الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira -
ⵜⴰⵙⴷⴰⵡⵉⵜ ⴰⴽⵍⵉ ⵎⵓⵃⴻⵏⴷ ⵓⵍⵃⴰⵊ - ⵜⵓⴱⵉⵔⴻⵜ -

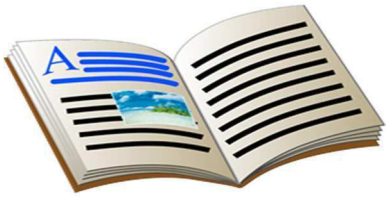
وزارة التعليم العالي والبحث العلمي
جامعة أكلي محند أولحاج
- البويرة -

جامعة البويرة

Faculté des Sciences et des Sciences Appliquées

كلية العلوم والعلوم التطبيقية

Département de Génie électrique



Polycopié de cours

En : Electronique

Spécialité :.Electronique des systèmes embarqués

Niveau : 1ère année master

Processeurs des signaux numériques

DSP

Par MADI Saida

Expertisé par :

Dr. Nom Prénom, grade, Université de Bouira

Pr. Nom Prénom, Professeur, Université de Nom

Année : 2022/2023

Table des matières

Chapitre 3 : Architecture des DSP TMS320C6x

Chapitre 4 : Gestion de la mémoire

Liste des figures

Listes des tableaux

Chapitre 1 : Généralités sur les processeurs DSP

1.1 Définition : Processeur de signal numérique DSP

Le traitement du signal signifie l'analyse et la manipulation du signal. Il est effectué pour obtenir un signal clair et pur. Le traitement du signal est exécuté par l'ordinateur ou des circuits intégrés, comme : les circuits ASCI (Application Specific Integrated Circuits), FPGA (Field Programmable Gate Array) ou DSP (Digital Signal Processing).

Le processeur de signal numérique DSP est un microprocesseur optimisé pour exécuter des applications de traitement numérique du signal (filtrage, extraction de signaux, … etc.) le plus rapidement possible.

1.2 Présentation des différentes familles de DSP

Le marché de DSP est partagé en quatre constructeurs principaux :

1. Texas Instruments,
2. Analog Devices,
3. Freescale (Motorola), et
4. Lucent.

Les DSP se différencient par :

1. Le format de calcul (fixe ou entier),
2. La taille du bus de données (16, 24 ou 32 bits),
3. La puissance en millions d'instructions par secondes (mips),
4. Les fonctionnalités spécifiques directement intégrées (traitement du son, de l'image, … etc.).

Il est impossible d'effectuer une classification « définitive » des DSP, car chaque constructeur met sur le marché chaque année un nouveau composant qui surclasse les anciens ou le conçurent par la puissance de calcul, la rapidité (gestion du pipeline et fréquence d'horloge), le nombre de registres, le nombre de ports séries.

Un point essentiel des DSP est la représentation des nombres (les données) qu'ils peuvent manipuler. Il est possible de distinguer deux grandes familles : La première famille regroupe les processeurs à virgule fixe : le programmeur doit rester concentré à chaque étape d'un calcul. Ces DSP sont plus difficiles à programmer. Le second regroupe les processeurs à virgule flottante. Les DSP à virgule flottante fournissent une très grande dnamique et ils sont plus chers et consomment plus d'énergie.

En termes de rapidité, les DSP à virgule fixe se placent d'ordinaire devant leurs homologues à virgule flottante, ce qui constitue un critère de choix important.

1.3 Classification des DSP

Voyons le cas des DSP fabriqués par Texas Instruments (désignés par TMS) et Analog Devices (désignées par ADSP). Le classement du tableau 1.1 est effectué selon le nombre de bits du bus de données et le temps d'exécution d'un cycle, puis d'une opération complexe, comme la transformée de Fourier rapide à 1024 points de calcul.

- ✓ Les TMS320Clx sont à 16 bits à virgule fixe et sont utilisés pour le contrôle des disques durs dans les ordinateurs.
- ✓ Les TMS320C2x ou ADSP-2105 servent au fonctionnement de fax.
- ✓ Les TMS320C5x ou ADSP-2101 sont utilisés dans les modems.
- ✓ Les TMS320C3x ou ADSP-21010 sont utilisés pour les systèmes Hi-Fi, à synthèse vocale, et dans les processeurs graphiques à 3 dimensions.
- ✓ Les TMS320C4x, TMS320C6x ou ADSP-21020 sont conçus pour le fonctionnement en parallèle, avec d'autres systèmes processeurs (applications : la réalité virtuelle et la reconnaissance d'image).

Tableau 1.1 : Comparaison entre diverses catégories de DSP

Nom	Critère de choix	Virgule fixe ou flottante	Durée d'une instruction [ns]	Durée de calcul FFT [µs]
ADSP2105	Faible coût	16 bits – Fixe	100	3.46
TMS320C2x	Faible coût	16 bits – Fixe	80	9.01
ADSP2101	Haute performance	16 bits – Fixe	60	2.07
TMS320C5x	Haute performance	16 bits – Fixe	35	2.97
ADSP2199x	Haute performance	16 bits – Fixe	6	0.4
ADSP21010	Faible coût	32 bits - Flottante	80	1.54
TMS320C3x	Faible coût	32 bits - Flottante	50	3.08
ADSP21020	Haute performance	32 bits - Flottante	40	0.77
TMS320C4x	Haute performance	32 bits - Flottante	40	1.55

1.4 Domaines d'applications des DSP

Les domaines d'applications du traitement numérique du signal sont nombreux et variés (traitements du son, de l'image, synthèse et reconnaissance vocale, analyse, compression de données, télécommunications, automatisme, … etc.). Chacun de ces domaines nécessite un système de traitement numérique, dont le cœur est un (parfois plusieurs) DSP ayant une puissance de traitement adaptée, pour un coût économique approprié.

Le tableau 1.2 représente une liste de quelques applications typiques pour la famille de DSP TMS320. Ces DSP offrent des approches adaptables aux problèmes traditionnels de traitement du signal. Ils prennent également en charge des applications complexes qui nécessitent souvent l'exécution simultanée de plusieurs opérations.

Tableau 1 .2 : Applications typiques des DSP 'TMS320'

Contrôle des consommateurs automobiles	Contrôle de conduite adaptative, Freins antidérapants, Téléphones cellulaires, Radios numériques, Contrôle du moteur, Positionnement global, La navigation, Analyse vibratoire, Commandes vocales
Consommateur	Radios/téléviseurs numériques, Jouets éducatifs, Synthétiseurs de musique, Téléavertisseurs, Outils électroportatifs, Détecteurs de radars, Répondeurs à semi-conducteurs
Contrôler	Contrôle du lecteur de disque, Contrôle du moteur, Contrôle de l'imprimante laser, Contrôle moteur, Contrôle robotique, Servocommande
Usage général	Filtrage adaptatif, Convolution, Corrélation, Filtrage numérique, Transformées de Fourier rapides, Hilbert se transforme, Génération de formes d'onde, Fenêtrage
Graphiques/Imagerie	Rotation 3D, Animation/cartes numériques, Traitement homomorphe, Compression/transmission d'images ; Amélioration d'images, La reconnaissance de formes, Vision robotique, Postes de travail
Industriel	Commande numérique, Surveillance de ligne électrique, Robotique, Accès sécurisé
Instrumentation	Filtrage numérique, Génération de fonctions, Correspondance de motifs, Boucles à verrouillage de phase, Traitement sismique, Analyse de spectre, Analyse transitoire

Médical	Équipement de diagnostic, Surveillance fœtale, Prothèses auditives, Surveillance des patients, Prothèses, Équipement d'échographie
Militaire	Traitement d'images, Guidage des missiles, La navigation, Traitement radar, Modems radiofréquence, Communications sécurisés, Traitement du sonar
Télécommunications	Modems 1200 à 56 600 bps, Égaliseurs adaptatifs, Transcodeurs ADPCM, Stations de base, Téléphones cellulaires, Multiplexage des canaux, Cryptage des données, PBX numériques, Interpolation numérique de la parole (Digital speech interpolation 'DSI'), Encodage/décodage DTMF, Annulation d'écho, Télécopier, Futurs terminaux, Répéteurs de ligne, Communications personnelles systèmes (Personal communications Systems 'PCS'), Assistants numériques personnels (Personal digital assistants 'PDA'), Téléphones à haut-parleur, Communications à spectre étalé, Boucle d'abonné numérique (Digital subscriber loop 'xDSL'), Vidéo conférence commutation de paquets X.25
Voix/Discours	Vérification du haut-parleur, Amélioration du discours, Reconnaissance de la parole, Synthèse de discours, VO codage de la parole, Texte pour parler, Messagerie vocale

1.5 Principaux algorithmes traités

Les algorithmes typiques des processeurs DSP sont :

- **Finite Impulse Response Filter (FIR)**

$$y(i) = \sum_{k=0}^{N-1} h(k)\, x(i-k) = h(n) * x(n) \quad (1.1)$$

  - x est la séquence d'entrée
  - y est la séquence de sortie
  - h est la réponse impulsionnelle (coefficient de filtre)
  - N est le nombre de prises (coefficient) ² le filtre
  - La séquence de sortie dépend uniquement de la séquence d'entrée et de la réponse impulsionnelle.

- **Infinite Impulse Response Filter(IIR)**

$$y(i) = \sum_{k=1}^{M-1} a(k)\, y(i-k) + \sum_{k=0}^{N-1} b(k)\, x(i-k) \quad (1.2)$$

- **Convolution**

$$y(n) = \sum_{k=0}^{N} x(k) * h(n-k) \quad (1.3)$$

- **Discrete Fourier Transform (DFT)**

$$X(k) = \sum_{n=0}^{N-1} x(n)\, exp^{-jnk\left(\frac{2\pi}{N}\right)} \quad (1.4)$$

1.6 Processeurs DSP et autres approches

Le traitement numérique permet facilement de :

- ✓ Changer les applications.
- ✓ Corriger les applications.
- ✓ Mettre à jour des applications.

Il permet également de réduire :

- ✓ Les erreurs de calcul.
- ✓ Le temps de développement.
- ✓ Le coût de traitement (les DSP peuvent être combinés avec d'autres composants dans le même boîtier. Par exemple, un ou plusieurs DSP peuvent être combinés avec un microprocesseur classique et des convertisseurs CAN et CNA. Ce type d'assemblage (circuits intégrés dédiés) permet de réduire les coûts dans des fabrications de grande série).
- ✓ La consommation d'énergie.

Les autres solutions numériques qui conçurent les DSP sont (voir tableau 1.3) :

- ✓ Les ASIC (Application Specific Integrated Circuits), qui sont des circuits intégrés pour des applications spécifiques, elles ont été depuis longtemps la technologie la mieux adaptée pour réaliser des applications nécessitant des performances élevées.
- ✓ Les FPGA (Field Programmable Gate Array), qui sont des composants électroniques qui comportent un grand nombre de fonctions logiques de base (ET, OU, etc.) que l'utilisateur peut combiner entre elles en fonction des besoins des applications.

Tableau 1.3 : Les DSP face aux ASIC et aux FPGA

	ASIC	FPGA	DSP
Performances	Très élevées	Elevées	Faibles
Tailles et poids Consommation	Faibles	Moyens	Elevées
Intégration	Faible	Modérés	Très élevées
Souplesse	Sur puce	Sur puce	Composants associés
	Fonctions figées	Reconfigurable	Programmable

Chapitre 2 : Arithmétique à virgule fixe et à virgule flottante

2.1 Introduction

Un système autonome ou embarqué qui nécessite un traitement de signal, se résume en général par le schéma suivant :

Figure 2.1. Les éléments principaux de DSP.

- ✓ L'entrée $x_a(t)$ c'est un signal analogique, comme : la tension, la température, l'intensité lumineuse, … etc
- ✓ CAN : Convertisseur Analogique Numérique
- ✓ $x[n]$ et $y[n]$ sont des numéros.
- ✓ CNA : Convertisseur Numérique Analogique
- ✓ La sortie $y_a(t)$ est un signal analogique.

Les processeurs DSP (Digital Signal Processors) sont des microprocesseurs spécifiquement conçus pour des applications de traitement numérique de signal. Ils ont été initialement développés dans les années 70 pour des applications de radars militaires et de télécommunications cryptées. Ces composants ont connu une croissance très importante ces dernières années.

2.2 Numérisation des signaux

La numérisation d'un signal est l'opération qui consiste à faire passer un signal de la représentation dans le domaine des temps et des amplitudes continus au domaine des temps et des amplitudes discrets.

L'interface nécessaire entre le monde analogique et un traitement numérique donné est réalisé par des convertisseurs analogiques – numériques CAN et numériques – analogiques CNA. Le rôle d'un CAN est de convertir un signal analogique en un signal numérique pouvant être traité par une logique numérique, et le rôle d'un CNA est de reconvertir le signal numérique une fois traité en un signal analogique (voir la figure 2.2).

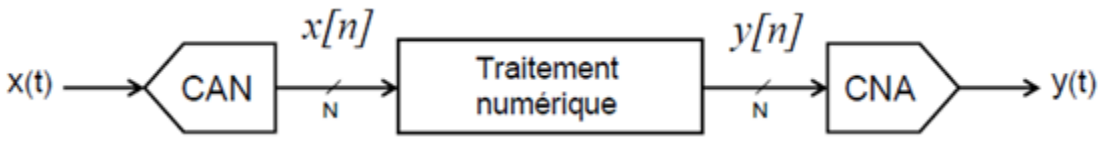


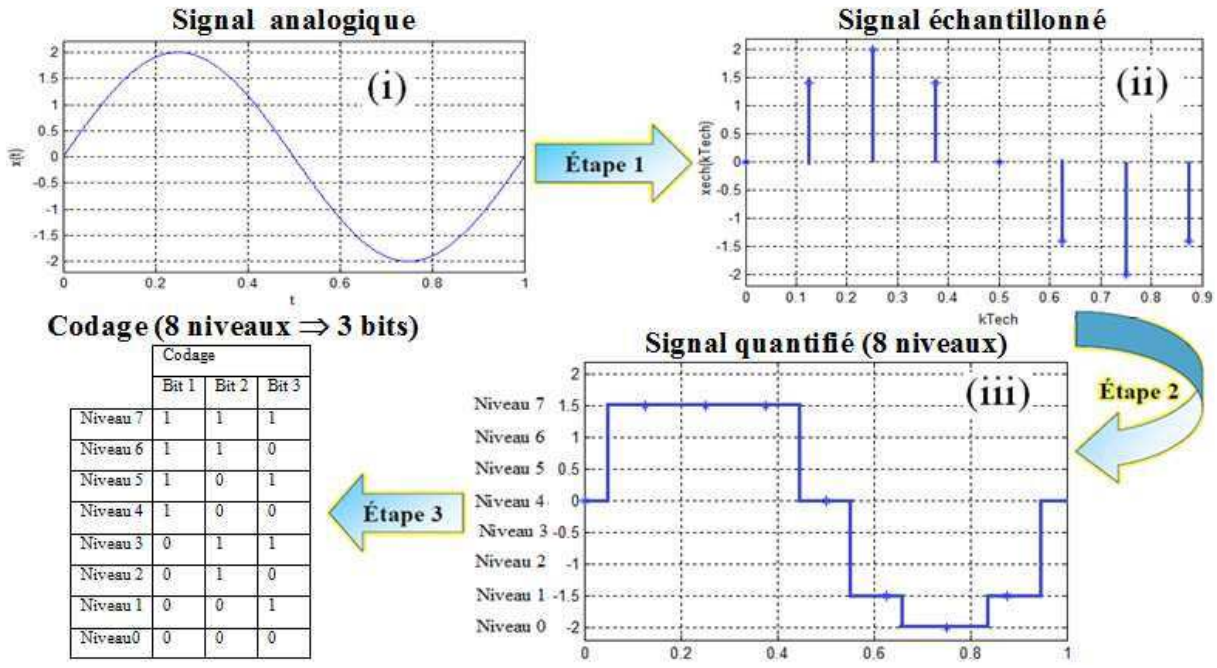
Figure 2.2 Conversions et traitements numériques des données.

2.3 Principe de la conversion analogique numérique

Le convertisseur analogique – numérique CAN est un dispositif électronique permettant la conversion d'un signal analogique en un signal numérique. La conversion analogique – numérique peut être divisée en trois étapes : (1) l'échantillonnage temporel, (2) la quantification et (3) le codage (voir la figure 2.3).

Un signal analogique, *x(t)* continu en temps et en amplitude (i) est échantillonné à une période d'échantillonnage constante $T_{éch}$. On obtient alors un signal échantillonné $x_{éch}$ (k.$T_{éch}$) discret en temps et continu en amplitude (ii). Ce dernier est ensuite quantifié, on obtient alors un signal numérique *xq[n]* discret en temps et en amplitude (iii).

La quantification est liée à la résolution du CAN (son nombre de bits) ; sur la figure 2.3 *xq[n]* peut prendre 8 amplitudes différentes (soit 2^3, 3 étant le nombre de bits du CAN). La figure. 2.3 présente également le code numérique sur 3 bits (en code binaire naturel) associé à *xq[n]* en fonction du temps.



	Codage		
	Bit 1	Bit 2	Bit 3
Niveau 7	1	1	1
Niveau 6	1	1	0
Niveau 5	1	0	1
Niveau 4	1	0	0
Niveau 3	0	1	1
Niveau 2	0	1	0
Niveau 1	0	0	1
Niveau0	0	0	0

Figure 2.3 Les trois étapes de la conversion pour un CAN à 3 bits.

La figure suivante présente le symbole d'un CAN à N bits qui sera utilisé dans la suite de ce cours.

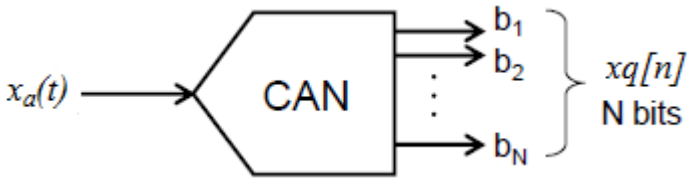


Figure 2.4 – Convertisseur analogique numérique

2.3.1 Echantillonnage d'un signal analogique

L'obtention d'un signal échantillonné $x_{éch}(kT_{éch})$ à partir d'un signal analogique x(t) peut être modélisée mathématiquement dans le domaine temporel par la multiplication de x(t) par un peigne de Dirac de période $T_{éch}$ (noté $\delta_{Téch}(t)$):

$$x_{éch}(kT_{éch}) = x(t)\delta_{T_{éch}}(t) = x(t) \sum_k \delta(t - kT_{éch}) \qquad (2.1)$$

L'échantillonnage est illustré graphiquement dans le domaine temporel aux points (i), (ii) et (iii) de la figure 2.5. Il peut également être décris graphiquement dans le domaine fréquentiel.

Au signal analogique x(t), est associé dans le domaine fréquentiel, le spectre X(f) s'étendant sur une bande de fréquence de $-f_{max}$ à f_{max} (voir la figure 2.5).

On rappelle un certain nombre de résultats démontrés en cours d'analyse de Fourier :

- ✓ Une multiplication dans le domaine temporel correspond à un produit de convolution dans le domaine spectral (et inversement),
- ✓ La transformée de Fourier d'un peigne de Dirac temporel, de période $T_{éch}$, et d'amplitude 1, est un peigne de Dirac dans le domaine fréquentiel, de période $f_{éch} = 1 / T_{éch}$ et d'amplitude $1 / T_{éch}$.

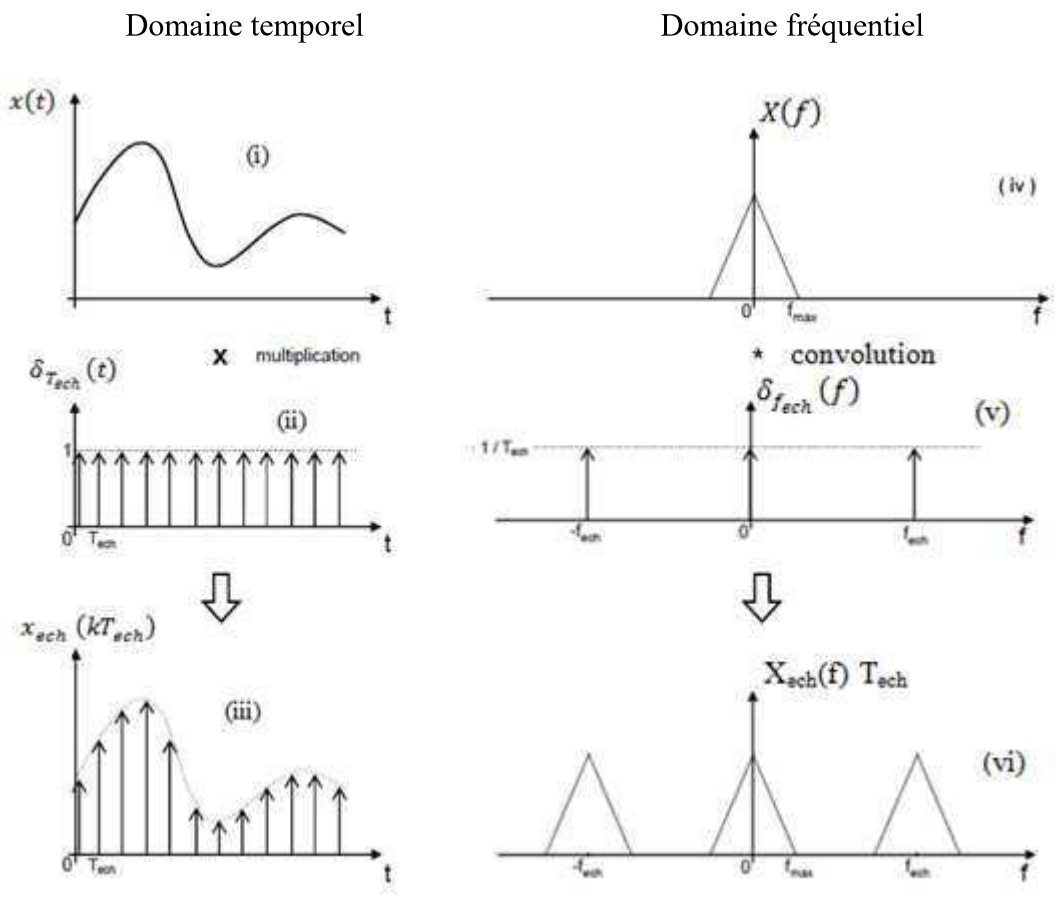


Figure. 2.5 Aspects temporels et fréquentiels de l'échantillonnage.

Ainsi, à la multiplication temporelle $x(t).\delta_{T_{éch}}(t)$ on fait correspondre dans le domaine fréquentiel, le produit de convolution $X(f) * \delta_{f_{éch}}(f)$ ($\delta_{f_{éch}}(f)$ étant la transformée de Fourier de $\delta_{T_{éch}}(t)$, (voir la figure 2.5.v). Le résultat de ce produit de convolution (est la transformée de Fourier du signal échantillonné $x_{éch}(k.T_{éch})$ (voir la figure 2.5.vi). On obtient le spectre X(f) répété à toutes les fréquences multiples de la fréquence d'échantillonnage (centrés sur les $k.f_{éch}$, k entier), à un facteur multiplicatif près sur l'amplitude $T_{éch}$ introduit par le peigne fréquentiel de Dirac.

Une approche graphique dans le domaine spectrale permet d'illustrer la récupération de l'information contenue dans un signal échantillonné par un filtrage passe bas (voir la figure. 2.6). En supposant un filtrage passe bas parfait sur la bande de fréquence de $\left(\frac{-f_{éch}}{2}\right)$ à $\left(\frac{f_{éch}}{2}\right)$ (appelée bande de Nyquist, la fréquence $\left(\frac{f_{éch}}{2}\right)$ étant appelée fréquence de Nyquist), on retrouve le spectre X(f) et donc le signal temporel qui y correspond x(t).

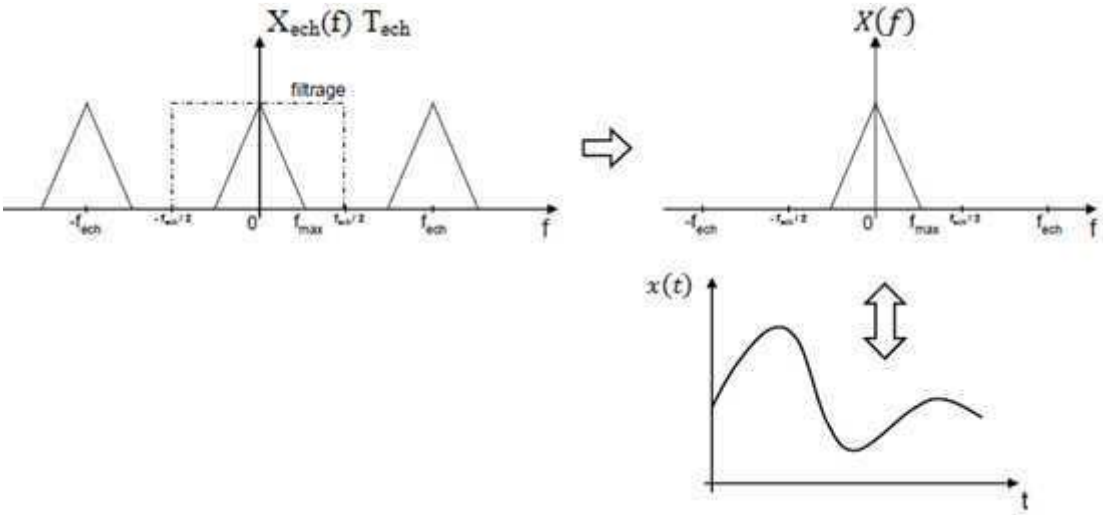


Figure 2.6 Récupération de l'information par filtrage passe bas.

Exemple

Soit le signal analogique $x(t) = 2\sin(\omega\, t)$, avec $f = 1Hz$

- Pour t = 1s, dessiner le signal échantillonné $x_{éch}\ (k\ T_{éch})$. On donne $T_{éch} = 0.125s$.

Solution

On a $x(t) = 2\sin(\omega\, t)$, avec $f = 1Hz$ donc $x(t) = 2\sin(2\pi f\ t) = 2\sin(2\pi\ t)$

Pour t=1s, et $T_{éch} = 0.125s$ donc $K = \frac{t}{T_{éch}} = \frac{1}{0.125} = 8$ éch/s, donc ; k ϵ {1, …, K-1=7}

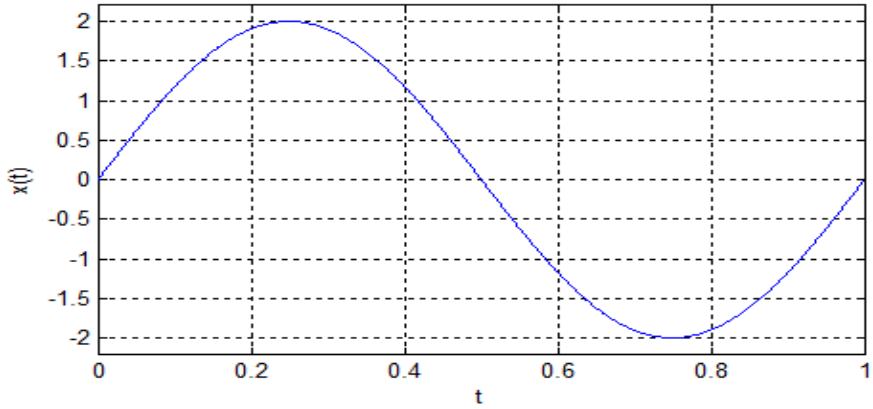


Figure 2.7 Signal analogique $x(t)$

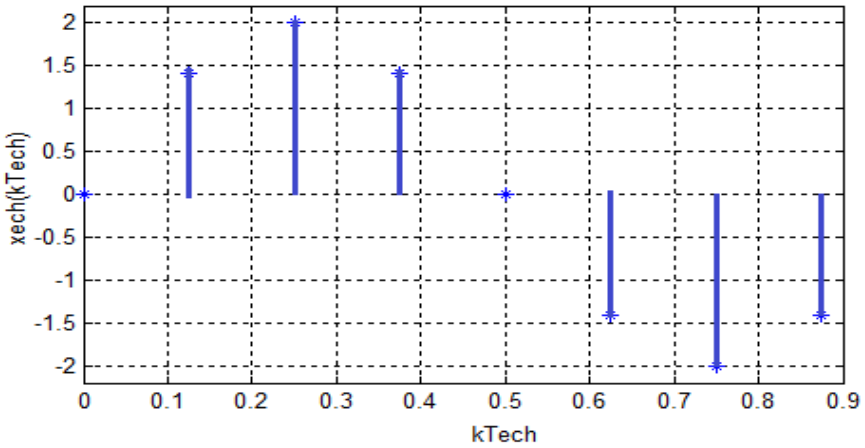


Figure 2.8 Signal échantillonné $x_{éch}$ $(k\ T_{éch})$.

2.3.2 Théorème de l'échantillonnage

Les illustrations graphiques précédentes correspondent au cas où $\frac{f_{éch}}{2} \geq f_{max}$. Dans le cas où on augmente la période d'échantillonnage (on a alors $f_{éch}$ qui diminue) il apparaît un phénomène de recouvrement spectral illustré (voir la figure 2.9).

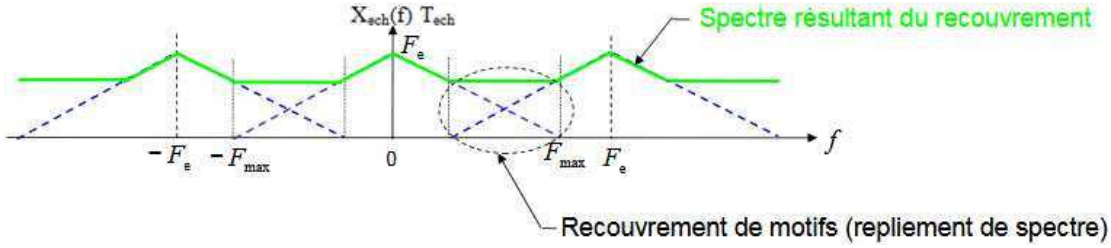


Figure 2.9 Repliement de spectre.

Ce phénomène apparaît *(immédiatement)* dès lors que f_{max} (la plus grande fréquence de la partie du spectre centrée sur 0) devient supérieur à $f_{éch} - f_{max}$ (la plus basse fréquence de la partie du spectre centrée sur $f_{éch}$) ; les parties du spectre qui se superposent s'ajoutent, et on

obtient le spectre replié de la figure précédente. Il n'est plus possible de récupérer le signal analogique de départ par le filtrage passe bas.

La contrainte qui en découle sur la fréquence d'échantillonnage pour éviter le repliement s'écrit mathématiquement :

$$f_{éch} \geq 2\, f_{max} \qquad (2.2)$$

Elle s'énonce sous la forme du théorème de Shannon, ou théorème de l'échantillonnage : "La condition nécessaire et suffisante pour échantillonner un signal sans perte d'information est que la fréquence d'échantillonnage $f_{éch}$ soit supérieure ou égale au double de la fréquence maximale du signal. Plus précisément, si on note f_{max} la fréquence maximale du signal, il faut et il suffit que : $f_{éch} \geq 2\, f_{max}$".

Le spectre réel est généralement de largeur infinie (à cause du bruit, ou de signaux interférents non désirés), il y a donc toujours un phénomène de repliement spectral susceptible de ramener dans la bande de Nyquist, du bruit ou un signal d'interférence. D'où la nécessité de toujours inclure un filtre passe-bas anti-repliement ayant une fréquence de coupure à ($f_{éch}$/ 2) devant un CAN.

Exemple

Considérons le signal analogique suivant :

$$x_{éch}\,(t) = 2\cos(100\pi t) + 5\sin\left(250\pi t + \frac{\pi}{6}\right) - 4\cos(380\pi t) + 16\sin\left(600\pi t + \frac{\pi}{4}\right)$$

- Quelle valeur minimum faut-il choisir pour $f_{éch}$ si l'on veut respecter le théorème d'échantillonnage ?

Solution

Ce signal comporte quatre composantes spectrales situées en f = 50, 125, 190, 300 Hz. La fréquence d'échantillonnage devra donc valoir au moins

$$f_{éch\,min} = 2\, f_{max} = 2 \text{ x } 300 = 600 \text{ Hz.}$$

2.4 Quantification uniforme

La conversion analogique numérique implique une opération qui consiste à remplacer la valeur exacte analogique de l'échantillon par la plus proche valeur approximative extraite d'un ensemble fini de valeurs discrètes.

Chaque nombre *xq*, représente un ensemble de valeurs analogiques contenues dans un intervalle de largeur Δ appelé "pas de quantification" ou "le quantum q". Lorsque la plage de conversion est subdivisée en pas de quantification égale, on parle de quantification uniforme.

La loi de quantification la plus fréquemment utilisée est la loi uniforme (ou linéaire) dans laquelle les pas de quantification Δ_i sont constants.

2.4.1 Caractéristique de la Quantification uniforme.

Le pas de quantification et la précision d'un CAN dépendent du nombre de bits en sortie (appelé résolution). Pour un CAN à N bits, le nombre d'états possibles en sortie est $\mathbf{2^N}$, ce qui permet d'exprimer des signaux numériques de 0 à $\mathbf{2^N-1}$ en code binaire naturel.

Un CAN est caractérisé également par la plage de variation acceptable de la tension analogique d'entrée, appelée Pleine Echelle (FS pour *Full Scale* en anglais) et que nous noterons V_{PE}.

La pleine échelle est divisée en autant de plages d'égale dimension (cas de la quantification uniforme) qu'il y a d'états possibles de la sortie numérique. Chaque plage est associée à un code numérique représentant la tension analogique d'entrée.

On définit le quantum q, comme étant la dimension de ces plages. On le note q et on l'obtient par :

$$q = \frac{V_{PE}}{2^N} \qquad \text{(il y a bien } 2^N \text{ "marches" à "l'escalier")} \qquad (2.3)$$

Les tensions de seuil V_{Sn}, correspondant aux transitions entre les codes de sortie, sont telles que :

$$V_{Sn} = n.q \qquad n \in \{0, 1, \ldots, 2^N-1\} \qquad (2.4)$$

La figure 2.10 représente la caractéristique de transfert idéal (sans défaut) en escalier d'un CAN à 3 bits.

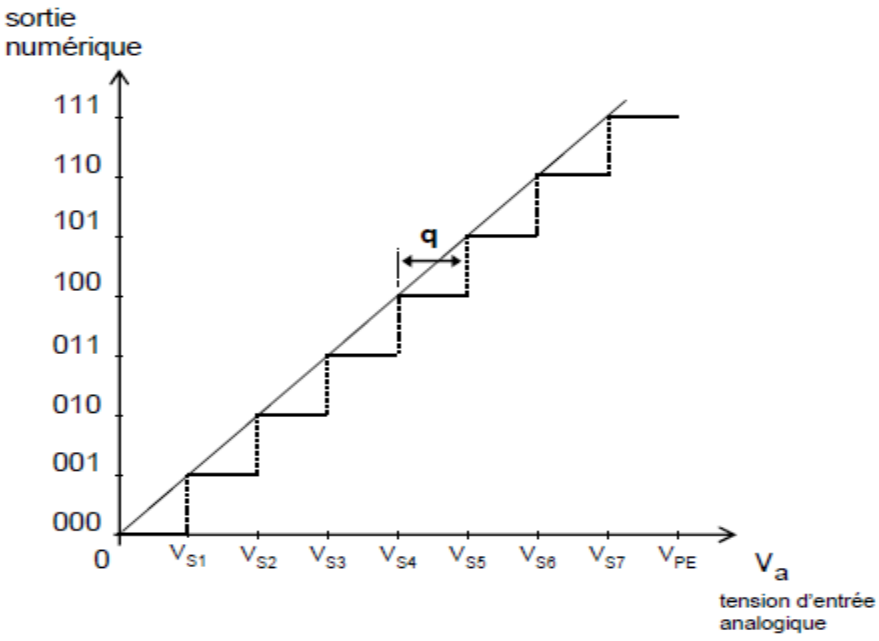


Figure 2.10 Caractéristique de transfert idéal d'un CAN à quantification linéaire par défaut.

Remarque

Plus la résolution d'un CAN est élevée, plus la sortie numérique est une image précise du signal analogique d'entrée comme l'illustre le tableau 2.1 pour une tension de pleine échelle de 5V.

$$q = (V_{PE}/2^N) = (5/2^N) \qquad (2.5)$$

Tableau 1.1: Quantum d'un CAN en fonction de sa résolution

N	4		16	32
Quantum q	312.5 mV		76.29 µ mV	1.26 nV

Exemple

Soit le signal analogique $x(t) = 2 + 2\sin(\omega\, t)$, avec $f = 1Hz$. Son signal échantillonné est $x_{éch}\ (k\ T_{éch})$. Pour t=1s, N=3 et $T_{éch} = 0.125s$:

- Dessiner le signal échantillonné $x_{éch}\ (k\ T_{éch})$.
- Calculer le quantum q et les tensions de seuil V_{Sn}.
- Dessiner le signal xq(n).

Solution

On a $x(t) = 2 + 2\sin(\omega\, t)$, avec $f = 1Hz$ donc $x(t) = 2 + 2\sin(2\pi ft) = 2 + 2\sin(2\pi t)$

Pour t=1s, et $T_{éch} = 0.125s$: $K = \frac{t}{T_{éch}} = \frac{1}{0.125} = 8$ éch/s ;

On a ; $V_{PE} = V_{max} - (-V_{max}) = 2 - (-2) = 4$ V, et N = 3, donc le quantum q : $q = (V_{PE}/2^N)$

$q = 4/(2^3) = 0.5$ V; les tensions de seuil V_{Sn} : $V_{Sn} = n\ q$, pour $n \in \{0,\ldots, 2^3\text{-}1\text{=}7\}$, donc :

Tableau 2.2: Les tensions de seuil

n	0	1	2	3	4	5	6	7
V_{Sn} (V)	0	0.5	1	1.5	2	2.5	3	3.5

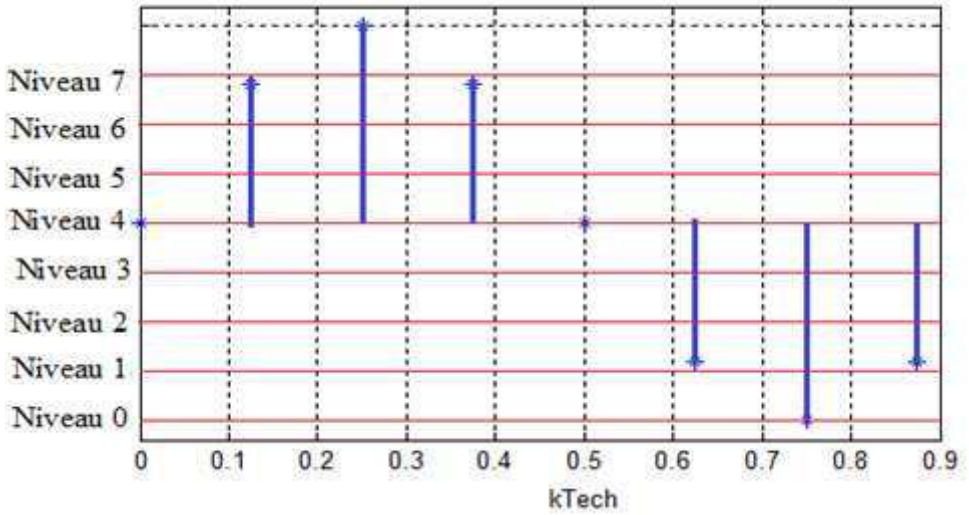


Figure 2.11 Le signal échantillonné $x_{éch}(kT_{éch})$ avec V_{Sn}

Les différents niveaux pour N = 3 sont donnés dans le tableau suivant.

Tableau 2.2: Les différents niveaux pour N=3

Niveau 0	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5	Niveau 6	Niveau 7
000	001	010	011	100	101	110	111

Le signal quantifié xq(n) du signal échantillonné $x_{éch}(kT_{éch})$ est représenté par la **figure 2.12**.

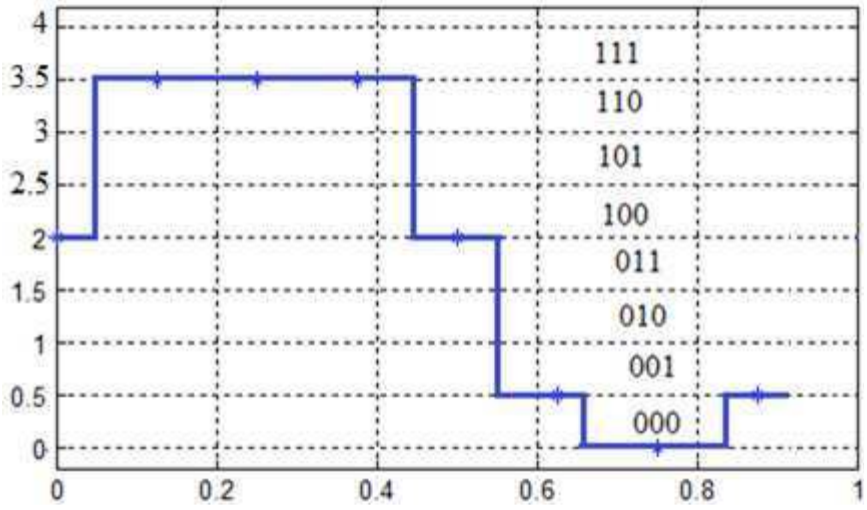


Figure 2.12 : Le signal quantifié xq(n)

2.4.2 Erreur de quantification :

L'erreur de quantification est la différence entre la valeur du signal échantillonné et la valeur analogique d'entrée correspondant au code de sortie (correspondance donnée par la droite de transfert idéal), l'erreur de codage est calculée par l'équation suivante :

$$e_q = x_q - x_{éch} \qquad (2.6)$$

La Figure 2.13 donne l'erreur de codage d'un CAN à 3 bits pour une quantification linéaire par défaut. L'erreur de quantification est comprise entre 0 et 1 LSB. Ainsi, tous les signaux analogiques compris entre V_{S2} et V_{S3}, par exemple, sont représentés par le code binaire 010. Ce type d'erreur est inhérent aux CAN, il est lié à l'étape de quantification.

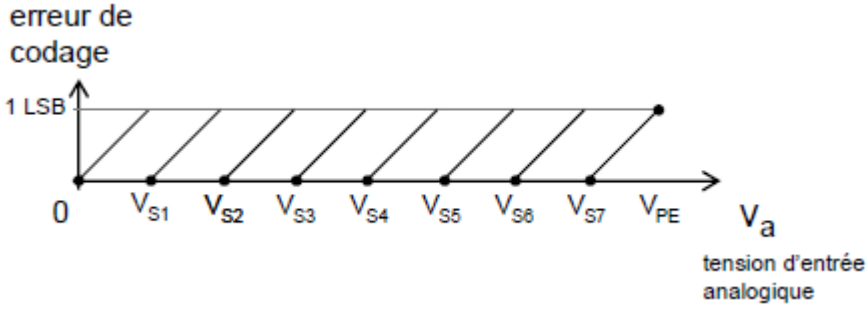


Figure 2.13 Erreur de codage de la quantification linéaire par défaut.

Remarque

Plus la résolution (le nombre de bits) d'un CAN est élevée plus l'erreur de quantification est réduite.

Un simple changement de convention, dans la fixation des tensions de seuil, permet de réduire l'erreur de quantification en valeur absolue. Ainsi, on utilisera plutôt la quantification linéaire centrée, pour laquelle la droite de transfert idéal passe par le centre des "marches" de la caractéristique (voir la figure 2.14).

La droite de transfert idéal coupe la caractéristique idéale de transfert pour $va = n\,q$ tel que $n \in \{0, \ldots, 2^N\text{-}1\}$ (cf. points sur la figure). On obtient la caractéristique pour une quantification linéaire centrée en décalant vers la gauche de ½ q la caractéristique correspondant à une quantification linéaire par défaut. A noter que le premier palier mesure ½ q et le dernier $\frac{3}{2}$ q.

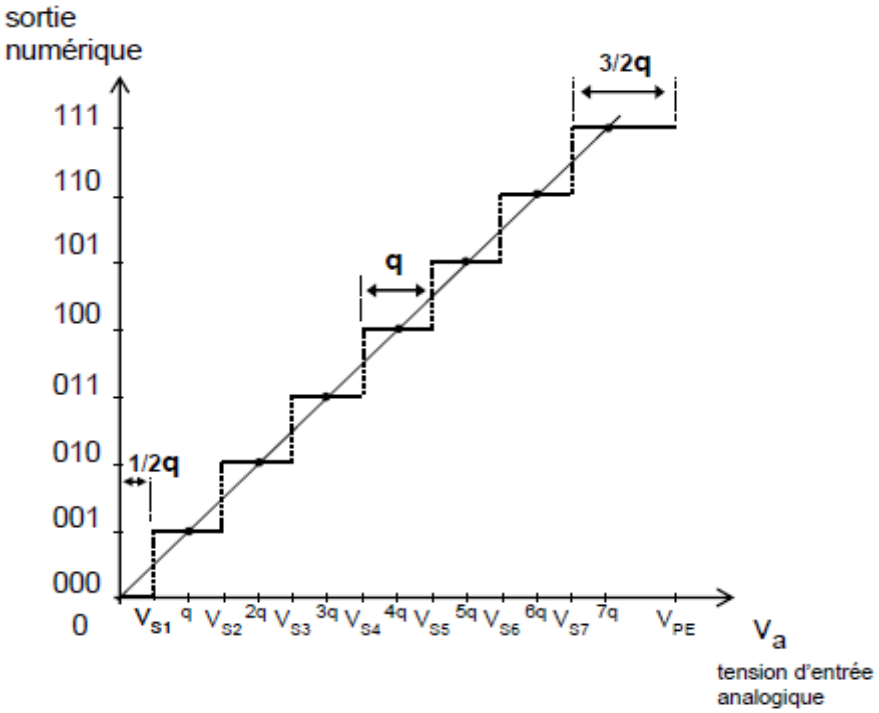


Figure 2.14 Caractéristique de transfert d'un CAN à quantification linéaire centrée.

La figure 2.15 donne l'erreur de codage pour une quantification linéaire centrée. Elle varie entre (-½q) et (+½q) (sauf pour le dernier palier ou elle peut atteindre 1q pour la pleine échelle).

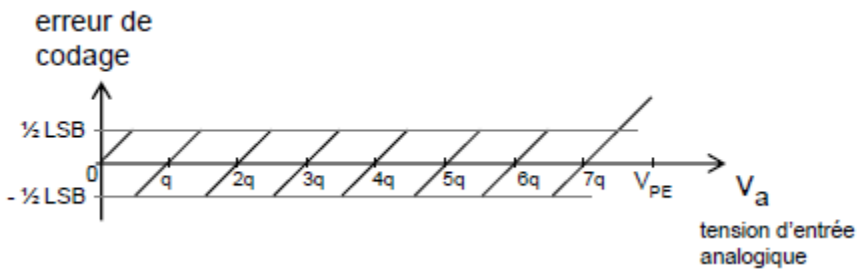


Figure 2.15 Erreur de codage de la quantification linéaire centrée.

Exemple

Soit le signal analogique donné par l'équation $x(t) = 2 + 2\sin(\omega\, t)$, avec une fréquence de $1Hz$. Le signal échantillonné est donné par l'équation $x_{éch}\ (k\ T_{éch}) == 2 + 2\sin(\omega\ k\ T_{éch})$ avec une période d'échantillonnage est $T_{éch} = 0.125s$.

- Calculer l'erreur de quantification eq pour une période (On donne N = 3 bits).

Solution

On a, $x(t) = 2 + 2\sin(\omega\, t)$, avec $f = 1Hz$ donc $x(t) = 2 + 2\sin(2\pi\, t)$, et T = 1s.

$T_{éch} = 0.125s$, l'erreur de quantification est égal à $eq = x_q - x_{éch}$, et ces valeurs numériques sont données dans le tableau 2.3.

Tableau 2.3: Erreur de quantification

k	0	1	2	3	4	5	6	7
$X_{éch}$	2	3.414	4	3.414	2	0.5858	0	0.5858
x_q	2	3.5	3.5	3.5	2	0.5	0	0.5
Valeur binaire	100	111	111	111	100	001	000	001
Erreur eq	0	0.086	- 0.5	0.086	0	-0.0858	0	-0.0858

2.4.3 Convertisseur analogique-numérique CAN bipolaire.

Les caractéristiques précédentes sont celles du CAN unipolaire dont la tension analogique d'entrée est positive. Bien souvent, un même CAN peut être configuré également en mode bipolaire de façon à accepter une tension analogique d'entrée négative ou positive (la plage de variation est alors symétrique entre -½V_{PE} et +½V_{PE}). La figure 2.16 présente la caractéristique de transfert correspondante. 2^N

$$q = \frac{V_{PE}}{2^N - 1} \qquad (2.7)$$

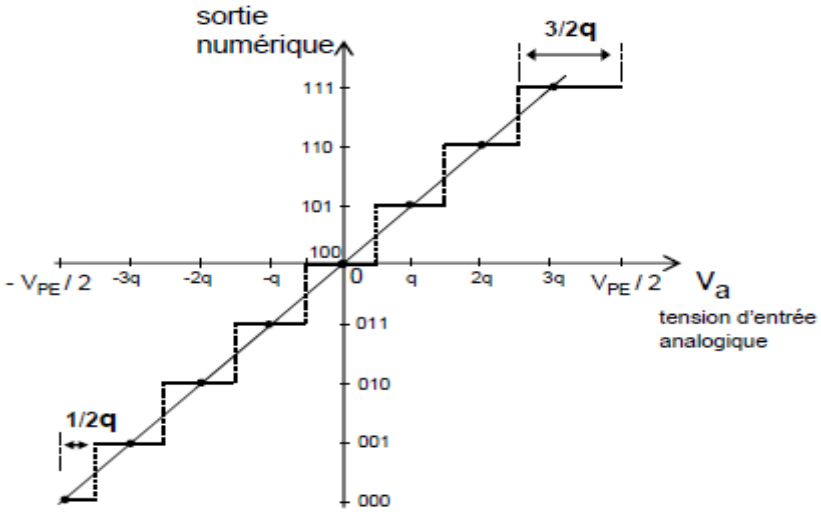


Figure 2.16 Caractéristique de transfert d'un CAN bipolaire.

2.4.4 Bruit de quantification

Une illustration est donnée par la figure 2.17 dans le cas du CAN bipolaire (cas de l'arrondi). Les amplitudes multiples impaires de $q/2$ sont appelées amplitudes de décision.

L'amplitude du signal d'erreur est comprise entre – $q/2$ et $q/2$. Sa puissance mesure la dégradation que subit le signal.

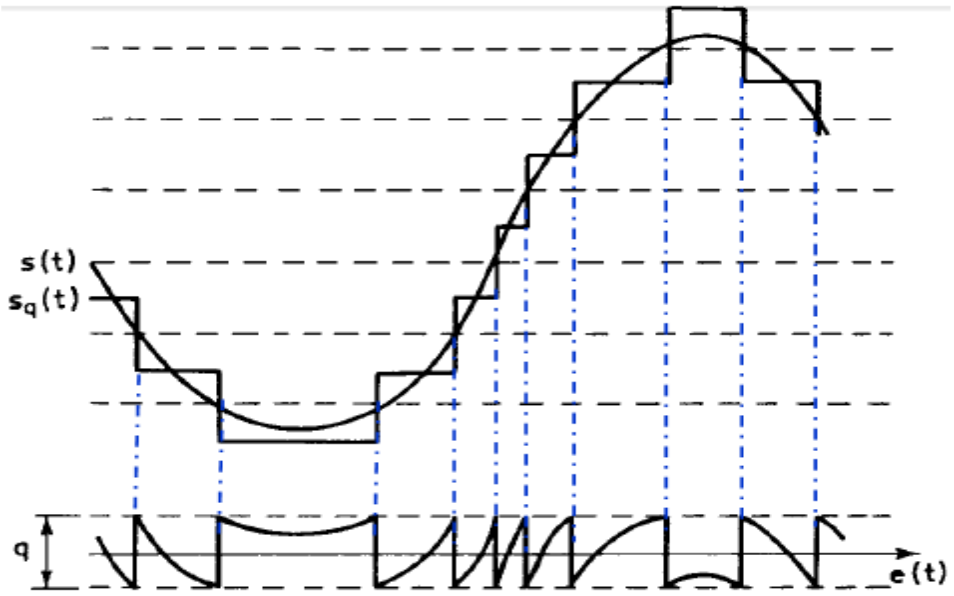


Figure 2.17 Erreur de quantification

Quand les variations du signal sont grandes par rapport à l'échelon de quantification, q, c'est-à-dire que la quantification est faite avec suffisamment de finesse, le signal d'erreur est équivalent à un ensemble de signaux élémentaires, constitués chacun par un segment de droite (voir la figure 2.18).

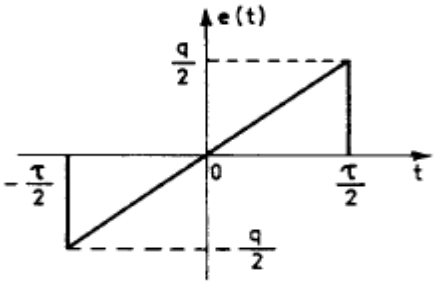


Figure 2.18 Signal d'erreur élémentaire

Sous l'hypothèse, relativement générale, que l'erreur de quantification e_q est uniformément répartie entre (-q/2) et (q/2), on peut calculer la valeur moyenne et l'écart type de cette erreur, σ_q. La densité de probabilité de e_q (notée $P(e_q)$) est dessinée ci-dessous :

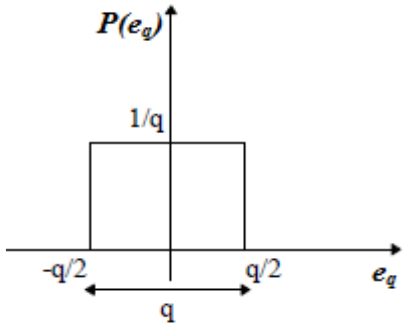


Figure 2.19. La densité de probabilité

Sous cette hypothèse ; on a la densité de probabilité de e_q qui est égal à $P(e_q) = \frac{1}{q}$

$$E(e_q) = \int_{-q/2}^{q/2} e_q \; P(e_q)\, de_q = \frac{1}{q} \int_{-q/2}^{q/2} e_q \; de_q = 0$$

$$P_{Bruit} = E(e_q^2) = \sigma_q^2 = \int_{-q/2}^{q/2} e_q^2 \; P(e_q)\, de_q = \frac{1}{q} \int_{-q/2}^{q/2} e_q^2 \; de_q = \frac{q^2}{12}$$

Donc ; pour $V_{PE} = 2*V_{max}$

$$P_{Bruit} = \sigma_q^2 = \frac{1}{3} \frac{V_{max}^2}{(2^N-1)^2} = \frac{q^2}{12} \qquad (2.8)$$

La valeur de la puissance obtenue, $P_{Bruit} = \frac{q^2}{12}$, est une estimation de la puissance du bruit de quantification suffisante dans la plupart des cas réels.

La plupart du temps les conditions sont remplies pour que la densité spectrale énergétique du bruit de quantification soit constante et l'on retiendra le résultat suivant : Le bruit produit dans l'opération de quantification uniforme avec un échelon q a une puissance qui s'exprime en général par $P_{Bruit} = \frac{q^2}{12}$ et présente une répartition spectrale constante dans la bande de fréquences $[-f_{éch}/2, f_{éch}/2]$.

Le bruit de quantification est le facteur qui limite la précision des échantillons numériques. Celui-ci est défini par le rapport signal au bruit, noté RSB, exprimé de la manière suivante :

$$RSB = \frac{P_{Signal}}{P_{Bruit}} \qquad (2.9)$$

avec P_{Signal}, la puissance du signal (la puissance de crête du codeur) et P_{Bruit}, la puissance du bruit de quantification (la puissance de l'erreur).

Dans le cas général, ou le signal d'entrée $x(t)$ a pour écart type σ_x, le rapport signal au bruit devient :

$$RSB = \frac{P_{Signal}}{P_{Bruit}} = \frac{\sigma_x^2}{\sigma_q^2} = 12\left(\frac{\sigma_x}{q}\right)^2 \quad (2.10)$$

Cette relation générale est applicable à n'importe quel signal analogique.

2.4.5 Dynamique d'une quantification uniforme N bits

Le rapport signal au bruit RSB peut s'exprimer en décibels par la relation suivante :

$$D_{dB} = RSB_{dB} = 10\,log_{10}\left(\frac{P_{Signal}}{P_{Bruit}}\right) = 10\,log_{10}\left(\frac{\sigma_x^2}{\sigma_q^2}\right) \quad (2.11)$$

Où, D, est la dynamique D du codeur [un codeur est un système effectuant une quantification uniforme (arrondi au plus proche voisin) puis une numérisation sur N bits].

En remplaçant P_{Bruit} par sa valeur en fonction de V_{max} :

$$D_{dB} = RSB_{dB} = 10\,log_{10}\left(\frac{P_{Signal}}{P_{Bruit}}\right) = 10\,log_{10}(P_{Signal}) - 10\,log_{10}(P_{Bruit})$$

$$P_{Bruit} = \frac{q^2}{12} = \frac{\left(\frac{V_{PE}}{2^N-1}\right)^2}{12} = \frac{(2*\mathrm{Vmax})^2}{12\,(2^N-1)^2} = \frac{4*\mathrm{Vmax}^2}{12\,(2^N-1)^2} = \frac{1}{3}\frac{V_{max}^2}{(2^N-1)^2}$$

$$D_{dB} = 10\,log_{10}(P_{Signal}) - 10\,log_{10}(V_{max}^2) - [-10\,log_{10}(3(2^N-1)^2)]$$

$$D_{dB} \approx 10\,log_{10}(P_{Signal}) + 10\,log_{10}(3*2^{2N}) - 10\,log_{10}(V_{max}^2)$$

$$D_{dB} \approx 10\,log_{10}(P_{Signal}) + 10\,log_{10}(3) + 10\,log_{10}(2^{2N}) - 10\,log_{10}(V_{max}^2)$$

$$D_{dB} \approx 6.02N + 10\,log_{10}(3) - 10\,log_{10}\left(\frac{V_{max}^2}{P_{Signal}}\right)$$

$$D_{dB} \approx 6.02N + 10\,log_{10}(3) - 20\,log_{10}(\Gamma) \quad (2.12)$$

Avec le facteur de surcharge exprimé par l'équation suivante : $\Gamma = \frac{V_{max}}{\sigma_x}$

Le rapport signal au bruit en dB dépend donc de façon linéaire de la puissance du signal de dB. Il est d'autant plus grand que la puissance du signal est grande.

Cette définition du RSB est valable pour les CAN fonctionnant à la fréquence de Nyquist.

Exemple

Calculer le rapport signal sur bruit d'un CAN idéal avec une entrée sinusoïdale pleine échelle.

Solution

Le rapport signal au bruit d'un CAN idéal avec une entrée sinusoïdale pleine échelle est le quotient entre la valeur efficace du signal $V_{eff,sinus}$ et celle du bruit $V_{eff,bruit}$ (s'agissant d'un CAN idéal le bruit se réduit au bruit de quantification) :

On a ; RSB $= \frac{V_{eff,sinus}}{V_{eff,bruit}}$, Et en *dB*, $D_{dB} = 20\ log_{10}(RSB) = 20\ log_{10}\left(\frac{V_{eff,sinus}}{V_{eff,bruit}}\right)$

Avec, $V_{eff,bruit} = \sqrt{P_{Bruit}} = \frac{q}{\sqrt{12}}$

Pour un codeur N bits, il peut convertir sans saturation des valeurs x compris entre $-V_{max}$ et V_{max} avec $q = \frac{V_{PE}}{2^N-1} = \frac{2*V_{max}}{2^N-1}$, de ce fait : $V_{eff,sinus} = \frac{V_{PE}}{2\sqrt{2}} = \frac{(2^N-1)q}{2\sqrt{2}}$

D'où $RSB = \frac{V_{eff,sinus}}{V_{eff,bruit}} = \frac{(2^N-1)q\sqrt{12}}{2q\sqrt{2}} = (2^N-1)\sqrt{\frac{3}{2}} \approx 2^{N-1}\sqrt{\frac{4*3}{2}} = 2^{N-1}\sqrt{6} = 2^{N-1}\sqrt{6}$

En dB : $D_{dB} = 20\ log_{10}(RSB) = 20\ log_{10}\left((2^N-1)\sqrt{\frac{3}{2}}\right)$

$$D_{dB} = 20\ log_{10}(2^{N-1}) + 20\ log_{10}(\sqrt{3}) - 20\ log_{10}(\sqrt{2})$$

$$D_{dB} \approx 20\ log_{10}(2^N) + 20\ log_{10}(\sqrt{3}) - 20\ log_{10}(\sqrt{2})$$

$$D_{dB} \approx 6.02N + 4.77 - 3.01 = 6.02N + 1.76$$

Donc ; rajouter 1 bit au convertisseur revient à rajouter 6 dB à la dynamique. Il s'agit ici d'une formule bien connue des concepteurs de système d'acquisition.

Conclusion : Choix du pas de quantification et du nombre de bits de codage

On choisira le pas de quantification q en fonction de la précision désirée lors de la conversion et le nombre de bits N en fonction de la dynamique du signal à coder.

Remarque : Dans le cas d'un CAN fonctionnant avec un facteur de sur-échantillonnage (Over-Sampling Ratio - OSR) égal a ($f_{éch}$ / F_{nyq}), le RSB en dB devient :

$$D_{dB} = 6.02N + 1.76 + 10\ \log_{10}(OSR) \qquad (2.13)$$

2.5 Formats des représentations des nombres

Les nombres entiers ''y'' sont représentés dans une base ''b'' quelconque, comme suit :

$$y = \pm(a_1 b^0 + a_2 b^1 + \cdots + a_n b^{N-1}) = \pm \sum_{i=1}^{N} a_i b^{i-1} \qquad (2.14)$$

avec : $a_i \in \{0,1, \ldots, b-1\}$, et, $N \in \mathbb{N}$.

Exemple :

- ✓ **En base 10,**

$(5876)_{10} = 6 * 10^0 + 7 * 10^1 + 8 * 10^2 + 5 * 10^3 = + \sum_{i=1}^{4} a_i 10^{i-1}$, avec, $a_i \in \{0,1,\ldots, 9\}$,

- ✓ **En base 2,**

$$(1101)_2 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = + \sum_{i=1}^{4} a_i 2^{i-1}, avec\ a_i \in \{0,1\}$$

La conversion des nombres décimaux en nombres binaires est obtenue par l'équation suivante :

$$(y)_{10} = + \sum_{i=1}^{n} a_i 2^{i-1}, avec\ a_i \in \{0,1\}\ et\ n \in N \qquad (2.15)$$

Les nombres binaires sont donc écrits avec des 0 et des 1, qui sont représentés physiquement par des bits dans la machine. Le schéma d'une mémoire à n+1 bits (au minimum 8 bits dans un micro-ordinateur) (voir la figure 2.20) :

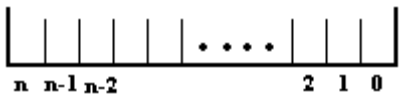


Figure 2.20. Mémoire à n+1 bits

Où ;

- ✓ Les cases du schéma représentent les bits ;
- ✓ Le chiffre marqué en dessous d'une case indique la puissance de 2 à laquelle est associé ce bit (on dit aussi le *rang* du bit) ;
- ✓ Le bit de rang **0** est appelé le bit de **poids faible** ;
- ✓ Le bit de rang **n** est appelé le bit de **poids fort**.

Exemple

$$(13)_{10} = 1 + 4 + 8 = + \sum_{i=1}^{4} a_i 2^{i-1}, avec\ a_i \in \{0,1\} = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3$$

$$(13)_{10} = + \sum_{i=1}^{4} a_i 2^{i-1}, avec\ a_i \in \{0,1\} = \mathbf{1} * 2^0 + \mathbf{0} * 2^1 + \mathbf{1} * 2^2 + \mathbf{1} * 2^3 = (1101)_2$$

2.5.1 Codage des nombres entiers

Il existe quatre types de codage, à savoir : a) codage des nombres entiers positifs, b) codage des nombres entiers relatifs, c) codage complément à 2, et d) codage binaire codé décimal (BCD).

2.5.1.1 Codage des nombres entiers positifs ou non signés

Ce codage est celui dans lequel les nombres entiers naturels sont écrits en numération binaire. Il s'appelle aussi : le code binaire pur, le code binaire simple, et code naturel.

Pour une mémoire à n+1 bits (n>1), tous les entiers naturels de l'intervalle [0, 2^{n+1} -1] seront représentés. Donc :

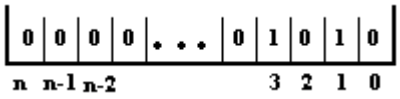
- ✓ Soit pour n + 1 = 8 bits, tous les nombres naturels de l'intervalle [0, 255] seront représentés.
- ✓ Soit pour n + 1 = 16 bits, tous les nombres naturels de l'intervalle [0, 65535] seront représentés.

Exemple

1) $(1010)_2 = +\sum_{i=1}^{4} a_i 2^{i-1} = a_1 2^0 + a_2 2^1 + a_3 2^2 + a_4 2^3$

Donc ; $(1010)_2 = 0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3$

Dans la mémoire il est codé :



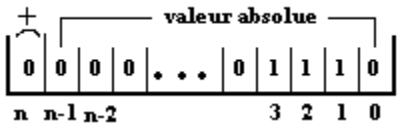
2) Le tableau 2.4 représente un exemple sur le codage binaire pour n+1 = 8 bits.

Tableau 2.4: Codage binaire simple pour n+1 = 8 bits

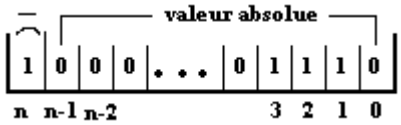
Signal électrique analogique	⇔	Code binaire
0 mV	CAN ⇒	0000 0000
…		…
53 mV		0011 0101
…		…
255 mV	⇐ CNA	1111 1111

2.5.1.2 Codage des nombres entiers relatifs

Ce codage permet la représentation des nombres entiers relatifs (les nombres entiers signés). Dans la représentation en binaire signé, le bit de poids fort sert à représenter le signe (0 pour un entier positif et 1 pour un entier négatif), les **n** autres bits représentent la valeur absolue du nombre en binaire pur (voir la figure 2.21).



*+14 est représenté par **0**0000...01110*



*-14 est représenté par **1**000...01110*

Figure 2.21 Représentation en binaire signé des nombres (+14) et (-14).

Pour une mémoire à **n+1** bits **(n > 1)**, tous les entiers naturels de l'intervalle $\left[-\left(\frac{2^n-1}{2}\right), \frac{2^n-1}{2}\right]$ seront représentés. Donc :

- Soit pour n + 1 = 8 bits, tous les nombres entiers de l'intervalle [-127, 127] seront représentés.
- Soit pour n + 1 = 16 bits, tous les nombres entiers de l'intervalle [-32767, 32767] seront représentés.
- Le nombre zéro est représenté dans cette convention (dites du zéro positif) par : **0**000...00000.
- Il reste malgré tout une configuration mémoire inutilisée : **1**000...00000. Cet état binaire ne représente à priori aucun nombre entier ni positif ni négatif de l'intervalle $[-(2^n - 1), 2^n - 1]$. Afin de ne pas perdre inutilement la configuration " **1**000...00000 ", les informaticiens ont décidé que cette configuration représente malgré tout un nombre négatif parce que le bit de signe est **1**, et en même temps la puissance du bit contenant le "**1**", donc $-\mathbf{2^n}$. L'intervalle de représentation se trouve alors augmenté d'un nombre, il devient $\left[-\left(\frac{2^n}{2}\right), \frac{2^n-1}{2}\right]$. Donc :
  - ✓ Soit pour n + 1 = 8 bits, tous les nombres entiers relatifs de l'intervalle [-128, 127] seront représentés.
  - ✓ Soit pour n + 1 = 16 bits, tous les nombres entiers relatifs de l'intervalle [-2768, 32767] seront représentés.

Remarque

Ce codage n'est pas utilisé tel quel, il est donné ici à titre pédagogique. En effet le traitement spécifique du signe coûte cher en circuits électroniques et en temps de calcul. C'est une version améliorée de ce codage qui est utilisée dans la plupart des calculateurs : elle se nomme le complément à deux.

2.5.1.3 Codage complément à 2 (complément vrai)

Ce codage, purement conventionnel et très utilisé de nos jours, il est dérivé du binaire signé, sert à représenter en mémoire les entiers relatifs.

Comme dans le binaire signé, la mémoire est divisée en deux parties inégales ; le bit de poids fort représentant le signe, le reste représente la valeur absolue avec le codage suivant :
Supposons que la mémoire soit à **n + 1** bits, soit **x** un entier relatif à représenter.

- Si **x > 0,** alors c'est la convention en *binaire signé* qui s'applique (le bit de signe vaut 0, les **n** bits restants codent le nombre), soit pour le nombre +14, il est représenté par **0000...01110**

- Si x < 0, alors :
  - ✓ On code |**x**| en binaire signé.
  - ✓ Puis l'on complémente tous les bits de la mémoire (complément à 1 ou complément restreint). Cette opération est un non logique effectué sur chaque bit de la mémoire.
  - ✓ Enfin l'on additionne 1 au nombre binaire de la mémoire (addition binaire).

Exemple

1) Soit à représenter le nombre (-14)

La figure 2.22 représente les différentes étapes pour obtenir la présentation du nombre (-14).

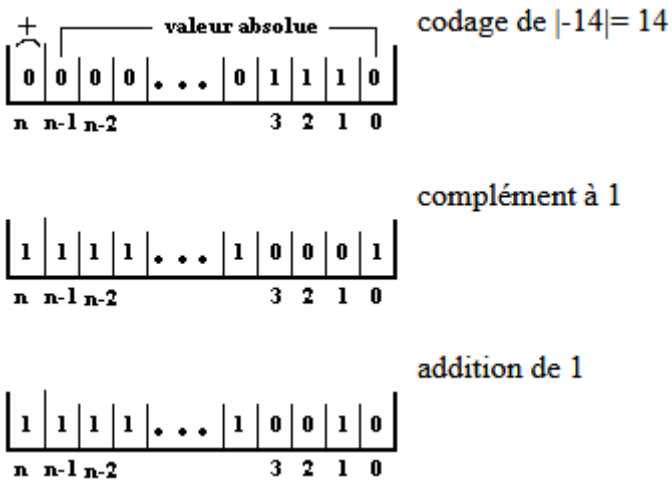


Figure 2.22 Représentation en binaire signé des nombres (+14) et (-14).

Donc, le nombre -14 s'écrit en complément à 2 par : 1111...10010.

2) Le tableau 2.5 représente un exemple sur le codage binaire pour n+1 = 8 bits.

Tableau 2.5 : Codage complément à 2 pour n+1 = 8 bits

Signal électrique analogique	⇔	Code binaire
-128 mV		1000 0000
-127 mV		1000 0001
…	CAN ⇒	…
-1 mV		1111 1111
0 mV		0000 0000
1 mV	⇐ CNA	0000 0001
…		…
127 mV		0111 1111

Remarque

Un des intérêts majeurs de ce codage est d'intégrer la soustraction dans l'opération de codage et de n'effectuer que des opérations simples et rapides (non logique, addition de 1). En effet, la somme d'un nombre et de son complément à 2 donne bien un résultat nul.

2.5.1.4 Codage binaire codé décimal (BCD)

Le nombre décimal est codé en considérant chaque chiffre du nombre et en donnant sa représentation sur 4 bits (0 à 9). Ce codage conduit à des nombres binaires plus longs puisque toutes les combinaisons binaires ne sont pas utilisées. Ce codage est très utile lors de la sortie des informations sur des afficheurs, l'information étant disponible directement sous forme décimale (voir la figure 2.6).

Tableau 2.6 : Codage binaire codé décimal pour n+1 = 8 bits

Signal électrique analogique	⇔	Code binaire
0 mV		1000 0000
…		…
9 mV	CAN ⇒	0000 1001
10 mV		0001 0000
11 mV		0001 0001
…	⇐ CNA	…
99 mV		1001 1001

2.5.2 Représentation des nombres réels dans un calculateur

La représentation à virgule en base ''**b**'' d'un nombre réel est donnée par l'égalité suivante :

$$x = \pm \left(a_{n-1} \dots a_0, d_1 d_2 \dots d_p\right)_b \text{ , avec, } a_i, d_k \in \{0, \dots, b-1\} \qquad (2.16)$$

Et l'écriture à virgule en base ''**b**'' du nombre réel est donnée par l'équation suivante :

$$x = \pm \left(\sum_{i=0}^{n-1} a_i b^i + \sum_{k=1}^{p} \frac{d_k}{b^k}\right) \qquad (2.17)$$

On identifie x et sa représentation à virgule en base ''**b**'', donc on peut écrire :

$$x = \pm\, a_{n-1} \dots a_0, d_1 d_2 \dots d_p = \pm \left(\sum_{i=0}^{n-1} a_i b^i + \sum_{k=1}^{p} \frac{d_k}{b^k}\right) \qquad (2.18)$$

Exemple

$$(1010{,}011)_2 = \left(\sum_{i=0}^{4-1} a_i 2^i + \sum_{k=1}^{3} \frac{d_k}{2^k}\right) = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + \frac{0}{2^1} + \frac{1}{2^2} + \frac{1}{2^3}$$

Donc ; $(1010{,}011)_2 = (10.375)_{10}$

2.5.2.1 Virgule fixe

L'écriture normalisée en base ‘‘b’’ du nombre $x \in D_b$ non nul s'écrit de façon unique $x = \pm m * b^e$, avec $m \in D_b$, $m \in [1, b[$, et, $e \in Z$. Où ;

- ✓ On appelle m la mantisse de x et e l'exposant de x.
- ✓ Les nombres de l'ensemble $D_b = \left\{ \frac{k}{b^n} / k \in Z \text{ et } n \in N \right\}$ ont une représentation à virgule finie (un nombre fini de chiffres après la virgule) en base ‘‘b’’.
- ✓ Cette écriture normalisée n'est qu'un décalage de la virgule.

Exemple

- ✓ **En base 10 :** $5287,34 = 5,28734 \times 10^3$ (m = 5.28734 et e = 3)
- ✓ **En base 2 :** $(1010,011)_2 = (1,010011)_2 \times 2^{(11)_2} = (1,010011 \times 10^{11})_2$, avec ; m = $(1,010011)_2$ et e = $(0011)_2$.

2.5.2.2 Virgule flottante

Les nombres flottants sont l'ensemble des nombres x à virgule en base b ayant une écriture normalisée du type donné par l’équation suivante :

$$x = \pm m * b^e \qquad (2.19)$$

Avec, $m \in D_b$, $m \in [1, b[$, et, $e \in Z$, la valeur de m et e dans une certaine plage, précisée dans le paragraphe suivant.

Dans la base 2, l’écriture normalisée des nombres à virgule flottants x est donnée par l’équation suivante :

$$x = \pm m * 2^e \qquad (2.20)$$

Avec, $m \in D_2$, $m \in [1, 2[$, et, $e \in Z$, la valeur de m et e dans une certaine plage.

Exemple

Représenter le réel 12.575 en virgule flottante normalisée.

Correction

$(17.575)_{10} = (00010001.1001001\ldots)_2 = (1.00011001001\ldots \times 10^{100})_2$

Où ; m = $(1.00011001001)_2$, et e = $(100)_2 = (4)_{10}$

2.5.2.3 Flottants sur 32 bits ou sur 64 bits

On veut représenter un nombre flottant (normalisé) x sur un certain nombre de bits (32 ou 64 typiquement aujourd'hui). Il faut partager les bits disponibles entre :

- ✓ Le signe de x (facile, un bit suffit) ;
- ✓ L’exposant de x (la puissance de x) ;
- ✓ La mantisse de x.

La norme IEEE 754 de1985, complétée en 2008, (Institute of Electrical and Electronics Engineers) décrit entre autres la représentation informatique des nombres flottants sur :

- **Le nombre *x* codé sur 32 bits** sous la forme $x = s\ e_1\ e_2 \dots e_8 f_1 f_2 \dots f_{23}$, (32=1+8+23)

Le nombre flottant, x, est calculé par l'équation suivante :

$$x = (-1)^S\ 2^E\ m = (-1)^S\ 2^{e-127}\ [1 + (f_1 f_2 \dots . f_{23})_2 * 2^{-23}] \qquad (2.21)$$

Avec,

- ✓ $S = 0$ si $x > 0$ et $S = 1$ si $x < 0$. Le bit S est le bit de signe.
- ✓ $E = (e_1\ e_2 \dots e_8)_2 - 127$. Les bits $e_1\ e_2 \dots e_8$ sont les bits d'exposant et e ∈ {−127, ..., +128}. Où ; $\{-127 = -(2^8 - 1)/2), \dots, +128 = (2^8/2)\}$
- ✓ $m = (1, f_1 f_2 \dots f_{23})_2$. Les bits $f_1 f_2 \dots f_{23}$ sont les bits de mantisse.

On appelle ces nombres les nombres flottants (normalisés) "simple précision".

S	e	e	e	e	e	e	e	e	f	f	f	f	f	···	f	f	f	f	f	f	f	f
31	30	29	28	27	26	25	24	23	22	21	20	19	18		7	6	5	4	3	2	1	0

Figure 2.23 Format simple précision

- **Le nombre *x* codé sur 64 bits** sous la forme $x = s\ e_1\ e_2 \dots e_{11} f_1 f_2 \dots f_{52}$, (64=1+11+52)

Le nombre flottant, x, est calculé par l'équation suivante :

$$x = (-1)^S\ 2^E\ m = (-1)^S\ 2^{e-1023}\ [1 + (f_1 f_2 \dots . f_{52})_2 * 2^{-52}] \qquad (2.22)$$

Avec ;

- ✓ $S = 0$ si $x > 0$ et $s = 1$ si $x < 0$. Le bit s est le bit de signe.
- ✓ $E = (e_1\ e_2 \dots e_{11})_2 - 1023$. Les bits $e_1\ e_2 \dots e_{11}$ sont les bits d'exposant et e ∈ {−1023, ..., + 1024}. $\{-1023 = -(2^{11} - 1)/2), \dots, +1024 = (2^{11}/2)\}$
- ✓ $m = (1, f_1 f_2 \dots f_{52})_2$. Les bits $f_1 f_2 \dots f_{52}$ sont les bits de mantisse.

On appelle ces nombres les nombres flottants (normalisés) "double précision".

S	e	e	e	e	e	e	e	e	e	e	e	e	f	f	f	f	···	f	f	f	f	f
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48		4	3	2	1	0

Figure 2.24 Format double précision

Remarque

Dans les deux cas, le premier bit de la mantisse (parfois appelé bit caché) vaut toujours 1 et n'est pas compté dans la mantisse.

Exemple

Soit la série suivante : **00111111100000000000000000000000**

- Quel est le nombre flottant, x, donné par cette série ?
- Quel est son code ?

Correction

0	**0 1 1 1 1 1 1 1**	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
S	e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8	f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 f_9 f_{10} f_{11} f_{12} f_{13} f_{14} f_{15} f_{16} f_{17} f_{18} f_{19} f_{20} f_{21} f_{22} f_{23}

Le nombre flottant, x, donné par la série : 00111111100000000000000000000000, est calculé par l'équation suivante : $x = (-1)^S\, 2^E\, m = (-1)^S\, 2^{e-127}\, [1 + (f_1 f_2 \ldots . f_{23})_2 * 2^{-23}]$

Avec ;

- ✓ Le bit de signe : $S = 0$.
- ✓ $E = (e_1\, e_2 \ldots e_8)_2 - 127 = (01111111)_2 - 127 = 2^{0+}2^{1+}2^{2+}2^{3+}2^{4+}2^5 + 2^6 = 127\text{-}127 = 0$.
- ✓ La mantisse : $m = (1, f_1 f_2 \ldots f_{23})_2 = (1{,}00000000000000000000000)_2$.

Donc : $x = (-1)^0\, 2^0\, (1.0)_2 = (1.0)_2 = 1.0$

X=1.0 est codée par 0x3F800000,

0	**0 1 1**	**1 1 1 1**	**1**000	0000	0000	0000	0000	0000
S	3	F	8	0	0	0	0	0

Chapitre 3 : Architecture des DSP TMS320C6x

3.1 Introduction

Parmi les constructeurs principaux de DSP : Texas Instruments et Analog Devices (ADSP), Motorola, et Lucent…

Notre chapitre est concerné par les DSP fabriqués par Texas Instruments (désignés par TMS).

La première génération du processeur de signal numérique est TMS32010 en 1982, le TMS320C25 en 1986 et le TMS320C50 en 1991. Plusieurs versions de chacun de ces processeurs (C1x, C2x et C5x) sont disponibles avec des caractéristiques différentes, telle qu'une vitesse d'exécution plus rapide. Ces processeurs de 16 bits sont tous des processeurs à virgule fixe et sont compatibles avec le code. Les applications DSP typiques nécessitent plusieurs accès à la mémoire au cours d'un cycle d'instructions. Les processeurs à point fixe C1x, C2x et C5x sont basés sur une architecture de Harvard modifiée avec des espaces mémoires séparés pour les données et les instructions permettant des accès simultanés.

Le processeur à virgule flottante TMS320C30 a été introduit à la fin des années 1980. Le C31, le C32 et le C33 appartiennent tous à la famille de processeurs à virgule flottante C3x. Les processeurs à virgule flottante C4x, présentés ultérieurement, sont compatibles avec le code des processeurs C3x et sont basés sur l'architecture de Harvard modifiée.

La deuxième génération est apparue il y'a quelques années avec l'apparition du TMS320C6x de Texas Instruments. Le TMS320C6201 (C62x), annoncé en 1997, est le premier membre de la famille des processeurs de signaux numériques à virgule fixe C6x. Contrairement aux processeurs à point fixe précédents, C1x, C2x et C5x, le C62x est basé sur une architecture Very-Long-Instruction-Word (VLIW), utilisant toujours des espaces mémoires séparés pour les instructions et les données, comme dans l'architecture de Harvard. L'architecture VLIW a des instructions plus simples, mais il en faut plus pour une tâche qu'avec une architecture DSP conventionnelle.

Plusieurs facteurs, tels que le coût, la consommation d'énergie et la vitesse, entrent en jeu lors du choix d'un processeur de signal numérique spécifique. Les processeurs C6x sont particulièrement utiles pour les applications nécessitant des calculs intensifs. Les membres de la famille du C6x incluent des processeurs à virgule fixe (par exemple, C62x, C64x) et à virgule flottante (par exemple, C67x).

Un processeur à virgule flottante est généralement plus cher, car il a plus propriété réel « ''Real estate'' ou est une puce plus grosse en raison des circuits supplémentaires nécessaires pour gérer l'arithmétique des nombres entiers et en virgule flottante.

Un processeur à virgule fixe convient mieux aux appareils utilisant des batteries, tels que les téléphones cellulaires, car il consomme moins d'énergie qu'un processeur à virgule flottante équivalent. Les processeurs à virgule fixe C1x, C2x et C5x sont des processeurs de 16 bits à plage dynamique et précision limitées. Le processeur à virgule fixe C6x est un processeur de 32 bits à plage dynamique et précision améliorées. Dans un processeur à virgule fixe, il est nécessaire de mettre à l'échelle les données (pour éviter le problème du débordement (Overflow)). Le tableau 3.1 résume les différentes générations des processeurs TMS320Cx.

Tableau 3.1: Les processeurs TMS320Cx

TMS320	Virgule fixe ou flottante	Architecture	Utilisation
C1x	16 bits – Fixe	Harvard modifiée avec des espaces mémoires séparés pour les données et les instructions permettant des accès simultanés.	Pour le contrôle des disques durs dans les ordinateurs.
C2x	16 bits – Fixe	Harvard modifiée avec des espaces mémoires séparés pour les données et les instructions permettant des accès simultanés.	Servent au fonctionnement de fax
C5x	16 bits – Fixe	Harvard modifiée avec des espaces mémoires séparés pour les données et les instructions permettant des accès simultanés.	Dans les modems
C3x	32 bits – Flottante	Harvard modifiée	Pour les systèmes Hi-Fi, à synthèse vocale, et dans les processeurs graphiques à 3 dimensions
C4x	32 bits – Flottante	Harvard modifiée	Pour le fonctionnement en parallèle, avec d'autres systèmes processeurs (applications : la « réalité virtuelle » et la reconnaissance d'image).

C6x	**C62x** **C64x**	32 bits – Fixe	Very – Long - Instruction -Word (VLIW) utilisant toujours des espaces mémoire séparés pour les instructions et les données, comme dans l'architecture de Harvard.	Sont particulièrement utiles pour les applications nécessitant des calculs intensifs.
	C67x	32 bits – Flottante		

3.2 Le processeur

La plateforme TMS320C6000 des processeurs de signaux numériques fait partie de la famille TMS320. Elle comporte les processeurs TMS320C62x à arithmétique fixe et TMS320C67x à arithmétique flottante. Afin de bien comprendre le fonctionnement des DSP, il faut voir ce processeur comme un ensemble de blocs interconnectés (voire la figure 3.1).

Les principales caractéristiques d'un DSP ''TMS320C6x'' sont :

- Les architectures de ces processeurs sont Harvard (mémoire de programme et mémoire de données séparées) et VLIW (Very Long Instruction Word) d'ordre 8, c'est-a-dire qu'ils sont capables d'exécuter jusqu'a 8 instructions de 32-bits en parallèle par les 8 unités fonctionnelles.
- Le CPU (Central Process Unit) est constitué de 32 registres généraux de 32-bits, et de 8 unîtes de traitement, 2 multiplieurs et 6 ALU (Arithmétique and Logic Units).
- DMA (Direct Memory Access) : 4/16 channels : Elle permet de faire des transferts des données sans passer par le CPU.

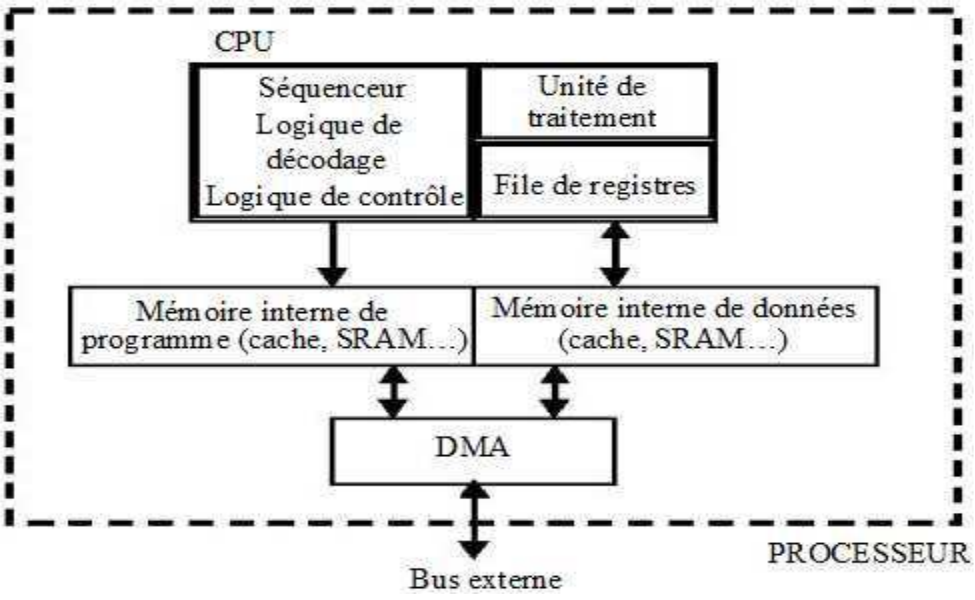


Figure 3.1. L'organisation en blocs d'un DSP basé sur une architecture VLIW et Harvard.

❖ Une EMIF (External Memory Interface): Glue less access to async/sync memory EPROM, SRAM, SDRAM, SBSRAM: Elle permet de faire l'interfaçage entre les différents types de mémoire externe et le DSP.

3.3 Architecture interne du C6000

Le détail de l'architecture interne du DSP basé sur le cœur C6000 (dont est dérivé le C6x) est représenté dans la figure 3.2.

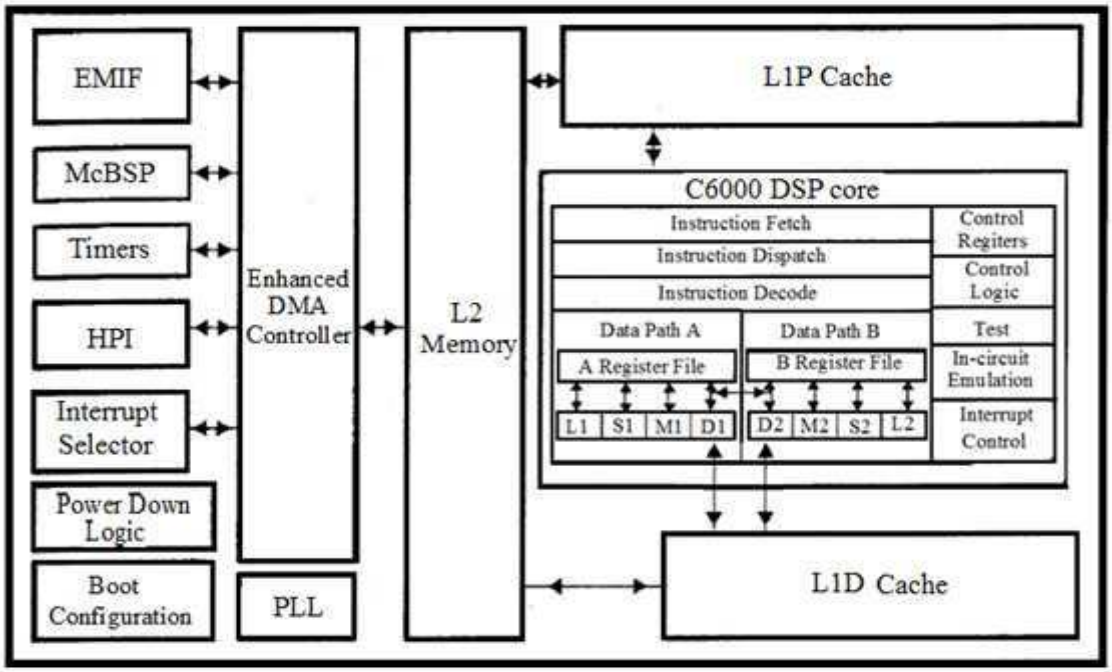


Figure 3.2. Schéma fonctionnel de TMS320C6x.

Le DSP TMS320C6000 intègre un CPU (Central Processing Unit) des mémoires internes de programme et de donnée, des bus internes de programmes et de données et des périphériques internes.

- Le cœur C6000 qui est représenté dans la figure 3.3, possède :
  - ✓ Une unité d'extraction de programme (programme Fetch) ;
  - ✓ Une unité d'exécution des instructions (Instruction Dispatch) ;
  - ✓ Une unité de décodage des instructions (Instruction Decode) ;
  - ✓ 32 registres de 32- bits séparent en deux files contenantes chacun de 16 registres (Register File A et Register File B) ;
  - ✓ Deux voies pour les données (data path) contenant chacune quatre unités de traitement (L, S, M, D) ;
  - ✓ Deux registres de contrôle (Control Regiters) ;

- ✓ Une logique de contrôle (Control Logic) ;
- ✓ Le test (Test), Émulation en circuit (In-circuit Emulation), Contrôle d'interruption (Interrupt Control).
- ✓ Les unités exécutent des instructions logiques, de décalage (Shifting), des multiplications hardwares et des opérations sur les adresses de données. Tous les transferts de données entre la file de registre et la mémoire se font exclusivement par les unités ≪Data adressing≫.
- ✓ Les quatre chemins sont croisés, ce qui permet l'échange de données entre les deux files de registre.

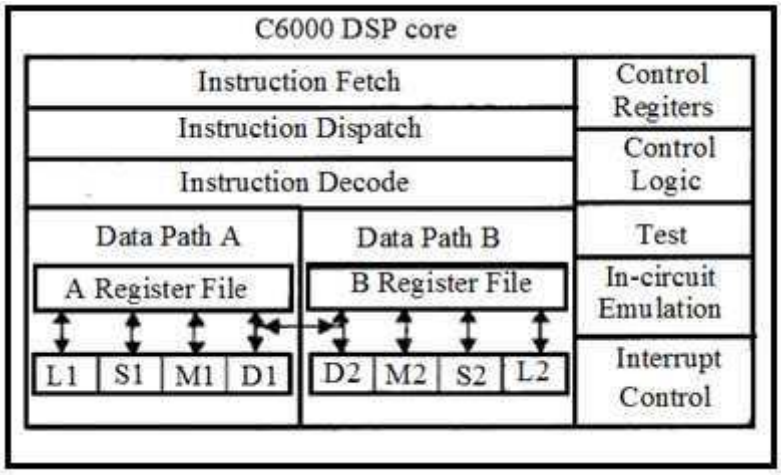


Figure 3.3 Le cœur C6000.

- Les périphériques sur puce comprennent :
  - ✓ Deux ports sériés multicanaux à mémoire tampon (McBSP) (Two Multichannel Buffered Serial Ports) ;
  - ✓ Deux minuteries (Two Timers) ;
  - ✓ Une interface de port hôte de 16 bits (HPI) (Host Port Interface) ;
  - ✓ Une interface de mémoire externe 32 bits (EMIF) (External Memory Interface).

  Il nécessite 3,3 V pour les E/S et 1,8 V pour le noyau (interne).
- Les bus internes comprennent :
  - ✓ Un bus d'adresses de programme de 32 bits ;
  - ✓ Un bus de données de programme de 256 bits pour recevoir les huit instructions de 32 bits ;
  - ✓ Deux bus d'adresses de données de 32 bits ;
  - ✓ Deux bus de données de 64 bits ;
  - ✓ Deux bus de données de stockage de 64 bits ;

- ✓ Avec un bus d'adresse de 32 bits, l'espace mémoire total est de $\mathbf{2^{32} = 4294967296}$ donc $\mathbf{2^{32} \approx 4\ GB}$, y compris les quatre espaces mémoires externes : CE0, CE1, CE2 et CE3.

- Les banques de mémoires indépendantes sur le C6x permettent deux accès mémoires en un cycle d'instruction. Deux banques de mémoires indépendantes sont accessibles via deux bus indépendants. La mémoire interne étant organisée en banques de mémoires, deux instructions de chargement ou de stockage peuvent être exécutées en parallèle. Aucun conflit ne se produit si les données consultées se trouvent dans différentes banques de mémoire. Des bus distincts pour les programmes, les données et l'accès direct à la mémoire (DMA) (Direct Memory Access) permettent au C6x d'exécuter simultanément des opérations d'extraction de programmes, de lecture et d'écriture de données et d'opérations DMA. Avec des données et des instructions résidant dans des espaces mémoire séparés, des accès mémoires simultanés sont possibles. Le C6x dispose d'un espace mémoire adressable sur des octets. La mémoire interne est organisée en espaces de mémoires de programmes et de données séparés, avec deux ports internes de 32 bits (deux ports de 64 bits avec le C64x) pour accéder à la mémoire interne.
- La fréquence d'horloge est correspondant ou nombre des impulsions par seconde s'exprimé en Hertz (Hz) :

$$\mathbf{Fréquence\ (Hz) = Nbr\ (IMP/Sec)} \qquad (3.1)$$

Où ; Nbr (IMP/Sec) est le nombre des impulsions par seconde.

Donc pour une fréquence d'horloge égal à 150 MHz le nombre d'impulsions par seconde est égale à :

150 GHz = $150*10^6$ Hz = 150 000 000 IMP/Sec =150 millions d'IMP/Sec.

- Le temps de cycle d'instruction t_{CI} est calculé par l'équation suivante :

$$\mathbf{t_{CI} = 1 / Fréquence\ (Hz)} \qquad (3.2)$$

Fonctionnant à 150 MHz correspond à un temps de cycle d'instructions égal à :

$t_{CI} = 1 / 150\ 000000 = 6.67 * 10^{-9}$ s = 6.67 ns.

3.4 Cartographie de mémoire

Le tableau 3.2 représente un résumé de la carte mémoire (cartographie de mémoire) d'un DSP 'TMS320C6000'.

Tableau 3.2 Cartographie de mémoire d'un DSP 'TMS320C6000'

Plage d'adresses (Hex)	Taille (octets)	Description du bloc de mémoire
0000 0000—0000 FFFF	64K	RAM interne (L2)
0001 0000—017F FFFF	24M–64K	Réservé
0180 0000—0183 FFFF	256K	Registres EMIF du bus de configuration interne
0184 0000—0187 FFFF	256K	Registres de contrôle L2 du bus de configuration interne
0188 0000—018B FFFF	256K	Registre HPI de bus de configuration interne
018C 0000—018F FFFF	256K	Registres McBSP 0 du bus de configuration interne
0190 0000—0193 FFFF	256K	Registres McBSP 1 du bus de configuration interne
0194 0000—0197 FFFF	256K	Registre 0 minuterie du bus de configuration interne
0198 0000—019B FFFF	256K	Registre 1 minuterie du bus de configuration interne
019C 0000—019F FFFF	256K	Registres de sélecteur d'interruption du bus de configuration interne
01A0 0000—01A3 FFFF	256K	Bus de configuration interne EDMA RAM et registres
01A4 0000—01FF FFFF	6M–256K	Réservé
0200 0000—0200 0033	52	Registres de QDMA
0200 0034—2FFF FFFF	736M–52	Réservé
3000 0000—3FFF FFFF	256M	McBSP 0/1 données
4000 0000—7FFF FFFF	1G	Réservé
8000 0000—8FFF FFFF	256M	Interface de mémoire externe CE0
9000 0000—9FFF FFFF	256M	Interface de mémoire externe CE1
A000 0000—AFFF FFFF	256M	Interface de mémoire externe CE2
B000 000—BFFF FFFF	256M	Interface de mémoire externe CE3
C000 0000—FFFF FFFF	1G	Réservé

3.5 Unités fonctionnelles

Le CPU se compose de huit unités fonctionnelles indépendantes divisées en deux chemins de données A et B, comme illustré dans le tableau 3.3. Chaque chemin comporte :

- ✓ Une unité pour les opérations de multiplication (.M),
- ✓ Une unité pour les opérations logiques et arithmétiques (.L),
- ✓ Une unité pour les opérations de branche, de manipulation de bits et arithmétiques (.S),
- ✓ Une unité pour les opérations de chargement / stockage et arithmétiques (.D).

- Les unités .S et .L sont destinées aux instructions arithmétiques, logiques et de branchement. Tous les transferts de données utilisent les .D units.

- Les opérations arithmétiques, telles que la soustraction ou l'addition (SUB ou ADD), peuvent être effectuées par toutes les unités, à l'exception des unités .M (une par chemin de données).
- Les huit unités fonctionnelles sont constituées de quatre ALU à virgule flottante / fixe (.L units et .S units), de deux ALU à virgule fixe (.D units) et de deux multiplicateurs à virgule flottante / fixe (.M units).
- Chaque unité fonctionnelle peut lire ou écrire directement dans le fichier de registre dans son propre chemin.
- Les unités se terminant par 1 écrivent dans le fichier de registre A et les unités se terminant par 2 écrivent dans le fichier de registre B.
- Deux chemins croisés (1x et 2x) permettent aux unités fonctionnelles d'un chemin de données d'accéder à un opérande de 32 bits à partir du fichier de registres situé du côté opposé.
- Il peut y avoir un maximum de deux lectures de sources croisées par cycle.
- Chaque côté de l'unité fonctionnelle peut accéder aux données des registres du côté opposé en utilisant un chemin croisé (c'est-à-dire que les unités fonctionnelles d'un côté peuvent accéder au jeu de registres de l'autre côté).

Tableau 3.3 Les unités fonctionnelles et ces opérations

Unité fonctionnelle	**Opérations**
Unité .L (.L1,.L2)	Arithmétique et opérations de comparaison sur 32/40 bits 1 ou 0 bit le plus à gauche comptant pour 32 bits Compte de normalisation pour 32 et 40 bits Opérations logiques sur 32 bits
Unité .S (.S1, .S2)	Opérations arithmétiques sur 32 bits Décalages sur 32/40 bits et opérations de champ binaire sur 32 bits Opérations logiques sur 32 bits Branches Génération constante Transferts de registre vers/depuis le fichier de registre de contrôle (.S2 uniquement)
Unité .M (.M1, .M2)	Opérations de multiplication 16 × 16 bits
Unité .D (.D1, .D2)	Addition, soustraction, calcul d'adresse linéaire et circulaire sur 32 bits Charge et stocke avec un décalage constant de 5 bits Charge et stocke avec un décalage constant de 15 bits (.D2 uniquement)

- Le nombre des multiplications et des accumulations par seconde N_{MAC} est donné par l'équation suivante :

$$\mathbf{N_{MAC} = Nbr\ (IMP/Sec) * Nbr\ (U_{MAC})} \quad (3.3)$$

Où, **Nbr (U_{MAC})** est le nombre des unités fonctionnelles capables de gérer des opérations de multiplication et d'accumulions par cycle.

Avec une horloge de 150 MHz à bord du DSK, on peut atteindre idéalement deux multiplications et accumulations par cycle, donc :

N_{MAC} = 150000000 * 2 = 300000000 MAC/sec = 300 millions MAC/sec.

- Le nombre des opérations en virgule flottante N_{MFLOPS} (Million Floating-Point Operations Per Second) est donné par l'équation suivante :

$$\mathbf{N_{MFLOPS} = Nbr\ (IMP/Sec) * Nbr\ (U_{FPOU})} \quad (3.4)$$

Où ; **Nbr (U_{FPOU})** est le nombre des unités fonctionnelles capables de gérer des opérations en virgule flottante.

Avec six des huit unités fonctionnelles de la figure 3.2 (et non les unités .D décrites ci-dessous) capables de gérer des opérations en virgule flottante. Le nombre des opérations en virgule flottante par seconde est calculé par l'équation suivante :

N_{MFLOPS} = 150 000000 * 6 = 900 000000 FLOPS = 900 MFLOPS

- Le nombre total d'instruction par second N_{MIPS} est calculé par l'équation suivante :

$$\mathbf{N_{MIPS} = Nbr\ (IMP/Sec) * Nbr\ (Us)} \quad (3.5)$$

Où ; **Nbr (Us)** est le nombre des unités.

Fonctionnant à 150 MHz avec huit unités fonctionnelles, cela correspond à nombre total d'instruction par second égal à

N_{MIPS} = 150 000000*8 = 1200 000000 IMPS = 1200 MIMPS

3.6 Paquets d'extraction et de *fetch*

Un paquet d'exécution EP (Execute Packet) est constitué d'un groupe des instructions pouvant être exécuté en parallèle au cours du même temps de cycle.

Le nombre des paquets d'exécution EP dans un paquet d'extraction FP (Fetch Packet) peut varier d'un (avec huit instructions en parallèles) à huit (sans instructions en parallèles). L'architecture VLIW a été modifiée pour permettre à plusieurs EP d'être inclus dans un EP.

Le bit 0 (le bit de poids faible) de chaque instruction de 32 bits est contient un «p» qui indique s'il est en parallèle avec une instruction ultérieure (voir la figure 3.4). Donc, le bit «p» est utilisé pour déterminer si l'instruction suivante appartient au même EP (si 1) ou fait partie de l'EP suivant (si 0).

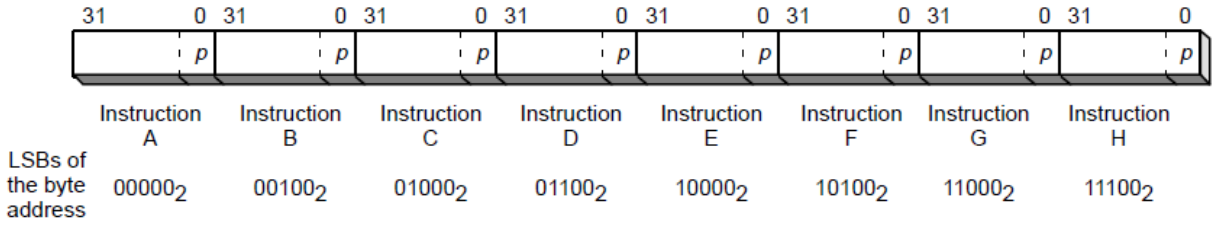


Figure 3.4 Format de base d'un paquet de récupération.

Remarque :

- ✓ Le bit " p" contrôle l'exécution parallèle des instructions.
- ✓ Les p-bits sont balayés de gauche à droite (adresse inférieure à supérieure).
- ✓ Si le p -bit de l'instruction i est 1, alors l'instruction i + 1 doit être exécutée en parallèle avec (dans le même cycle que) l'instruction i. Si le p-bit de l'instruction i est 0, alors l'instruction i + 1 est exécutée dans le cycle suivant l'instruction i.
- ✓ Toutes les instructions s'exécutant en parallèle constituent un paquet d'exécution.
- ✓ Un paquet d'exécution peut contenir jusqu'à huit instructions. Chaque instruction d'un paquet d'exécution doit utiliser une unité fonctionnelle différente.
- ✓ Un paquet d'exécution ne peut pas franchir une limite de 8 mots. Par conséquent, le dernier p-bit d'un paquet d'extraction est toujours défini sur 0, et chaque paquet d'extraction démarre un nouveau paquet d'exécution.

Il existe trois types de modèles de p bits pour les paquets d'extraction. Ces trois modèles de p bits entraînent les séquences d'exécution suivantes pour les huit instructions :

3.6.1 Totalement en série

Les huit instructions sont exécutées séquentiellement (voir la figure 3.5). Les résultats de cette séquence d'exécution sont donnés dans le tableur 3.4.

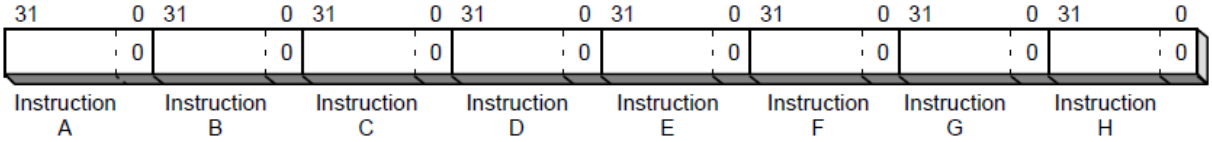


Figure 3. 5 : Les huit instructions sont exécutées séquentiellement

Tableau 3.4 : Paquet d'exécution totalement en série

Cycle/Execute Packet	Instructions
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

3.6.2 Totalement en parallèle

Les huit instructions sont exécutées en parallèle (voir la figure 3.6). Les résultats de cette séquence d'exécution sont donnés dans le tableur 3.5.

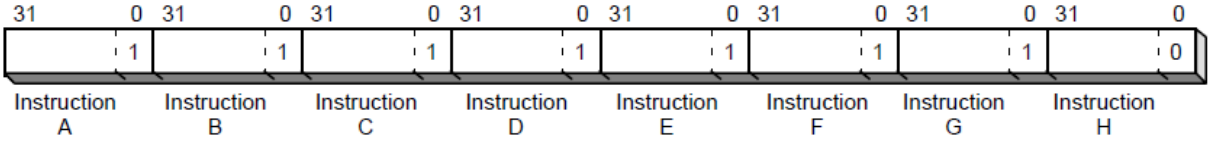


Figure 3.6 Les huit instructions sont exécutées en parallèle

Tableau 3.5 Paquet d'exécution totalement en parallèle

Cycle/Execute Packet	Instructions
1	A B C D E F G H

3.6.3 Partiellement en série

L'exécutées des huit instructions est partiellement en parallèle (voir la figure 3.7). Les résultats de cette séquence d'exécution sont donnés dans le tableur 3.6. Par conséquent les instructions C, D et E n'utilisent aucune des mêmes unités fonctionnelles, chemins croisés ou autres ressources de chemin de données. Ceci est également vrai pour les instructions F, G et H.

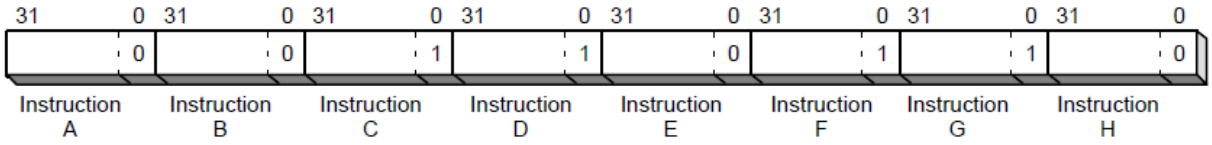


Figure 3.7 Les huit instructions sont exécutées partiellement en parallèle

Tableau 3.6 Paquet d'exécution totalement partiellement en série

Cycle/Execute Packet	Instructions
1	A
2	B
3	C D E
4	F G H

Exemple

Considérons un paquet d'extraction FP avec trois paquets d'exécution EP : EP1, avec deux instructions en parallèles, et EP2 et EP3, chacun avec trois instructions en parallèles, comme suit :

```
    Instruction A
||  Instruction B

    Instruction C
||  Instruction D
||  Instruction E

    Instruction F
||  Instruction G
||  Instruction H
```

Où ; EP1 contient les deux instructions en parallèles A et B; EP2 contient les trois instructions en parallèles C, D et E; et EP3 contient les trois instructions en parallèles F, G et H.

Le paquet d'extraction FP serait comme indiqué à la figure 3.4.

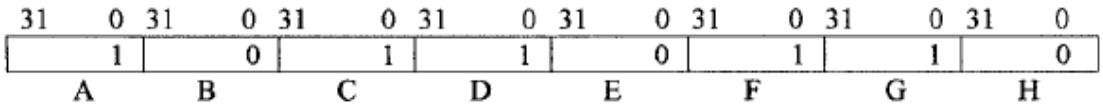


Figure 3.8 : Un paquet d'extraction avec trois paquets d'exécution, montrant le "p" bit de chaque instruction.

Remarque :

Dans cet exemple, le bit «p» de l'instruction B est égal à zéro, ce qui signifie qu'il ne se trouve pas dans le même paquet d'exécution EP que l'instruction C suivante. De même, l'instruction E n'est pas dans le même paquet d'exécution EP que l'instruction F.

3.7 Architecture pipeline

Le traitement en pipeline est une caractéristique clé dans un processeur de signal numérique pour obtenir le bon fonctionnement des instructions en parallèles, ce qui nécessite un minutage minutieux. Le traitement en pipeline comporte trois étapes : l'extraction, le décodage et l'exécution du programme.

1. L'étape d'extraction du programme (*program fetch stage*) est composée de quatre phases :
   a) PG (*program address generate*) : adresse du programme généré (dans le CPU) pour récupérer une adresse.
   b) PS (*program address send*) : adresse du programme envoyé (en mémoire) pour envoyer l'adresse.
   c) PW (*program address ready wait*) : adresse du programme prête attendre (lecture en mémoire) pour attendre les données.

d) PR (*program fetch packet receive*) : programme de récupération du paquet reçu (au niveau du CPU) pour lire le code de l'opération dans la mémoire

2. L'étape de décodage (*decode stage*) est composée de deux phases :
   a) PD (*dispatch packet*) : pour envoyer toutes les instructions d'un FP aux unités fonctionnelles appropriées.
   b) DC (*instruction decode*) : décodage d'instruction.
3. L'étape d'exécution (*execute stage*) est composée de **six phases** (avec virgule fixe) à **dix phases** (avec virgule flottante), en raison des retards (latences) associés aux instructions suivantes :
   a) Instruction de multiplication, composée de deux phases dues à un délai.
   b) Instruction de charge, qui comprend cinq phases en raison de quatre retards.
   c) Instruction de branche, qui comprend six phases en raison de cinq retards.

Le tableau 3.7 montre les phases du pipeline et le tableau 3.8 montre les effets de pipeline.

- La première ligne du tableau 3.8 représente les cycles 1, 2, . . . , 12.
- Chaque ligne suivante représente un paquet d'extraction FP.
- Les colonnes représentées PG, PS, . . . , illustrent les phases associées à chaque paquet d'extraction FP.
- Le PG du premier paquet d'extraction FP commence au cycle 1 et le PG du deuxième paquet d'extraction FP commence au cycle 2, et ainsi de suite.
- Chaque paquet d'extraction FP prend quatre phases pour l'extraction du programme et deux phases pour le décodage.
- Nous supposons que chaque paquet d'extraction FP contient un paquet d'exécution EP.

Tableau 3.7 Phase du Pipeline

Programme d'extraction	Décodage	Exécution
PG PS PW PR	DP DC	E1–E6 avec virgule fixe (E1–E10 avec virgule flottante)

Tableau 3.8 Effets de Pipelining (avec virgule fixe)

Cycle d'horloge											
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

Remarque :

- Au cycle 7, les instructions du premier paquet d'extraction FP sont dans la première phase d'exécution E1 (qui peut être la seule), les instructions du deuxième paquet d'extraction FP sont en phase de décodage, les instructions du troisième paquet d'extraction FP sont en la phase d'expédition, et ainsi de suite. Les sept instructions suivent les différentes phases. Par conséquent, au **cycle 7**, « **le pipeline est plein** ».
- La plupart des instructions ont une phase d'exécution FP. Les instructions telles que multiplier (MPY), charger (Load High/ Load Low LDH/LDW) et brancher (B) prennent respectivement **deux, cinq et six** phases. Des phases d'exécution supplémentaires sont associées à des types d'instructions à virgule flottante et à double précision, pouvant prendre jusqu'à **10** phases. Par exemple, l'opération de multiplication double précision (MPYDP), disponible sur le C67x, comporte neuf intervalles de retard, de sorte que la phase d'exécution prend au total 10 phases.
- La latence de l'unité fonctionnelle (The *functional unit latency*), qui représente le nombre de cycles pendant lesquels une instruction liée à une unité fonctionnelle, est égale à 1 pour toutes les instructions, à l'exception des instructions double précision, disponibles avec C67x à virgule flottante. La latence de l'unité fonctionnelle est différente d'un intervalle de délai. Par exemple, l'instruction MPYDP a quatre latences d'unités fonctionnelles mais neuf intervalles de retard. Cela implique qu'aucune autre instruction ne peut utiliser l'unité fonctionnelle multiplicatrice associée pendant quatre cycles. Un magasin n'a pas de créneau de délai mais termine son exécution dans la troisième phase d'exécution du pipeline.
- Si le résultat d'une instruction de multiplication tel que MPY est utilisé par une instruction ultérieure, un NOP (aucune opération de pipeline n'est exécutée) doit être inséré après l'instruction MPY pour que le traitement en pipeline fonctionne correctement. Quatre ou cinq NOP doivent être insérés au cas où une instruction utilisera le résultat d'une instruction de chargement ou d'une instruction de branche, respectivement. Le tableau suivant résume l'intervalle de retard.

Tableau 3.9 Les intervalles de retard

Type d'instruction	L'intervalle de retard
NOP (aucune opération de pipeline d'exécution)	0
Boutique	0
Un cycle	0
Multiplier	1
Charger (LD) (la modification de l'adresse se produit dans E1)	4
Branche (Le cycle lorsque la cible entre en E1)	5

Le temps total d'exécution des instructions :

Le temps total t_T , d'exécution des instructions est calculé par l'équation suivante :

Séquentiel : $\boldsymbol{t_T = m * t_p, avec;\ t_p = n * t_e}$ (3.5)

Pipeline : $t_T = t_p + (m-1) * t_e = n * t_e + (m-1) * t_e$

$$\boldsymbol{t_T = t_p + (m-1) * t_e = (n+m-1) * t_e} \quad (3.6)$$

Où; t_e, est le temps d'exécution d'une phase.

n est le nombre des phases que compose une instruction

t_p, est le temps d'exécution d'une instruction

m est le nombre des instructions

Exemple : Nous supposons que chaque paquet d'extraction FP contient un paquet d'exécution EP.

- L'exécution séquentielle de **2** instructions sans pipeline est en 24 cycles.

  $t_T = m * n * t_e$ = 2 * 12 * 1 = 24 cycles

- L'exécution séquentielle de **8** instructions avec pipeline est en 19 cycles.

  $t_T = (n+m-1) * t_e = (12+8-1) * 1 = 19$ cycles

1.8 Les registres

Les registres sont également appelés mémoires internes ou mémoires à accès immédiat. Un registre est une petite quantité de mémoire temporaire rapide dans le processeur ou l'ALU ou le CPU peuvent stocker et modifier les valeurs nécessaires pour exécuter des instructions.

Différents processeurs ont différents ensembles de registres. Un registre important est le compteur de programmes (the program conter). Cela permet de garder une trace de l'ordre d'exécution des instructions et montre quelle instruction du programme doit être exécutée ensuite.

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

- ✓ Les registres A0, A1, B0, B1 et B2 sont utilisés comme registres conditionnels.
- ✓ Les registres A4 à A7 et B4 à B7 sont utilisés pour l'adressage circulaire.
- ✓ Les registres A0 à A9 et B0 à B9 (sauf B3) sont des registres temporaires.
- ✓ Tous les registres utilisés d'A10 à A15 et de B10 à B15 sont sauvegardés puis restaurés avant de revenir d'un sous-programme.

La figure 3.9 représente les interconnexions entre les blocks des fichiers de registres (A et B) et les unités fonctionnelles.

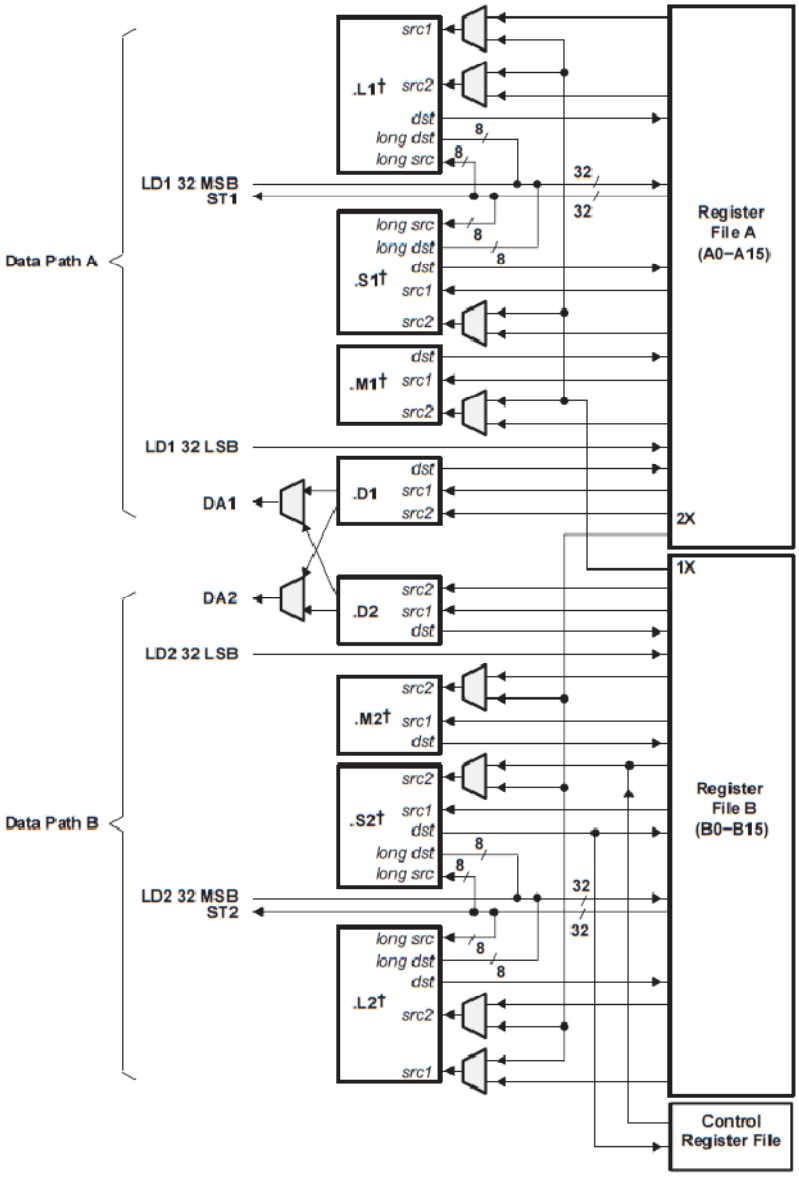


Figure 3.9. TMS320C67x CPU Data Paths

Remarque

- ✓ Une valeur de données de 40 bits peut être contenue dans une paire de registres (voir le tableau 3.10). Les 32 bits de poids faible sont stockés dans le registre pair (par exemple, A2) et les 8 bits restants sont stockés dans les 8 LSB du registre supérieur suivant (impair) (A3).
- ✓ Un schéma similaire est utilisé pour conserver une valeur double précision de 64 bits dans une paire de registres (pair et impair).
- ✓ Les 32 registres sont considérés comme des registres à usage général. Plusieurs registres spéciaux sont également disponibles pour le contrôle et les interruptions : par exemple, le registre de mode d'adresse (AMR) utilisé pour les registres d'adressage circulaire et de contrôle d'interruption.

Tableau 310 Paire de registres

Register File	
A	B
A1 : A0	B1 : B0
A3 : A2	B3 : B2
A5 : A4	B5 : B4
A7 : A6	B7 : B6
A9 : A8	B9 : B8
A11 : A10	B11 : B10
A13 : A12	B13 : B12
A15 : A14	B15 : B14

3.10 Les registres de contrôle

Un certain nombre de registres spéciaux disponibles sur le processeur C6x, comme :

- **Le registre de contrôle du port série SPCR** (serial port control register) est illustré dans la figure 3.10.

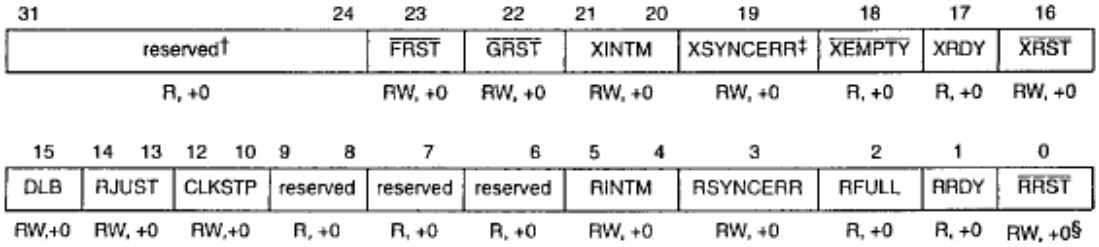


Figure 3.10 Registre de contrôle de port série (SPCR).

- **Les registres de contrôle d'interruption.**

Les registres de contrôle des interruptions sont les suivants.

1. Le registre de contrôle d'état CSR (Control Status Register) (voir la figure 3.11) : contient le bit d'activation d'interruption globale GIE (the global interrupt enable) et d'autres bits de contrôle d'état.
2. Le registre d'activation d'interruption IER (interrupt enable register) (voir la figure 3.12) : active / désactive les interruptions individuelles.
3. Le registre d'indicateur d'interruption IFR (interrupt flag register) (voir la figure 3.13) : affiche l'état des interruptions.
4. Le registre de l'ensemble des interruptions ISR (interrupt set register) (voir la figure 3.14) : définit les interruptions en attente.
5. Le registre d'effacement d'interruption ICR (interrupt clear register) (voir la figure 3.15) : efface les interruptions en attente

6. Le registre de pointeur de tableau de service d'interruption ISTP (interrupt service table pointer) : localise la routine de service d'interruption (un ISR)) (voir la figure 3.16).
7. Le registre de pointeur de retour d'interruption IRP (interrupt return pointer)
8. Le registre de pointeur de retour d'interruption non masquable NIRP (nonmaskable interrupt return pointer)

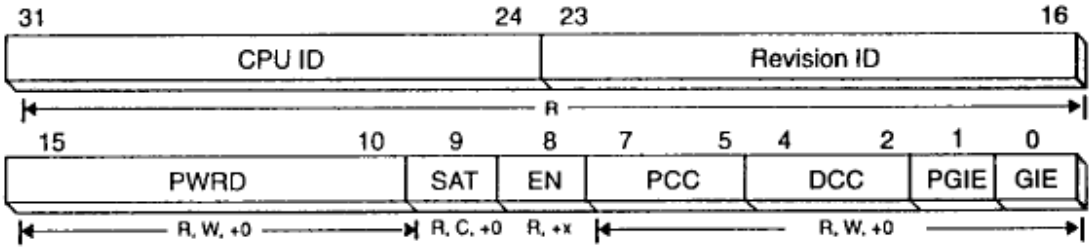


Figure 3.11 Registre d'état de contrôle CSR

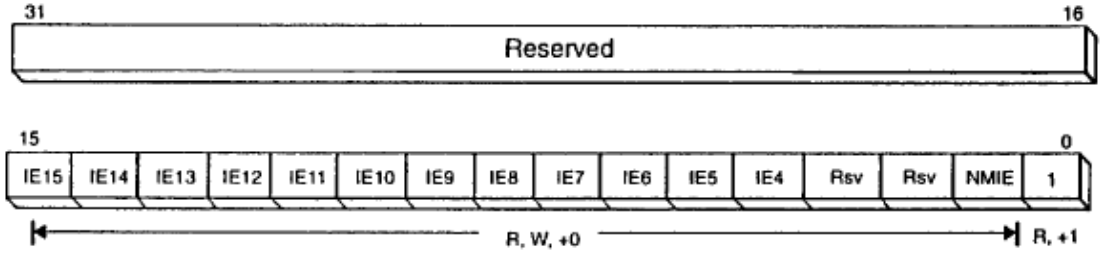


Figure 3.12 Registre d'activation d'interruption IER.

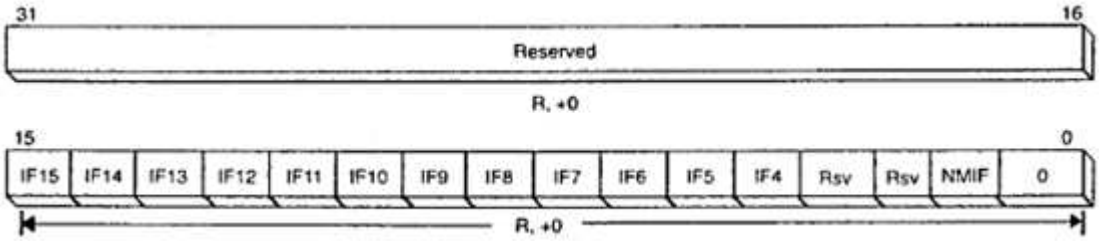


Figure 3.13 Registre d'indicateur d'interruption IFR

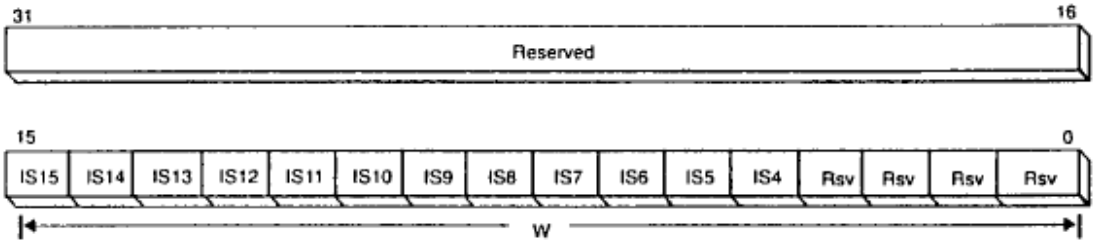


Figure 3.14 Registre de l'ensemble des interruptions ISR.

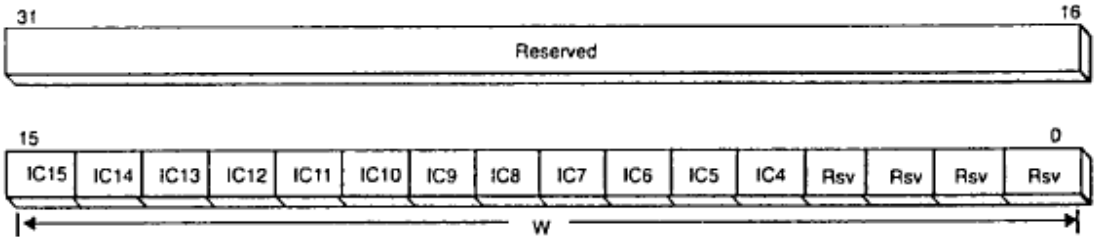


Figure 3.15 Registre d'effacement d'interruption ICR

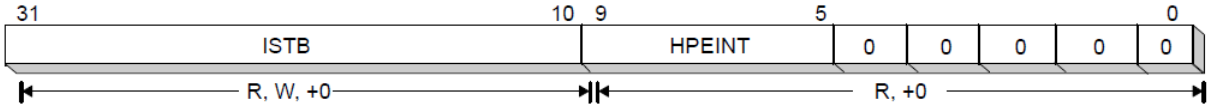


Figure 3.16 Registre de pointeur de tableau de service d'interruption ISTP.

Remarque :

- Pour traiter une interruption masquable, les règles suivantes s'appliquent :
  - ✓ Le bit GIE dans le registre d'état de contrôle CSR est mis à 1
  - ✓ Le bit NMIE dans le registre d'activation d'interruption IER est mis à 1.
  - ✓ Le bit IE approprié dans le registre d'activation d'interruption IER est défini sur 1.
  - ✓ Le bit IF correspondant dans le registre d'indicateur d'interruption IFR est mis à 1.
- Pour qu'une interruption se produise, le CPU ne doit pas exécuter un intervalle de retard associé à une instruction de branchement.
- Le signal de réinitialisation est un signal actif-bas utilisé pour arrêter le CPU et le signal NMI alerte le CPU en cas de problème matériel potentiel. Douze interruptions du CPU avec des priorités inférieures sont disponibles, correspondant aux signaux masquables INT4 à INT15. Les priorités de ces interruptions sont les suivantes : INT4, INT5, ..., INT15, avec INT4 ayant la priorité la plus haute et INT15 la priorité la plus basse.

Le tableau de service d'interruption IST présenté dans le tableau 3.11 est utilisé au début d'une interruption.

Tableau 3.5 : Le tableau de service d'interruption

Interrupt	Offset
RESET	000h
NMI	020h
Reserved	040h
Reserved	060h
INT4	080h
INT5	0A0h
INT6	0C0h
INT7	0E0h
INT8	100h
INT9	120h
INT10	140h
INT11	160h
INT12	180h
INT13	1A0h
INT14	1C0h
INT15	1E0h

- Chaque emplacement est associé à un paquet de recherche FP associé à chaque interruption. Donc le tableau contient 16 FP, chacun avec huit instructions de 32 bits (32*8 = 256 bits) ou de 32 octets.
- Les adresses du côté droit correspondent à un décalage associé à chaque interruption spécifique. Le décalage de chaque adresse du tableau est incrémenté de 20 h = 32.
- Le paquet de recherche FP de la réinitialisation doit être à l'adresse 0. Toutefois, les FP associés aux autres interruptions peuvent être déplacés. L'adresse déplaçable peut être spécifiée en écrivant cette adresse dans le registre de la base du tableau de services des interruptions ISTB (interrupt service table base) du pointeur du tableau de services des interruptions ISTP, illustrée à la figure 3.16. À la réinitialisation, ISTB est zéro. Pour déplacer le tableau de vecteurs, le protocole ISTP est utilisé ; l'adresse relocalisable est ISTB plus l'offset.

3.10 Les périphériques (Timers, Interruptions, HPI, PLL)

La figure 3.17 représente le schéma block du CPU et les signaux des périphériques. Parmi les périphériques internes des processeurs TMS320C6x :

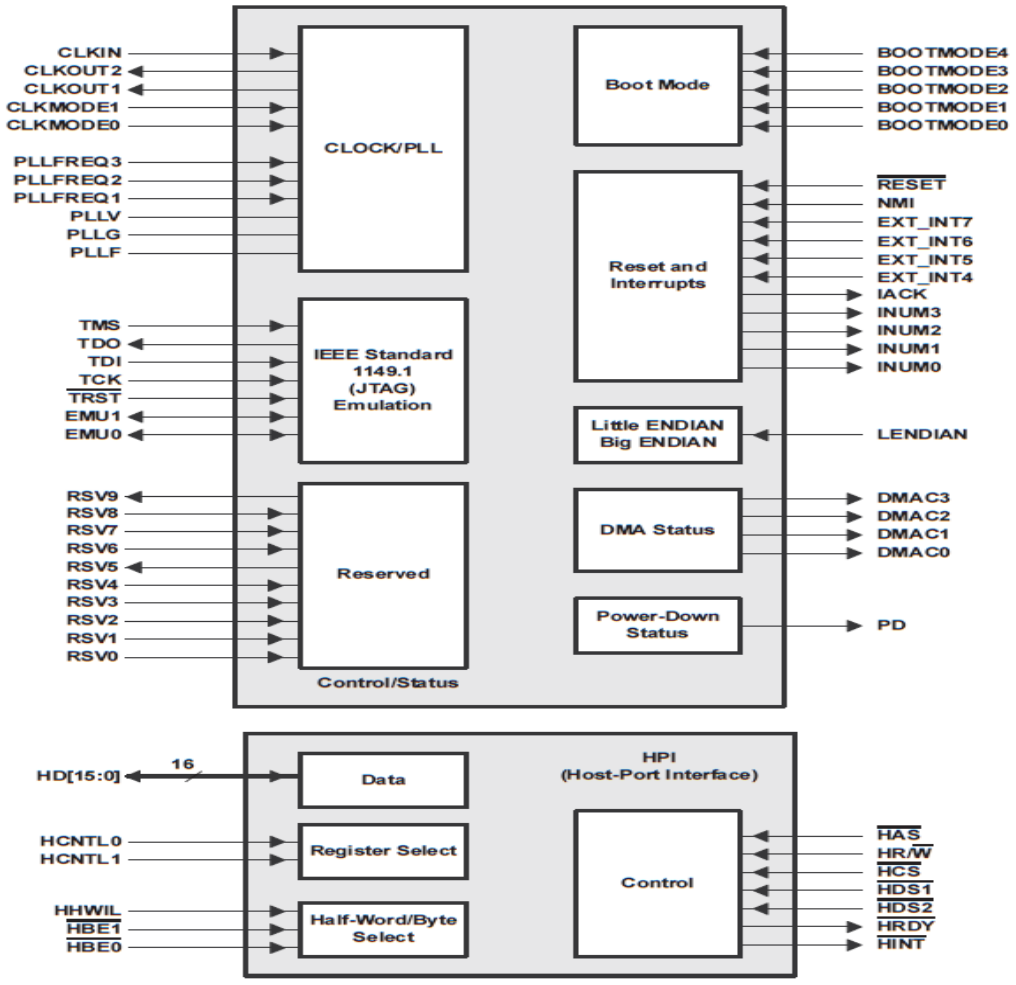


Figure 3.17: CPU et les signaux des périphériques.

➢ La minuterie (Timer)

Deux minuteries de 32 bits peuvent être utilisées pour chronométrer et compter des événements ou pour arrêter le CPU. Une minuterie peut ordonner à un CAN externe de démarrer la conversion ou au contrôleur DMA de démarrer un transfert de données.

La minuterie comprend :

- ✓ Un registre de la fréquence de la minuterie.
- ✓ Un registre de compteurs de temps contenant la valeur du compteur incrément.
- ✓ Un registre de contrôle de la minuterie, qui surveille l'état de la minuterie.

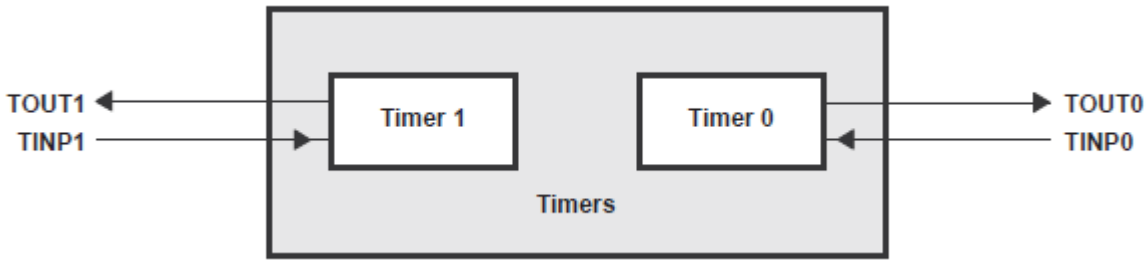


Figure 3.18: Timers.

➢ Les interruptions

Une interruption peut être émise en interne ou en externe. Une interruption arrête le processus en cours afin qu'il puisse exécuter une tâche requise lancée par l'interruption. Le flux de programme est redirigé vers une routine de service d'interruption (ISR). La source de l'interruption peut être un CAN, une minuterie, etc. Lors d'une interruption, les conditions du processus en cours doivent être enregistrées pour pouvoir être restaurées après l'exécution de la tâche d'interruption. Lors de l'interruption, les registres sont sauvegardés et le traitement continue vers un ISR. Ensuite, les registres sont restaurés.

Il y a 16 sources d'interruption. Ils incluent deux interruptions de minuterie, quatre interruptions externes, quatre interruptions McBSP et quatre interruptions DMA. Douze interruptions de processeur sont disponibles. Un sélecteur d'interruption est utilisé pour choisir parmi les 12 interruptions.

Le tableau 3.12 indique les valeurs de sélecteur d'interruptions nécessaires pour choisir un type d'interruption spécifique.

Tableau 3.12 : Sélection d'interruptions à l'aide du sélecteur d'interruption

Interrupt Selector	Type	Description
00000	DSPINT	Host port to DSP interrupt
00001	TINT0	Timer 0 interrupt
00010	TINT1	Timer 1 interrupt
00011	SD_INT	EMIF SDRAM timer interrupt
00100	EXT_INT4	External interrupt pin 4
00101	EXT_INT5	External interrupt pin 5
00110	EXT_INT6	External interrupt pin 6
00111	EXT_INT7	External interrupt pin 7
01000	DMA_INT0	DMA channel 0 interrupt
01001	DMA_INT1	DMA channel 1 interrupt
01010	DMA_INT2	DMA channel 2 interrupt
01011	DMA_INT3	DMA channel 3 interrupt
01100	XINT0	McBSP0 transmit interrupt
01101	RINT0	McBSP0 receive interrupt
01110	XINT1	McBSP1 transmit interrupt
01111	RINT1	McBSP1 receive interrupt

Remarque :

Les interruptions définies par le logiciel INT4 – INT15 sont associées à un signal d'interruption physique utilisant les registres multiplex d'interruption IML et IMH. Les valeurs de sélection d'interruption souhaitées dans le tableau 3.5 sont stockées dans les registres IML ou IMH appropriés pour INT4 – INT15.

- **L'interface du port hôte HPI (Host Port Interface)**.

La figure 3.19 représente le schéma block de l'interface du port hôte d'un processeur DSP ‘‘TMS320C6x’’. L'interface du port hôte de 16 bits permettant à l'hôte d'accéder à la mémoire et aux périphériques C6x.

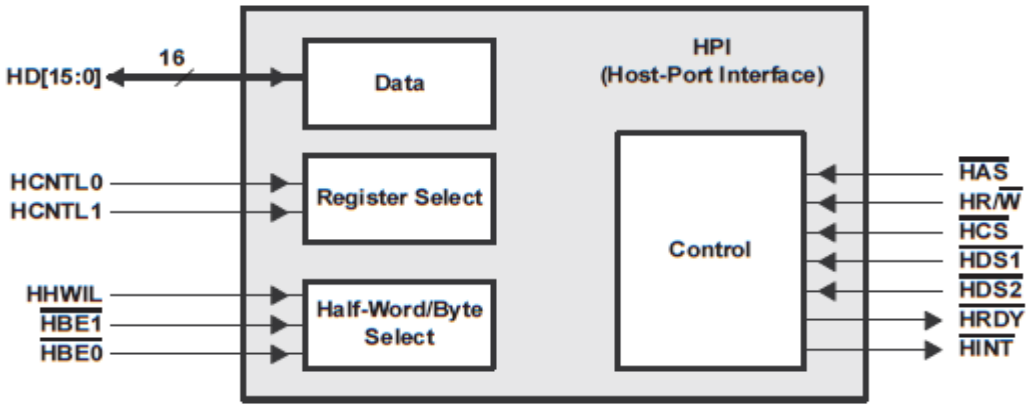
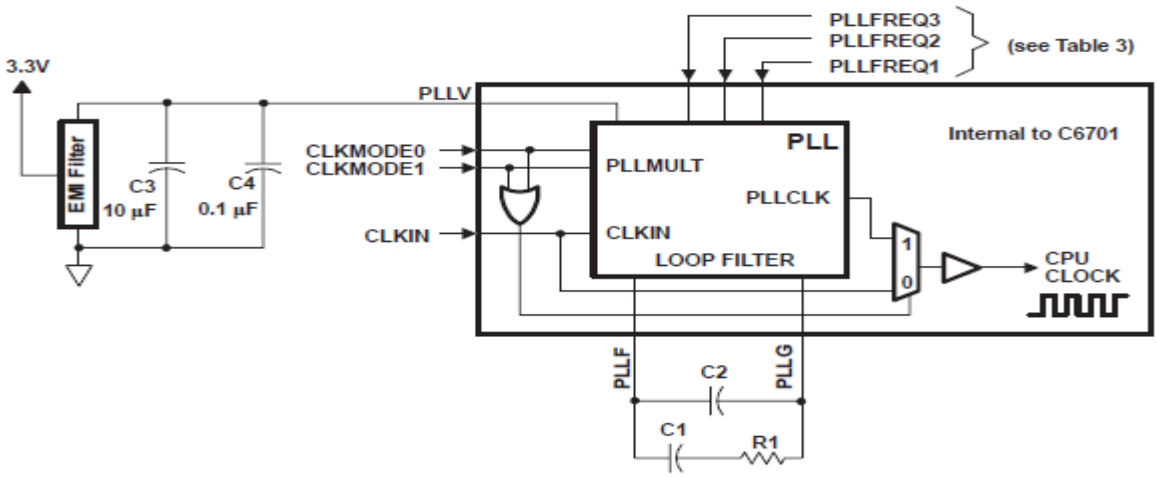


Figure 3.19: Schéma block de l'interface du port hôte d'un processeur DSP ‘‘TMS320C6x’’.

- **Boucle à verrouillage de phase PLL (Phase-Locked Loop)**

Figure 3.20 montre le circuit de PLL externe pour le mode PLL x4 ou le mode x1 (Bypass).



AVAILABLE MULTIPLY FACTORS			
CLKMODE1	CLKMODE0	PLL MULTIPLY FACTORS	CPU CLOCK FREQUENCY F(CPUCLOCK)
0	0	x1(BYPASS)	1 x f(CLKIN)
0	1	Reserved	Reserved
1	0	Reserved	Reserved
1	1	x4	4 x f(CLKIN)

Figure 3.20. Circuit de PLL externe pour le mode PLL x4 ou le mode x1 (Bypass)

Pour des performances optimales, TI recommande que :

- ✓ Tous les composants externes PLL soient sur un seul côté de la carte, sans cavaliers, commutateurs ou composants autres que ceux illustrés. Maintenez la longueur du câble et le nombre de via entre la broche PLLF, la broche PLLG et R1, C1 et C2 au minimum. De plus, placez tous les composants externes PLL aussi près que possible du dispositif C6000 DSP.

- ✓ Pour réduire la gigue PLL, maximisez l'espacement entre les signaux de commutation et les composants externes PLL (R1, C1, C2, C3, C4 et le filtre EMI).
- ✓ L'alimentation 3,3 V du filtre EMI doit provenir du même plan d'alimentation 3,3 V fournissant la tension d'E/S, DVDD.
- ✓ Fabricant du filtre EMI : Numéro de pièce TDK ACF451832-333, 223, 153, 103. Numéro de pièce Panasonic EXCCET103U.

3.11 La liaison série (*multichannel buffered serial port*)

Deux ports série multicanaux en mémoire tampon McBSP (multichannels buffered serial ports) sont disponibles (voir la figure 3.21). Ils fournissent une interface aux périphériques externes peu coûteux (standards de l'industrie). Les McBSP possèdent des fonctionnalités telles que la communication en duplex intégral (full-duplex communication), synchronisation et cadrage indépendants pour la réception et la transmission, et une interface directe avec les périphériques compatibles AC97 et IIS. Il permet plusieurs tailles de données entre 8 et 32 bits.

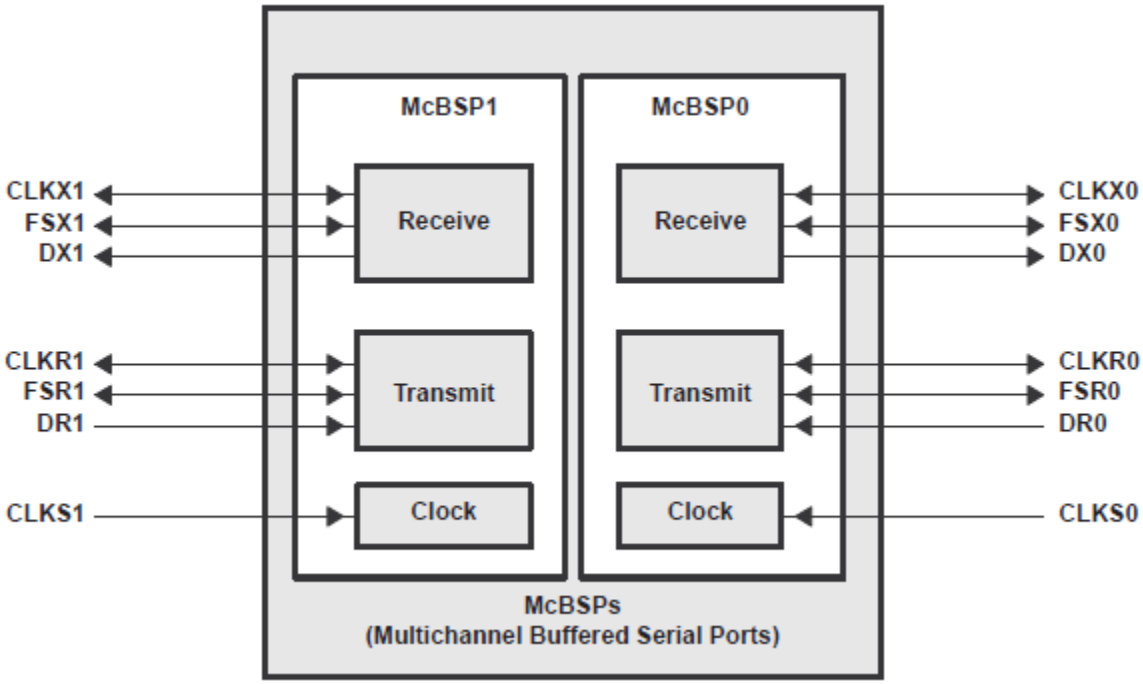


Figure 3.21 : McBSP (Multi-Channel Buffered Serial Port).

Une communication de données externe peut se produire pendant que les données sont déplacées en interne. La figure 3.22 montre un schéma de principe interne d'un McBSP. Les broches de transmission de données DX (data transmit) et les broches de réception de données DR (data receive) sont utilisées pour la communication de données. Les informations de contrôle (clocking and frame synchronization) passent par CLKX, CLKR, FSX et FSR. Le CPU ou le contrôleur DMA lit les données du registre de réception de données DRR (data receive

register) et écrit les données à transmettre au registre de transmission de données DXR (data transmit register). Le registre à décalage de transmission XSR (transmit shift register) transmet ces données vers DX. Le registre à décalage de réception RSR (receive shift register) copie les données reçues sur DR dans le registre de tampon de réception RBR (receive buffer register). Les données dans RBR sont ensuite copiées dans DRR pour être lues par le CPU ou le contrôleur DMA.

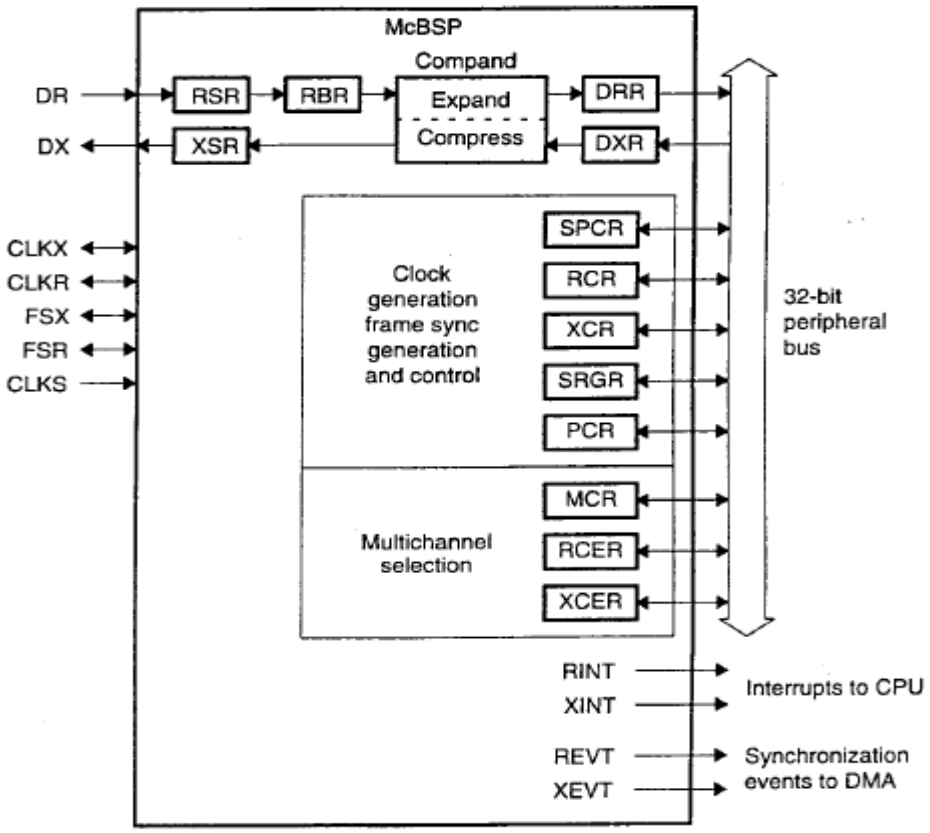


Figure 3.22 : Schéma de principe interne d'un McBSP (Courtesy of Texas Instruments).

Autres registres prisent en charge autre communication de données, comme :

- ✓ Registre de contrôle de port série **SPCR** (serial port control register).
- ✓ Registre de contrôle de réception/transmission **RCR/XCR** (receive/transmit control register).
- ✓ Registre d'activation de canal de réception/transmission **RCER/XCER** (receive/transmit channel enable register).
- ✓ Registre de contrôle de broche **PCR** (pin control register).
- ✓ Registre de générateur de fréquence d'échantillonnage **SRGR** (sample rate generator register).

3.12 Présentation du jeu d'instructions

Le langage d'assemblage est un langage de bas niveau qui reflète étroitement le fonctionnement des opcodes et des opérandes. Il est beaucoup plus rapide pour les ordinateurs de convertir des instructions d'assemblage en code machine que de convertir un code source de haut niveau en code machine.

3.12.1 Format du code d'assemblage

Un format de code d'assemblage est représenté par le champ

Label || [ ] Instruction Unit Operands ; comments

Example:

|| [A1] ADD .L1 A0,A1,A5 ;add A0 + A1 → A5 (accum in A5) si A1 # 0

Remarque :

- ✓ Une étiquette (Label), représente une adresse spécifique ou un emplacement de mémoire contenant une instruction ou des données. Elle doit être dans la première colonne.
- ✓ Les barres parallèles (||) sont présentes si l'instruction est exécutée parallèlement à l'instruction précédente.
- ✓ Le champ suivant est facultatif pour rendre l'instruction associée conditionnelle. Cinq des registres (A1, A2, B0, B1 et B2) peuvent être utilisés comme registres conditionnels. Par exemple, [A1] spécifie que l'instruction associée s'exécute si A1 n'est pas zéro. Par contre, avec [! A1], l'instruction associée s'exécute si A1 est égal à zéro.
- ✓ Toutes les instructions C6x peuvent être conditionnées avec les registres A1, A2, B0, B1 et B2 en déterminant à quel moment le registre conditionnel est à zéro.
- ✓ Le champ d'instruction peut être une directive assembleur ou une mnémonique. Une directive assembleur est une commande pour l'assembleur. Une mnémonique est une instruction réelle qui s'exécute au moment de l'exécution.
- ✓ L'instruction (directive mnémonique ou assembleur) ne peut pas commencer dans la colonne 1.
- ✓ Le champ Unité, qui peut être l'une des huit unités fonctionnelles, est facultatif.
- ✓ Les commentaires commençant dans la colonne 1 peuvent commencer par un astérisque (*) ou un point-virgule, tandis que les commentaires commençant par une autre colonne doivent commencer par un point-virgule.
- ✓ Le code des processeurs à virgule flottante C3x / C4x n'est pas compatible avec le code des processeurs à virgule fixe C1x, C2x et C5x / C54x. Cependant, le code pour le point

fixe C62x est compatible avec le code pour le C67x à virgule flottante. Le code C62x est en réalité un sous-ensemble du code C67x.

- ✓ Des instructions supplémentaires permettant de gérer les opérations en double précision et en virgule flottante ne sont disponibles que sur le processeur C67x (certaines instructions supplémentaires sont également disponibles sur le processeur C64x à point fixe).
- ✓ Plusieurs segments de code sont présentés pour illustrer le jeu d'instructions C6x. Le code d'assemblage des processeurs C6x est très similaire au code C3x / C4x. Les types des instructions à tâche unique disponibles pour le C62x / C67x facilitent la programmation par rapport à la génération précédente de processeurs à virgule fixe ou à virgule flottante. Cela contribue à l'efficacité du compilateur.
- ✓ Des instructions supplémentaires sont disponibles sur le C64x (mais pas sur le C62x) ressemblent aux types d'instructions multitâches des processeurs C3x / C4x.

3.12.2 Types des instructions

Les typiques instructions du langage d'assemblage incluent :

a) L'instruction Ajouter

```
ADD   .L1      A3,A7,A7     ;add A3 + A7 → A7 (accum in A7)
```

Additionne les valeurs dans les registres A3 et A7 et place le résultat dans le registre A7. L'unité .L1 est optionnelle. Si la destination ou le résultat est en B7, l'unité serait .L2.

b) L'instruction Soustraire

```
SUB    .S1     A1,1,A1     ; soustrait 1 de A1
```

Soustrait 1 de A1 pour le décrémenter, en utilisant l'unit .S.

c) Les instructions parallèles (Multiplier)

```
MPY     .M2    A7,B7,B6    ; multiplier 16  LSBs of A7,B7 → B6
||  MPYH   .M1    A7,B7,A6    ; multiplier 16MSBs of A7,B7→A6
```

Multiplie les 16 bits les plus faibles ou les moins significatifs (LSB) de A7 et B7 et place le produit dans B6, en parallèle (simultanément dans le même paquet d'exécution) avec une seconde instruction multipliant les 16 bits les plus significatifs ou les plus significatifs (MSB) de A7 et B7 et place le résultat dans A6. De cette manière, deux opérations de multiplication / accumulation peuvent être exécutées dans un seul cycle d'instruction. Ceci peut être utilisé pour décomposer une somme de produits en deux ensembles de somme de produits : un ensemble utilisant les 16 bits inférieurs pour fonctionner sur le premier, le troisième, le cinquième, ... nombre, et un autre ensemble utilisant les 16 bits les plus élevés pour fonctionner sur les

deuxième, quatrième, sixième, ... nombre. Notez que le symbole parallèle ne figure pas dans la colonne 1.

d) L 'instruction Charger

```
LDH   .D2    *B2++,B7    ;load (B2) →B7, increment B2
||  LDH   .D1    *A2++,A7    ;load (A2) →A7, increment A2
```

Charger dans B7 le demi-mot (16 bits) dont l'adresse en mémoire est spécifiée / indiquée par B2. Le registre B2 est alors incrémenté (post incrémenté) pour pointer vers l'adresse de mémoire immédiatement supérieure. En parallèle, une autre instruction de mode d'adressage indirect permet de charger dans A7 le contenu en mémoire, dont l'adresse est spécifiée par A2. Ensuite, A2 est incrémenté pour pointer vers l'adresse mémoire la plus haute suivante.

Remarque

- ✓ L'instruction LDW charge un mot (word) de 32 bits. Deux chemins utilisant les unités fonctionnelles .D1 et .D2 permettent le chargement de données de la mémoire vers les registres A et B à l'aide de l'instruction LDW.
- ✓ L'instruction à virgule flottante LDDW à double mot sur le C6711 permet de charger simultanément deux registres de 32 bits dans la face A et deux registres de 32 bits dans la face B.

e) L'instruction Stocker

```
STW    .D2    A1,*+A4[20]    ;store A1→(A4) offset by 20
```

Stocker le mot A1 de 32 bits dans la mémoire dont l'adresse est spécifiée par A4 décalée de 20 mots (de 32 bits). Le registre d'adresse A4 est pré-incrémenté avec offset, mais il n'est pas modifié.

f) Brancher / Déplacer.

Le segment de code suivant illustrer la création de branches et le transfert de données.

```
Loop  MVK     .S1     x,A4          ;move 16 LSBs of x address →A4
      MVKH    .S1     x,A4          ;move 16 MSBs of x address →A4
      .
      .
      .
      SUB     .S1     A1,1,A1       ;decrement A1
[A1]  B       .S2     Loop          ;branch to Loop if A1 # 0
      NOP             5             ;five no-operation instructions
      STW     .D1     A3,*A7        ;store A3 into (A7)
```

La première instruction déplace les 16 bits inférieurs (LSB) de l'adresse x dans le registre A4. La deuxième instruction déplace les 16 bits les plus élevés (MSB) de l'adresse x vers A4, qui contient maintenant l'adresse complète à 32 bits de x. Donc ; il faut utiliser les instructions MVK / MVKH pour obtenir une constante de 32 bits dans un registre.

Le registre A1 est utilisé comme compteur de boucle. Après qu'il soit décrémenté avec l'instruction SUB, il est testé pour une branche conditionnelle. L'exécution de l'instruction branche sur l'étiquette ou sur la boucle d'adresse si A1 n'est pas nul. Si le registre A1 = 0, l'exécution continue et les données du registre A3 sont enregistrées dans la mémoire dont l'adresse est spécifiée par A7.

3.12.3 Les opérations conditionnelles

Toutes les instructions peuvent être conditionnelles. La condition est contrôlée par un champ d'opcode de 3 bits (creg) qui spécifie le registre de condition testé, et un champ de 1 bit (z) qui spécifie un test pour zéro ou non zéro.

Remarque :

- ✓ Les quatre MSB de chaque opcode sont creg et z.
- ✓ Le registre est testé au début de l'étape de pipeline E1 pour toutes les instructions.
- ✓ Si z = 1, le test est d'égalité avec zéro. Si z = 0, le test est pour non nul.
- ✓ Le cas de creg = 0 et z = 0 est traité comme toujours vrai pour permettre l'exécution inconditionnelle des instructions.
- ✓ Le champ creg est codé dans l'opcode de l'instruction comme indiqué dans le Tableau suivant.

Tableau 3.13. Registres pouvant être testés par des opérations conditionnelles

Registre conditionnelle	creg			Z
	Bit 31	Bit 30	Bit 29	Bit 28
Conditionnelle	0	0	0	0
Réserver	0	0	0	1
B0	0	0	1	Z
B1	0	1	0	Z
B2	0	1	1	Z
A1	1	0	0	Z
A2	1	0	1	Z
Réserver	1	1	X	X

Remarque :

x peut être n'importe quelle valeur dans des cas réservés.

- Les instructions conditionnelles sont représentées à l'aide de crochets, **[ ]**, entourant le registre de condition. Le paquet d'exécution suivant contient deux instructions ADD en parallèle. Le premier ADD est conditionnel à ce que B0 soit différent de zéro. Le deuxième ADD est conditionnel à ce que B0 soit zéro. Le personnage ! indique-le "non" de la condition.

```
        [B0]   ADD   .L1    A1,A2,A3
   ||   [!B0]  ADD   .L2    B1,B2,B3
```

- Les instructions ci-dessus sont mutuellement exclusives. Cela signifie qu'un seul sera exécuté. Si elles sont programmées en parallèle, les instructions mutuellement exclusives sont contraintes. Si des instructions mutuellement exclusives partagent des ressources elles ne peuvent pas être ordonnancées en parallèle (mises dans le même paquet d'exécution), même si une seule s'exécutera.

3.12.4 Contraintes des ressources

Deux instructions dans le même paquet d'exécution ne peuvent pas utiliser les mêmes ressources. De plus, deux instructions ne peuvent pas écrire dans le même registre au cours du même cycle.

a) Contraintes sur les instructions utilisant la même unité fonctionnelle

Deux instructions utilisant la même unité fonctionnelle ne peuvent pas être émises dans le même paquet d'exécution.

Le paquet d'exécution suivant n'est pas valide :

```
        ADD .S1     A0, A1, A2 ; \ L'unité .S1 est utilisée pour
   ||   SHR .S1     A3, 15, A4 ; / les deux instructions
```

Le paquet d'exécution suivant est valide :

```
        ADD .L1     A0, A1, A2 ; \ Deux unités fonctionnelles
   ||   SHR .S1     A3, 15, A4 ; / différentes sont utilisées
```

b) Contraintes sur les chemins croisés (1X et 2X)

Une unité (soit une unité .S, .L ou .M) par chemin de données, par paquet d'exécution, peut lire un opérande source à partir de son fichier de registre opposé via les chemins croisés (1X et 2X). Par exemple, .S1 peut lire les deux opérandes d'une instruction à partir du fichier de registre A, et elle peut lire un opérande à partir du fichier de registre B en utilisant le chemin croisé 1X et l'autre à partir du fichier de registre A. Ceci est indiqué par un X après le nom de l'unité dans la syntaxe de l'instruction.

Deux instructions utilisant le même chemin croisé entre les fichiers de registre ne peuvent pas être émises dans le même paquet d'exécution, car il n'y a qu'un seul chemin de A à B et un chemin de B à A.

Le paquet d'exécution suivant n'est pas valide :

```
        ADD  .L1X   A0,B1,A1 ; \ Le chemin croisé 1X est utilisé
||      MPY  .M1X   A4,B4,A5 ; / pour les deux instructions
```

Le paquet d'exécution suivant est valide :

```
        ADD  .L1X   A0,B1,A1 ; \ Les instructions utilisent les
||      MPY  .M2X   B4,A4,B2 ; / chemins croisés 1X et 2X
```

L'opérande proviendra d'un fichier de registre opposé à la destination si le bit x dans le champ d'instruction est défini (indiqué dans la carte d'opcode située à la section 3.3).

c) Contraintes sur les chargements et les stockages (Loads and Stores)

Les chargements et les stockages peuvent utiliser un pointeur d'adresse d'un fichier de registre lors du chargement ou du stockage à partir de l'autre fichier de registre. Deux chargements et/ou stockages utilisant un pointeur d'adresse à partir du même fichier de registre ne peuvent pas être émis dans le même paquet d'exécution.

Le paquet d'exécution suivant n'est pas valide :

```
        LDW.D1      *A0,A1 ; \ Registres d'adresses à partir
||      LDW.D1      *A2,B2 ; / des mêmes fichiers de registres.
```

Le paquet d'exécution suivant est valide :

```
        LDW.D1      *A0,A1 ; \ Registres d'adresses de différents
||      LDW.D2      *B0,B2 ; / fichiers de registre
```

Deux chargements et/ou stockages chargés et/ou stockés à partir du même fichier de registre ne peuvent pas être émis dans le même paquet d'exécution.

Le paquet d'exécution suivant n'est pas valide :

```
        LDW .D1     *A4,A5 ; \ Chargement et stockage à partir
||      STW .D2     A6,*B4 ; / du même fichier de registre
```

Le paquet d'exécution suivant est valide :

```
        LDW .D1     *A4,B5 ; \ Chargement et stockage à partir
||      STW .D2     A6,*B4 ; / de différents fichiers de registre
```

d) Contraintes sur les données longues (40 bits)

Étant donné que les unités .S et .L partagent un port de registre de lecture pour les opérandes source longs et un port de registre d'écriture pour les résultats longs, un seul résultat long peut être émis par fichier de registre dans un paquet d'exécution. 2

Le paquet d'exécution suivant n'est pas valide :

```
        ADD .L1     A5:A4,A1,A3:A2 ; \ Deux longues écritures sur
||      SHL .S1     A8,A9,A7:A6 ; / un fichier de registre
```

Le paquet d'exécution suivant est valide :

```
        ADD .L1     A5:A4,A1,A3:A2 ; \ Une longue écriture pour
||      SHL .S2     B8,B9,B7:B6 ; / chaque fichier de registre
```

Étant donné que les unités .L et .S partagent leur port de lecture long avec le port de stockage, les opérations qui lisent une valeur longue ne peuvent pas être émises sur les unités .L et/ou .S dans le même paquet d'exécution qu'un stockage.

Le paquet d'exécution suivant n'est pas valide :

```
        ADD .L1     A5:A4,A1,A3:A2 ; \ Opération de lecture
||      STW .D1     A8,*A9 ; / longue et un stockage
```

Le paquet d'exécution suivant est valide :

```
        ADD.L1      A4, A1, A3:A2 ; \ Pas longtemps lu avec
||      STW.D1      A8,*A9 ; / le stockage
```

e) Contraintes sur les lectures de registre

- Plus de quatre lectures du même registre ne peuvent pas se produire sur le même cycle. Les registres conditionnels ne sont pas inclus dans ce décompte.

La séquence de code suivante n'est pas valide :

```
            MPY .M1     A1,A1,A4 ; cinq lectures du registre A1
||          ADD .L1     A1,A1,A5
||          SUB .D1     A1,A2,A3
```

Cette séquence de code est valide :

```
            MPY .M1     A1,A1,A4 ; seulement quatre lectures de A1
||      [A1] ADD .L1    A0,A1,A5
||          SUB .D1     A1,A2,A3
```

- Des écritures multiples dans le même registre sur le même cycle peuvent se produire si des instructions avec des latences différentes écrivant dans le même registre sont émises sur des cycles différents. Par exemple, un MPY émis au cycle i suivi d'un ADD au cycle i + 1 ne peut pas écrire dans le même registre car les deux instructions écrivent un résultat au cycle i + 1. Par conséquent, la séquence de code suivante n'est pas valide :

```
        MPY .M1     A0,A1,A2
        ADD .L1     A4,A5,A2
```

Chapitre 4 : Gestion de la mémoire

4.1 Présentation et intérêt de l'architecture Harvard

L'architecture d'un microprocesseur, et donc d'un DSP, est un élément important qui conditionne directement les performances d'un processeur. Il existe deux types fondamentaux de structures, dites « Von Neuman » et «Harvard ». La structure « Harvard » se distingue de l'architecture « Von Neuman » uniquement par le fait que les mémoires programmes et données sont séparées (voire les figures 4.1-4.2). L'accès à chacune des deux mémoires se fait via un chemin distinct. Cette organisation permet de transférer une instruction et des données simultanément, ce qui améliore les performances.

L'architecture généralement utilisée par les microprocesseurs est la structure Von Neuman (exemples : la famille Motorola 68XXX, la famille Intel 80X86). L'architecture Harvard est plutôt utilisée dans des microprocesseurs spécialisés pour des applications temps réels, comme les DSP. Il existe cependant quelques rares DSP à structure Von Neuman. La raison de ceci est liée au coût supérieur de la structure de type Harvard. En effet, elle requiert deux fois plus de bus de données, d'adresses, et donc de broches sur la puce. Or un des éléments augmentant le coût de productions des puces est précisément le nombre de broches à implanter.

Pour réduire le coût de la structure Harvard, certains DSP utilisent l'architecture dite « Structure de Harvard modifiée ». À l'extérieur, le DSP ne propose qu'un bus de données et un bus d'adresse, comme la structure Von Neuman. Toutefois, à l'intérieur, la puce DSP dispose de deux bus distincts de données et de deux bus distincts d'adresses. Le transfert des données entre les bus externes et internes est effectué par multiplexage temporel.

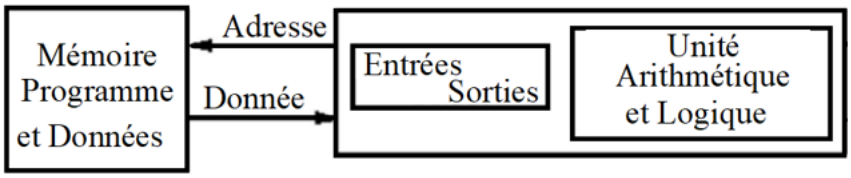


Figure 4.1 : Architecture Von Neuman

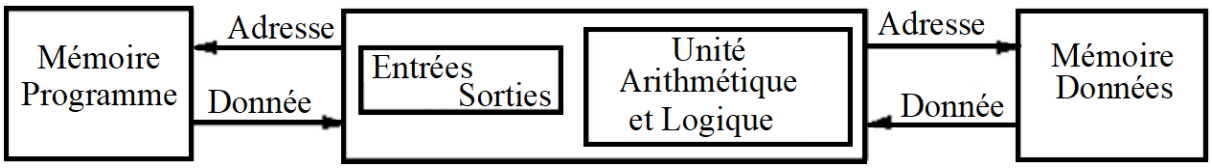


Figure 4.2 : Architecture Harvard

4.2 Mémoires internes (niveaux L1 et L2).

Le schéma fonctionnel de la mémoire interne d'un microprocesseur DSP est représenté dans la figure 4.3. La mémoire interne comprend une architecture de cache à deux niveaux avec 4 KB de cache de programme de niveau 1 (**L1P**) (level 1 program cache), 4 KB de cache de données de niveau 1 (**L1D**) (level 1 data cache) et 64 KB de RAM ou cache de niveau 2 pour l'allocation de données / programme (**L2**).

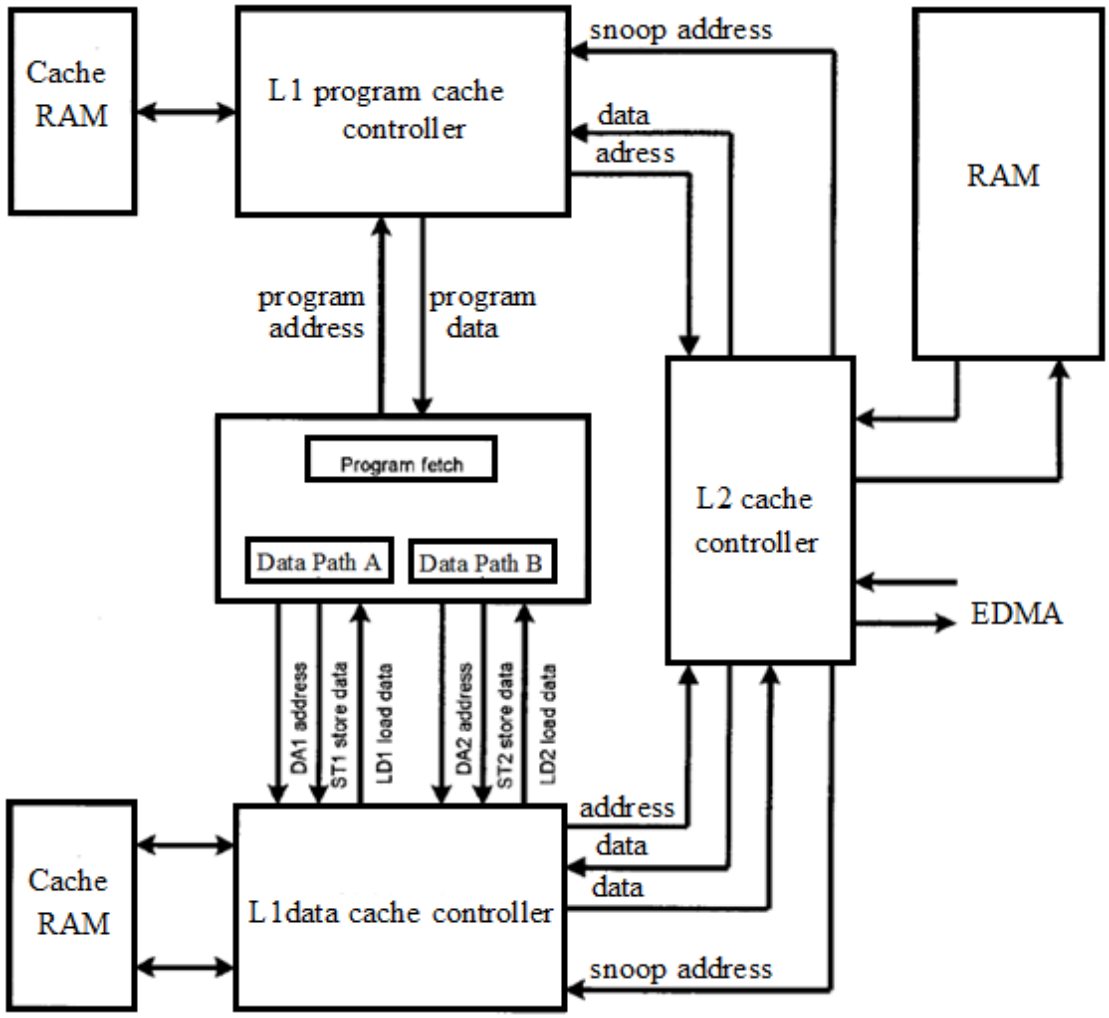


Figure 4.3. Schéma fonctionnel de la mémoire interne.

Exemple :

Le microprocesseur TMS320C6711 du DSK comprend **72 KB** de mémoire interne, commençant à **0x00000000.**

4.3 Mémoires externes (SRAM, Flash, DDRAM, ...)

Le microprocesseur DSP possède une interface sans colle (directe) avec les mémoires synchrones (synchronous memories) (**SDRAM** et **SBSRAM**) et les mémoires asynchrones

(asynchronous memories) (**SRAM** et **EPROM**). La mémoire synchrone nécessite une horloge, mais fournit un compromis entre la SRAM statique et la SDRAM dynamique, la SRAM étant plus rapide, mais plus chère que la DRAM.

Exemple :

Le microprocesseur TMS320C6711 du DSK comprend **16 MB** de **SDRAM** externe, mappée via **CE0** à partir de **0x80000000**. Le **DSK** comprend également **128 KB** de mémoire Flash intégrée, à partir de **0x90000000**.

4.4 Plan d'adressage des mémoires.

Le tableau 4.1 indique l'espace d'adressage de la RAM de programme interne pour le mode mappage sélectionné lors de la réinitialisation de l'appareil. En mode mappé, le CPU et le DMA peuvent accéder à tous les emplacements de la RAM. Tout accès en dehors de l'espace d'adressage auquel la RAM interne est mappée est transmis à l'EMIF. Si le CPU et le DMA tentent d'accéder au même bloc de RAM en même temps, alors le DMA est bloqué jusqu'à ce que le CPU termine ses accès à ce bloc. Une fois l'accès au processeur terminé, le DMA est autorisé à accéder à la RAM.

Pour le C6202(B)/C6203(B), le DMA ne peut accéder qu'à l'un des deux blocs de RAM à la fois. Le CPU et le DMA peuvent accéder à la RAM interne sans interférence tant que chacun accède à un bloc différent. Le DMA ne peut pas traverser le bloc 0 et le bloc 1 en un seul transfert. Des transferts DMA séparés doivent être utilisés pour franchir les limites des blocs.

Tableau 4.1. Mappage d'adresse RAM de programme interne en mode mappé en mémoire

Device	Block	Map 0	Map 1
C6201	—	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6202(B)	Block 0	0140 0000h – 0141 FFFFh	0000 0000h – 0001 FFFFh
	Block 1	0142 0000h – 0143 FFFFh	0002 0000h – 0003 FFFFh
C6203(B)	Block 0	0140 0000h – 0143 FFFFh	0000 0000h – 0003 FFFFh
	Block 1	0144 0000h – 0145 FFFFh	0004 0000h – 0005 FFFFh
C6204	—	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6205	—	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6701	—	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh

Exemple :

Le microprocesseur TMS320C621x/C671x n'a qu'une seule carte mémoire, qui est illustrée dans le tableau 4.2. La mémoire interne est toujours située à l'adresse 0, mais peut être utilisée

à la fois comme mémoire de programme et de données. Le registre de configuration des périphériques communs aux C620x/C670x et C621x/C671x se trouve aux mêmes adresses dans les deux processeurs. Les plages d'adresses de mémoire externe commencent à l'emplacement 8000 0000h dans le C621x/C671x, qui est l'emplacement de la mémoire de données interne dans le C620x.

Tableau 4.2. Résumé de la carte mémoire TMS320C621x/C671x

Address Range (Hex)	Size (Bytes)	Description of Memory Block
0000 0000 – 0000 FFFF	64K	Internal RAM (L2)
0001 0000 – 017F FFFF	24M–64K	Reserved
0180 0000 – 0183 FFFF	256K	Internal configuration bus EMIF registers
0184 0000 – 0187 FFFF	256K	Internal configuration bus L2 control registers
0188 0000 – 018B FFFF	256K	Internal configuration bus HPI register
018C 0000 – 018F FFFF	256K	Internal configuration bus McBSP 0 registers
0190 0000 – 0193 FFFF	256K	Internal configuration bus McBSP 1 registers
0194 0000 – 0197 FFFF	256K	Internal configuration bus timer 0 registers
0198 0000 – 019B FFFF	256K	Internal configuration bus timer 1 registers
019C 0000 – 019F FFFF	256K	Internal configuration bus interrupt selector registers
01A0 0000 – 01A3 FFFF	256K	Internal configuration bus EDMA RAM and registers
01A4 0000 – 01FF FFFF	6M–256K	Reserved
0200 0000 – 0200 0033	52	QDMA registers
0200 0034 – 2FFF FFFF	736M–52	Reserved
3000 0000 – 3FFF FFFF	256M	McBSP 0/1 data
4000 0000 – 7FFF FFFF	1G	Reserved

4.5 Gestion de la mémoire externe par L'EMIF (*External Memory Inter Face*).

Les interfaces de mémoire externe (EMIF) de tous les appareils TMS320C6000 prennent en charge une interface sans colle vers une variété d'appareils externes, notamment :

- ✓ Pipeline synchronous-burst SRAM (SBSRAM)
- ✓ Synchrone DRAM (SDRAM)
- ✓ Périphériques asynchrones, y compris SRAM, ROM et FIFOs
- ✓ Un périphérique de mémoire partagée externe

Les requêtes de services EMIF TMS320C620x/C670x du bus externe proviennent de quatre demandeurs :

- ✓ Contrôleur de mémoire de programme sur puce qui gère les récupérations de programmes CPU
- ✓ Contrôleur de mémoire de données sur puce qui gère les extractions de données du processeur Contrôleur DMA sur puce
- ✓ Contrôleur de périphérique de mémoire partagée externe (via des signaux d'arbitrage EMIF).

Si plusieurs requêtes arrivent simultanément, l'EMIF les hiérarchise et effectue le nombre d'opérations nécessaires. Le contrôle de l'EMIF et des interfaces mémoire qu'il prend en charge est maintenu via des registres mappés en mémoire dans l'EMIF. Les registres mappés en mémoire sont répertoriés dans le tableau 4.3.

Tableau 4.3. Registres mappés en mémoire EMIF

Byte Address		Abbreviation	EMIF Register Name
EMIF/EMIFA	**EMIFB**		
0180 0000h	01A8 0000h	GBLCTL	EMIF global control
0180 0004h	01A8 0004h	CE1CTL	EMIF CE1 space control
0180 0008h	01A8 0008h	CE0CTL	EMIF CE0 space control
0180 000Ch	01A8 000Ch		Reserved
0180 0010h	01A8 0010h	CE2CTL	EMIF CE2 space control
0180 0014h	01A8 0014h	CE3CTL	EMIF CE3 space control
0180 0018h	01A8 0018h	SDCTL	EMIF SDRAM control
0180 001Ch	01A8 001Ch	SDTIM	EMIF SDRAM refresh control
0180 0020h	01A8 0020h	SDEXT	EMIF SDRAM extension§
0180 0024h to	01A8 0024h to	–	Reserved
0180 0040h	01A8 0040h		
0180 0044h	01A8 0044h	CE1SEC	EMIF CE1 space secondary control
0180 0048h	01A8 0048h	CE0SEC	EMIF CE0 space secondary control
0180 004Ch	01A8 004Ch	–	Reserved
0180 0050h	01A8 0050h	CE2SEC	EMIF CE2 space secondary control
0180 0054h	01A8 0054h	CE3SEC	EMIF CE3 space secondary control

Remarques

- ➢ Les signaux EMIF du C6201/C6701 sont illustrés à la figure 4.4. Le C620x/C670x EMIF dispose d'une interface de bus de données 32 bits. Le C6201/C6701 fournit des signaux d'horloge et de commande séparés pour l'interface SBSRAM et SDRAM. La SDRAM

s'exécute sur SDCLK, tandis que la SBSRAM s'exécute sur SSCLK. Les trois types de mémoire (SDRAM, SBSRAM et périphériques asynchrones) peuvent être inclus dans un système. L'interface asynchrone est prise en charge sur tous les espaces CE, mais CE1 est utilisé uniquement pour l'interface asynchrone.

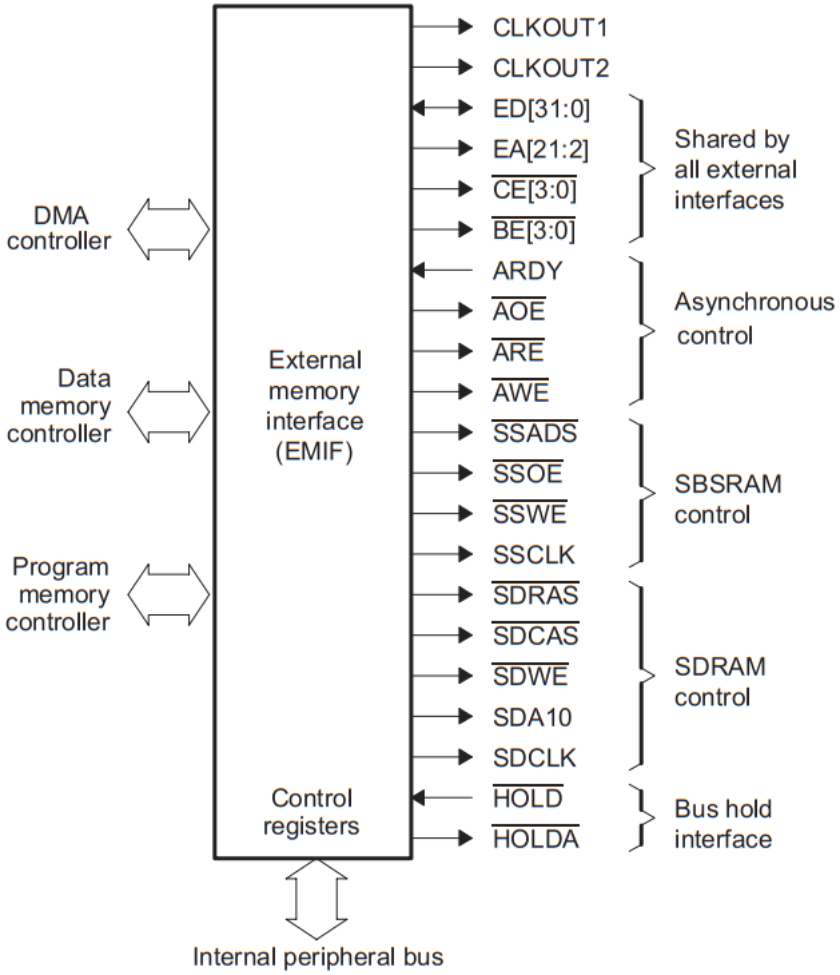


Figure 4.4. Interface mémoire externe TMS320C6201/C6701

- Les requêtes de services C621x/C671x/C64x (la figure 4.5) du bus externe proviennent de deux demandeurs :
  - ✓ Contrôleur d'accès direct à la mémoire amélioré (EDMA) sur puce
  - ✓ Contrôleur de périphérique de mémoire partagée externe
- Le C64x EMIF offre une flexibilité supplémentaire en remplaçant le mode SBSRAM par un mode synchrone programmable, qui prend en charge les interfaces sans colle vers les éléments suivants :
  - ✓ ZBT (Zero Bus Turnaround) SRAM
  - ✓ Synchronous FIFOs
  - ✓ Pipeline and flow-thru SBSRAM

- Le C64x dispose de deux EMIF, EMIFA et EMIFB, par appareil. Les suffixes « A » et « B » indiquent la largeur du bus de données EMIF comme suit :
  - ✓ EMIFA - 64 bits Interface de bus de données
  - ✓ EMIFB - 16 bits Interface de bus de données

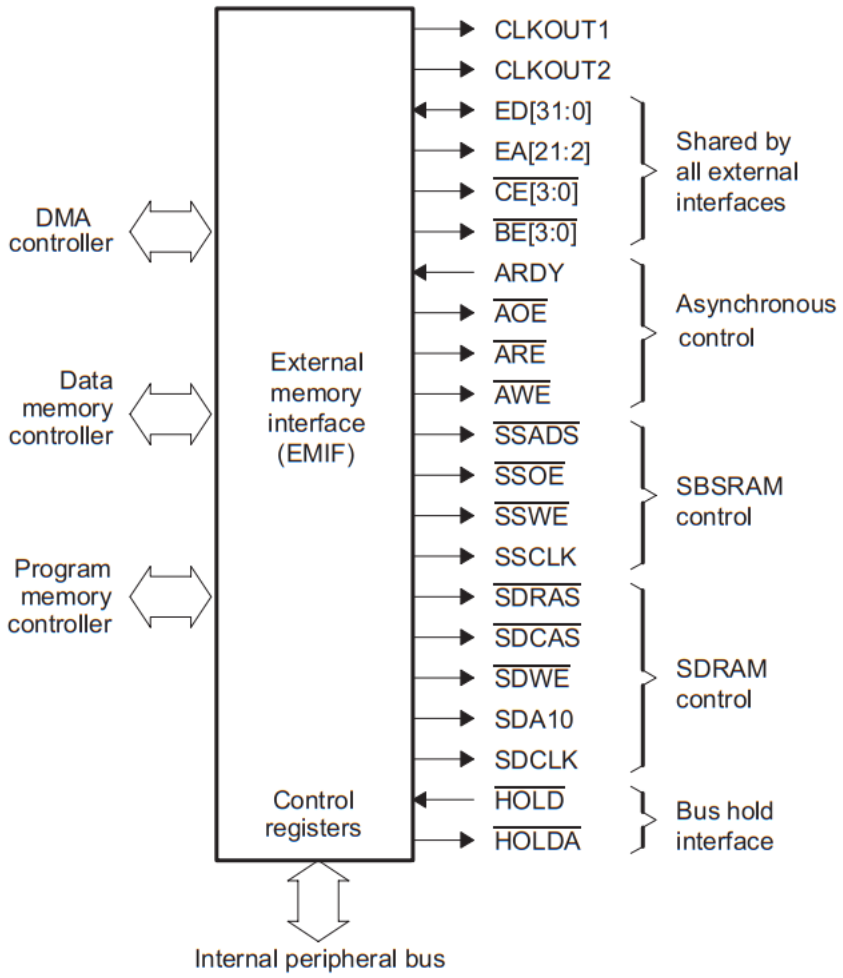


Figure 4.5. Interface mémoire externe TMS320C621x/C671x/C64x

4.6 Modes d'adressage (indirect, circulaire).

Les modes d'adressage déterminent comment on accède à la mémoire. Ils spécifient comment les données sont accédées, telles que la récupération indirecte d'un opérande depuis un emplacement de mémoire. Les modes d'adressage linéaire et circulaire sont pris en charge. Le mode le plus couramment utilisé est l'adressage indirect de la mémoire.

4.6.1 Adressage indirect

L'adressage indirect peut être utilisé avec ou sans déplacement. Le registre R représente l'un des 32 registres A0 à A15 et B0 à B15 pouvant spécifier ou pointer des adresses de mémoire. En tant que tels, ces registres sont des pointeurs. Le mode d'adressage indirect utilise un « *»

en conjonction avec l'un des 32 registres. Pour illustrer, considérons R comme un registre d'adresses.

1. *** R.** Le registre R contient l'adresse d'un emplacement mémoire où une valeur de données est stockée.
2. *** R ++ (d).** Le registre R contient l'adresse de la mémoire (emplacement). Une fois l'adresse mémoire utilisée, R est post incrémenté (modifié), de sorte que la nouvelle adresse est l'adresse actuelle décalée de la valeur de déplacement d. Si d = 1 (par défaut), la nouvelle adresse est R + 1 où R est incrémenté à l'adresse immédiatement supérieure en mémoire. Un double moins (- -) au lieu d'un double plus mettraient à jour ou post-décrémente l'adresse en R - d.
3. *** ++ R (d).** L'adresse est pré-incrémentée ou compensée par d, de sorte que l'adresse actuelle est R + d. Un double moins pré-décrémente l'adresse de la mémoire de sorte que l'adresse actuelle soit R - d.
4. *** + R (d).** L'adresse est pré-incrémentée par d, de sorte que l'adresse actuelle est R + d (comme dans le cas précédent). Cependant, dans ce cas, R pré-incrémentée sans modification. Contrairement au cas précédent, R n'est ni ne mis à jour ni modifié.

4.6.2 Adressage circulaire

L'adressage circulaire est utilisé pour créer un tampon circulaire. Ce tampon est créé dans le matériel et est très utile dans plusieurs algorithmes DSP, tels que les algorithmes de filtrage numérique ou de corrélation où les données doivent être mises à jour.

Le microprocesseur TMS320C6x dispose d'un matériel dédié pour permettre un adressage du type circulaire. Ce mode d'adressage peut être utilisé avec un tampon circulaire pour mettre à jour des échantillons en décalant les données sans la surcharge créée par le décalage direct des données. Lorsqu'un pointeur atteint l'emplacement final ou « inférieur » d'un tampon circulaire contenant le dernier élément du tampon, puis est incrémenté, le pointeur est automatiquement renvoyé à la ligne ou pointe vers le début ou l'emplacement « supérieur » du tampon qui contient le premier élément.

Nous pouvons utiliser deux mémoires-tampons circulaires indépendantes utilisant BK0 et BK1 dans le registre de mode d'adresse (AMR). Les huit registres A4 à A7 et B4 à B7, associés aux deux unités .D, peuvent servir de pointeurs. (Tous les registres peuvent être utilisés pour l'adressage linéaire). Le segment de code suivant illustrer l'utilisation d'un tampon circulaire à l'aide du registre B2 (seul le côté B peut être utilisé) pour définir les valeurs appropriées dans AMR :

MVK .S2 0x0004,B2 ; inférieur de 16 bits à B2. Sélectionnez A5 comme pointeur

MVKLH .S2 0x0005,B2 ; supérieur de 16 bits à B2. Sélectionnez B0, définissez N = 5

MVC .S2 B2,AMR ; déplacez 32 bits de B2 vers AMR

Les deux instructions de déplacement MVK et MVKLH (à l'aide de l'unité .S) déplacent 0x0004 dans les 16 LSB du registre B2 et 0x0005 dans les 16 MSB de B2. L'instruction MVC (constante de déplacement) est la seule à pouvoir accéder à AMR et aux d'autres registres de contrôle et s'exécute uniquement du côté B en association avec les unités fonctionnelles et des registres du côté B. Une valeur de 32 bits est créée dans B2, qui est ensuite transférée à AMR avec l'instruction MVC d'accès AMR.

La valeur $0x0004 = (0100)_b$ dans les 16 LSB d'AMR met le bit 2 (troisième bit) sur 1 et tous les autres bits sur zéro. Ceci règle le mode sur 01 et sélectionne le registre A5 comme pointeur sur un tampon circulaire en utilisant le bloc BK0.

Le tableau 4-4 présente les modes associés aux registres A4 à A7 et B4 à B7. La valeur $0x0005 = (0101)_b$ dans les 16 MSo de AMR définit les bits 16 et 18 sur 1 (les autres bits étant à zéro). Ceci correspond à la valeur de N utilisée pour sélectionner la taille du tampon sous la forme 2N + 1 = 64 octets avec BK0. Par exemple, si on utilise une taille de tampon de 128 à l'aide de BK0, les 16 bits supérieurs d'AMR sont définis sur $(0110)_b = 0x0006$. Si le code d'assemblage est utilisé pour le tampon circulaire, lorsque l'exécution retourne à une fonction C appelante, AMR doit être réinitialisé sur le mode linéaire par défaut. Par conséquent, l'adresse du pointeur doit être sauvegardée.

Tableau 4.4 Mode AMR et description

Mode	Description
0 0	Pour l'adressage linéaire (par défaut lors de la réinitialisation)
0 1	Pour un adressage circulaire avec BK0
1 0	Pour un adressage circulaire avec BK1
1 1	Réservé

4.7 Technique de transfert par blocs.

Le contrôleur d'accès direct à la mémoire (Direct Demory Access DMA) transfère les données entre les régions de la carte mémoire sans intervention de la CPU. Le contrôleur DMA permet le déplacement de données vers et depuis la mémoire interne, les périphériques internes ou les périphériques externes en arrière-plan du fonctionnement de la CPU. Le contrôleur DMA

dispose de quatre canaux programmables indépendants, permettant quatre contextes différents pour le fonctionnement DMA. De plus, un cinquième canal (auxiliaire) permet au contrôleur DMA de répondre aux demandes de l'interface de port hôte (HPI).

- **Transfert de bloc :** chaque canal DMA a également un nombre programmable indépendamment de trames par bloc. Lors de l'exécution d'un bloc-transfert, le contrôleur DMA déplace toutes les trames qu'il a été programmé pour déplacer.
- **Transfert d'un grand bloc unique :** ELEMENT COUNT peut être utilisé en conjonction avec FRAME COUNT pour permettre des transferts de blocs d'une seule image de plus de 65535 octets. Le produit de ELEMENT COUNT et FRAME COUNT peut former un plus grand nombre d'éléments effectifs. Ce qui suit doit être effectué :
  - ✓ Si l'adresse doit être ajustée à l'aide d'une valeur programmable (DIR = 11b), FRAME INDEX doit être égal à ELEMENT INDEX si l'ajustement d'adresse est déterminé par un registre d'index global DMA. Ceci s'applique aux adresses source et destination. Si l'adresse ne doit pas être ajustée par une valeur programmable, cette contrainte ne s'applique pas, car le même ajustement d'adresse se produit par défaut aux limites d'élément et de trame.
  - ✓ La synchronisation de trame doit être désactivée (c'est-à-dire que FS doit être défini sur 0 dans le registre de contrôle principal du canal DMA). Cela évite les exigences de synchronisation au milieu du grand bloc.
  - ✓ Le nombre d'éléments dans la première trame est Ei. Le nombre d'éléments dans les trames successives est ((F – 1) × Er). Le nombre d'éléments effectif est ((F – 1) × Er) + Ei où:

    F = valeur initiale de FRAME COUNT

    Er = ELEMENT COUNT valeur de rechargement

    Ei = valeur initiale de ELEMENT COUNT
  - ✓ Ainsi, pour transférer 128K + 1 éléments, vous pouvez définir F sur 5, Er sur 32K et Ei sur 1.
- **Lancement d'un transfert de bloc**

Chaque canal DMA peut être démarré indépendamment, soit manuellement via un accès direct au processeur, soit automatiquement via une initialisation automatique. Chaque canal DMA peut être arrêté ou mis en pause indépendamment via un accès direct au processeur. L'état d'un canal DMA peut être observé en lisant le champ STATUS dans le registre de contrôle primaire du canal DMA (PRICTL).

Une fois la valeur de START modifiée, le registre de contrôle primaire ne doit plus être modifié tant que STATUS n'est pas égal à START.

- ✓ **Opération de démarrage manuel :** Pour démarrer l'opération DMA pour un canal particulier, une fois que les valeurs souhaitées sont écrites dans tous les autres registres de contrôle DMA, la valeur souhaitée doit être écrite dans le PRICTL avec START = 01b. L'écriture de cette valeur sur un canal DMA déjà démarré n'a aucun effet. Une fois démarré, la valeur sur STATUS est 01b.
- ✓ **Opération de pause :** Une fois démarré, un canal DMA peut être mis en pause en écrivant START = 10b. Lorsqu'il est en pause, le canal DMA termine tous les transferts d'écriture dont les demandes de transfert de lecture sont terminées. De plus, si le canal DMA a toutes les synchronisations de lecture nécessaires, un transfert d'élément supplémentaire est autorisé à se terminer. Une fois en pause, la valeur sur STATUS devient 10b après que le DMA a terminé tous les transferts d'écriture en attente.
- ✓ **Arrêter l'opération :** Le contrôleur DMA peut être arrêté en écrivant START = 00b. L'opération d'arrêt est identique à l'opération de pause. Une fois qu'un transfert DMA est terminé, à moins que l'initialisation automatique ne soit activée, le canal DMA revient à l'état arrêté et STATUS devient 00b une fois que le DMA a terminé tous les transferts d'écriture en attente.

➢ **Auto-initialisation DMA :** Le contrôleur DMA peut se réinitialiser automatiquement après l'achèvement d'un bloc-transfert. Certains des registres de contrôle DMA peuvent être préchargés pour le bloc-transfert suivant via des registres de données globales DMA sélectionnés. En utilisant cette capacité, certains des paramètres du canal DMA peuvent être réglés bien avant le transfert de bloc suivant. L'initialisation automatique permet :

- ✓ **Fonctionnement continu :** Le CPU dispose d'un temps mort long, pendant lequel il peut reconfigurer le contrôleur DMA pour un transfert ultérieur. Normalement, la CPU devrait réinitialiser le contrôleur DMA immédiatement après l'achèvement du dernier transfert d'écriture dans le bloc-transfert en cours et avant la première synchronisation de lecture pour le bloc-transfert suivant. Avec les registres de rechargement, il peut réinitialiser ces valeurs pour le bloc-transfert suivant à tout moment après le début du bloc-transfert en cours.
- ✓ **Fonctionnement répétitif :** Ce fonctionnement est un cas particulier du fonctionnement continu. Une fois qu'un bloc-transfert est terminé, le contrôleur DMA répète le bloc-transfert précédent. Dans ce cas, la CPU ne précharge pas les

registres de rechargement avec de nouvelles valeurs pour chaque bloc-transfert. Au lieu de cela, la CPU charge les registres uniquement avant le premier bloc-transfert. Activation de l'initialisation automatique : En écrivant START = 11b dans le registre de contrôle primaire de la voie, l'initialisation automatique est activée. Dans ce cas, après l'achèvement d'un transfert de bloc, les registres de canal DMA sélectionnés sont rechargés et le canal DMA est redémarré. En cas de redémarrage après une pause, START doit être réécrit en tant que 11b pour que l'initialisation automatique soit activée.

- ✓ **Passage de l'initialisation automatique à l'initialisation non automatique :** il est possible de passer d'un transfert initialisé automatiquement à un transfert initialisé non automatique pour terminer l'activité DMA sur un canal particulier. Pour changer de mode, le canal actif doit être mis en pause en restaurant le registre de contrôle principal avec START = 10b, puis redémarré dans le nouveau mode en restaurant le registre de contrôle principal avec START = 01b. Si le canal actif fonctionne en mode divisé, il est alors nécessaire de s'assurer que le passage de l'initialisation automatique à l'initialisation non automatique ne se produit pas à une limite de trame. Si le canal est en pause avec la source d'émission dans la trame n et la destination de réception dans la trame n – 1 ; puis le canal doit être redémarré avec auto initialisation (START = 11b), puis remis en pause avant que le changement de mode ne se produise. Cela permet de s'assurer que les flux de données d'émission et de réception complètent tous deux le même nombre de trames.

Remarque :

Pour l'auto-initialisation, les blocs-transferts successifs sont supposés similaires.

- ➢ **Verrouiller le transfert 2D synchronisé (FS=1) :** Pour un transfert 2D, le bloc complet est transféré lorsque l'événement de la voie se produit et FS = 1. La synchronisation de bloc entraîne l'implémentation de l'index de tableau (FRMIDX) par la logique de génération/transfert d'adresse. Cette mise à jour d'adresse est transparente pour l'utilisateur et n'est pas reflétée dans la RAM de paramètres. L'adresse est mise à jour après chaque élément d'une rafale. La logique met d'abord à jour les adresses en fonction du réglage de SUM/DUM. Si un élément est le dernier d'un tableau particulier et qu'un mode de mise à jour est sélectionné (SUM/DUM ≠ 00b), la ou les adresses sont indexées en fonction de l'index du tableau. L'index est ajouté à l'adresse après la mise à jour de

l'adresse. FRMIDX est donc égal à l'espace entre les tableaux d'un bloc, comme illustré à la figure 4.6.

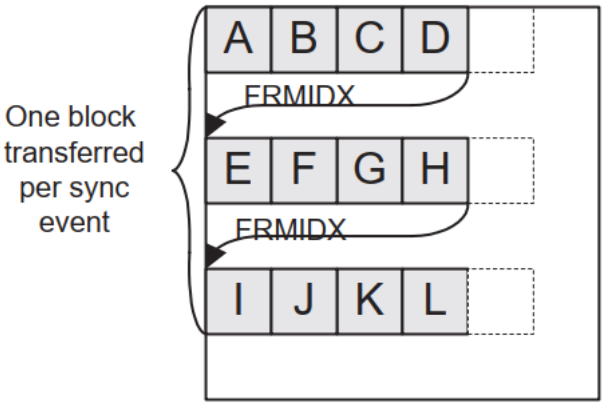


Figure 4.6. 2-D Transfer with Block Synchronization (FS=1)

Remarque

Si la liaison est activée (LINK=1), le transfert de bloc EDMA suivant dans la liaison (comme spécifié par l'adresse de liaison) est effectué dès que la synchronisation de bloc suivante arrive.

4.8. Lier les transferts EDMA

Le contrôleur EDMA fournit une liaison, une fonctionnalité particulièrement utile pour les applications de type tri complexe et mise en mémoire tampon circulaire. Si LINK = 1, à la fin d'un transfert, la fonction de liaison EDMA recharge les paramètres de transfert actuels avec le paramètre pointé par l'adresse de liaison 16 bits. La totalité de la RAM de paramètres EDMA se trouve dans la zone 01A0 xxxxh. Par conséquent, l'adresse de liaison 16 bits, qui correspond à l'adresse physique 16 bits inférieure, est suffisante pour spécifier l'emplacement de la prochaine entrée de transfert. L'adresse de liaison doit être alignée sur une limite de 24 octets. Un exemple de transfert EDMA lié est illustré à la figure 4.7.

L'adresse du lien n'est évaluée que si LINK est égal à 1 et seulement après épuisement des paramètres d'événement. Les paramètres d'un événement sont épuisés lorsque le contrôleur EDMA a terminé le transfert associé à la requête. Le tableau 4–5 montre les conditions d'achèvement du canal lorsque la liaison des paramètres est effectuée. Il n'y a pratiquement aucune limite à la durée des transferts liés. Cependant, la dernière entrée de paramètre de transfert doit avoir son LINK = 0 afin que le transfert lié s'arrête après le dernier transfert. La dernière entrée doit être liée à un jeu de paramètres NULL.

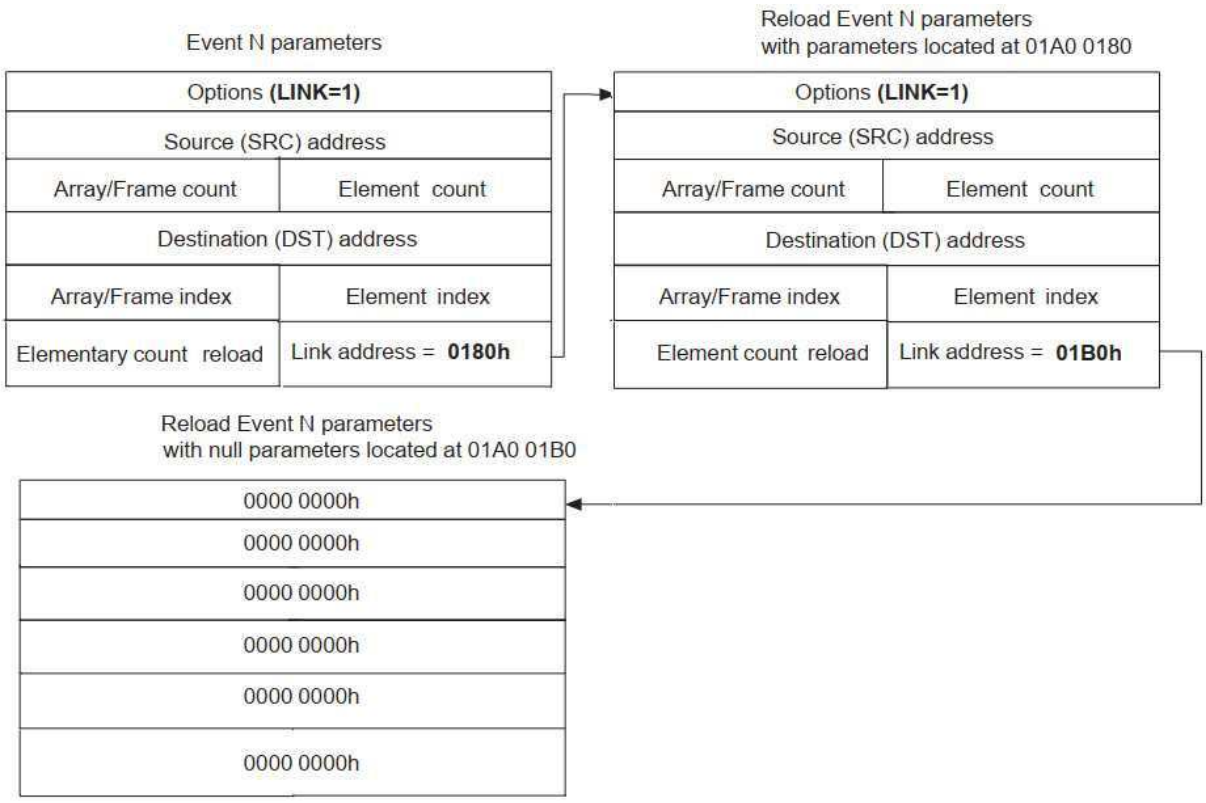


Figure 4.7. Transfert EDMA lié

Tableau 4.5. Conditions d'achèvement du canal

LINK = 1	1D Transfers	2D Transfers
Element/array sync	Frame count == 0 &&	Frame count == 0
(FS = 0)	Element count == 1	
Frame sync (FS = 1)	Frame count == 0	Always

Remarque

- Lier une entrée à elle-même reproduit le comportement de l'auto-initialisation pour faciliter l'utilisation de la mise en mémoire tampon circulaire et des transferts répétitifs. Après qu'un canal EDMA a épuisé son entrée actuelle, le jeu de paramètres est rechargé et le transfert recommence.
- Une fois que les conditions d'achèvement du canal sont remplies pour un événement, les paramètres de transfert situés à l'adresse de liaison sont chargés dans l'un des 16 espaces de paramètres d'événement (C621x/C671x) ou 64 espaces de paramètres d'événement (C64x) pour l'événement correspondant. Maintenant, l'EDMA est prêt à démarrer le prochain transfert. Pour éliminer d'éventuelles fenêtres temporelles posées pendant ce mécanisme de rechargement de paramètres, le contrôleur EDMA n'évalue pas le registre

d'événements pendant ce temps. Cependant, les événements sont toujours capturés dans l'ER et seront traités une fois le rechargement des paramètres terminé.

- Toute entrée dans la PaRAM peut être utilisée pour un jeu de paramètres de transfert lié. Les entrées dans les 16 premiers emplacements (C621x/C671x) ou 64 (C64x) ne doivent être utilisées pour la liaison que si l'événement correspondant et l'événement en chaîne sont désactivés.

4.9. Terminer un transfert EDMA

Tous les transferts EDMA sont terminés par une liaison à un jeu de paramètres NULL après le dernier transfert.

L'ensemble de paramètres NULL sert de point de terminaison de tout transfert EDMA. Un jeu de paramètres NULL est défini comme un jeu de paramètres EDMA où tous les paramètres (options, adresse source/destination, nombre de trames/éléments, etc.) sont mis à zéro. Plusieurs transferts EDMA peuvent être liés au même ensemble de paramètres NULL de terminaison. Par conséquent, un seul jeu de paramètres NULL est requis dans la RAM de paramètres EDMA. La figure 4.8 est un exemple de terminaison de transfert EDMA.

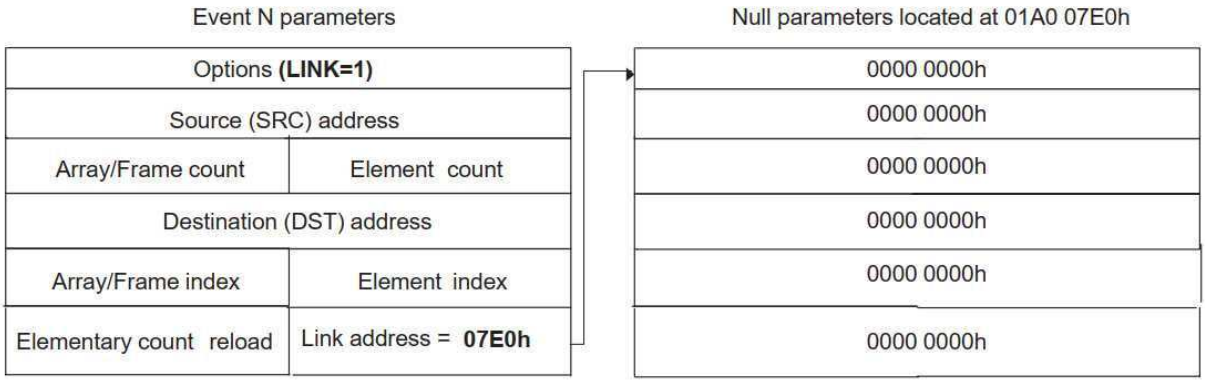


Figure 4.8. Terminer les transferts EDMA

4.10. Paramètres et options pour l'EDMA.

Tous les demandeurs de transfert vers l'EDMA sont connectés à la chaîne de demande de transfert. Ceci est illustré à la figure 4.9. Une demande de transfert, une fois soumise, est déplacée à travers la chaîne vers la barre transversale de transfert (TC), où elle est priorisée et traitée. La demande de transfert peut être pour un seul élément de données ou pour un grand nombre d'éléments.

La chaîne de demande fournit un schéma de priorité inhérent aux demandeurs. En supposant que chacun fait une soumission sur le même cycle, le demandeur le plus proche du TC (demandeur en aval) arrive en premier et le plus éloigné (demandeur en amont) arrive en

dernier. Une fois qu'une demande se trouve dans la chaîne de demandes, elle a la priorité sur les nouvelles soumissions, de sorte que les demandes à la fin de la chaîne ne soient pas privées de service.

Pour éviter d'éventuelles situations de blocage qui se produiraient si un demandeur en aval n'était pas soumis en raison de soumissions continues par des demandeurs en amont, il existe un système de tourniquet mis en œuvre dans la logique de la chaîne. Un jeton est passé autour de la chaîne (pour le jeton, c'est une boucle) dans le sens aval à chaque cycle d'horloge du processeur. Le nœud de demande de transfert qui possède le jeton inverse les niveaux de priorité de ses deux demandeurs. Plutôt que de donner la priorité à une requête existante dans la chaîne, située dans le nœud amont, la priorité est donnée au demandeur local avec le jeton pour soumettre une nouvelle requête. Bien qu'il s'agisse d'une protection implantée dans l'EDMA, la bande passante élevée de l'EDMA par rapport à la vitesse à laquelle les demandes sont soumises a montré que cela était sans conséquence.

Les demandeurs incluent le contrôleur L2, les canaux EDMA et le HPI/PCI.

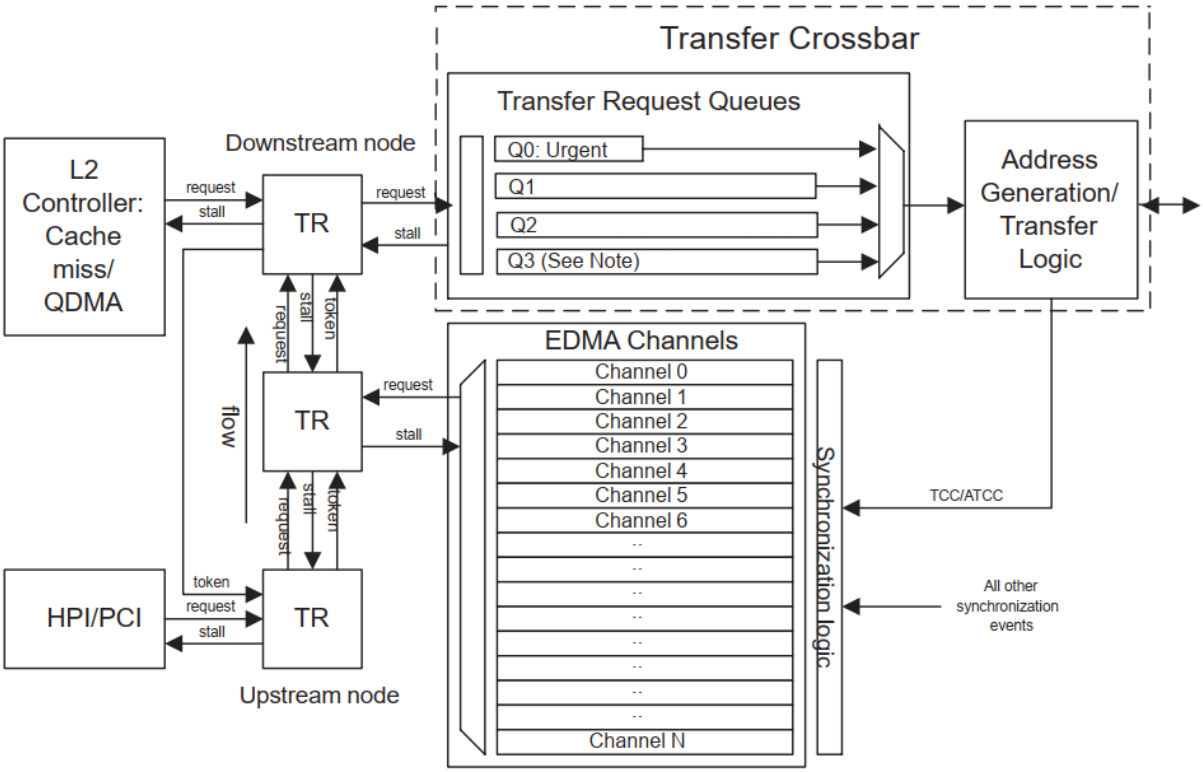


Figure 4.9. Schéma fonctionnel EDMA

Le champ ESIZE dans les options d'une entrée de paramètre d'événement permet à l'utilisateur de spécifier la taille d'élément que l'EDMA doit utiliser pour un transfert. Le contrôleur EDMA

peut transférer des mots de 32 bits, des demi-mots de 16 bits ou des octets de 8 bits dans un transfert.

Les adresses doivent être alignées sur la limite de taille d'élément. Les accès mot et demi-mot doivent être alignés sur une limite de mot (multiple de 4) et de demi-mot (multiple de 2), respectivement. Des valeurs non alignées peuvent entraîner un fonctionnement indéfini.

Lors du transfert d'une rafale d'éléments vers ou depuis un périphérique large de 64 bits (par exemple L2 ou EMIFA), les éléments 64 bits sont transférés quel que soit l'ESIZE programmé. Cela permet à l'EDMA de maximiser la bande passante disponible.

4.11. Considérations relatives au transfert en mode adresse fixe

La taille maximale des éléments EDMA est de 32 bits. Cependant, les chemins de données suivants ont une largeur de 64 bits :

- ✓ SRAM L2
- ✓ EMIFA (EMIF 64 bits, C64x uniquement)

Des précautions doivent être prises lors de l'exécution d'un accès en rafale via les chemins de données 64 bits qui ont les configurations EDMA suivantes :

- ✓ La taille de l'élément est de 32 bits (ESIZE = 00b)
- ✓ Mode adresse fixe (SUM ou DUM = 00b dans le paramètre options)
- ✓ Accès synchronisé en trame (FS = 1 dans le paramètre options), ou transfert source ou destination bidimensionnel (soit 2DS, soit 2DD est réglé sur 1).
- ✓ Le nombre d'éléments est supérieur à 1 (ELECNT > 1).

Les accès à un bus de données de 64 bits de large avec les configurations EDMA ci-dessus sont fixés sur une frontière de 64 bits. Par exemple, lors de l'exécution d'un nombre N d'accès 32 bits à la SRAM L2 ou à l'EMIFA en mode d'adresse fixe (ELECNT = N, N> 1), l'EDMA effectue en fait un nombre N/2 d'accès 64 bits au mot double fixe adresse. C'est donc en fait un mot double de 64 bits qui est transféré.

Pour une écriture sur un bus de données de 64 bits de large avec les conditions ci-dessus, le mot de 32 bits est écrit à la fois sur le mot 0 et le mot 1 de l'adresse de mot double fixe. Par exemple, une écriture 32 bits à l'adresse SRAM L2 0x00000000 met à jour le mot 0 (à l'adresse 0x00000000) et le mot 1 (à l'adresse 0x00000004) avec les nouvelles données.

Pour une lecture à partir d'un bus de données de 64 bits de large avec les conditions ci-dessus, à la fois le mot 0 et le mot 1 de l'adresse de mot double fixe sont extraits. Par exemple, lors de l'exécution d'un transfert de mot EDMA de l'adresse fixe SRAM L2 0x00000000 vers une mémoire externe via un EMIF 32 bits, les données extraites de la SRAM L2 s'afficheront sur

les broches EMIF ED[31:0] en tant que "mot 0", "mot 1", "mot 0", "mot 1"…etc. Les considérations ci-dessus ne s'appliquent qu'aux accès à un bus de données de 64 bits.

Pour l'accès par mot en mode adresse fixe EDMA à un registre interne 32 bits ou à un dispositif de mémoire externe 32/16/8 bits, l'adresse est fixée sur une limite de mot 32 bits. Les lectures et les écritures ne sont effectuées que sur l'adresse de mot spécifiée.

4.12 Exemples de transfert de données.

Une grande variété de transferts peut être effectuée par l'EDMA en fonction de la configuration des paramètres. Les transferts plus basiques peuvent être effectués soit par un canal EDMA, soit en soumettant un QDMA. Les transferts plus compliqués ou les transferts répétitifs nécessitent l'utilisation d'un canal EDMA.

Exemple 1 : Déplacement de bloc

Le transfert le plus élémentaire pouvant être effectué par l'EDMA est celui d'un déplacement de bloc. Souvent, pendant le fonctionnement de l'appareil, il est nécessaire de transférer un bloc de données d'un emplacement à un autre, généralement entre la mémoire sur puce et la mémoire hors puce. Dans cet exemple, une section de données doit être copiée de la mémoire externe vers la SRAM L2 interne. Le bloc de données est de 256 mots et réside à l'adresse 0xA0000000 (CE2). Il doit être transféré à l'adresse interne 0x00002000 (bloc L2 0). Le transfert de données est illustré à la figure 4.10.

Le moyen le plus rapide d'effectuer ce transfert consiste à effectuer une requête QDMA. La demande QDMA peut être soumise de plusieurs manières différentes, la plus basique étant un transfert 1-D à 1-D synchronisé avec les trames. Ce type de transfert est valable pour des tailles de blocs inférieures à 64k éléments. Le transfert doit être synchronisé en trame afin que tous les éléments soient transférés une fois l'entrée soumise. QDMA soumet toutes les demandes sous forme de transferts synchronisés sur les trames, quelle que soit la valeur du bit FS.

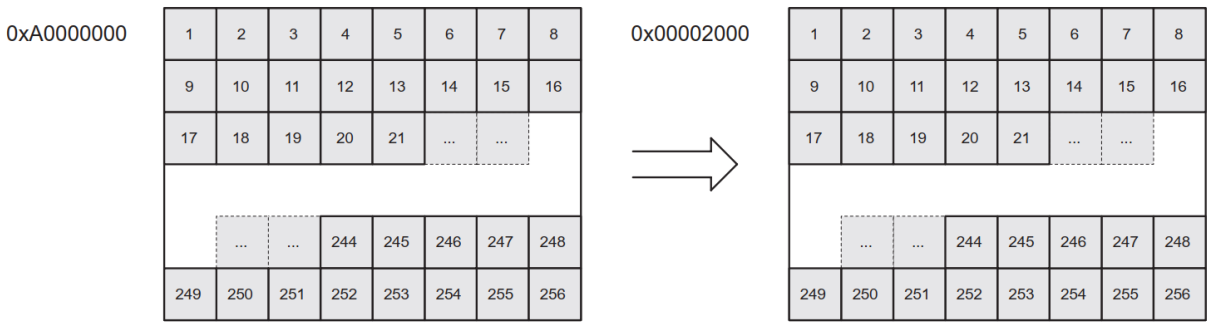


Figure 4.10. Diagramme de mouvement de bloc

Les paramètres de ce transfert sont illustrés à la figure 4.11. Ceux qui doivent être configurés sont les options QDMA, l'adresse source, l'adresse de destination et le nombre d'éléments.

L'adresse source du QDMA est définie au début du bloc de données dans la mémoire externe et l'adresse de destination est définie au début du bloc de données dans L2. Étant donné que toutes les données sont contiguës, SUM et DUM sont tous deux définis sur 01b (incrément). La priorité (PRI) est définie sur faible priorité pour le transfert en arrière-plan.

La CPU a besoin de quatre cycles pour soumettre la demande de ce transfert, un cycle pour chaque écriture de registre. Moins de registres sont requis si l'un des registres QDMA est déjà configuré, avec un minimum d'un cycle. Trois des paramètres QDMA doivent être écrits dans leurs propres registres QDMA et un paramètre doit être écrit dans son pseudo-registre, qui initie le transfert. Voici un exemple de soumission QDMA pour le transfert ci-dessus :

```
...
QDMA_SRC = 0xA0000000; /* Set source address */
QDMA_DST = 0x00002000; /* Set destination address */
QDMA_CNT = 0x00000100; /* Set frame/element count */
QDMA_S_OPT = 0x41200001; /* Set options and submit */
...
```

Un bloc contenant plus de 64k éléments nécessite l'utilisation à la fois du nombre d'éléments et du nombre de tableaux/images. Étant donné que le champ de comptage d'éléments n'est que de 16 bits, la plus grande valeur de comptage pouvant être représentée est 65535. Tout comptage supérieur à celui-ci doit également être représenté par un comptage de tableau. Afin de transmettre cette quantité de données, un QDMA peut toujours être utilisé. Plutôt qu'un transfert 1-D à 1-D synchronisé par trame, le QDMA doit être configuré comme un transfert 2-D à 2-D synchronisé par bloc (FS=1).

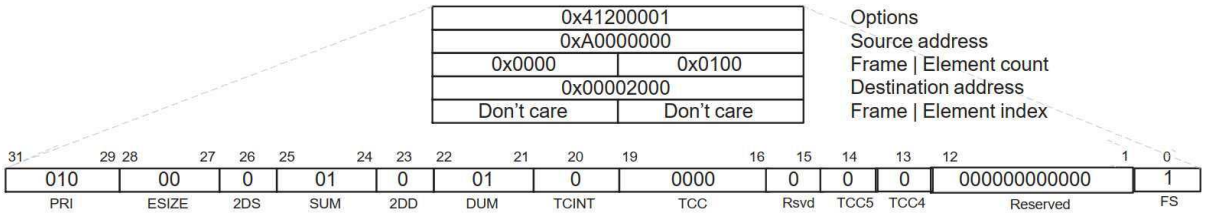


Figure 4.11. Paramètres QDMA de déplacement de bloc

Remarque

PRI (bits 31:29) est défini sur 010b (C621x/C671x) ou 011b (C64x) pour un transfert en arrière-plan de faible priorité. TCCM (bits 14:13) sont réservés sur C621x/C671x.

Exemple 2 : Sub-frame Extraction

L'EDMA a un moyen efficace d'extraire une petite trame de données d'une plus grande. En effectuant un transfert 2-D vers 1-D, l'EDMA peut récupérer une partie des données à traiter par la CPU. Dans cet exemple, une image de 640 x 480 pixels de données vidéo est stockée dans la mémoire externe, CE2. Chaque pixel est représenté par un demi-mot de 16 bits. Une sous-trame de 16 x 12 pixels de l'image est extraite pour être traitée par le CPU. Pour faciliter un temps de traitement plus efficace par le CPU, l'EDMA place la sous-trame dans la SRAM L2 interne. La figure 4.12 illustre le transfert de la sous-trame de la mémoire externe vers L2.

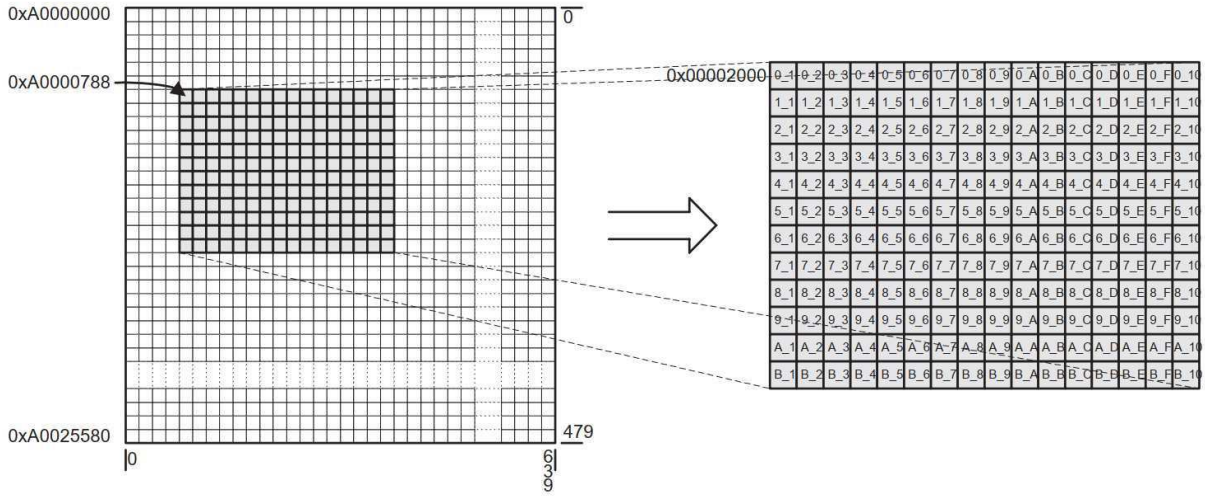


Figure 4.12. Extraction du sous-châssis

Pour effectuer ce transfert, la CPU peut émettre une requête QDMA pour un transfert 2D vers 1D synchronisé par bloc (FS=1). Étant donné que la source est 2D et que le transfert est synchronisé par bloc, le QDMA demande un transfert de la sous-trame entière. Les paramètres requis pour que les registres QDMA demandent ce transfert sont illustrés à la figure 4.13.

Toutes les mises à jour d'adresses se produisent dans la logique de génération/transfert d'adresses. L'indice de tableau fourni est donc l'espace entre les tableaux de la sous-trame. Étant donné que chaque tableau de l'image vidéo a une longueur de 640 pixels et que chaque tableau de la sous-trame a une longueur de 16 pixels, l'indice de tableau est défini sur 2 octets/élément * (640 – 16) éléments = 1248 octets (0x4=0). La sous-trame est transférée dans un bloc de mémoire contigu. Le nombre d'éléments (ELECNT) est fixé à 16, le nombre d'éléments par tableau de sous-trame et le nombre de tableaux (FRMCNT) est fixé à 11, un de moins que le nombre de tableaux. La demande QDMA est envoyée à la file d'attente de faible priorité afin qu'elle n'interfère pas avec l'acquisition de données qui pourrait se produire.

Inversement, un transfert 1-D vers 2-D peut être utilisé pour effectuer l'insertion d'une sous-trame dans une plus grande trame de données. Par exemple, avec cet exemple, la sous-trame pourrait être réinsérée dans l'image plus grande après un certain traitement par le CPU.

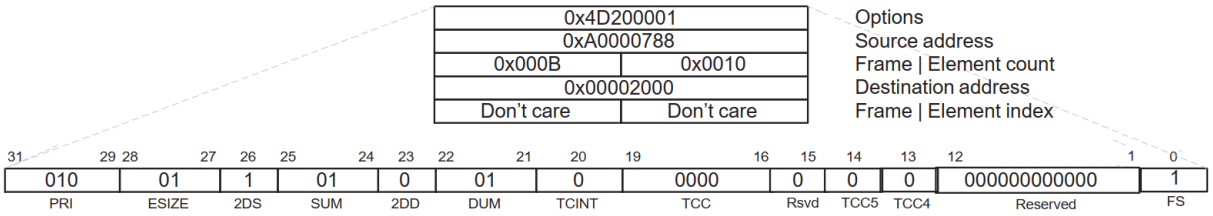


Figure 4.13. Paramètres QDMA d'extraction de sous-trame.

Remarque

PRI (bits 31:29) est défini sur 010b (C621x/C671x) ou 011b (C64x) pour un transfert de faible priorité. TCCM est réservé sur C621x/C671x

Chapitre 5 : Environnement de développement : 'Code Composer Studio' (CCS)

5.1 Introduction

Le Code Composer Studio (CCS) fournit un environnement de développement intégré (IDE) pour incorporer les outils logiciels. CCS comprend des outils de génération de code, tels qu'un compilateur C, un assembleur et un éditeur de liens. Il a des capacités graphiques et prend en charge le débogage en temps réel. Il fournit un outil logiciel facile à utiliser pour créer et déboguer des programmes.

Le compilateur C compile un programme source C avec l'extension .c pour produire un fichier source d'assemblage avec l'extension.asm. L'assembleur assemble un fichier source.asm pour produire un fichier objet en langage machine avec l'extension.obj. L'éditeur de liens combine des fichiers objets et des bibliothèques d'objets en entrée pour produire un fichier exécutable avec l'extension *.out*. Ce fichier exécutable représente un format de fichier objet commun lié (Common Object File Format COFF), populaire dans les systèmes basés sur Unix et adopté par plusieurs fabricants de processeurs de signaux numériques. Ce fichier exécutable peut être chargé et exécuté directement sur le processeur C6711.

Pour créer un projet d'application, on peut « ajouter » les fichiers appropriés au projet. Les options du compilateur/éditeur de liens peuvent facilement être spécifiées. Un certain nombre de fonctionnalités de débogage sont disponibles, notamment la définition de points d'arrêt et la surveillance des variables, la visualisation de la mémoire, des registres et du code mixte C et assembleur, la représentation graphique des résultats et la surveillance du temps d'exécution. On peut avancer dans un programme de différentes manières (entrer, ou dépasser, ou sortir).

L'analyse en temps réel peut être effectuée à l'aide d'un échange de données en temps réel (Real-Time Data EXchange RTDX) associé à DSP/BIOS. RTDX permet l'échange de données entre l'hôte et la cible et l'analyse en temps réel sans arrêter la cible. Les statistiques clés et les performances peuvent être surveillées en temps réel. Grâce au Joint Team Action Group (JTAG), la communication avec le support d'émulation sur puce se produit pour contrôler et surveiller l'exécution du programme. La carte DSK C6711 comprend une interface d'émulation JTAG.

5.2 Configuration de base 'Basic Setup'

Utilisez le câble parallèle (imprimante) DB25 pour connecter la carte DSK (J2) au port parallèle du PC, tel que LPT1 ou LPT2. Utilisez l'adaptateur 5-V fourni avec le package DSK pour vous

connecter au connecteur d'alimentation J4, pour allumer le DSK. Installez CCS avec le CD-ROM fourni avec le DSK, en utilisant de préférence la structure *c:\ti* (par défaut).

L'icône CCS doit être sur le bureau en tant que "CCS 2 ['C 6000]" et est utilisée pour lancer CCS. Les outils de génération de code (C compiler, assembler, linker) version 4.1 sont utilisés.

A la mise sous tension, les trois LED situées près des quatre commutateurs DIP utilisateur doivent compter de 1 à 7 (binaire).

CCS fournit des documentations utiles incluses avec le package DSK sur les éléments suivants (voir l'icône d'aide) :

1. Outils de génération de code (compiler, assembler, linker, etc.)
2. Tutoriels sur CCS, compilateur, RTDX, DSP/BIOS avancé
3. Les instructions et registres de DSP
4. Outils sur RTDX, DSP/BIOS, etc.

Une grande quantité de matériel de support (*fichiers pdf*) est inclus avec CCS. Il existe également quelques exemples inclus avec CCS, tels qu'un exemple de test de confiance pour le DSK, un exemple audio et un exemple associé au flash intégré.

Un certain nombre de fichiers inclus dans les sous-dossiers/répertoires suivants dans *c:\ti* peuvent être très utiles :

1. *docs* : contient la documentation et les manuels.
2. *myprojects* : fourni pour vos projets. Tous les programmes et projets discutés dans ce livre peuvent être placés dans ce sous-répertoire.
3. *c6000\cgtools* : contient des outils de génération de code.
4. *bin* : contient de nombreux utilitaires.
5. *c6000\examples* : contient des exemples inclus avec CCS.
6. *c6000\RTDX* : contient des fichiers de support pour le transfert de données en temps réel.
7. *c6000\bios* : contient les fichiers de support pour DSP/BIOS.

5.3 Types de fichiers utiles

Il existe un certain nombre de fichiers avec différentes extensions. Ils comprennent :

1. *file.pjt* : pour créer et construire un projet nommé file.
2. *fichier.c* : programme source de C.
3. *file.asm* : programme source d'assemblage créé par l'utilisateur, par le compilateur C ou par l'optimiseur linéaire.

4. *fichier.sa* : programme source d'assemblage linéaire. L'optimiseur linéaire utilise *file.sa* comme entrée pour produire un programme d'assemblage *file.asm*.
5. *file.h* : fichier de support d'en-tête.
6. *file.lib* : fichier de bibliothèque, tel que le fichier de bibliothèque de support d'exécution *rts6701.lib*.
7. *file.cmd* : fichier de commande de l'éditeur de liens qui mappe les sections sur la mémoire.
8. *file.obj* : fichier objet créé par l'assembleur.
9. *file.out*: fichier exécutable créé par l'éditeur de liens pour être chargé et exécuté sur le processeur.

5.4 Création d'un nouveau projet sous CCS

Dans cette section, nous illustrons comment créer un projet, en ajoutant les fichiers nécessaires à la construction du projet *sine8_intr.* Accédez à CCS (depuis le bureau).

1. Pour créer le fichier projet *sine8_intr.pjt*. Sélectionnez Projet → Nouveau. Tapez *sine8_intr* pour le nom du projet, comme illustré à la figure 5.1. Ce fichier de projet est enregistré dans *sine8_intr* (le dossier que vous avez créé dans *c:\ti\myprojects).* Le fichier *.pjt* stocke les informations du projet sur les options de génération, les noms de fichiers source et les dépendances.
2. Pour ajouter des fichiers au projet. Sélectionnez Project → Add Files to Project. Regardez dans *sine8_intr*, Fichiers de type C Source Files. Ouvrez les deux fichiers source C *C6xdskinit.c* et *sine8_intr*.c. Ouvrir (pour ajouter au projet) un fichier à la fois ; ou placez le curseur sur l'un de ces fichiers, puis sur l'autre tout en maintenant la touche Maj (The Shift key) enfoncée et appuyez sur Open. Cliquez sur le symbole "+" à gauche de la fenêtre Project Files dans CCS pour développer et vérifier que les deux fichiers source C ont été ajoutés au projet.
3. Sélectionnez Projet → Add Files to Project. Regardez dans *sine8_intr.* Utilisez le menu déroulant pour Fichiers de type : et sélectionnez ASM Source Files. Double-clique sur le fichier source de l'assemblage *vectors_11.asm* pour l'ouvrir/l'ajouter au projet.
4. Répétez l'étape 3 mais sélectionnez Fichiers de type: Fichier de commande (Linker Command File) de l'éditeur de liens et ajoutez le fichier de commande de l'éditeur de liens *C6xdsk.cmd* au projet.
5. Répétez l'étape 3, mais sélectionnez Fichiers de type: Fichiers objet et bibliothèque (Object and Library Files). Regardez dans *c:\ti\c6000\cgtools\lib* et sélectionnez le

fichier de bibliothèque de support d'exécution *rts6701.lib* (qui prend en charge l'architecture C67x/C62x) à ajouter au projet. Cela suppose que vous avez utilisé la destination par défaut de *c:\ti* lors de l'installation de CCS.

6. Vérifiez que le fichier de commande de l'éditeur de liens *(.cmd),* le fichier de projet (*.pjt*), le fichier de bibliothèque *(.lib),* les deux fichiers source C *(.c)* et le fichier d'assemblage *(.asm)* ont été ajoutés au projet. Le fichier GEL *dsk6211_6711.gel* est ajouté automatiquement lorsque vous créez le projet. Il initialise le DSK.
7. Notez qu'il n'y a pas encore de fichiers "inclus". Sélectionnez Projet → Scan All Dependencies. Cela ajoute/inclut les fichiers d'en-tête : *C6xdsk.h, C6xdskinit.h, C6xinterrupts.h* et *C6x.h.* Les trois premiers fichiers d'en-tête ont été copiés (transférés) à partir du disque d'accompagnement, et *C6x.h* est inclus avec CCS.

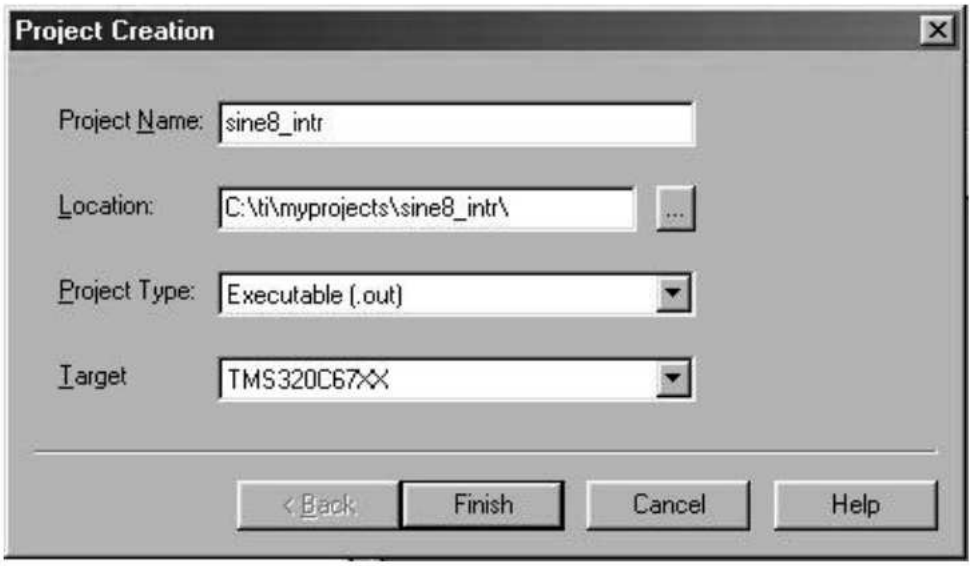


Figure 5.1. Fenêtre d'affichage du projet CCS pour sine8_intr (création du projet).

La fenêtre Fichiers dans CCS devrait ressembler à la figure 5.2. Tous les fichiers (à l'exception du fichier de bibliothèque) de la fenêtre Fichiers de CCS peuvent être affichés en cliquant dessus. Vous ne devez pas ajouter d'en-tête ni inclure de fichiers au projet. Ils sont automatiquement ajoutés au projet lorsque vous sélectionnez : Scan All Dependencies.

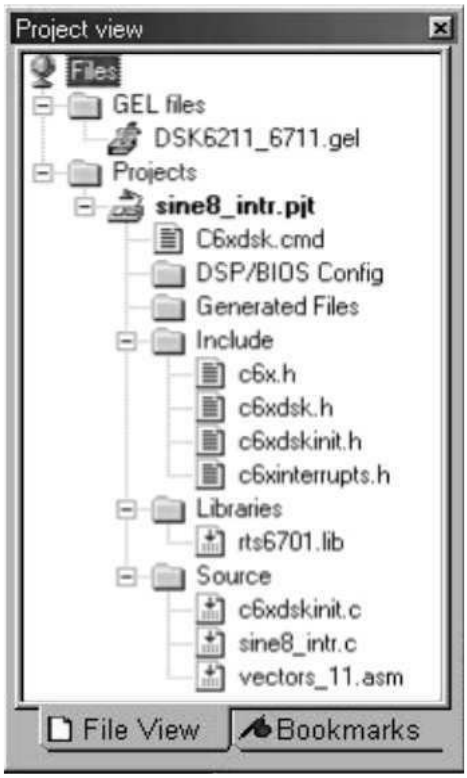


Figure 5.2. Fenêtre d'affichage du projet CCS pour sine8_intr (dossiers de projet).

5.5 Exécution du programme

Le projet *sine8_intr* peut maintenant être construit et exécuté.

1. Générez ce projet en tant que *sine8_intr*. Sélectionnez Project → Rebuild All. Ou appuyez sur la barre d'outils avec les trois flèches vers le bas. Cela compile et assemble tous les fichiers C en utilisant *cl6x* et assemble le fichier d'assemblage *vectors_11.asm* en utilisant *asm6x*. Les fichiers objets résultants sont ensuite liés au fichier de support de la bibliothèque d'exécution *rts6701.lib* à l'aide de *lnk6x*. Cela crée un fichier exécutable *sine8_intr.out* qui peut être chargé dans le processeur C6711 et exécuté. Notez que les commandes de compilation, d'assemblage et de liaison sont exécutées avec l'option Build. Un fichier journal *cc_build_Debug.log* est créé et affiche les fichiers qui sont compilés et assemblés, ainsi que les options de compilateur sélectionnées. Il répertorie également les fonctions de support qui sont utilisées. La figure 5.3 montre plusieurs fenêtres dans CCS pour le projet *sine8_intr*.
2. Sélectionnez File → Load Program afin de charger *sine_intr.out* en cliquant dessus (CCS inclut une option pour charger le programme automatiquement après une construction). Il devrait se trouver dans le dossier du projet *sine8_intr*. Sélectionnez Debug → Run, ou utilisez la barre d'outils avec le "running man.". Connectez un haut-parleur au OUT connecteur (J6) du DSK. Vous devriez entendre une tonalité.

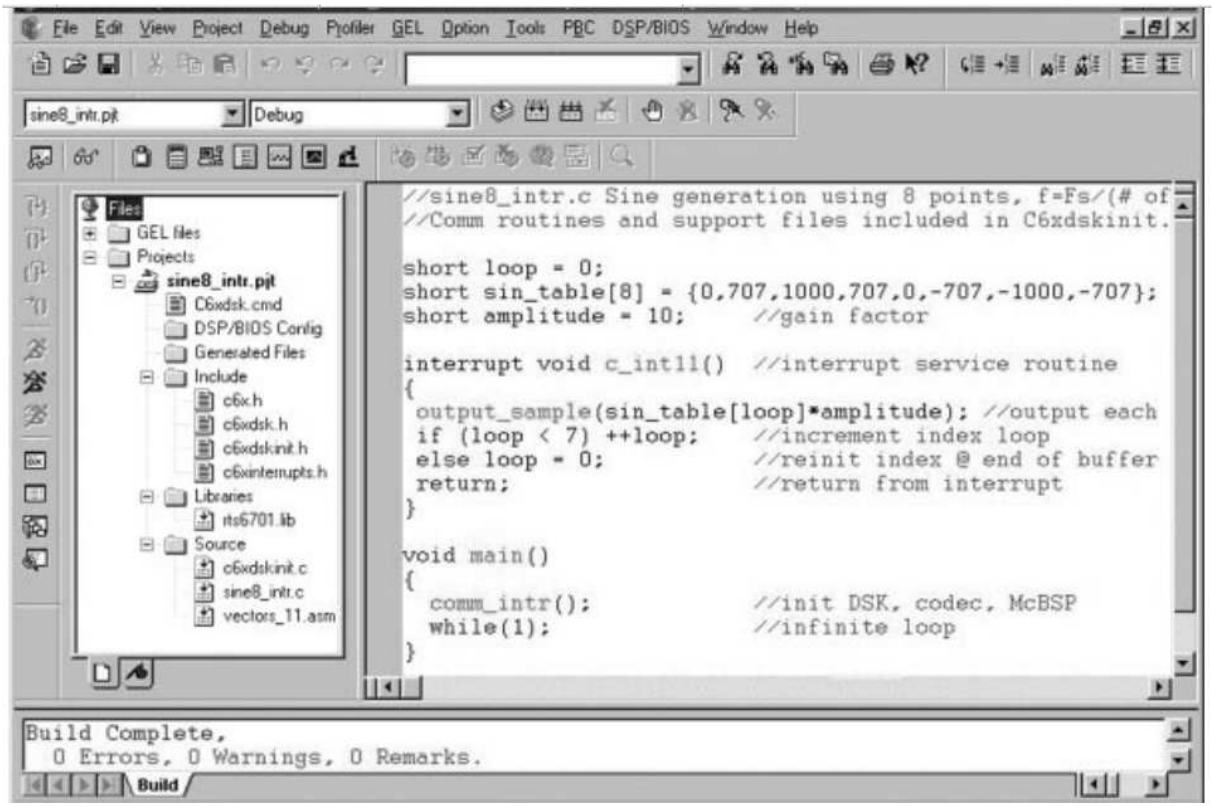


Figure 5.3. Fenêtres CCS pour le projet *sine8_intr.*

5.5.1 Surveillance de la *fenêtre de surveillance* (*Monitoring the Watch Window*)

Vérifiez que le processeur fonctionne toujours. Notez l'indicateur "DSP RUNNING" en bas à gauche de CCS. La fenêtre Watch permet de changer la valeur d'un paramètre ou de surveiller une variable :

1. Sélectionnez la fenêtre View → Quick Watch, qui doit être affichée dans la partie inférieure de CCS. Tapez *amplitude,* puis cliquez sur "Add to Watch". La valeur d'amplitude de 10 définie dans le programme de la figure 5.4 doit apparaître dans la fenêtre Watch.
2. Modifiez l'*amplitude* de 10 à 30.
3. Vérifiez que le volume de la tonalité générée a augmenté (notez que le processeur fonctionnait toujours). L'amplitude de l'onde sinusoïdale est passée d'environ 0,85 V p-p à environ 2,6 V p-p.
4. Modifiez l'*amplitude* à 33 (comme à l'étape 2). Vérifiez une tonalité plus aiguë, ce qui implique que la fréquence de l'onde sinusoïdale a changé simplement en changeant son amplitude. Ce n'est pas le cas. Vous avez dépassé la capacité du codec 16 bits AD535. Étant donné que les valeurs du tableau sont mises à l'échelle par 33, la plage de ces valeurs est désormais comprise entre + et -33 000. La plage des valeurs de sortie est

limitée de -215 à (215 - 1), ou de -32 768 à +32 767, en raison du codec AD535. N'essayez pas d'envoyer plus de 16 bits de données au codec. Le codec intégré utilise un format de complément à 2.

```
//sine8_intr.c Sine generation using 8 points, f=Fs/(# of points)
//Comm routines and support files included in C6xdskinit.c

short loop = 0;
short sin_table[8] = {0,707,1000,707,0,-707,-1000,-707}; //sine values
short amplitude = 10;                          //gain factor

interrupt void c_int11()                       //interrupt service routine
{
        output_sample(sin_table[loop]*amplitude); //output each sine value

        if (loop < 7) ++loop; //increment index loop
        else loop = 0; //reinit index @ end of buffer
        return; //return from interrupt
}
void main()
{
comm_intr();                                   //init DSK, codec, McBSP
while(1);                                              //infinite loop
}
```

Figure 5.4. Programme de génération de sinus utilisant huit points (*sine8_intr.c*)

5.5.2 *Plotting with CCS*

Le tampon de sortie est mis à jour en continu tous les 256 points (vous pouvez facilement modifier la taille du tampon). Utilisez CCS pour tracer les données de sortie actuelles stockées dans le tampon *out_buffer*.

1. Sélectionnez View → Graph → Time/Frequency.
2. Modifiez la **boîte de dialogue des propriétés du graphique** (**the Graph Property Dialog**) afin que les options de la figure 5.5a soient sélectionnées pour un tracé dans le domaine temporel (utilisez le menu déroulant le cas échéant). L'adresse de début du

tampon de sortie est *out_buffer*. Les autres options peuvent être laissées par défaut. La figure 5.6 montre un tracé dans le domaine temporel du signal sinusoïdal.

3. La figure 5.5B montre l'affichage des propriétés du graphique de CCS (**Graph Property Display**) pour un tracé dans le domaine fréquentiel. Choisissez un ordre FFT de sorte que 2eme corresponde à la taille de la trame (frame size). Appuyez sur OK et vérifiez que le tracé de la magnitude FFT est comme indiqué sur la figure 5.6. Le pic à 1000 Hz représente la fréquence de la sinusoïde générée.

Remarque

Pour modifier la taille de l'écran, cliquez avec le bouton droit sur « Build window » et désélectionnez « Allow Docking ». Vous pouvez alors obtenir de nombreuses fenêtres différentes dans CCS.

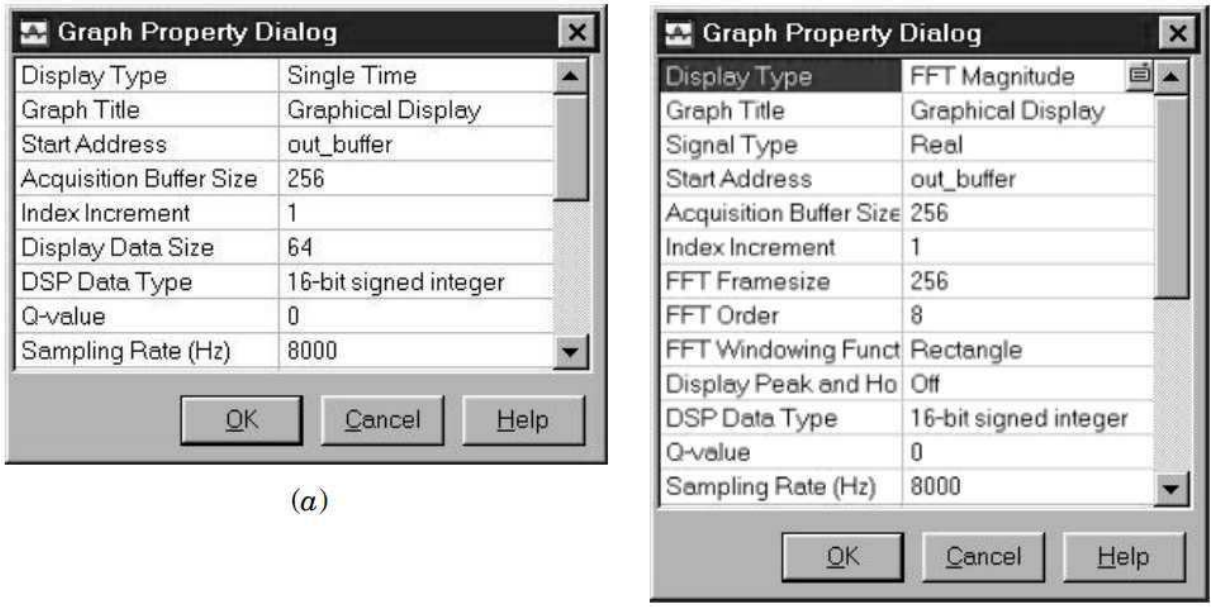


Figure 5.5. Boîte de dialogue des propriétés du graphique CCS pour sine8_buf : (a) pour le tracé dans le domaine temporel ; (b) pour le tracé du domaine fréquentiel.

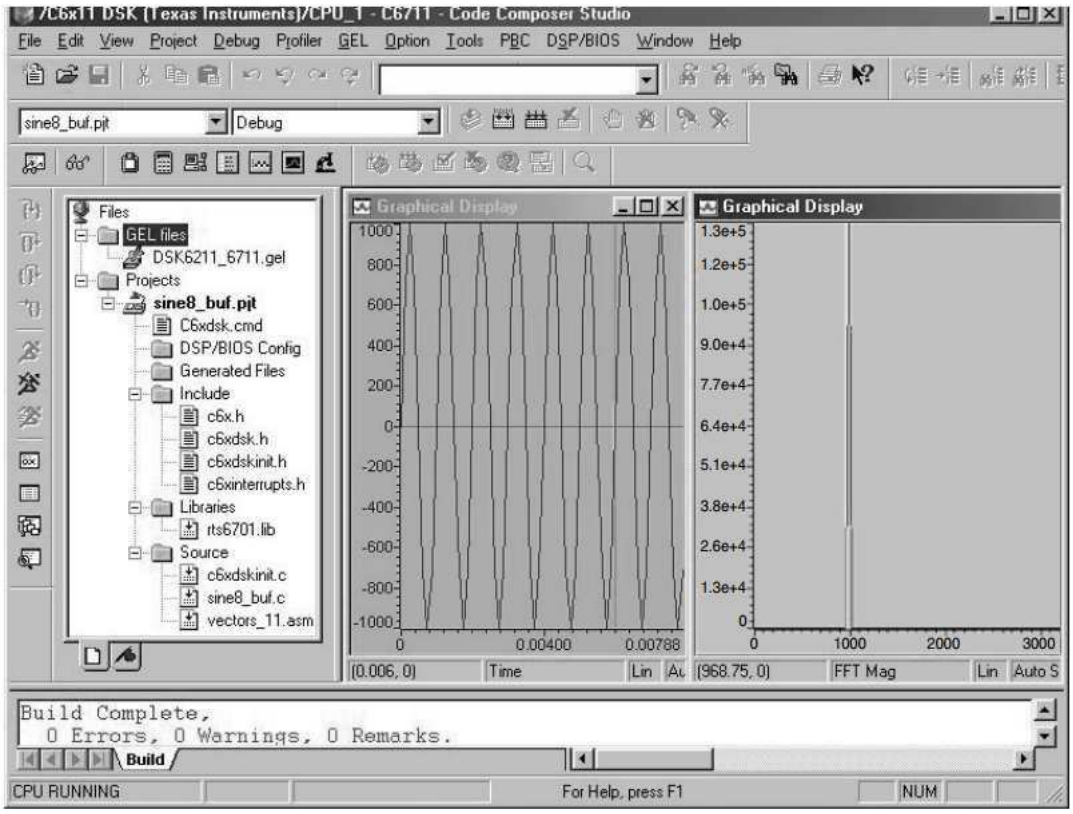


Figure 5.6. Fenêtres CCS avec tracés dans les domaines temporel et fréquentiel d'une onde sinusoïdale de 1 kHz.

5.5.3 *Implementing a Variable Watch*

1. Sélectionnez Project → Options:
   Compiler: *–gs*
   Linker: *–c –o dotp4.out*
2. Tout reconstruire en sélectionnant la barre d'outils avec les trois flèches (ou sélectionnez Debug → Build).
3. Sélectionnez View→ Quick Watch. Tapez *sum* pour regarder le variable *sum*, et cliquez sur "Add to Watch." Un message "identifier not found" associé à *sum* s'affiche (comme valeur) car cette variable locale « n'existe pas encore » puisque nous sommes toujours dans la fonction *main*.
4. Définissez un point d'arrêt (a breakpoint) sur la ligne de code [*sum += a[i] * b[i];*] en plaçant le curseur de la souris (en cliquant) sur cette ligne, puis cliquez avec le bouton droit et sélectionnez Basculer le point d'arrêt (**Toggle breakpoint**). Un cercle à gauche de cette ligne de code devrait apparaître.

5. Sélectionnez Debug → Run (ou utilisez la barre d'outils "running man "). Le programme s'exécute jusqu'à la ligne de code avec le point d'arrêt défini. Une flèche jaune pointera également vers cette ligne de code.
6. Une seule étape à l'aide de F8 (ou utilisez la barre d'outils). Répétez ou continuez en une seule étape et observer/regarder la valeur de la somme variable à 0, 4, 16, 40. Sélectionnez Debug → Run, et vérifiez que la valeur résultante de la somme est imprimée comme ***[sum = 40 (decimal)]***
7. Notez l'instruction *printf* dans le programme C *dotp4.c* pour imprimer le résultat. Une telle instruction doit être évitée, car son exécution peut prendre 3000 cycles.

5.6 Scriptes GEL (*General Extension Language*) du CCS

Le langage d'extension général (GEL) est un langage d'interprétation similaire à (un sous-ensemble de) C. Il vous permet de modifier une variable telle que l'amplitude, en glissant sur différentes valeurs pendant que le processeur est en cours d'exécution. Toutes les variables doivent d'abord être définies dans votre programme.

1. Sélectionnez File → Load GEL et ouvrez le fichier *amplitude.gel* que vous avez copié (depuis le disque fourni) dans le dossier *sine8_intr*. Double-cliquez sur le fichier *amplitude.gel* pour le visualiser dans CCS. Il devrait être affiché dans la fenêtre Fichiers (Files window). Ce fichier est illustré à la figure 5.7. En créant l'amplitude de la fonction de curseur illustrée à la figure 5.7, vous pouvez commencer avec une valeur initiale de 10 (première valeur) pour l'amplitude variable définie dans le programme C, jusqu'à une valeur de 35 (deuxième valeur), incrémentée de 5 (troisième valeur).
2. Sélectionnez GEL → Sine Amplitude → Amplitude. Cela devrait faire apparaître la fenêtre du curseur illustrée à la figure 5.8, avec la valeur minimale de 10 définie pour l'amplitude.
3. Appuyez sur la touche fléchée vers le haut pour augmenter la valeur d'amplitude de 10 à 15, comme affiché dans la fenêtre Curseur. Vérifiez que le volume de l'onde sinusoïdale générée a augmenté. Appuyez à nouveau sur la touche fléchée vers le haut pour continuer à augmenter le curseur, en incrémentant de 5 à 30. L'amplitude de l'onde sinusoïdale doit être environ 2,6 V p-p avec une valeur d'amplitude fixée à 30. Utilisez maintenant la souris pour cliquer sur la fenêtre du curseur (Slider window) et augmentez lentement la position du curseur jusqu'à 31, puis 32, et vérifiez que la fréquence générée est toujours de 1 kHz. Augmentez le curseur à 33 et vérifiez que vous ne générez plus une onde sinusoïdale de 1 kHz (plutôt un signal avec deux tonalités : 1 et 3kHz). Les

valeurs du tableau, mises à l'échelle par amplitude, sont maintenant entre + et -33 000 (au-delà de la plage acceptable par le codec).

Deux curseurs peuvent facilement être utilisés, l'un pour modifier l'amplitude et l'autre pour changer la fréquence. Une fréquence différente peut être générée en changeant la boucle index dans le programme C. Lorsque vous quittez CCS après avoir créé un projet, toutes les modifications apportées au projet peut être enregistré. Vous pouvez ensuite revenir au projet avec le statut vous l'avez laissé avant.

```
/*Amplitude.gel Create slider and vary amplitude of sinewave*/
menuitem "Sine Amplitude"
slider Amplitude(10,35,5,1,amplitudeparameter) /*start at 10,up to 35*/
    {

        amplitude = amplitudeparameter; /*vary amplit of sine*/
    }
```

Figure 5.7. Fichier GEL pour "glisser" à travers différentes valeurs d'amplitude dans le programme de génération de sinus (*amplitude.gel*).

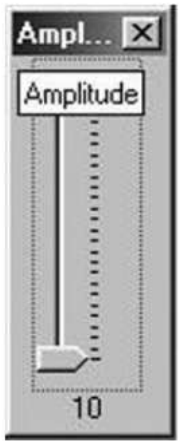


Figure 5.8. Fenêtre coulissante CCS pour faire varier l'amplitude d'une onde sinusoïdale.

5.7 Utilisation des *switches* DIP (*Interrupt-Driven Program*)

La fonction *comm_intr* dans main dans le programme source C se trouve dans le fichier de communication *c6xdskinit.c*, dont une liste partielle est illustrée à la figure 5.9. Le DSK est initialisé, puis l'interruption de transmission INT11 est configurée et activée.

Avec un programme piloté par interruption, une interruption est sélectionnée (nous avons sélectionné INT11). Le bit d'interruption non masquable (nonmaskable) doit être activé ainsi que le bit d'activation d'interruption globale (GIE). Les fonctions de support appropriées pour les interruptions se trouvent dans le fichier de support *C6xdskinterrupts.h* et sont appelées à partir de la fonction *comm_intr* dans le fichier *C6xdskinit.c*.

```
//C6xdskinit.c Partial listing. Init DSK,AD535,McBSP
#include <c6x.h>
#include "c6xdsk.h"
#include "c6xdskinit.h"
#include "c6xinterrupts.h"
void mcbsp0_write(int out_data)                 //function for writing
{
        int temp;
        if (polling)                             //bypass if interrupt-driven
        {
                temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
                while ( temp == 0)

                        temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
        }
        *(unsigned volatile int *)McBSP0_DXR = out_data;

}
int mcbsp0_read()                               //function for reading

        int temp;
        if (polling)                            //bypass if interrupt-driven
        {
                temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
                while ( temp == 0)
                        temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
        }
        temp = *(unsigned volatile int *)McBSP0_DRR;
        return temp;
}
void comm_poll()                                //communication with polling
{
        polling = 1;                            //setup for polling
        c6x_dsk_init();                         //call init DSK function
}
void comm_intr()                                //communication with interrupt
{
        polling = 0;                            //if interrupt-driven
        c6x_dsk_init();                         //call init DSK function
```

```
        config_Interrupt_Selector(11,XINT0);  //using transmit interrupt INT11
        enableSpecificINT(11);                //for specific interrupt
        enableNMI();                          //enable NMI
        enableGlobalINT();                    //enable GIE global interrupt
        mcbsp0_write(0);                      //write to SP0
}
void output_sample(int out_data)              //added function for output
{
        mcbsp0_write(out_data & 0xfffe);      //mask out LSB
}
int input_sample()                            //added function for input
{
        return mcbsp0_read();                 //read from McBSP0
}
```

Figure 5.9. Liste partielle du programme d'aide à la communication (communication support program) (*C6xdskinit.c*).

Chapitre 6 : Algorithmes de traitement du signal sur DSP

6.1 L'adéquation algorithme-architecture.

Les processeurs de signaux numériques tels que la famille de processeurs TMS320C6x (C6x) sont comme des microprocesseurs rapides à usage spécial avec un type spécialisé d'architecture et un jeu d'instructions approprié pour le traitement du signal. La notation C6x est utilisée pour désigner un membre de la famille de processeurs de signaux numériques TMS320C6000 de Texas Instruments (TI). Le C6x est considéré comme le processeur le plus puissant de TI.
Le TMS320C6711 (C6711) est basé sur l'architecture de mot d'instruction très long (VLIW), qui est très bien adaptée aux algorithmes numériquement intensifs. La mémoire de programme interne est structurée de manière à ce qu'un total de huit instructions puissent être récupérées à chaque cycle. Par exemple, avec une fréquence d'horloge de 150 MHz, le C6711 est capable de récupérer huit instructions 32 bits tous les 1/(150 MHz) ou 6,66 ns. Les caractéristiques du C6711 incluent 72 Ko de mémoire interne, huit unités fonctionnelles ou d'exécution composées de six ALU et de deux unités multiplicatrices, un bus d'adresse 32 bits pour adresser 4 Go (Giga octets) et deux ensembles de registres à usage général 32 bits. Les C67xx (comme les C6701 et C6711) appartiennent à la famille des processeurs à virgule flottante C6x ; tandis que les C62xx et C64xx appartiennent à la famille des processeurs à virgule fixe C6x. Le C6711 est capable de traitement à la fois en virgule fixe et en virgule flottante.
Le système de base consiste en un convertisseur analogique-numérique (ADC) pour capturer un signal d'entrée. La représentation numérique résultante du signal capturé est ensuite traitée par un processeur de signal numérique tel que le C6x, puis émise via un convertisseur numérique-analogique (DAC). Le système de base comprend également un filtre d'entrée spécial pour l'anticrénelage afin d'éliminer les signaux erronés, et un filtre de sortie pour lisser ou reconstruire le signal de sortie traité.

6.2 Filtrage RIF et RII

Le filtrage est l'une des opérations de traitement du signal les plus utiles. Des processeurs de signaux numériques sont maintenant disponibles pour implémenter des filtres numériques en temps réel. Le jeu d'instructions et l'architecture du TMS320C6x le rendent bien adapté à de telles opérations de filtrage. Un filtre analogique fonctionne sur des signaux continus et est généralement réalisé avec des composants discrets tels que des amplificateurs opérationnels, des résistances et des condensateurs. Cependant, un filtre numérique, tel qu'un filtre à réponse impulsionnelle finie (FIR), fonctionne sur des signaux à temps discret et peut être mis en œuvre avec un processeur de signal numérique tel que le TMS320C6x. Cela implique l'utilisation

d'une ADC pour capturer un signal d'entrée externe, le traitement des échantillons d'entrée et l'envoi de la sortie résultante via un DAC.

Au cours des dernières années, le coût des processeurs de signaux numériques a été considérablement réduit, ce qui ajoute aux nombreux avantages des filtres numériques par rapport à leurs homologues analogiques. Ceux-ci incluent une fiabilité, une précision et une sensibilité moindres à la température et au vieillissement. Des caractéristiques d'amplitude et de phase strictes peuvent être réalisées avec un filtre numérique. Les caractéristiques du filtre telles que la fréquence centrale, la bande passante et le type de filtre peuvent être facilement modifiées. Deux types de filtre sont utilisés dans DSP :

1. Filtrage RIF (Finite Impulse Response (FIR) filter)
2. Filtrage RII (Infinite Impulse Response (IIR) filter)

6.2.1 Filtrage RIF

L'équation (6.1) peut être utilisée pour définir le filtre FIR. Il montre qu'un filtre FIR peut être mis en œuvre en connaissant l'entrée x(n) à l'instant n et les entrées retardées x(n - k). Il est non récursif et aucune rétroaction ou sortie passée n'est requise. Les autres noms utilisés pour les filtres FIR sont les filtres transversaux et à retard de prise.

$$y(n) = \sum_{k=0}^{N-1} h(k)\, x(n-k) \tag{6.1}$$

La transformée en z de (6.1) avec des conditions initiales nulles est donnée par l'équation suivant :

$$Y(z) = h(0)\, X(z) + h(1)\, z^{-1} X(z) + \cdots + h(N-1)\, z^{-(N-1)} X(z) \tag{6.2}$$

L'équation (6.1) représente une convolution dans le temps entre les coefficients et les échantillons d'entrée, qui équivaut à une multiplication dans le domaine fréquentiel, (voir l'équation (6.3)).

$$Y(z) = X(z) * H(z) \tag{6.3}$$

où $H(z) = ZT[h(k)]$ est la fonction de transfert est donnée par l'équation (6.4).

$$H(z) = \sum_{k=0}^{N-1} h(k)\ z^{-k} = h(0) + h(1)\, z^{-1} + \cdots + h(N-1)\, z^{-(N-1)}$$

$$H(z) = \frac{h(0)\, z^{(N-1)} + h(1)\, z^{N-2} + \cdots + h(N-1)}{z^{N-1}} \tag{6.4}$$

L'équation (6.4) montre qu'il y a (N – 1) pôles, tous situés à l'origine. Par conséquent, ce filtre FIR est intrinsèquement stable, avec ses pôles situés uniquement à l'intérieur du cercle unité.

Nous décrivons généralement un filtre FIR comme un filtre « sans pôles ». La figure 6.1 montre une structure de filtre FIR représentant (6.1) et (6.3).

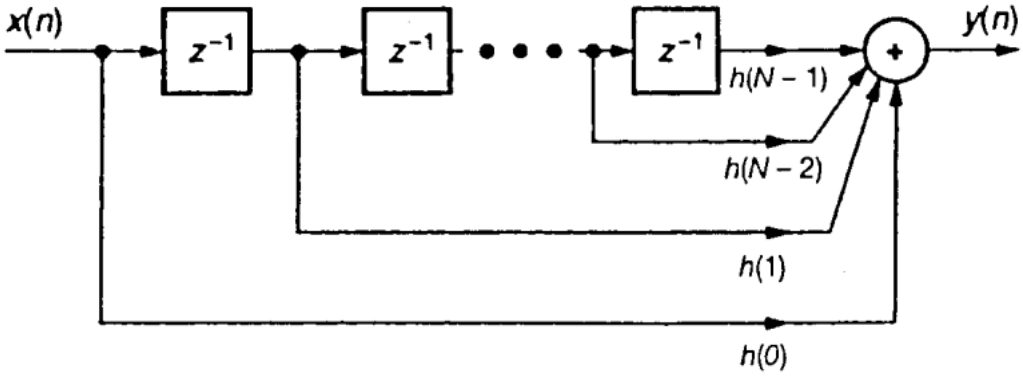


Figure 6.1 Structure du filtre FIR montrant les retards

Une caractéristique très utile d'un filtre FIR est qu'il peut garantir une phase linéaire. La fonction de phase linéaire peut être très utile dans des applications telles que l'analyse de la parole, où la distorsion de phase peut être très critique. Par exemple, avec une phase linéaire, toutes les composantes sinusoïdales d'entrée sont retardées de la même quantité. Sinon, une distorsion harmonique peut se produire. La transformée de Fourier d'un échantillon d'entrée retardé $x(n-k)$ est $e^{-j\omega kT}\,X(j\omega)$, produisant une phase de $\theta = -\omega kT$, qui est une fonction linéaire en termes de w. Notez que la fonction de retard de groupe, définie comme la dérivée de la phase, est une constante, ou $\frac{d\theta}{d\omega} = -kT$

Remarque

Un certain nombre d'outils sont disponibles pour concevoir et mettre en œuvre en quelques minutes un filtre FIR en temps réel à l'aide du DSK basé sur TMS320C6x. La conception du filtre consiste en l'approximation d'une fonction de transfert avec un ensemble résultant de coefficients.

6.2.2 Filtrage RII

Considérons une équation générale d'entrée-sortie de la forme suivante

$$y(i) = \sum_{k=0}^{N} a_k\, x(i-k) - \sum_{k=1}^{M} b_k\, y(i-k) \qquad (6.5)$$

$$y(i) = a_0\, x(i) + a_1\, x(i-1) + \cdots + a_N\, x(i-N) - b_1\, y(i-1) - b_2\, y(i-2) - \cdots - b_M\, y(i-M) \qquad (6.6)$$

Ce type d'équation récursive représente un filtre à réponse impulsionnelle infinie (IIR). La sortie dépend des entrées ainsi que des sorties passées (avec rétroaction). La sortie à l'instant n dépend non seulement de l'entrée actuelle à l'instant n et des entrées passées $x(n - 1), x(n - 2), \ldots, x(n - N)$, mais aussi des sorties passées $y(n - 1), y(n - 2), \ldots, y(n - M)$.

Si nous supposons que toutes les conditions initiales sont nulles dans (6.6), la transformée en z de (6.6) devient donné par l'équation (6.7).

$$Y(z) = a_0\, X(z) + a_1\, z^{-1}X(z) + \cdots + a_N\, z^{-N}X(z) - b_1\, z^{-1}Y(z) - b_2\, Y(i-2) - \cdots - b_M\, z^{-M}Y(z) \quad (6.7)$$

Soit N = M dans (6.7) ; alors la fonction de transfert H(z) est donnée par l'équation (6.8).

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1\, z^{-1} + \cdots + a_N z^{-N}}{1 + b_1\, z^{-1} + \cdots + b_N z^{-N}} = \frac{N(z)}{D(z)} \quad (6.8)$$

où N(z) et D(z) représentent respectivement le polynôme numérateur et dénominateur. Multipliant et divisant l'équation (6.8) par z^N, H(z) devient donné par l'équation (6.9).

$$H(z) = \frac{a_0\, z^N + a_1\, z^{N-1} + \cdots + a_N}{z^N + b_1\, z^{N-1} + \cdots + b_N} = C \prod_{i=1}^{N} \frac{z - z_i}{z - p_i} \quad (6.9)$$

Cette équation est une fonction de transfert à N zéros et N pôles. Si tous les coefficients *bj* dans (6.9) sont nuls, cette fonction de transfert se réduit à la fonction de transfert à N pôles à l'origine dans le plan z représentant le filtre FIR. Pour qu'un système soit stable, tous les pôles doivent résider à l'intérieur du cercle unité ; Ainsi, pour qu'un filtre IIR soit stable, la magnitude de chacun de ses pôles doit être inférieure à 1, ou :

1. Si |Pi| < 1, alors h(n) → 0, comme n → ∞, donnant un système stable.
2. Si |Pi| > 1, alors h(n) → ∞, comme n → ∞, donnant un système instable.
3. Si |Pi| = 1, le système est légèrement stable, produisant une réponse oscillatoire. De plus, les pôles d'ordre multiple sur le cercle unitaire donnent un système instable. Remarquons à nouveau qu'avec tous les coefficients bj = 0, le système se réduit à un filtre FIR non récursif et stable.

Le filtre donné par l'équation (6.6) peut être réalisé en utilisant la structure de forme directe I illustrée à la figure 6.2. Il y a un sommateur implicite (non illustré) dans la figure 6.2. Pour un filtre d'ordre N, sa structure comporte 2N éléments de retard, représentés par z^{-1}. Par exemple, un filtre de second ordre avec N = 2 aura quatre éléments de retard.

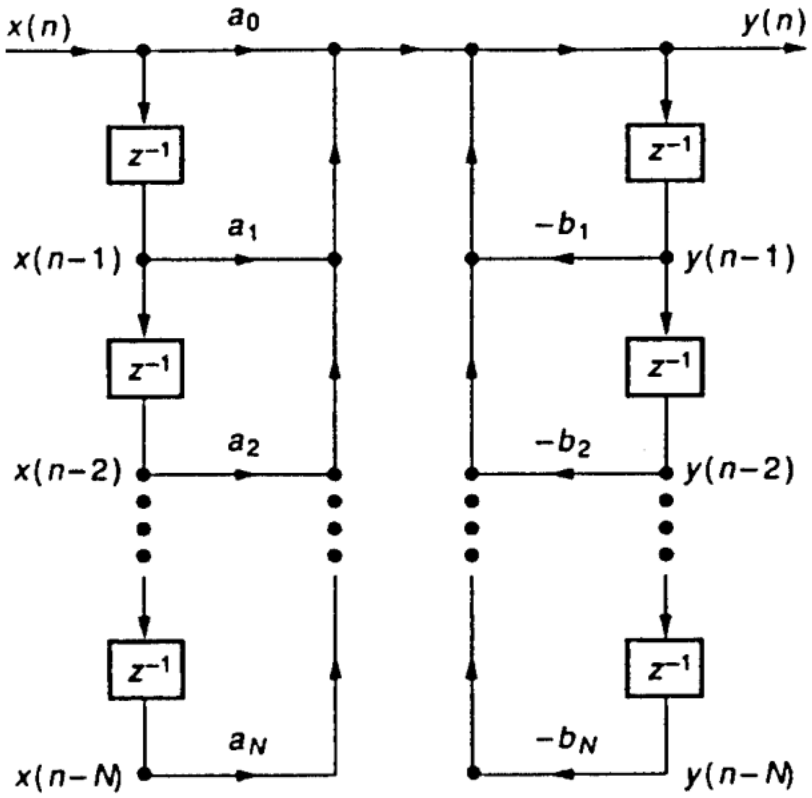


Figure 6.2 Forme directe I Structure du filtre IIR.

6.3 Contraintes temps-réel

Les processeurs DSP sont principalement concernés par le traitement du signal en **temps réel**. Le traitement en temps réel signifie que le traitement doit suivre le rythme d'un événement externe ; tandis que le traitement non en temps réel n'a pas une telle contrainte de temps. L'événement externe à suivre est généralement l'entrée analogique. Alors que les systèmes analogiques avec des composants électroniques discrets tels que des résistances peuvent être plus sensibles aux changements de température, les systèmes basés sur DSP sont moins affectés par les conditions environnementales telles que la température. Les processeurs DSP bénéficient des avantages des microprocesseurs. Ils sont faciles à utiliser, flexibles et économiques.

Diverses technologies ont été utilisées pour le traitement en temps réel, de la fibre optique pour la très haute fréquence aux processeurs DSP très adaptés à la gamme audio-fréquence. Les applications courantes utilisant ces processeurs ont été pour des fréquences de 0 à 20 kHz. La parole peut être échantillonnée à 8 kHz (vitesse d'acquisition des échantillons), ce qui implique que chaque valeur échantillonnée est acquise à une fréquence de 1/(8 kHz) ou 0,125 ms. Un taux d'échantillonnage couramment utilisé d'un disque compact est de 44,1 kHz. Des cartes

basées sur A/N dans la plage de fréquences d'échantillonnage en mégahertz sont actuellement disponibles.

6.4 Exemple Implémentation FIR avec un programme C appelant la fonction ASM à l'aide d'un tampon circulaire (*FIRcirc*)

Le programme *FIRcirc.c* (voir la figure 6.3) appelle la fonction ASM *FIRcircfunc.asm* (voir la figure 6.4), qui implémente un filtre FIR utilisant un tampon circulaire (Circular Buffer). Les coefficients du fichier *bp1750.cof* ont été conçus avec MATLAB à l'aide de la fenêtre Kaiser et représentent un filtre passe-bande FIR à 128 coefficients avec une fréquence centrale de 1750 Hz. La figure 6.5 affiche les caractéristiques de ce filtre, obtenues à partir du concepteur de filtres SPTOOL de MATLAB.

```
//FIRcirc.c C program calling ASM function using circular buffer

#include "bp1750.cof"                  //BP at 1750 Hz coeff file
int yn = 0;                            //init filter's output

interrupt void c_int11()               //ISR
{
short sample_data;

sample_data = input_sample();          //newest input sample data
yn = fircircfunc(sample_data,h,N);     //ASM func passing to A4,B4,A6
output_sample(yn >> 15);               //filter's output
return;                                //return to calling function
}

void main()
{

comm_intr();                           //init DSK, codec, McBSP
while(1);                              //infinite loop
}
```

Figure 6.3 Programme C appelant une fonction ASM à l'aide d'un tampon circulaire (FIRcirc.c)

Au lieu de déplacer les données pour mettre à jour les échantillons de retard, un pointeur est utilisé. Les 16 LSB du registre de mode d'adresse (AMR) sont définis avec une valeur de [0x0040 = 0000 0000 0100 0000]. Cela sélectionne le mode A7 comme registre de pointeur de tampon circulaire. Les 16 MSB d'AMR sont définis avec N = 0x0007 pour sélectionner le bloc BK0 en tant que tampon circulaire.

```
;FIRcircfunc.asm ASM function called from C using circular
addressing
;A4=newest sample, B4=coefficient address, A6=filter order
;Delay samples organized: x[n-(N-1)]...x[n]; coeff as h(0)...h[N-1]

           .def  _fircircfunc
           .def  last_addr
           .def  delays
           .sect "circdata"      ;circular data section
           .align 256       ;align delay buffer 256-byte boundary
Delays     .space 256            ;init 256-byte buffer with 0's
last_addr  .int  last_addr-1     ;point to bottom of delays buffer
           .text                 ;code section
_fircircfunc:                    ;FIR function using circ addr
           MV    A6,A1           ;setup loop count
           MPY   A6,2,A6         ;since dly buffer data as byte
           ZERO  A8              ;init A8 for accumulation

           ADD   A6,B4,B4        ;since coeff buffer data as bytes
           SUB   B4,1,B4         ;B4=bottom coeff array h[N-1]

           MVKL  0x00070040,B6   ;select A7 as pointer and BK0
           MVKH  0x00070040,B6   ;BK0 for 256 bytes (128 shorts)

           MVC   B6,AMR          ;set address mode register AMR

           MVK   last_addr,A9    ;A9=last circ addr(lower 16 bits)
           MVKH  last_addr,A9    ;last circ addr (higher 16 bits)
           LDW   *A9,A7          ;A7=last circ addr
           NOP   4
           STH   A4,*A7++        ;newest sample-->last address

loop:                            ;begin FIR loop
           LDH   *A7++,A2        ;A2=x[n-(N-1)+i] i=0,1,...,N-1
    ||     LDH   *B4--,B2        ;B2=h[N-1-i] i=0,1,...,N-1
           SUB   A1,1,A1         ;decrement count

     [A1]  B     loop            ;branch to loop if count # 0
           NOP   2
           MPY   A2,B2,A6        ;A6=x[n-(N-1)+i]*h[N-1+i]
           NOP
           ADD   A6,A8,A8        ;accumulate in A8

           STW   A7,*A9          ;store last circ addr to last_addr
           B     B3              ;return addr to calling routine
           MV    A8,A4           ;result returned in A4
           NOP   4
```

Figure 6.4 Fonction ASM appelée C utilisant un tampon circulaire pour mettre à jour les échantillons (`FIRcircfunc.asm`)

La taille de la mémoire tampon est $2^{N+1} = 256$. Une mémoire tampon circulaire est utilisée dans cet exemple uniquement pour les échantillons de retard. Il est également possible d'utiliser un second tampon circulaire pour les coefficients. Par exemple, en utilisant `[0x0140 = 0000 0001 0100 0000]` sélectionnerait deux pointeurs, B4 et A7.

Dans un programme C, un code assembleur en ligne peut être utilisé avec l'instruction *asm.* Par exemple : `asm(" MVK 0x0040,B6").`

Notez l'espace vide après la première citation afin que l'instruction ne commence pas dans la colonne 1. Le mode d'adressage circulaire élimine le déplacement des données pour mettre à jour les échantillons de retard, car le pointeur peut être déplacé pour obtenir le même résultat plus rapidement.

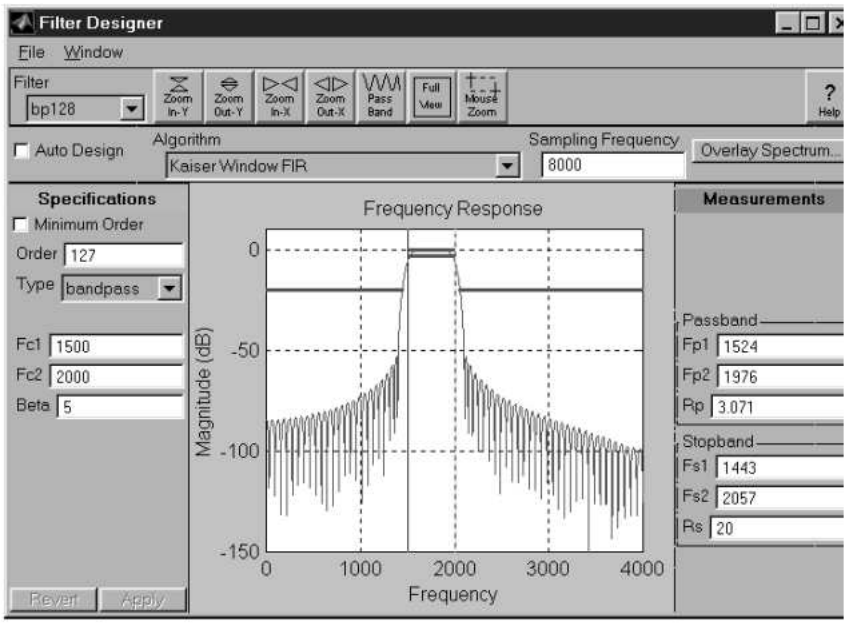


Figure 6.5 Caractéristiques de fréquence d'un filtre passe-bande FIR à 128 coefficients centré à 1750 Hz à l'aide du concepteur de filtres SPTOOL de MATLAB.