Université Mohamed Kheider Biskra
Faculté des Sciences et de la Technologie
Département Electronique et Automatique

DSP et FPGA

Spécialité : Master Réseaux de Télécommunications

Auteur : Dhiabi Fathi

4 avril 2026

Table des matières

Table des figures

CHAPITRE 1

Revue sur le microprocesseur

1.1 Introduction

Conçu pour remplacer des milliers de circuits logiques discrets, le microprocesseur intègre toutes leurs fonctionnalités dans un espace réduit. En tant que cur d'un micro-ordinateur, il fonctionne comme une machine à états séquentiels, exécutant les tâches de manière ordonnée et précise.

Le fonctionnement d'un ordinateur repose sur l'interaction de trois blocs fondamentaux, illustrés dans le schéma ci-dessous :

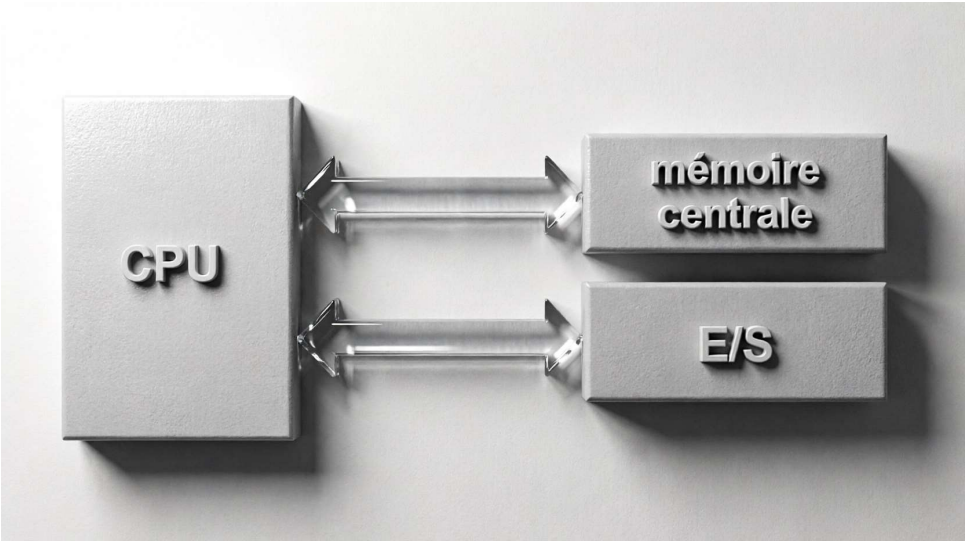


Figure 1.1 – Schéma fonctionnel simplifié d'un ordinateur

1.1.1 Unité centrale (CPU)

Véritable chef d'orchestre du système, l'unité centrale coordonne et supervise l'exécution de toutes les opérations internes et externes.

1.1.2 Périphériques d'entrée/sortie (E/S)

Les interfaces d'entrée/sortie assurent la communication entre la machine et son environnement :

— **Périphériques d'entrée** : capturent et transmettent les informations vers le processeur (ex. : clavier, lecteur optique).

— **Périphériques de sortie** : restituent les résultats ou les données traitées à l'utilisateur ou à un autre système (ex. : écran, imprimante).

1.1.3 Mémoire centrale

La mémoire centrale sert d'espace de travail temporaire et permanent : elle héberge simultanément le programme en cours d'exécution et les données nécessaires à son traitement. Physiquement, elle combine une RAM (mémoire volatile, modifiable) et une ROM (mémoire morte, conservant les données critiques hors tension).

1.2 Architecture du microprocesseur

En tant que CPU, le microprocesseur suit un cycle rigoureux en trois étapes : il récupère une instruction depuis la mémoire, l'analyse pour en comprendre la nature, puis la réalise en effectuant des calculs ou des transferts. Les données et les résultats transitent continuellement entre la mémoire et le processeur.

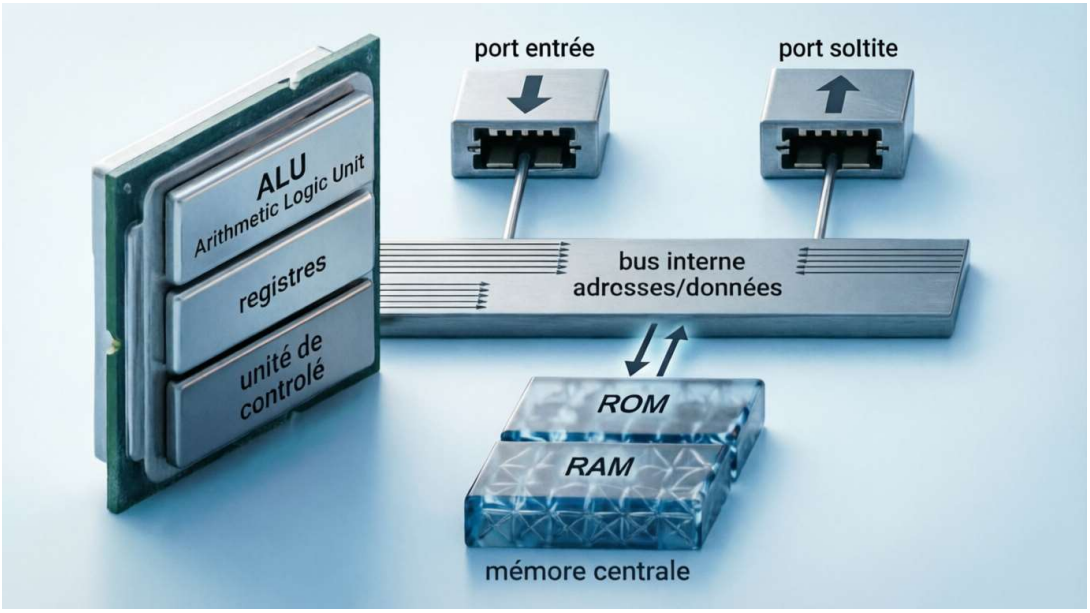


Figure 1.2 – Architecture interne typique d'un microprocesseur

Pour accomplir ces fonctions, le processeur s'organise en sous-unités spécialisées, reliées par des voies de communication appelées **bus**. Chaque bus transporte des signaux binaires

en parallèle (une ligne par bit) et joue un rôle précis :

- **Bus d'adresses** : unidirectionnel, il diffuse l'adresse cible depuis la CPU vers la mémoire ou les périphériques. Seul le destinataire désigné y répond.
- **Bus de données** : bidirectionnel, il assure le transfert des informations (données, instructions, résultats) dans les deux sens.
- **Bus de contrôle** : il véhicule les signaux de synchronisation et de commande qui orchestrent les échanges entre les composants.

1.2.1 Unité arithmétique et logique (ALU)

Cur de calcul du processeur, l'ALU est un circuit combinatoire dédié aux opérations mathématiques (addition, soustraction, etc.) et logiques (ET, OU, NON, comparaisons, décalages).

1.2.2 Registres internes

Pour accélérer les traitements, le microprocesseur intègre des mémoires ultra-rapides appelées **registres**. Elles servent de zones de stockage temporaire pour les données en cours de manipulation. On distingue :

- **Registres à usage général** : stockent temporairement des valeurs quelconques.
- **Accumulateur (ACC)** : registre dédié aux calculs ; il reçoit un opérande, participe à l'opération, et conserve le résultat.
- **Registre d'instructions (IR)** : accueille l'instruction courante pour la décoder et générer les signaux de commande.
- **Compteur ordinal (PC)** : pointe sur l'adresse de la prochaine instruction. Il s'incrémente automatiquement, sauf en cas de saut ou d'appel de sous-programme.
- **Pointeur de pile (SP)** : gère l'accès à la pile (structure LIFO), notamment pour sauvegarder les adresses de retour lors d'appels de fonctions.
- **Registre d'état (PSW/Flags)** : ensemble de bits indicateurs reflétant le résultat des dernières opérations :
  - `Z` (Zéro) : activé si le résultat est nul.
  - `S` (Signe) : indique si le bit de poids fort (MSB) vaut 1.
  - `P` (Parité) : signalé si le nombre de bits à 1 est pair.
  - `C` (Carry/Retenue) : activé en cas de dépassement de capacité.
- **Registres dédiés** : utilisés en interne par le matériel, ils restent invisibles pour le programmeur.

1.2.3 Décodeur et unité de contrôle

Le **décodeur d'instructions** analyse le contenu du registre d'instructions et le traduit en signaux exploitables. L'**unité de contrôle** exploite ces signaux, ainsi que l'état des flags, pour orchestrer le flux de données : elle active les transferts entre registres, commande l'ALU, et synchronise le tout via l'horloge principale qui cadence chaque cycle d'exécution.

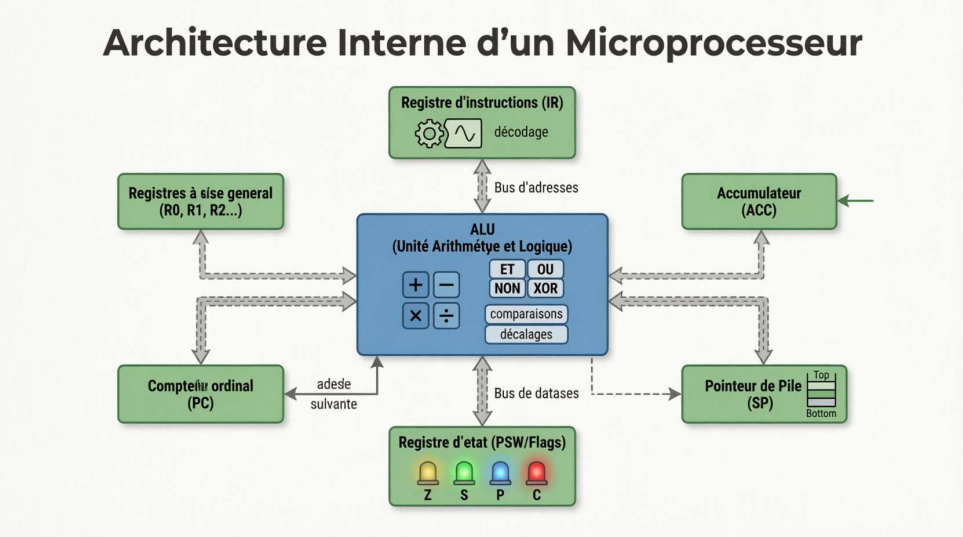


Figure 1.3 – Architecture interne d'un microprocesseur avec ses composants clés

1.3 Langage machine

Le matériel ne comprend que le binaire. En conséquence, toute instruction est encodée sous forme de suite de bits (souvent représentée en hexadécimal pour la lisibilité). Ce format brut constitue le **langage machine**. Une instruction s'y présente sous la forme :

— `OPCODE`

— `OPCODE OPERANDE(S)`

L'*opcode* spécifie l'action à exécuter, tandis que les *opérandes* désignent les cibles : données immédiates, registres internes ou adresses mémoire.

1.4 Langage assembleur (ASM)

Pour pallier la complexité du binaire, le **langage assembleur** remplace les codes numériques par des mnémoniques lisibles (ex : `ADD`, `MOV`, `INC`). Bien qu'il reste très proche du matériel, il est nettement plus ergonomique à écrire et à déboguer. Un logiciel dédié, l'**assembleur**, se charge de traduire automatiquement ce code symbolique en langage machine exécutable.

1.5 Jeu d'instructions

L'architecture du processeur définit un jeu d'instructions organisé en cinq catégories fonctionnelles :

1. **Transferts de données** : déplacement d'informations entre registres, entre registre et mémoire, ou entre deux zones mémoire.
2. **Opérations arithmétiques & logiques** : calculs numériques, comparaisons, opérations bit à bit, décalages et rotations.
3. **Branchements (sauts)** : modification du flux d'exécution. Ils peuvent être inconditionnels ou conditionnels (déclenchés par l'état des flags), et incluent les appels/retours de sous-programmes et les interruptions.
4. **Gestion de la pile & drapeaux** : manipulation du pointeur de pile, modification explicite des indicateurs d'état.
5. **Instructions système/diverses** : contrôle matériel (mise en veille, arrêt), ou opérations neutres (`NOP`).

1.5.1 Exemples de mnémoniques

```
; Additionne REGB àl'accumulateur et stocke le résultat dans ACCU
ADD ACCU, REGB

; Incrémente de 1 la valeur contenue dans le registre REGB
INC REGB
```

Listing 1.1 – Illustration d'instructions assembleur

CHAPITRE 2

Architecture et Périphériques des DSP

2.1 Introduction aux Processeurs de Signal Numérique (DSP)

2.1.1 Définition du DSP

Le **Processeur de Signal Numérique** (*Digital Signal Processor*) est un processeur spécialisé conçu pour traiter efficacement les signaux numériques. Contrairement aux processeurs généralistes (comme le CPU d'un ordinateur), le DSP se distingue par sa capacité à effectuer des opérations mathématiques complexes (multiplication et addition) en un seul cycle d'horloge.

2.1.2 Pourquoi avons-nous besoin des DSP ?

— Haute vitesse de traitement des données en temps réel
— Efficacité énergétique
— Capacité d'exécution des algorithmes de traitement du signal (filtres, transformée de Fourier, etc.)

Famille	Caractéristiques	Exemples
DSP à virgule fixe (Fixed-Point)	- Nombres entiers uniquement - Faible consommation d'énergie - Coût réduit - Complexité programmation élevée	TMS320C54x, TMS320C55x
DSP à virgule flottante (Floating-Point)	- Haute précision - Facilité de programmation - Consommation d'énergie élevée - Coût élevé	TMS320C3x, TMS320C67x
DSP multi-cur (Dual-Core)	- Deux curs sur une seule puce - Traitement parallèle - Performance doublée	OMAP, DaVinci

TABLE 2.1 – Classification des DSP selon l'architecture numérique

2.2 Les différentes familles de DSP

2.2.1 Classification selon l'architecture interne

2.2.2 Classification selon le fabricant

2.2.2.1 Texas Instruments (TI)

— **C2000** : Pour le contrôle industriel et les applications embarquées
— **C5000** : Pour les appareils mobiles (faible consommation d'énergie)
— **C6000** : Pour les applications haute performance (communications, vidéo)

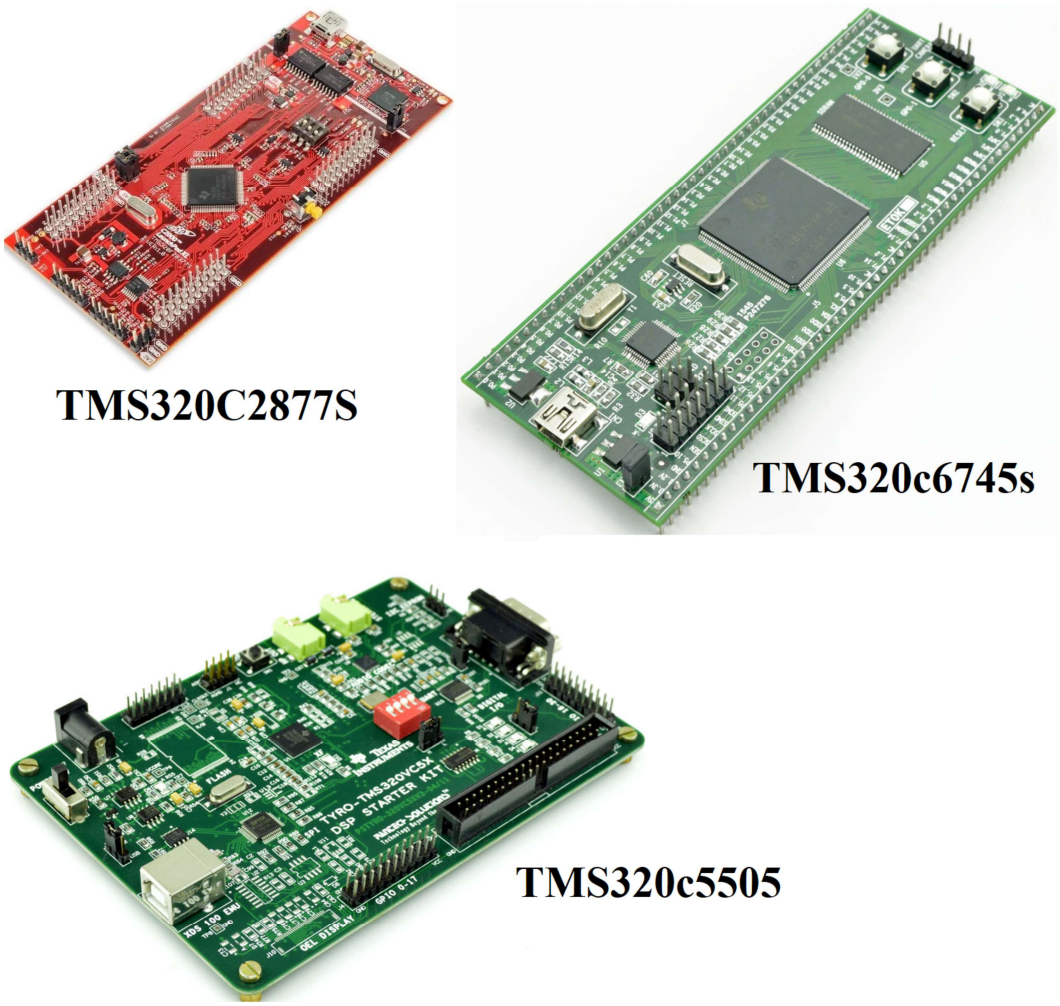


Figure 2.1 – Exemples de cartes de développement DSP : (a) **TMS320C2877S** basé sur larchitecture C2000, (b) **TMS320C6745** processeur DSP haute performance, (c) **TMS320C5505** processeur DSP basse consommation.

2.2.2.2 Analog Devices

— **SHARC** : Virgule flottante, pour l'audio professionnel
— **Blackfin** : Hybride entre DSP et microcontrôleurs

2.2.2.3 Motorola/Freescale (maintenant NXP)

— **DSP563xx** : Pour l'audio et les communications

2.3 L'unité de calcul MAC (Multiply and ACcumulate)

2.3.1 Définition de l'opération MAC

L'opération **MAC** est fondamentale dans le traitement du signal :

$$\text{MAC} : D = D + (A \times B) \tag{2.1}$$

où nous multiplions deux valeurs et additionnons le résultat à un résultat accumulé précédent, ce qui est essentiel pour les algorithmes de filtrage et de transformation, permettant d'effectuer des calculs complexes en un seul cycle d'horloge, optimisant ainsi les performances du DSP, et réduisant la latence dans les applications en temps réel, telles que le traitement audio, la vidéo et les communications. Pourquoi MAC est-elle importante ?

— **Filtrage numérique** : Chaque point de sortie nécessite plusieurs multiplications et additions

— **Transformations mathématiques** : FFT, DCT, etc.

— **Multiplication de matrices** : Dans les applications d'intelligence artificielle

la figure suivante représente la structure de l'unité MAC dans un DSP typique, montrant les registres d'entrée, le multiplieur, le résultat et l'accumulateur.

Figure 2.2 – Unité MAC : $D = D + (A \times B)$

2.3.2 Exemple pratique

Dans un DSP, l'opération MAC est souvent utilisée pour calculer des produits scalaires ou des convolutions, par utilisation d'un cycle d'horloge, par exemple calcul de :

$y = a_1x_1 + a_2x_2 + a_3x_3$

Cycle	Opération	Résultat accumulé
1	$a_1 \times x_1$	$D = a_1x_1$
2	$D + (a_2 \times x_2)$	$D = a_1x_1 + a_2x_2$
3	$D + (a_3 \times x_3)$	$D = a_1x_1 + a_2x_2 + a_3x_3$

TABLE 2.2 – Exécution séquentielle de l'opération MAC

2.4 Les mémoires internes – Architecture Harvard

2.4.1 Différence entre architecture Von Neumann et Harvard

L'architecture **Von Neumann** se caractérise par l'utilisation d'une seule mémoire pour stocker à la fois les données et le programme. Elle ne dispose que d'un seul bus de données, ce qui impose un accès séquentiel (plus lent). Cette architecture est principalement utilisée dans les ordinateurs généralistes.

En revanche, l'architecture **Harvard** utilise deux mémoires séparées et deux bus distincts. Cela permet un accès parallèle aux informations, rendant le système plus rapide. Elle est généralement réservée aux processeurs spécialisés.

2.4.2 Architecture Harvard modifiée dans les DSP

Dans l'architecture Harvard, il existe deux mémoires séparées, une pour les données et une pour les instructions du programme, et deux bus séparés qui les relient au processeur. Cela permet daccéder aux instructions du programme et aux données en même temps.

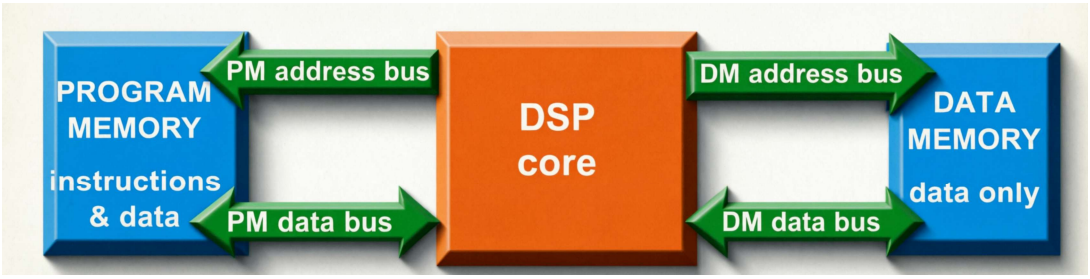


Figure 2.3 – Architecture Harvard

Dans la plupart des DSP commerciaux, l'architecture Harvard est améliorée en ajoutant une petite mémoire interne rapide, appelée ńcache d'instructionsż, dans laquelle sont relocalisées dynamiquement au moment de l'exécution les instructions du programme les plus fréquemment (ou les plus récemment) exécutées. Ceci est très avantageux par exemple si le DSP exécute une boucle suffisamment petite pour que toutes ses instructions puissent être contenues dans le cache d'instructions. Ces instructions sont transférées

dans le cache d'instructions lors de lexécution de la première itération de la boucle. Les autres itérations sont exécutées directement à partir du cache d'instructions. Une autre amélioration plus récente de larchitecture Harvard comme celle du DSP de Texas Instruments TMS320C6713, inclut un cache de programme et un cache de données. De plus Il existe deux niveaux de cache internes, appelés Niveau 1 L1 et Niveau 2 L2. Le temps daccès typique est de lordre de 1ns pour le cache L1, et 5ns pour le cache L2, tandis que le temps daccès à la mémoire externe (SDRAM/Flash) est de lordre de 100ns.

Les architectures DSP récentes, telles que celle du **Texas Instruments TMS320C6713** (étudiée dans ce module), poussent l'optimisation plus loin en adoptant une hiérarchie mémoire stricte inspirée des systèmes RISC avancés :

1. **Séparation L1 :** Existence de deux caches distincts au niveau 1 pour exploiter le parallélisme de Harvard :
   — **L1P (Level 1 Program) :** Dédié aux instructions.
   — **L1D (Level 1 Data) :** Dédié aux données.
2. **Mémoire Unifiée L2 :** Un deuxième niveau de cache (**L2**), plus grand mais légèrement plus lent, servant de zone tampon entre le cur du processeur et la mémoire externe (SDRAM/Flash). Une partie de la L2 peut être configurée comme SRAM mappée pour un accès déterministe.

Niveau	Type	Capacité	Vitesse	Fonction
L1	Cache	4–32 Ko	Très rapide	Stockage temporaire des données fréquentes
L2	Mémoire interne	64–512 Ko	Rapide	Stockage données et programme
Externe	SDRAM, Flash	Mégaoctets	Lente	Stockage permanent

TABLE 2.3 – Hiérarchie des mémoires dans les DSP

2.5 Le TMS320C6713 : Architecture

Le DSP de Texas Instruments TMS320C6713 est réalisé en technologie CMOS. Il utilise l'arithmétique à virgule flottante, et une architecture VLIW (Very Long Instruction Word).

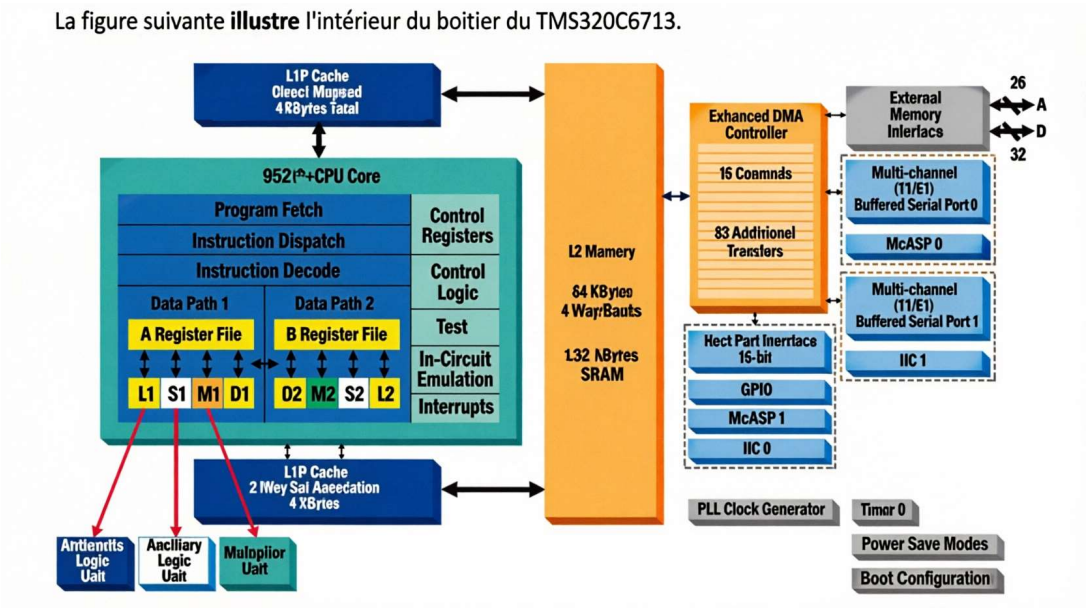


Figure 2.4 – Architecture interne du TMS320C6713, illustrant les chemins de données A et B, les unités fonctionnelles, les registres, et la hiérarchie mémoire.

La mémoire interne inclut deux niveaux de cache. Le cache L1 comprend 8 koctets de mémoire divisés en 4 koctets de cache de programme L1P et 4 koctets de cache de données L1D. Le cache L2 comprend 256 koctets de mémoire divisée en une mémoire SRAM de 192 koctets, et une mémoire à accès-dual de 64 koctets.

La mémoire interne de programme est structurée de sorte que le pipeline Fetch-Dispatch peut envoyer huit instructions parallèles au décodeur d'instructions à chaque cycle.

L'unité de traitement centrale du TMS320C6713 est divisée en deux sous-ensembles appelés chemin de données A et chemin de données B. Outre ces deux chemins de données, l'unité centrale comporte une logique de contrôle et des registres spécialisés nécessaires à son fonctionnement.

Chacun des deux chemins de données comporte quatre unités fonctionnelles autonomes, qui peuvent exécuter chacune une instruction séparée, et qui dispose de son jeu d'instructions dédié :

— une unité **.M** utilisée pour l'opération de multiplication
— une unité **.L** utilisée pour les opérations logiques et arithmétiques
— une unité **.S** utilisée pour les branchements et boucles du programme, les décalages de registres, la manipulation de bits, et les opérations arithmétiques
— une unité **.D** utilisée pour la lecture et l'écriture dans la mémoire de données, et pour les opérations arithmétiques

Il y a 16 registres 32 bits associés à chaque chemin de donnée, A0-A15 côté A, B0-B15 côté B. Toute interaction avec les unités fonctionnelles se fait obligatoirement à travers

ces registres.

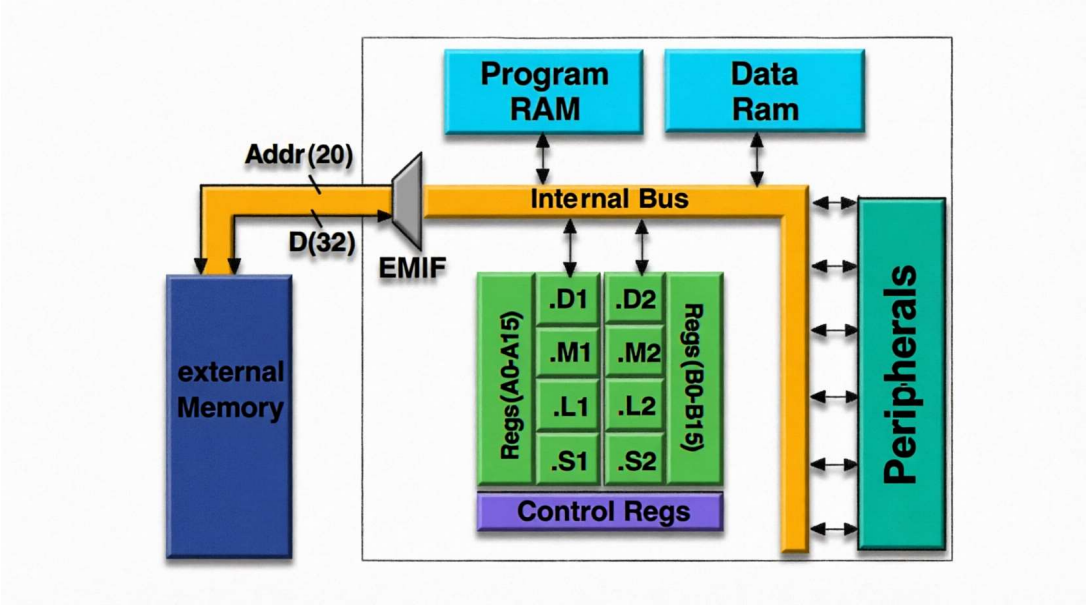


Figure 2.5 – Chemins de données A et B du TMS320C6713, montrant les unités fonctionnelles .M, .L, .S, .D, et les registres associés.

Le bus interne est constitué d'un bus 32 bits d'adresse de programme, un bus 256 bits d'instructions de programme pouvant recevoir huit instructions 32 bits, deux bus 32 bits d'adresses de données DA1 et DA2, deux bus 32 bits chargement de données LD1 et LD2, et deux bus 32 bits stockage de données ST1 et ST2. En outre, il existe un bus 32 bits de données DMA et un bus 32 bits d'adresses DMA. La mémoire hors-puce ou externe est accessible via un bus 20 bits d'adresses et un bus 32 bits de données contrôlés par EMIF (External Memory Interface).

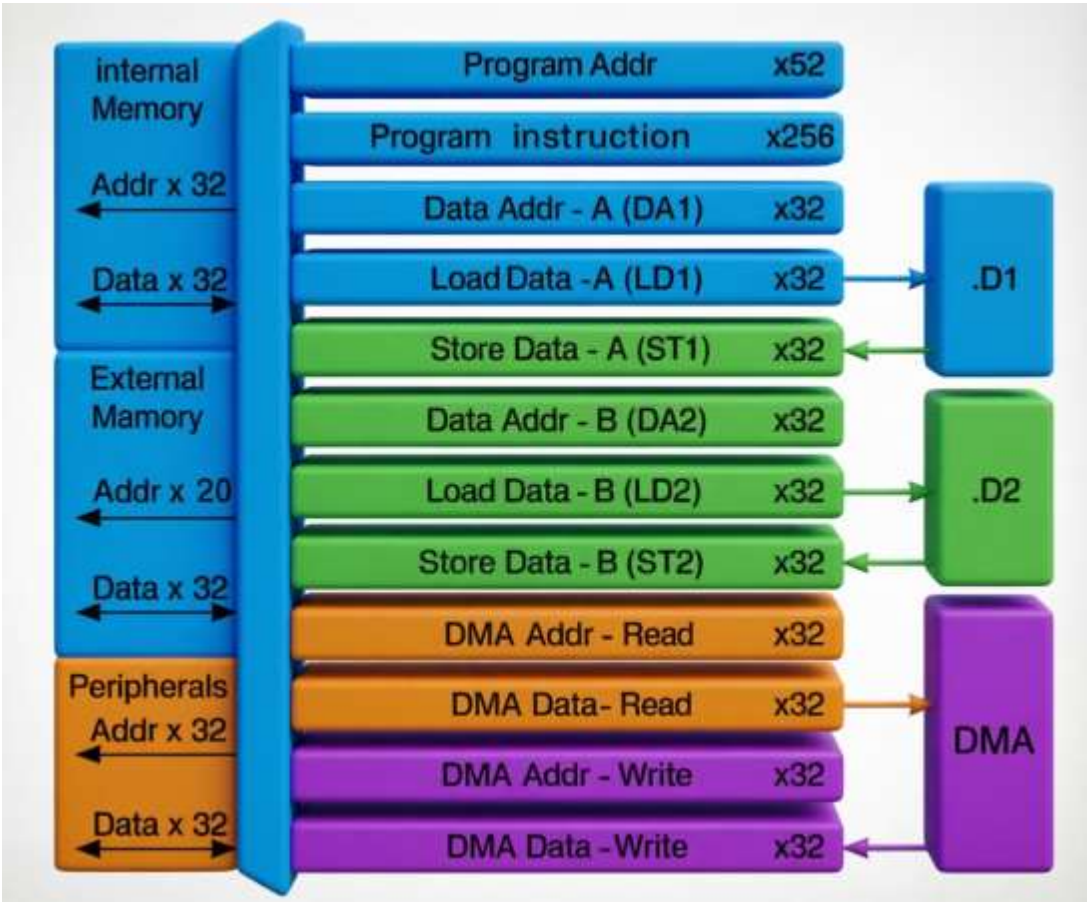


Figure 2.6 – Bus interne du TMS320C6713, illustrant les différents bus de données et adresses.

2.5.1 Concept VLIW

Le concept VLIW (Very Long Instruction Word) a été introduit pour la première fois par TI pour cette famille de DSP. Il désigne des codes instructions de taille fixe 256 bits, divisés en huit mots de 32 bits destinés chacun en parallèle à l'une des huit unités fonctionnelles, de sorte que idéalement les huit instructions devraient s'exécuter simultanément pour une pleine exploitation du parallélisme. En pratique, cet optimum est rarement réalisé.

2.5.2 Les périphériques

Les périphériques disponibles dans le TMS320C6713 sont répartis en deux catégories : Les interconnexions entrées-sorties, et les services système.

2.5.2.1 Interconnexions

— **EMIF (External Memory Interface)** : fournit les signaux nécessaires pour contrôler les accès à la mémoire externe SDRAM, SBSRAM, SRAM, ROM/Flash, FPGA.
— **DMA (Direct Memory Access)** : permet le déplacement des données d'un emplacement en mémoire vers un autre sans intervention de l'Unité Centrale.
— **McBSP (Multichannel Buffered Serial Port)** : 128 canaux de communication full-duplex programmables, à double registres tampons de données qui permettent un flux continu indépendant en réception et en transmission. Directement interfaçable avec les codecs TDM haut débit, l'interface analogique AIC, l'interface sérielle SPI, les Codecs AC97, et EEPROM sérielle.
— **McASP (Multichannel Audio Serial Port)** : port série polyvalent optimisé pour les applications audio multicanaux. Inclut le flux multiplexé par répartition dans le temps TDM, les protocoles Inter Integrated Sound I2S à ADC multicanal et Digital audio Interface Transmission DIT à sortie multiple, DAC, Codec, et DIR Digital Infrared.
— **HPI (Host Port Interface)** : permet à un autre circuit (hôte) d'accéder à la mémoire interne.
— **IIC** : bus série standard Inter-Integrated Circuit.
— **GPIO** : interface parallèle universelle General Purpose Input Output.

2.5.2.2 Services Système

— **Timer** : temporisateur à usage général constitué de deux compteurs 32 bits.
— **Contrôleur PLL** : assure la synchronisation d'horloge pour les périphériques à partir de signaux internes ou externes.
— **Power Down** : unité de mise hors tension pour économiser de l'énergie lorsque la CPU est inactive, ou de réduire la fréquence des signaux si besoin est car la dissipation de la puissance des circuits CMOS se produit lors du basculement d'un état logique à l'état inverse.
— **Boot Configuration** : permet la programmation par l'utilisateur du mode d'amorçage, à partir de la mémoire hors-puce, ou autre.

2.5.3 Exemple de fonctionnement (Opération MAC)

Avant d'illustrer par l'exemple le mode de fonctionnement de l'unité de traitement, il est utile de présenter à ce niveau de l'exposé une vue d'ensemble du jeu d'instructions du C6713 qui sera étudié en détail plus tard. Chaque unité fonctionnelle a ses instructions propres, à part celles écrites en bleu qui sont communes à plus d'une unité.

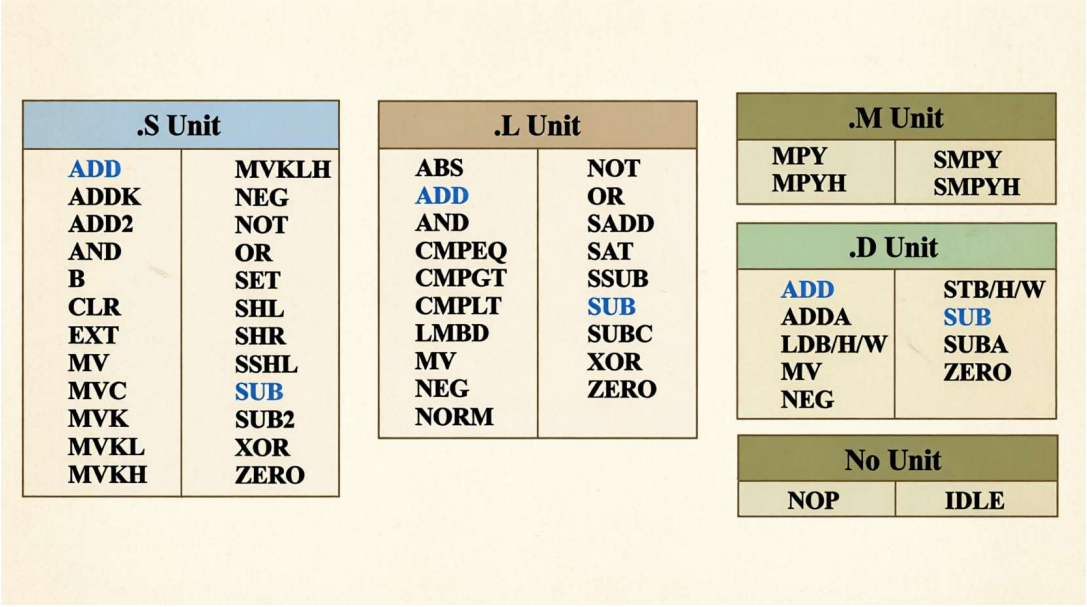


Figure 2.7 – Ensemble du jeu d'instructions du C6713, classé par unité fonctionnelle. Les instructions en bleu sont communes à plusieurs unités.

L'exemple montre comment l'unité de traitement centrale est mise en uvre et pourrait être utilisée pour effectuer l'opération MAC du filtre FIR :

$$y = \sum_{n=1}^{10} c_n x_n \tag{2.2}$$

En pratique les coefficients c_n sont préprogrammés et disponibles dans la mémoire de données, les échantillons x_k sont dans une autre zone mémoire après acquisition via une interface hardware.

— L'unité **.D1** effectue le chargement des opérandes c_n, x_n dans les registres à partir de la mémoire en utilisant une instruction de chargement telle que `LDH .D1 *Rn,Rm` qui signifie que l'adresse mémoire pointée par Rn est chargée dans le registre Rm.

— L'unité **.M1** effectue les multiplications de façon câblée entre les variables c_n et x_n, et met le résultat dans une variable prod, instruction assembleur `MPY .M1 c1, x1, prod`.

— L'unité **.L1** effectue l'addition, résultat cumulé dans Y, instruction `ADD .L1 Y, prod,Y`.

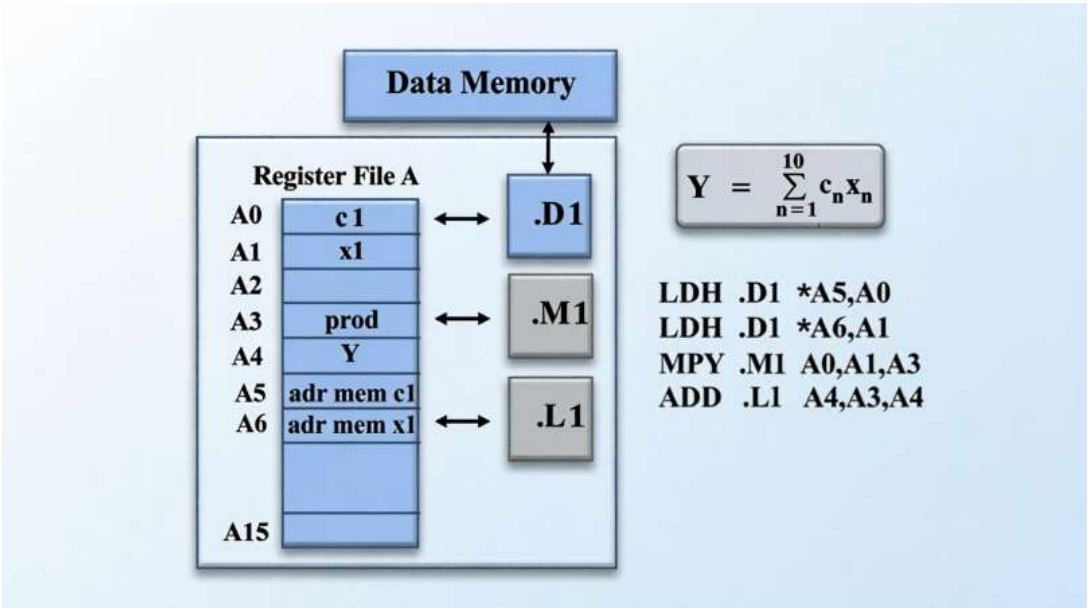


Figure 2.8 – l'opération MAC du filtre FIR : $y = \sum_{n=1}^{10} c_n x_n$ exécutée sur les unités fonctionnelles .D1, .M1, et .L1 du TMS320C6713.

2.6 Mappage Mémoire TMS320C6713

TABLE 2.4 – Cartographie Mémoire du TMS320C6713

Memory Block Description	**Block Size**	**Hex Address Range**
Internal RAM (L2)	192K	0000 0000 - 0002 FFFF
Internal RAM/Cache	64K	0003 0000 - 0003 FFFF
Reserved	$24M - 256K$	0004 0000 - 017F FFFF
EMIF Registers	256K	0180 0000 - 0183 FFFF
L2 Registers	128K	0184 0000 - 0185 FFFF
Reserved	128K	0186 0000 - 0187 FFFF
HPI Registers	256K	0188 0000 - 018B FFFF
McBSP0 Registers	256K	018C 0000 - 018F FFFF
McBSP1 Registers	256K	0190 0000 - 0193 FFFF
Timer0 Registers	256K	0194 0000 - 0197 FFFF
Timer1 Registers	256K	0198 0000 - 019B FFFF
Interrupt Selector Registers	512	019C 0000 - 019C 01FF
Device Configuration Registers	4	019C 0200 - 019C 0203

Suite dans la page suivante

Memory Block Description	Block Size	Hex Address Range
Reserved	$256K - 516$	019C 0204 - 019F FFFF
EDMA RAM and Registers	256K	01A0 0000 - 01A3 FFFF
Reserved	768K	01A4 0000 - 01AF FFFF
GPIO Registers	16K	01B0 0000 - 01B0 3FFF
Reserved	240K	01B0 4000 - 01B3 FFFF
I2C0 Registers	16K	01B4 0000 - 01B4 3FFF
I2C1 Registers	16K	01B4 4000 - 01B4 7FFF
Reserved	16K	01B4 8000 - 01B4 BFFF
McASP0 Registers	16K	01B4 C000 - 01B4 FFFF
McASP1 Registers	16K	01B5 0000 - 01B5 3FFF
Reserved	160K	01B5 4000 - 01B7 BFFF
PLL Registers	8K	01B7 C000 - 01B7 DFFF
Reserved	264K	01B7 E000 - 01BB FFFF
Emulation Registers	256K	01BC 0000 - 01BF FFFF
Reserved	4M	01C0 0000 - 01FF FFFF
QDMA Registers	52	0200 0000 - 0200 0033
Reserved	$16M - 52$	0200 0034 - 02FF FFFF
Reserved	720M	0300 0000 - 2FFF FFFF
McBSP0 Data Port	64M	3000 0000 - 33FF FFFF
McBSP1 Data Port	64M	3400 0000 - 37FF FFFF
Reserved	64M	3800 0000 - 3BFF FFFF
McASP0 Data Port	1M	3C00 0000 - 3C0F FFFF
McASP1 Data Port	1M	3C10 0000 - 3C1F FFFF
Reserved	$1G + 62M$	3C20 0000 - 7FFF FFFF
EMIF CE0	256M	8000 0000 - 8FFF FFFF
EMIF CE1	256M	9000 0000 - 9FFF FFFF
EMIF CE2	256M	A000 0000 - AFFF FFFF
EMIF CE3	256M	B000 0000 - BFFF FFFF
Reserved	1G	C000 0000 - FFFF FFFF

2.6.1 Exemples d'instructions de base

2.6.1.1 Instructions mathématiques

```
; Multiplication et accumulation (MAC)
MAC *AR0+, *AR1+, AC0  ; AC0 = AC0 + (*AR0 * *AR1)

; Addition
ADD #100, AC0          ; AC0 = AC0 + 100

; Multiplication
MPY T0, AC0            ; AC0 = T0 * AC0
```

Listing 2.1 – Instructions mathématiques MAC

2.6.1.2 Instructions de transfert de données

```
; Chargement depuis la memoire
MOV *AR0, T0           ; T0 = valeur a l'adresse AR0

; Stockage en memoire
MOV AC0, *AR1+         ; *AR1 = AC0 puis incrementation AR1

; Transfert entre registres
MOV T0, AC0            ; AC0 = T0
```

Listing 2.2 – Instructions de transfert

2.6.1.3 Instructions de contrôle

```
; Saut conditionnel
BCC label, T0 == #0    ; Si T0=0 aller a label

; Boucle
RPT #99                ; Repeter l'instruction suivante 100 fois
  MAC *AR0+, *AR1+, AC0

; Appel de fonction
CALL nom_fonction
```

Listing 2.3 – Instructions de contrôle

CHAPITRE 3

Code Composer Studio (CCS 12)

3.1 Qu'est-ce que Code Composer Studio (CCS) ? (Définition approfondie)

Code Composer Studio (CCS) est un environnement de développement intégré (IDE) complet, conçu et maintenu par **Texas Instruments (TI)**, spécifiquement dédié au développement, à la compilation, au débogage et à l'optimisation de logiciels embarqués destinés aux microcontrôleurs (MCU), processeurs numériques de signal (DSP) et systèmes multi-curs de l'écosystème TI.

Reposé sur la plateforme open-source **Eclipse**, CCS a été profondément remanié pour offrir une chaîne d'outils fluide, performante et adaptée aux contraintes du développement embarqué temps réel. Il prend en charge nativement les langages **C, C++ et Assembleur**, et s'interface avec les compilateurs optimisés de TI (TI CGT), GCC et Clang/LLVM.

Au-delà d'un simple éditeur de code, CCS constitue une **suite d'ingénierie logicielle embarquée** qui couvre l'intégralité du cycle de développement :

CCS est aujourd'hui la référence industrielle pour développer sur les familles TI : **MSP430, C2000, SimpleLink (CC13xx/CC26xx, CC32xx, MSPM0), Sitara (ARM Cortex-A)** et **DSPs C6000/C5000**.

3.2 Versions et Licences

Texas Instruments a progressivement simplifié son modèle de licence. Voici l'état actuel (valable pour CCS v10 et versions ultérieures) :

> **Note importante :** Depuis la version 10, **TI a supprimé toute limite de taille de code** dans la version gratuite. Le modèle Commercial ne concerne plus le déblocage de fonctionnalités, mais uniquement le **support technique**

Domaine	Fonctionnalités clés
Compilation & Build	Gestion automatique des dépendances, scripts de liaison personnalisés (`.cmd`), rapports d'optimisation, support C99/C11/C++17
Débogage matériel	Support JTAG/cJTAG/SWD via sondes XDS110, XDS100v3, J-Link, etc. Points d'arrêt conditionnels, exécution pas-à-pas, visualisation registres/périphériques en temps réel
Analyse de performance	Profiling cycle-accurate, analyseur de consommation mémoire (RAM/Flash), traçage matériel via ITM/SWO
EnergyTrace™	Mesure et visualisation en temps réel de la consommation énergétique du MCU, identification des états de veille, optimisation batterie
System Analyzer	Capture et analyse des événements système, suivi des tâches RTOS, logs UART/SWO non intrusifs
Écosystème TI	Intégration native de **Resource Explorer** (exemples, pilotes, SDK), support TI-RTOS/FreeRTOS/POSIX, outils de configuration automatique (SysConfig)
CCS Cloud	Version hébergée dans le navigateur, zéro installation, idéale pour la formation, la collaboration et les démos rapides

TABLE 3.1 – Fonctionnalités principales de Code Composer Studio

garanti, les services de conformité industrielle et l'accompagnement pour le déploiement en production.

Code Composer Studio n'est pas un simple éditeur de code, mais une **plateforme d'ingénierie embarquée complète** qui intègre compilation, débogage matériel, analyse énergétique, traçage système et gestion de projet. Sa gratuité actuelle, couplée à son écosystème TI étroitement intégré, en fait un choix stratégique pour les développeurs, les universités et les industriels travaillant sur des architectures Texas Instruments.

3.3 Architecture de CCS

CCS repose sur plusieurs composants essentiels :

— **Éditeur** : permet décrire le code source

— **Compilateur** : traduit le code en langage machine

— **Linker** : assemble les différents fichiers

— **Debugger** : permet de détecter et corriger les erreurs

Version	Description	Licence
CCS Free (Desktop)	Accès complet à l'IDE, au débogueur, à SysConfig, EnergyTrace™, System Analyzer et à toutes les familles de processeurs TI. **Aucune restriction de taille de code.**	Gratuit (compte TI recommandé)
CCS Academic	Identique à la version gratuite, mais inclut des ressources pédagogiques, des licences académiques étendues pour certains outils annexes, et un support dédié aux établissements d'enseignement.	Gratuit (sur vérification universitaire)
CCS Commercial / Enterprise	Même base logicielle gratuite, mais avec **support technique prioritaire**, accès à des outils de validation industrielle, mises à jour anticipées et accords de licence pour la production en série.	Payant (abonnement ou licence entreprise)
CCS Cloud	Environnement hébergé dans le navigateur, synchronisation automatique avec GitHub/GitLab, partage de projets en un clic. Idéal pour l'apprentissage et le prototypage rapide.	Gratuit (limité en temps d'exécution cloud)

TABLE 3.2 – Comparaison des versions et licences CCS

3.4 Installation et configuration

3.4.1 Étapes dinstallation

1. Télécharger CCS depuis le site officiel
2. Installer les packages nécessaires (MSP430, C2000, ARM, DSP)
3. Définir un *workspace*

3.4.2 Workspace

Le workspace est le dossier principal contenant tous les projets. Il est recommandé de créer un workspace par module ou matière.

3.5 Interface utilisateur

— Project Explorer : gestion des projets
— Editor : écriture du code
— Console : affichage des messages
— Debug View : suivi de lexécution

3.6 Création dun projet

3.6.1 Étapes

1. File → New → CCS Project
2. Choisir le microcontrôleur cible
3. Sélectionner le compilateur
4. Choisir un template

3.7 Compilation et Build

3.7.1 Définitions

— Compile : traduction du code source
— Build : génération du fichier exécutable

3.7.2 Types de build

— Full Build
— Incremental Build

3.8 Debugging

Le debugging est une étape essentielle pour analyser le comportement du programme.

3.8.1 Outils de debug

— Breakpoints : arrêt du programme
— Step Into / Step Over : exécution pas à pas
— Watch : surveillance des variables
— Memory View : visualisation mémoire

3.9 Connexion au matériel

3.9.1 Interfaces

— USB
— JTAG

3.9.2 Procédure

1. Connecter la carte
2. Lancer le mode Debug
3. Charger le programme

3.10 Outils avancés

— Graph Tool : visualisation des signaux
— Analyse mémoire
— Profiling (analyse des performances)

3.11 Problèmes fréquents

Problème	Solution
Erreur compilation	Vérifier les bibliothèques
Debug non connecté	Vérifier USB/JTAG
Programme ne fonctionne pas	Vérifier configuration

3.12 Exemple : LED Blink

```
#include <stdint.h>

int main(void) {
while(1) {
// Toggle LED
}
}
```

3.13 Explication détaillée

Ce programme contient une boucle infinie (`while(1)`) qui permet dexécuter en continu une action (comme allumer/éteindre une LED). En pratique, il faut configurer les registres du microcontrôleur pour contrôler une broche GPIO.

3.14 Bonnes pratiques

— Utiliser les exemples fournis
— Structurer le code en modules
— Tester régulièrement
— Documenter le code

3.15 Travaux pratiques

1. Création dun projet simple
2. Clignotement LED
3. Debug dun programme
4. Lecture dun capteur

3.16 Conclusion

CCS est un outil puissant pour le développement embarqué. Sa maîtrise est essentielle pour les ingénieurs en électronique et télécommunications.

CHAPITRE 4

Notion de base sur les circuits programmables

4.1 Introduction

Un **circuit logique programmable** (PLD – *Programmable Logic Device*) est un circuit intégré dont la logique interne peut être définie ou modifiée par l'utilisateur, même après fabrication. Contrairement aux microprocesseurs qui exécutent des programmes logiciels, les PLD sont configurés au niveau **matériel** (portes logiques, bascules, interconnexions). On parle donc de *reconfiguration matérielle*.

4.2 Architecture générale des circuits logiques programmables

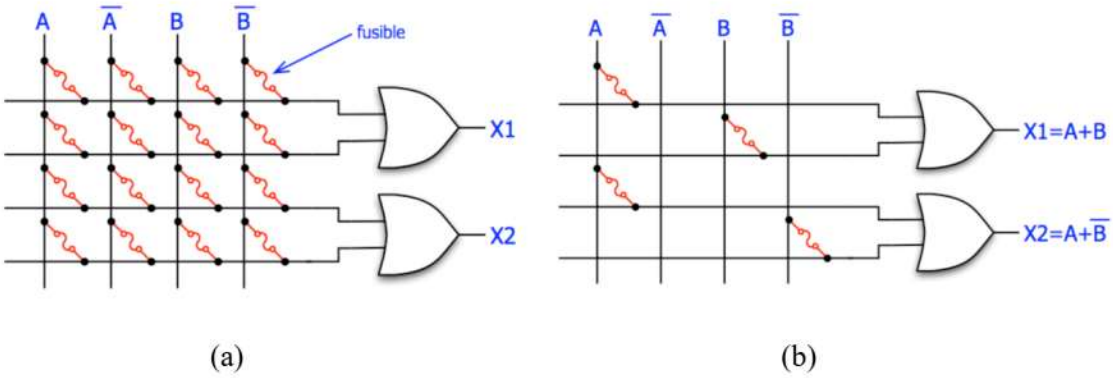
4.2.1 Les SPLD (Simple PLD)

Les SPLD regroupent les premiers circuits programmables. Leur principe repose sur :
— un réseau de portes ET programmables,
— un réseau de portes OU fixes.

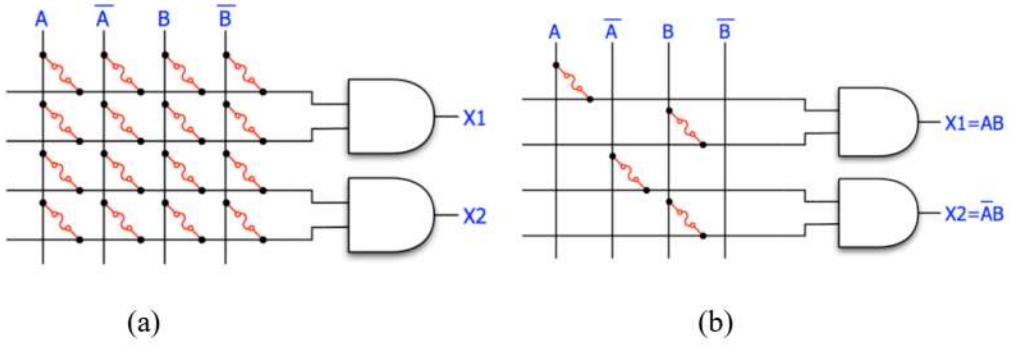
PAL (Programmable Array Logic). Développés par MMI (racheté par AMD) dans les années 70. Programmation par destruction de fusibles → non réinscriptibles. Utilisés dans le décodage et les machines à états.

GAL (Generic Array Logic). Évolution des PAL (par Lattice). Remplacement des fusibles par des transistors MOSFET → circuits reprogrammables. Plus souples et économiques.

PLA (Programmable Logic Array). Plus flexibles, car les deux réseaux (ET et OU) sont programmables. Adaptés à des fonctions logiques complexes.



Exemple de matrice OR : (a) non programmée. (b) programmée.



Exemple de matrice AND : (a) non programmée. (b) programmée.

Figure 4.1 – Matrices OR/AND programmables illustrant le principe SPLD (extrait du cours, p. 5).

Tableau comparatif des SPLD (adapté du cours, p. 6) :

Type	Plan ET	Plan OU	Technologie	Utilisation typique
PAL	Programmable	Fixe	Bipolaire	Décodage, machines à états
GAL	Reprogrammable	Fixe	CMOS	Décodage, machines à états
PLA	Programmable	Programmable	CMOS	Fonctions logiques complexes

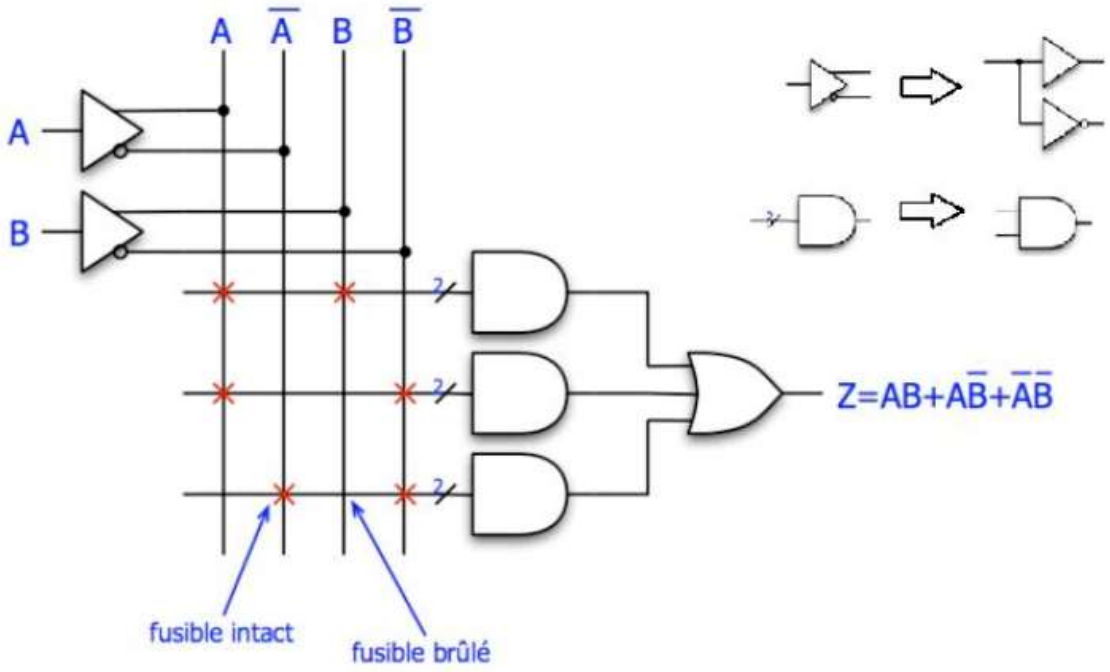


Figure 4.2 – Exemple de PAL à 2 entrées/1 sortie (extrait du cours, p. 6).

4.2.2 Les CPLD (Complex PLD)

Constitués de plusieurs SPLD interconnectés par une matrice programmée (PI – *Programmable Interconnect*). Chaque bloc logique (macro-cellule) contient des portes logiques et une bascule (*flip-flop*). Les temps de propagation sont **prédictibles**, ce qui les rend adaptés aux applications nécessitant stabilité et rapidité.

4.2.3 Les FPGA (Field Programmable Gate Arrays)

Inventés par Xilinx en 1985. Constitués de milliers/millions de blocs logiques configurables (CLB), de blocs d'entrées/sorties (IOB) et de canaux d'interconnexion flexibles.

Les FPGA modernes incluent :

— de la mémoire RAM,

— des blocs DSP (*Digital Signal Processing*),

— voire des processeurs embarqués.

Caractéristiques :

— très grande capacité logique,

— reconfiguration quasi illimitée,

— applications complexes : télécoms, imagerie médicale, cryptographie, prototypage ASIC, etc.

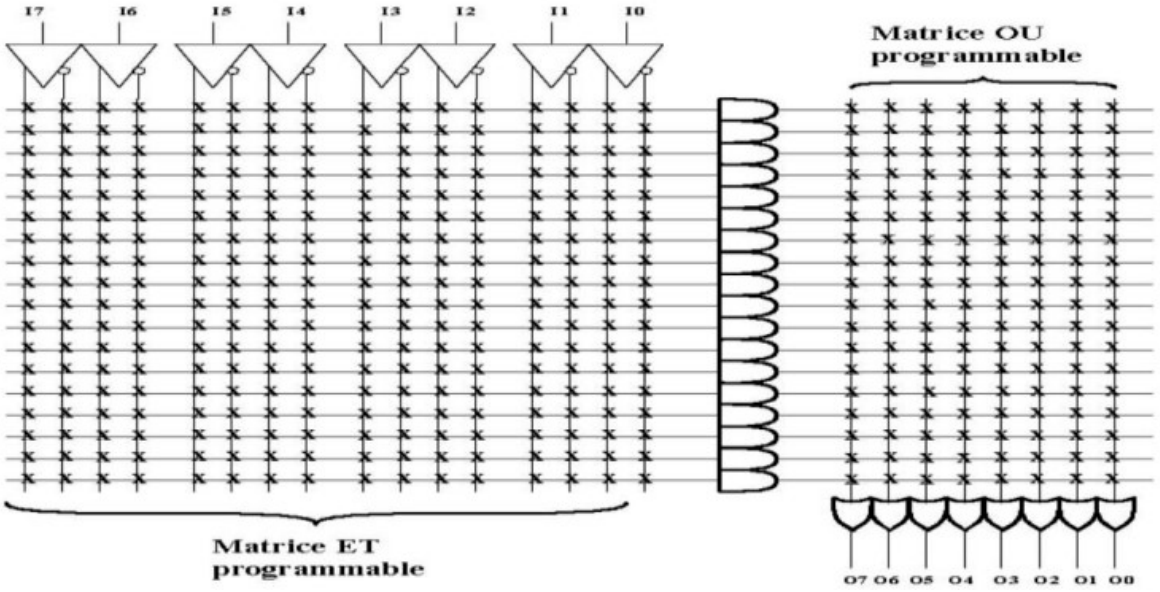


Figure 4.3 – Circuit PLA avec matrices ET/OU programmables (extrait du cours, p. 7).

4.3 Exemples de constructeurs et outils de programmation

4.3.1 Principaux constructeurs

- **Xilinx** : pionnier des FPGA (familles Spartan, Virtex, Artix...),
- **Altera (Intel)** : CPLD MAX, FPGA Cyclone/Stratix,
- **Lattice Semiconductor** : spécialiste GAL, CPLD et FPGA basse consommation,
- **Actel/Microsemi (Microchip)** : FPGA à anti-fusibles, utilisés en aéronautique/défense.

Tableau des technologies utilisées (adapté du cours, p. 10) :

Fabricant	Technologies utilisées
Actel	Anti-fusible, SRAM
Altera	EPROM, EEPROM, SRAM
AMD	EEPROM
Atmel	SRAM
Lattice	EPROM, EEPROM
Xilinx	SRAM, Anti-fusible, EPROM, EEPROM

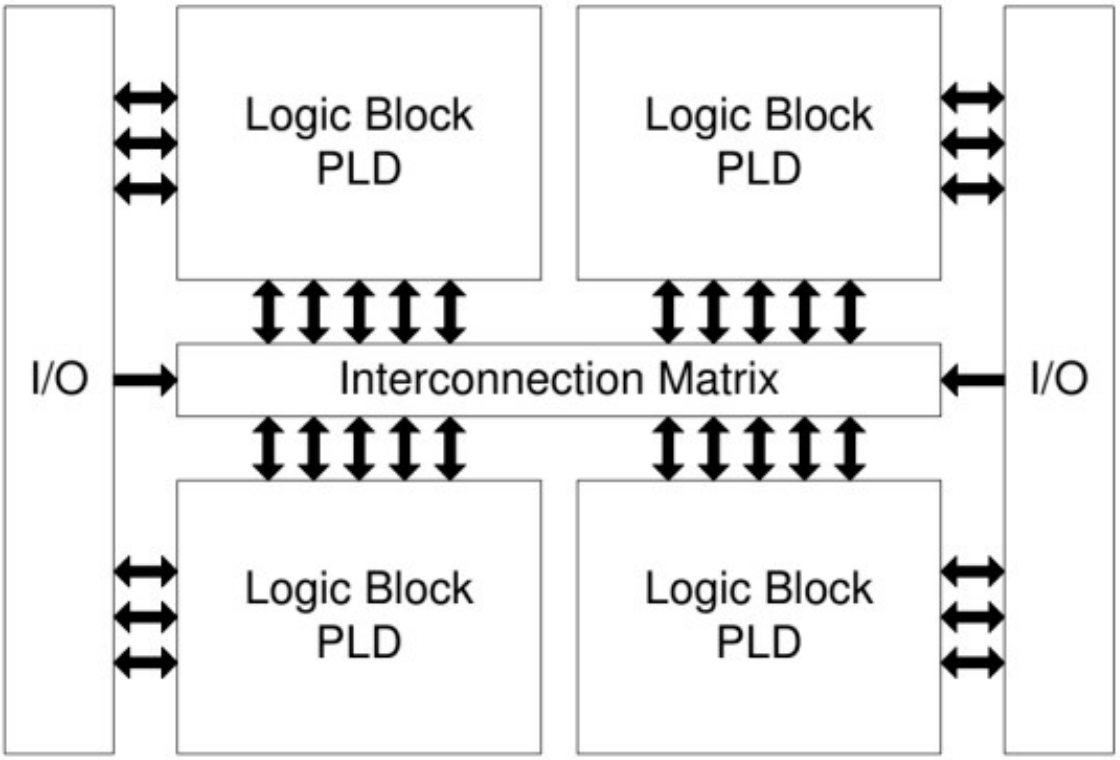


Figure 2 : physionomie d'un CPLD

Figure 4.4 – Physionomie d'un CPLD avec blocs logiques et matrice d'interconnexion (extrait du cours, p. 9).

4.3.2 Outils de programmation

— **Xilinx ISE** : ancien environnement de développement (remplacé par Vivado),
— **Vivado** : nouvelle génération pour FPGA Xilinx,
— **Altera Quartus II** : outil complet pour FPGA/CPLD Altera,
— **Lattice ISPLever** : pour circuits Lattice,
— **ModelSim** : simulateur VHDL/Verilog très utilisé en phase de test.

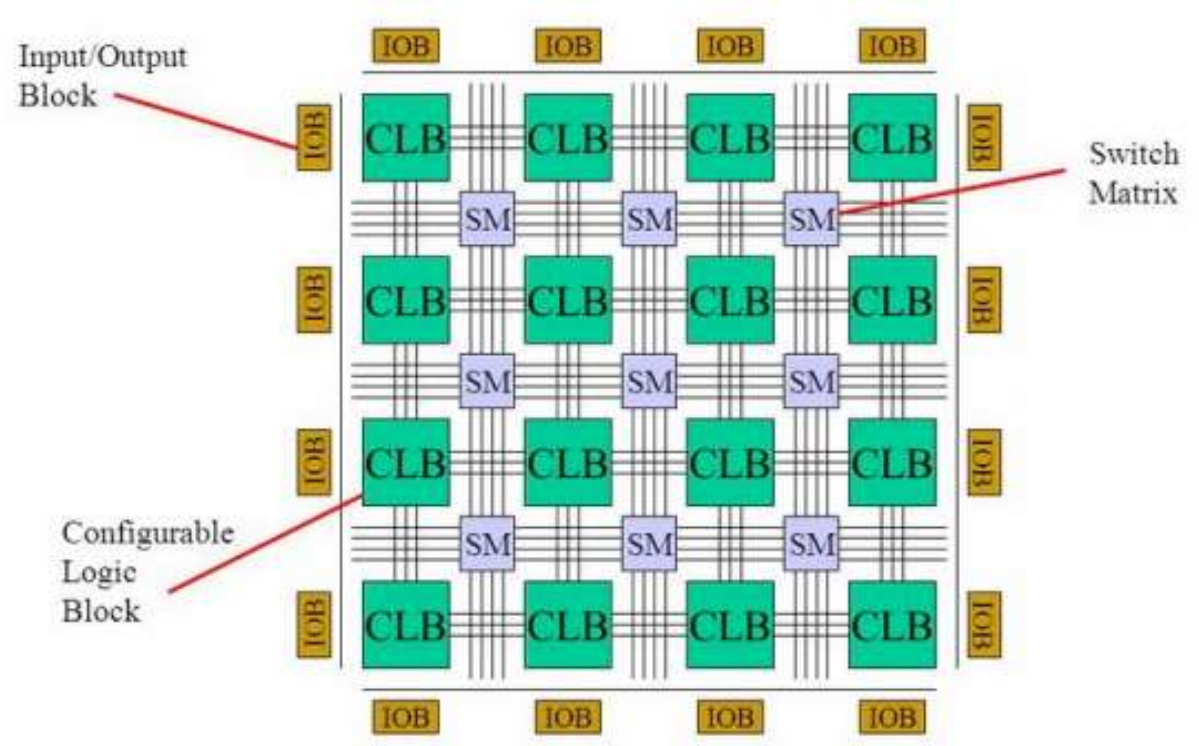


Figure 4.5 – Structure d'un FPGA de type matrice symétrique (extrait du cours, p. 16).

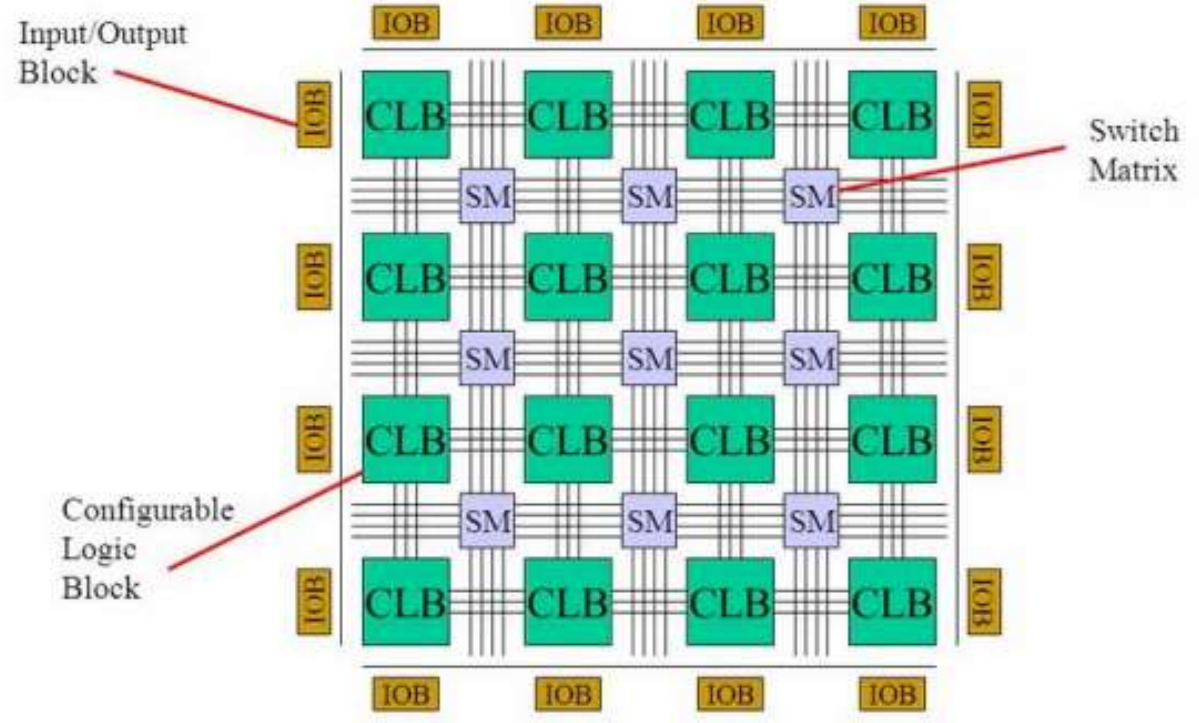


Figure 4.6 – Architecture interne d'un FPGA montrant l'interconnexion CLB/IOB (extrait du cours, p. 17).