) consiste en un groupe d'instructions qui peuvent être exécuté en parallèle dans le même temps de cycle.

Le nombre de paquets **EP** dans un paquet **FP** peut varier de un (avec huit instructions parallèles) à huit (sans aucune Instruction en parallèle).

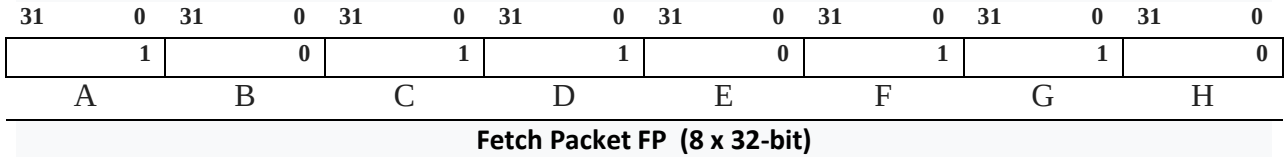
Le bit le moins significatif du code de chaque instruction 32 bits est utilisé pour déterminer si l'instruction suivante appartient au même EP (le bit LSB = 1) ou fait partie du prochain EP (le bit LSB = 0).

Considérons un FP composé de trois EP avec:

```
        Instruction A
||      Instruction B
        Instruction C
||      Instruction D
||      Instruction E
        Instruction F
||      Instruction G
||      Instruction H
```

EP1 : Deux instructions parallèles A et B,
EP2 : Trois instructions parallèles C, D et E
EP3 : Trois instructions parallèles F, G et H

Un **Fetch Packet** avec 3 **Execution Packet**, le bit LSB de chaque instruction sur 32 bits et positionné comme suit Voir figure 4 :



Fetch Packet FP (8 x 32-bit)

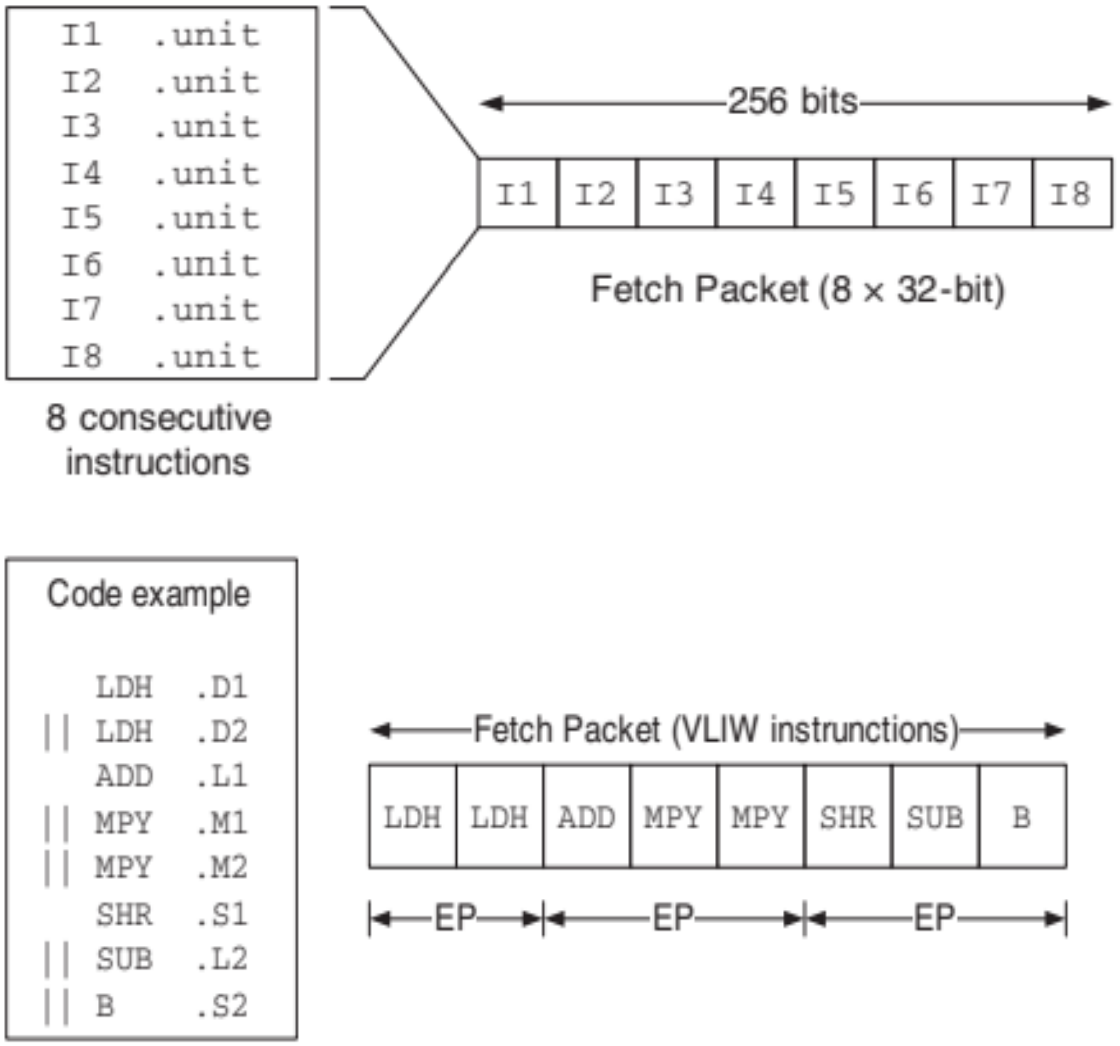


Fig 4: Fetch Packet et Execut Packet

III.1.9- Le principe du Pipeline

L'exécution d'une *instruction par le CPU d'un processeur s'effectue en trois* étapes *différentes :*

- Recherche ou récupération du code,
- Décodage et
- Exécution de l'instruction.

De nombreux processeurs à usage général effectuent ces étapes de manière série, l'instruction prend plusieurs cycles pour être achever.

Cependant, comme le processeur tms320c6x dispose de plusieurs unités fonctionnelles, de mémoires internes en structure Harvard et de bus multiples, ce processeur augmente ces performances par chevauchement des étapes d'exécution des instructions. Ce procédé est le pipeline.

Les trois étapes de pipelining sont :

Program Fetch				Decode		Execute
PG	PS	PW	PR	DP	DC	E1–E6 (E1–E10 for double precision)

1. **The program fetch stage** is composed of four phases:

   (a) **PG:** program address generate (in the CPU) to fetch an address
   (b) **PS:** program address send (to memory) to send the address
   (c) PW: program address ready wait (memory read) to wait for data
   (d) **PR:** program fetch packet receive (at the CPU) to read op-code from memory

2. **The decode stage** is composed of two phases:

   (a) **DP:** to dispatch all the instructions within an FP to the appropriate functional units
   (b) **DC:** instruction decode

3. **The execute stage** is composed of from six phases (with fixed point) to 10 phases (with floating point), due to delays (latencies) associated with the following instructions:

   (a) Multiply instruction, which consists of two phases due to one delay
   (b) Load instruction, which consists of five phases due to four delays
   (c) Branch instruction, which consists of six phases due to five delays

1	2	3	4	5	6	7	8	9	10	11	12
PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6
	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
		PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
			PG	PS	PW	PR	DP	DC	E1	E2	E3
				PG	PS	PW	PR	DP	DC	E1	E2
					PG	PS	PW	PR	DP	DC	E1
						PG	PS	PW	PR	DP	DC

- La plupart des instructions ont une phase d'exécution.
- Des instructions telles que multiplier (MPY), (LDH / LDW) et l'instruction de Branchement prennent respectivement deux, cinq et six phases d'exécution.
- Des phases d'exécution supplémentaires sont associées aux instructions de types virgule flottante à double précision jusqu'à 10 phases.

v. Mémoire Interne :

Les C62x / C64x / C67x ont un espace d'adressage à 32 bits **adressable en octet**. Ces processeurs conçus en technologie VLIW ont une mémoire interne, sur puce, organisée dans deux espaces séparés selon une Architecture de Harvard, en mémoire de données et mémoires de programmes.

La mémoire interne est composée de :

Les C62x / C64x / C67x ont un espace d'adressage à 32 bits **adressable en octet**.L'espace mémoire total adressable est d'une capacité totale de 4 Go ce qui correspondant à une représentation interne de l'adresse sur 32 bits. Cet espace mémoire est divisé en :

- Mémoire interne de programme,
- Mémoire interne de données,
- mémoire externes et
- périphérique interne.

La capacité et l'emplacement de la mémoire interne dépendent du processeur dsp choisi.

Les processeurs'C6201,' C6202 et 'C6701 ont des mémoires internes de programme et de données distinctes (architecture **Harvard**), tandis que pour le processeur ' C6211 une partie de sa mémoire interne peut être utilisée comme mémoire de programmes ou mémoire de données.

TMS320C6701

Le processeur TMS320C6701 possède 64 Ko de mémoire interne de programme et 64 Ko de mémoire interne de données,

- La mémoire programme, configurable comme cache ou programme, est organisé en 2K de paquets de récupération (Fetch Packet **FP**) de 256 bits. Les paquets sont traités à la vitesse maximale d'un paquet de huit instructions à 32 bits par cycle CPU , ou au moins une instruction par cycle.
- La mémoire de données interne se compose de deux blocs de huit banques à 16 bits, plutôt que de quatre banques à16 bits. Cette fonctionnalité permet des accès parallèles à double précision par le CPU dans le même cycle qu'un accès aux données par le DMA. Avec la nouvelle configuration de la mémoire, l'accès maximal aux données à chaque cycle est de deux accès CPU 64 bits (L'instruction LDDW uniquement) et d'un accès DMA à 32 bits.

vii. Accès à la mémoire des données

Pour les 'C6201,' C6202 et 'C6701, le contrôleur de mémoire de données traite toutes les requêtes vers la mémoire de données interne par le processeur ou le DMA. Le 'C6211 possède un contrôleur de cache de données de niveau un (L1D) et un contrôleur de cache de niveau deux (L2). Le contrôleur L1D traite les demandes de la CPU et les envoie au contrôleur L2 en cas de lecture ou d'écriture. Les demandes de données par l'EDMA vont directement au contrôleur L2.

La CPU envoie des requêtes au contrôleur de mémoire de données via les deux bus d'adresse (DA1 et DA2). Les données à stocker sont transmises via les bus de stockage de

données CPU (ST1 et ST2). Les données de chargement sont reçues via les bus de chargement de données CPU (LD1 et LD2). Les demandes de données CPU sont mappées en fonction de la plage d'adresses mémoire des données demandées à la mémoire de données interne, à l'espace périphérique interne (via le contrôleur de bus périphérique) ou à l'interface de mémoire externe. Le contrôleur de mémoire de données se connecte également à la mémoire de données interne et effectue un arbitrage CPU / DMA pour la RAM de données sur puce.

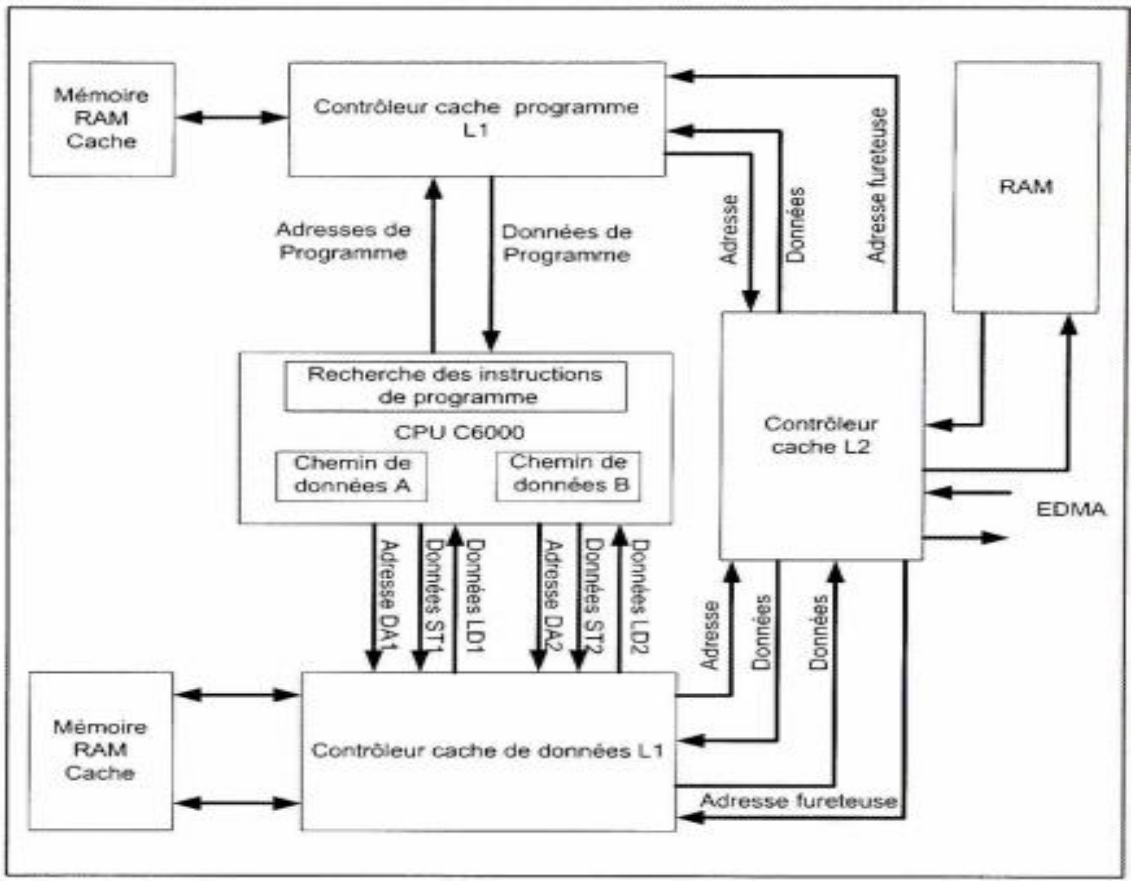


Figure 3.4 **Diagramme-bloc de la mémoire interne**

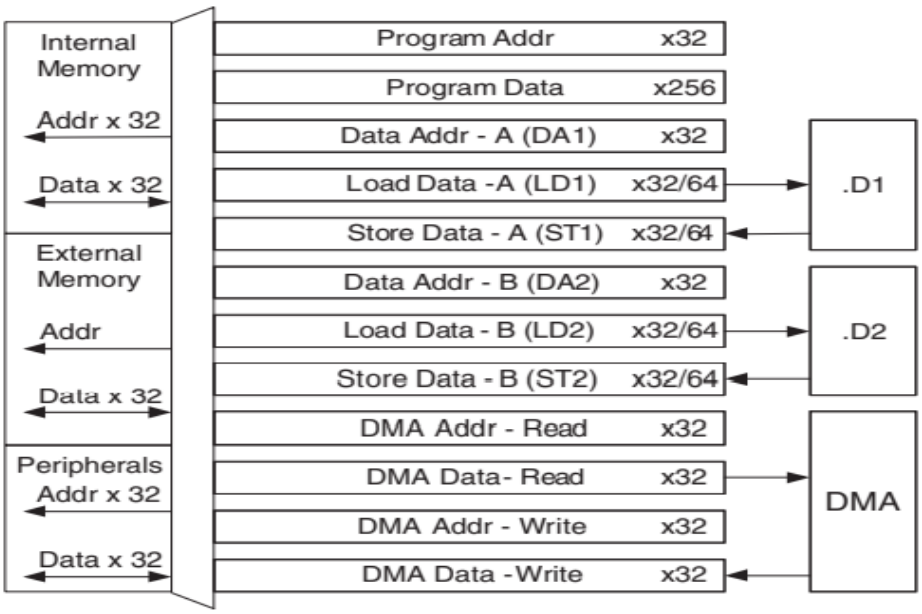
Le tableau ci-dessous donne la cartographie du Plan d'adressage du C6711

Attribut associé	Adresse	Taille (bytes)
RAM interne	0000 0000	64K
Réservé	0001 0000	24M
Registres de contrôle EMIF	0180 0000	32
Registres de configuration de la cache	0184 0000	4
Registres à compteurs et à adresses de la base L2	0184 4000	32
Registres à compteurs et à adresses de la base L1	0184 4020	32
Registres 'flush and clean' de L2	0184 5000	32
Registres d'attributs-mémoires CE0	0184 8200	16
Registres d'attributs-mémoires CE1	0184 8240	16
Registres d'attributs-mémoires CE2	0184 8280	16
Registres d'attributs-mémoires CE3	0184 82C0	16
Registres de contrôle HPI	0188 0000	4
Registres McBSP0	018C 0000	40
Registres McBSP1	0190 0000	40
Registres du timer 0	0194 0000	12
Registres du timer 1	0198 0000	12
Registres de sélection des interruptions	019C 0000	12
Paramètres RAM EDMA	01A0 0000	2M
Registres de contrôle EDMA	01A0 FFE0	32
Pseudo-registres QDMA	0200 0020	20
Données McBSP0	3000 0000	64M
Données McBSP1	3400 0000	64M
CE0, SDRAM	8000 0000	16M
CE1, 8 bits ROM	9000 0000	128K
CE1, 8 bits ports entrées/sorties	9008 0000	4
Carte fille CE2	A000 0000	256M
Carte fille CE3	B000 0000	256M

Plan d'adressage du C6711

VI. Bus interne :

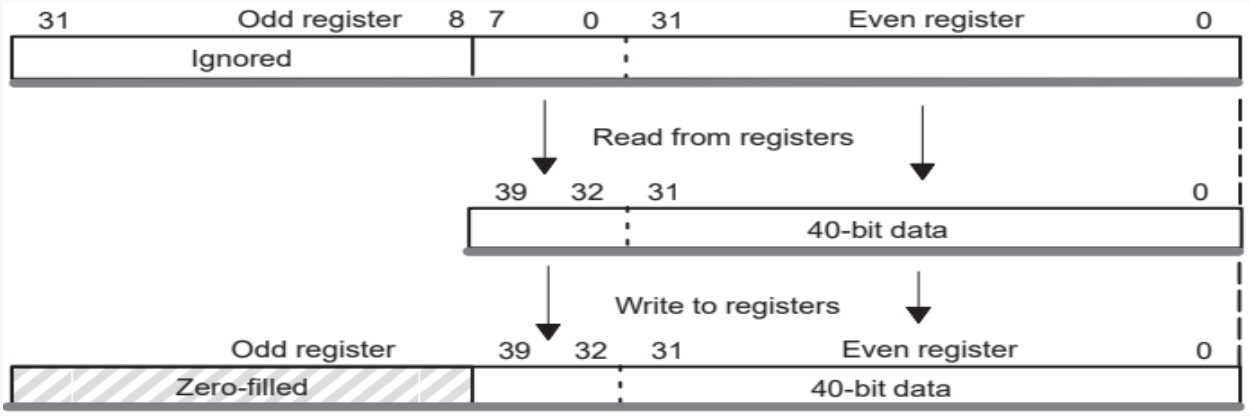
- **Les bus internes illustrés à la Figure 3 comprennent :**
- **Un bus d'adresses pour les programmes de 32 bits et un bus de 256 bits pour les données. Ce dernier accommode** Huit unités fonctionnelles à 32 bits pouvant opérer en parallèle : .LI, .SI, .MI, .DI pour la voie A d'une part et .L2, .S2, .M2 et .D2 pour la voie B d'autre part.
- **Deux voies de chargement depuis la mémoire à 32 bits LDI et LD2 (LD pour Load).**
- **Deux voies de stockage vers la mémoire à 32 bits STI et ST2 (ST pour Store)**
- **Deux chemins d'adresse de données à 32 bits DAI et DA2 (DA pour Data Address).**
- Un bus de de données à 32 bits et un bus d'adresse à 32 bits pour les accès DMA .
- Un bus d'adresse de 20 bits et un bus de données de 32 bits pour l'accès à la mémoire hors puce, ou mémoire externe.



Les registres

Deux ensembles de fichiers de registres, chacun comprenant 16 registres, sont disponibles: fichier de registres A (A0 à A15) et enregistrez le fichier B (B0 à B15).

- Registres A0, A1, B0, B1 et B2 sont utilisés comme registres conditionnels.
- Registres A4 à A7 et B4 à B7sont utilisés pour l'adressage circulaire.
- Registres A0 à A9 et B0 à B9 (sauf B3) sont des registres temporaires.
- L'un des registres A10 à A15 et Les B10 à B15 utilisés sont enregistrés puis restaurés avant de revenir d'un sous-programme.
- Une valeur de données de 40 bits peut être contenue dans une paire de registres.



- Les 32 bits les moins significatifs (LSB) sont stockés dans le registre pair (par exemple, A2) et les 8 bits restants sont stockés dans les 8 bits LSB du registre supérieur (impair) suivant (A3). Un schéma similaire est utilisé pour contenir une valeur de double précision 64 bits dans une paire de registres (même et étrange).

Ces 32 registres sont considérés comme des registres à usage général. Plusieurs registres spéciaux sont également disponibles pour le contrôle des interruptions: par exemple, le registre de mode d'adresse (AMR) utilisé pour l'adressage circulaire et la commande d'interruption registres, comme indiqué à l'annexe B.

Registres de contrôle tms320c6x

Abbreviation	Register Name	Description
AMR	Addressing mode register	Specifies whether to use linear or circular addressing for each of eight registers; also contains sizes for circular addressing
CSR	Control status register	Contains the global interrupt enable bit, cache control bits, and other miscellaneous control and status bits
IFR	Interrupt flag register	Displays status of interrupts
ISR	Interrupt set register	Allows manually setting pending interrupts
ICR	Interrupt clear register	Allows manually clearing pending interrupts
IER	Interrupt enable register	Allows enabling/disabling of individual interrupts
ISTP	Interrupt service table pointer	Points to the beginning of the interrupt service table
IRP	Interrupt return pointer	Contains the address to be used to return from a maskable interrupt
NRP	Nonmaskable interrupt return pointer	Contains the address to be used to return from a nonmaskable interrupt
PCE1	Program counter, E1 phase	Contains the address of the fetch packet that is in the E1 pipeline stage

Le VIII. Les périphériques :

En plus de la mémoire sur puce, les processeurs TMS320C62x et TMS320C67x disposent des périphériques internes pour la communication avec la mémoire hors puce, les coprocesseurs, les processeurs hôtes et les périphériques série. Les périphériques disponibles pour chaque processeur de la plateforme ’C6000 sont répertoriés dans le tableau 4–1

Périphérique DMA (Direct Memory Access):

Le contrôleur DMA élabore le transfert des données entre différent espaces mémoire sans intervention du CPU. Le DMA permet des mouvements de données vers et depuis la mémoire interne, les périphériques internes ou externes de se produire en arrière-plan du fonctionnement du processeur.

Le DMA dispose de quatre canaux programmables indépendamment permettant quatre contextes différents pour le fonctionnement. Un cinquième canal auxiliaire permet au DMA de répondre aux demandes de l'interface (HPI) ou du bus d'extension (XB).

Le processeur peut accéder au registre d'adresses d'hôte (HPIA) et au registre de données (HPID) pour accéder à l'espace mémoire interne du processeur DSP.

La figure est un schéma simplifié de l'interface entre l'hôte et le HPI C62x / C67x

Le Périphériques d'interface Host-Port Interface (HPI) :

Le HPI est un port d'accès paralléle à 16 bits par lequel un processeur hôte, externe, peut accéder lire et écrire directement dans l'espace de la mémoire interne de données du processeur «C62xx». Le périphérique

HPI fonctionne comme une interface asynchrone et utilise des signaux d'établissement de liaison pour synchroniser les transferts de données.

Le HPI est connecté à la mémoire interne via un ensemble de registres. Le processeur hôte ou le processeur dsp peuvent utiliser le registre de contrôle (HPIC) pour configurer l'interface HPI.

Périphérique EDMA (Enhanced Direct Memory Access): 'C6211

Le contrôleur EDMA (Enhanced Direct Memory Access), comme le contrôleur DMA, transfère les données entre les espaces des champs mémoire total accessible sans intervention du processeur.

En plus des fonctionnalités du contrôleur DMA, L'EDMA inclut plusieurs perfectionnements par rapport au DMA classique :

- ✓ **Nombre de canaux :**
  Un nombre de Seize canaux programmables indépendamment permettant seize contextes différents de fonctionnement.
- ✓ **Liaison :**
  La capacité de lier et d'enchaîner des transferts de données.
- ✓ **Synchronisation des événements:**
  Chaque canal est initié par un événement spécifique.Les transferts peuvent être synchronisés par élément ou par trame.

Périphérique EMIF :

L'interface de mémoire externe (EMIF) permet d'interfacer le 'C62xx à plusieurs types de mémoire externe:

L'interface de mémoire externe (EMIF) permet d'interfacer les processeurs 'C62xx à plusieurs types de mémoire externe:

EMIF fournit le timing nécessaire pour accéder à la mémoire externe

- ✓ Périphériques asynchrones. Cette interface est programmable pour s'adapter à diverses largeurs de configuration, de maintien et de stroboscope.
- ✓ Mémoire dynamique synchrones SDRAM ainsi que les RAM de type SBSRAM).

- ✓ Possibilité de lecture de mémoire 8 bits et 16 bits pour supporter les mémoires de pour le démarrage « boot » faible coût (FLASH, EEPROM, EPROM et PROM).

L'EMIF peut recevoir trois types de demandes d'accès. Les trois types sont classés par ordre de priorité dans l'ordre ci-dessous.

1) Accès aux données du processeur
2) Récupérations du programme CPU
3) Accès aux données DMA.

Lorsqu'il est disponible pour desservir un autre accès, l'EMIF traite le type de demande de priorité la plus élevée

Le tableau 4–1 présente les interfaces prises en charge par chacun des trois espaces de mémoire externe (0–2). Les espaces 0 et 2 sont destinés à une variété d'interfaces. L'espace 1 est destiné aux mémoires asynchrones, y compris le stockage ROM à faible coût.

Périphérique Boot Loader

Boot Loader démarre le chargement du code de la mémoire hors puce ou HPI vers la mémoire interne démarre le chargement du code ou du programme de démarrage de la mémoire externe hors puce ou HPI vers la mémoire interne

Ces configurations déterminent les actions que le C62x / C67x effectue, après une reset du processeur, pour préparer l'initialisation.

Ces configurations de boot sont positionnées par des broches externes « pins » en entrée, déterminent:

L'espace mémoire sélectionnée par le processeur

L'espace mémoire détermine si la mémoire interne ou externe est mappée à l'adresse 0.

Le type de mémoire externe à l'adresse 0 (si la mémoire externe est mappée à l'adresse 0)

Le processus de démarrage utilisé pour initialiser la mémoire à l'adresse 0 avant la

Le processeur est autorisé à s'exécuter.

Le processus de boot utilisé initialise la mémoire à l'adresse 0 avant l'autorisation du processeeur à démarrer .

Périphériques Timers Devices: Périphériques de temporisation: tous

Le 'C62x / C67x possède deux temporisateurs « Timer » 32 bits à usage général qui peuvent être utiliser comme :

1. Générateurs d'événements temporels
2. Compteur d'événements
3. Générateur d'impulsions
4. Générateur de signaux d'interruption de CPU
5. Générateur de signaux de synchronisation pour l'EDMA

Périphérique Multichannel Buffered Serial Port (McBSP),

Le port série multicanal bufférisé (McBSP) disponible sur les processeurs C62x / C67x est basé sur la liaison de port série standard des plates-formes TMS320C2000 et 'C5000.
Ce port série standard permet:

- ✓ des communications en full-duplex
- ✓ un flux de données continu grace aux double Registres de données à double tampon.
- ✓ une indépendance dans le timing et le format des données à l'émission et la réception,
- ✓ Interface directe avec les codecs standard de l'industrie,interfaces analogiques A / D et D / A
- ✓ une synchronisation par horloge interne ou externe

Le microprocesseur C67x dispose de deux modules de liaison série (McBSPo et McBSP1).

Références bibliographiques:

1. *R. Chassaing, D. Reay, Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK, John Wiley & Sons, 2008.*
2. *Steven A Tretter, Communication System Design Using DSP Algorithms, Springer 2008.*
3. *N. Dahnoun, Digital Signal Processing Implementation using the TMS320 C6000 DSP platform, Prentice Hall, 2000.*
4. *N. Kehtarnaz, N. Kim, Real Time Digital Signal Processing Based on TMS320C6000, Newnes, 2004.*
5. *N. Kehtarnaz, M. Keramat, DSP System Design using TMS320C6000, Prentice Hall, 2006.*
6. *P. Laspsley , J. Bier , A. Shoham, E. A. Lee, DSP Fundamentals: Architecture and Features, Berkley Design Technology, Inc, 1994.*
7. *Texas Instruments, Code Composer Studio Development Tools v3.3 Getting Started Guide (Rev. H), http://www.ti.com/lit/ug/spru509h/spru509h.pdf, 2008.*
8. *Texas Instruments, TMS320C6000 CPU and Instruction Set Reference Guide (Rev. G), http://www.ti.com/lit/ug/spru189g/spru189g.pdf, 2006.*