ETSET BARCELONA / ENST DE BRETAGNE

Etude Architecturale et Ecriture d'un Générateur de Multiplieurs Parallèle-Parallèle pour le Filtrage Transversal

RAPPORT DE STAGE

Jaume Guasch

4 Mars – 28 Juin 1991

Projet réalisé par: GUASCH, Jaume
Responsable CNET: JOANBLANQ, Christophe

Als meus pares

Avant d'exposer mon travail, je tiens à remercier Monsieur GERBER, chef du groupement CCI, et Monsieur SENN, chef du département CAT, pour avoir bien voulu m'accueillir dans leur service pour la réalisation de mon stage. Je remercie tout particulièrement Christophe JOANBLANQ pour l'aide et les conseils qu'il m'a apporté et Gilles PRIVAT pour son apport théorique sur le fond et la forme de mon travail.

Je remercie également Emmanuel BIDET qui a bien voulu m'initier aux arcanes du système UNIX, Hélène LE MAUX pour avoir "réalisé" la version française de ce rapport, tous les membres de l'équipe CAT du CNET pour leur aide, leurs conseils et leur amitié ainsi que Marc RENAUDIN et Liliane ANLIARD qui ont coordonné respectivement la partie académique et la partie administrative pendant les cours et le stage.

SOMMAIRE

1. Résumé

Le but de ce travail a été d'abord l'étude des structures de filtrage à réponse impulsionnelle finie à employer dans les applications envisagées au CNET de Grenoble. Tout de suite après l'étude et la génération d'un multiplieur en arithmétique redondante ont été réalisées. Le multiplieur obtenu est très simple dans le cas des coefficients codés en notation canonique signée (CSD). On en est venu à montrer finalement l'équivalence entre les structures classiques Carry-Save et les structures signées.

Pour la réalisation de ce travail, le système GDT (Generator Development Tools) a été utilisé. Ce système nous offre un environment d'édition et de vérification très adapté à l'écriture de générateurs, à l'édition en modification de cellules et à la vérification à plusieurs niveaux des dessins et structures.

Le résultat final est un générateur de multiplieurs basé sur une nouvelle structure redondante avec des chiffres signés qui pourra être employée dans le générateur de filtres en cours de développement au département CCI/CAT du CNET de Grenoble.

1. Resum

Per l'enginyer d'aplicacions, l'implantació VLSI ofereig possibilitats i prestacions sense precedents. L'actual tecnologia permet a l'enginyer de concebre el sistema complert dins un simple xip. Lluny de l'era on els sistémes constituïent un conjunt de circuits integrats sobre un placa impresa, la tendència actual és l'especialització de les tasques i els circuits per una aplicació específica anomenats ASICs.

Aquests circuits integrats son cada vegeda més complexes i per l'enginyer seria interessant de disposar d'eines de concepció per automatitzar el procès. De la mateixa manera que el programador informàtic disposa de compiladors, per comunicar-se amb la màquina en un llenguatge d'alt nivell i produir el codi corresponent al processador emprat, els anomenats Compiladors de Silici representent l'equivalent en el món de la concepció de circuits integrats. El resultat en lloc d'esser una sèrie d'instruccions en codi màquina és un conjunt de polígons que representent els diferents nivells de les màsqueres.

Malauradament, els compiladors de silici es troben, per causes diferents, en un estat primitiu, particularitzats sobre certes llibreries de cèl.lules estandar. La millor aproximació, la representen els Generadors. Un generador és bàsicament un conjunt de bucles de emplaçament i rutatge de cèl.lules en diferents nivells de representació com l'esquemàtic (elèctric o lògic) o el de layout (disseny a la micra). Un generador pot esser concebut en la seva totalitat en un llenguatge d'alt nivell com el C o l'ADA sobre una estació de treball tot gestionant la base de dades, els algoritmes de plaçament, rutatge, verificació i simulació dels circuits tot creant una interfase gràfica per l'utilitzador. Evidentment, això representa una gran quantitat de treball que per altra banda resultaria absurda de realitzar per cada generador. Així doncs, diferents cases de software posen a disposició de l'usuari eines per l'escriptura de generadors. Entre ells poden citar el sistema GDT (Generator Development Tools), emprat per la realització d'aquest stage.

El sistema GDT i els generadors

El sistema GDT constitueix un entorn complert de disseny, verificació i simulació de circuits que gira en torn d'una base de dades textual descrita en el llenguatge propi anomenat L. Aquesta base de dades conté una sèrie d'objectes com cèl.lules, instàncies, arrays, terminals, transistors, fils i contactes caracteritzats per paràmetres propis.

Bàsicament, es tracta de manipular cèl.lules com agrupacions d'objectes a diferents nivells jeràrquics. L'automatització i parametrització d'aquestes, constitueix un generador. Un generador pot esser cridat per un generador d'ordre superior, etc.

Així doncs, som capaços de generar de forma automàtica i a partir de paràmetres introduits per l'usuari (o a partir de processos algorítmics com el calcul dels coeficients d'un filtre) els esquemes i layouts corresponents al circuit desitjat, sempre i que aquest compleixi unes mínimes condicions de regularitat i repetibilitat.

Finalment, el sistema GDT ens permet realitzar simulacions a diferents nivells i extreure 'netlists' per altres simuladors com SPICE o per verificacions posteriors.

Generació d'un multiplicador

Al CNET de Grenoble estan interessats en l'obtenció d'un generador de filtres FIR de fase lineal en tecnologia CMOS de 1µm com a subsistema en la concepció de circuits complexes per aplicacions en sectors com el de la TV d'alta definició, el videotelèfon etc. La meva tasca ha estat, primerament, l'estudi de les arquitectures utilitzables pel filtre (optimitzant complexitat i regularirat tot respectant les especificacions previstes) i dins d'aquest, el multiplicador nucli de disseny del filtre ha estat l'objectiu principal d'aquest stage.

Hem arribat a la conclusió que un multiplicador que operi amb aritmètica redundant amb un operand en xifres amb signe (Signed Digit) i l'altre en complement a

dos (CA2) resultaria interessant tant per la realització de filtres FIR o IIR com per altres estructures generals.

L'estructura corresponent al multiplicador dissenyat és als annèxes. Cal adonar-se que malgrat operem en CA2 i aritmètica en SD, la complementació de les dades per implementar les sustraccions, típica en estructures clàssiques Carry-Save, es realitza de forma implícita. Això ens deixa una entrada suplementària lliure per una adició i en definitiva, la complexitat del circuit és molt similar a la del cas clàssic.

El Stage

El procès seguit ha començat per un estudi de les arquitectures pel filtratge FIR com les transversals directa i transposada, la descomposició Bit-Plane, la seva millora o Modified Bit-Plane i el tractament per blocs a alta velocitat o matriu-vector. Hem vist que per les aplicacions actuals en el terreny de les telecomunicacions tractades en aquest centre, una estructura del tipus transversal compleix amb les especificacions de regularitat, simplicitat i velocitat. Així doncs, partint d'aquesta base i deixant d'altres arquitectures més avançades per estudis posteriors, s'ha posat el multiplicador com objectiu central del stage.

Tot seguit, un estudi d'estructures i operadors ha estat realitzat per tal d'aprofondir en els conceptes de sistolisme, paral.lelisme i pipeline, així com en l'aritmètica redundant, els seus operadors i l'assimilació, arrodoniment i troncatura finals.

A partir de l'estudi dels operadors bàsics, s'ha iniciat la concepció en el mode descendent del multiplicador començant per la tria dels operadors redundants, el diagrama de fluxe o flow-graph de l'estructura (com matriu regular paral.lela-paral.lela d'operadors, descartant solucions poc regulars com els arbres de Wallace) fins arribar a l'especificació de les cèl.lules a nivell de transistor. Les verificacions i simulacions necessàries han estat realitzades al llarc d'aquest procès des dels models d'alt nivell fins els transistors MOS.

Conclusions

El resultat és doncs un estudi d'arquitectures de filtratge, un estudi sobre l'aritmètica redundant i un generador de multiplicadors fàcilment generalitzable a un generador de filtres FIR basat en l'aritmètica redundant en SD.

Cal remarcar, finalment, que el procès seguit i el resultats aconseguits venen a demostrar l'equivalència entre les estructures amb representació interna en SD i les clàssiques Carry-Save com diferents interpretacions del cas general de l'aritmètica redudant, la qual cosa no era del tot evident a priori.

2. Introduction au Contexte

2.1. Implantation VLSI et la génération automatique

L'implantation VLSI offre à l'ingénieur des performances sans précédents quant à sa capacité, sa puissance de traitement, son coût, sa consommation et sa taille réduites. La technologie permet au concepteur de placer des systèmes complets, de plus en plus complexes sur la même puce. Le niveau d'intégration a beaucoup changé pour les applications depuis les cartes imprimées pleines de circuits pour arriver finalement aux circuits spécifiques ou ASICs.

Cependant, cette complexité croissante comporte aussi un nombre de difficultés en augmentation notamment pour le dessin et la vérification du circuit. Le coût associé au produit, tombent fondamentalement sur l'étape de conception. On entrevoit des applications VLSI futures progressivement spécialisées et en conséquence produites en séries de plus en plus petites, or des outils de conception assistée performants capables de constituer un environnement complet de support au concepteur seront nécessaires. La communication entre l'utilisateur et la machine doit pouvoir être faite en un langage de haut niveau, voire un langage naturel.

L'ingénieur souhaiterait une méthode de conception plus automatisée, pour aboutir au résultat final sous la forme de schéma ou de layout en partant des spécifications générales du circuit. L'idée, appelée Compilateur de Silicium, est parallèle au concept de compilateur pour les informaticiens où une description de haut niveau génère une série d'instructions en code machine adaptées au processeur employé. Le compilateur de silicium aurait son résultat sous la forme d'ensembles de polygones qui représenteraient les masques. Les compilateurs de silicium se trouvent encore dans un état embryonnaire dû à la complexité et à la généralité du problème ainsi qu'à la difficulté pour définir ce que nous voulons. Les compilateurs de silicium existants sont basés souvent sur des bibliothèques de cellules standard ou traitent un spectre très limité de cas et d'applications.

Une première approche consisterait à employer la répétitivité de certains circuits pour les générer d'un façon automatique et les placer dans des systèmes plus complexes. Un générateur est un programme capable de produire un schéma ou un layout par placement et interconnexion des cellules à partir de certains paramètres fixés par l'utilisateur. La régularité et répétitivité des circuits à générer sont les contraintes principales pour l'emploi de générateurs, car la génération d'une structure quelconque sera d'autant plus difficile qu'elle possède de fortes irrégularités. Sa génération automatique sera déconseillée. En fait, la régularité peut s'exprimer pour un dessin comme la répétition d'une forme. En pratique, les structures matricielles rectangulaires ou linéaires sont des cas typiques de régularité, qui trouvent leur équivalent informatique par des boucles FOR et WHILE et permetent une certaine flexibilité avec des structures de contrôle IF-THEN-ELSE. Eléments de programmation des générateurs.

2.2. Les générateurs. Présentation du système GDT

Le système de dessin intégré GDT (Generator Development Tools) a été utilisé pour la réalisation du générateur de multiplieurs. GDT est un environnement complet de dessin, vérification et simulation de circuits autour d'une base de données textuellement décrite par son langage propre L. Cette base de données contient une partie technologique et une partie graphique.

La première comprend l'information reliée aux variables dépendantes de la technologie, ceci nous permet de travailler avec le même outil dans plusieurs technologies différentes.

La deuxième est une collection de données et d'objets qui constituent un dessin. Les objets sont les suivants :

- Cellules : groupes d'autres objets à un niveau d'hiérarchie donné
- Instances : appel d'une cellule à un niveau de hiérarchie supérieur
- Arrays : tableaux de cellules
- Terminals : points de connexion d'une cellule avec l'extérieur

- Transistors : dispositifs actifs MOS
- Fils : connexions physiques entre objets de même niveau physique
- Contacts : connexions entre objets de différents niveaux physiques
- Propriétés : informations supplémentaires

Chaque objet peut avoir des attributs associés tels que la taille des transistors, la largeur pour les fils, la position, la connexion et l'orientation, etc. L'information géométrique contenue dans la base de données est utilisée pour l'édition graphique, la compaction, la vérification et la simulation des circuits. L'organisation hiérarchique de la base de données nous permet d'effectuer et de gérer de grands circuits. Enfin, la vérification des règles de dessin et l'extraction de netlists pour des simulations logiques avec Lsim (partie très puissante de GDT) ou électriques avec SPICE sont prévues.

La description du contenu de la base de données est faite textuellement avec le langage propre L. Au niveau le plus bas, L comprend des primitives géométriques comme des rectangles des polygones et du texte, au niveau le plus élevé L comprend les primitives électriques et géométriques telles que les fils, les transistors, les contacts, les terminaux et les éléments de hiérarchie appelés cellules. Le langage L accepte des variables et des opérations arithmétiques de base avec l'inclusion des structures de contrôle conditionnel (IF-THEN-ELSE), de boucle (WHILE) et des instructions de routage automatique. Il nous permet d'appeler d'autres programmes écrits soit en L, soit exécutables sous UNIX et de construire ainsi des générateurs. Cependant, L n'est pas un langage évolué avec la puissance de ADA ou de C, car il est plutôt conçu pour gérer le placement, l'assemblage et le routage d'objets et de cellules. Ainsi, les traitements algorithmiques complexes doivent être effectués de façon externe avant la phase d'assemblage proprement dite.

3. Etude des Structures pour le Filtrage à Réponse Impulsionnelle Finie

Il est déjà classique de dire que le domaine du traitement du signal a pris récemment une très grande importance. Ce fait est accepté par tous, l'évolution de la technologie VLSI apporte une croissance de la puissance et des performances des applications. Cependant, la vraie importance de cette révolution est l'existence d'un passage direct entre l'algorithme et le matériel, entre l'équation et la puce en silicium. Autrement dit, le compilateur de silicium, comme on l'a vu dans le chapitre d'introduction à la génération automatique, permettra une conception rapide du layout du circuit à partir du cahier des charges ou des spécifications de haut niveau.

Les domaines classiques d'application sont très vastes et on peut les classer arbitrairement en :

- les applications des télécommunications avec la numérisation des réseaux, les codages, la compression et le traitement des informations, le traitement de la parole et des images, etc.

- les applications grand-public qui deviennent de plus en plus importantes grâce à l'amélioration des technologies (produits plus performants et moins chers), à l'apport des résultats professionnels en traitement du signal (CD vidéo-disque et TV numérique) et télécom (réseau numérique à intégration de services) dans le marché de consommation.

Une première tâche générique du traitement du signal est le filtrage, intervenant dans la plupart des applications classiques. Notamment, le filtrage transversal à réponse impulsionnelle finie (FIR) prend une importance capitale, notion à conserver pour la suite de l'ouvrage. En effet, malgré la nécessité d'un nombre plus élevé de coefficients pour atteindre les mêmes caractéristiques en fréquence que les filtres à réponse

impulsionnelle infinie (IIR), ils nous présentent les avantages de la stabilité et la linéarité de phase.

3.1. Les filtres FIR et FIR à phase linéaire

Dans certaines applications, la phase à comportement linéaire est une contrainte à atteindre. Ceci a posé beaucoup de problèmes aux théoriciens et ingénieurs à l'époque des études sur le filtrage analogique. L'apparition du traitement numérique et des études sur les filtres FIR a permis un contrôle plus précis de la phase. Le dessin des filtres à phase linéaire a alors été facilement réalisable.

En général, un filtre peut être représenté dans le domaine frequentiel comme un quotient de polynômes en z:

$$H(z) = \frac{N(z)}{D(z)}$$

N(z): Partie non récursive du filtre.
D(z): Partie récursive du filtre.

Et on sait que la phase peut être calculée comme:

$$\Phi = \frac{1}{2j} \ln\left[\frac{H(z)}{H(1/z)}\right]_{z=e^{jw}} = \frac{1}{2j} \ln\left[\frac{N(z)}{D(z)}\frac{D(1/z)}{N(1/z)}\right]_{z=e^{jw}}$$

Pour disposer d'une phase linéaire, on doit vérifier que :

$$\frac{N(z)}{N(1/z)}\frac{D(1/z)}{D(z)} = z^{-M}$$

.

On aura donc finalement :

$$\Phi(w) = \frac{1}{2j} \ln\left[z^{-M}\right]_{z=e^{jw}} = -\frac{M}{2}w$$

qui représente une ligne droite de pente -M/2 suivant la fréquence. Un système comme celui-ci doit avoir ses pôles et ses zéros en symétrie homothétique pour compenser les racines en (z) et en (1/z) de l'expression antérieure. La contrainte de la stabilité du système nous oblige à imposer D(z) = 1 pour ne pas avoir des pôles à l'extérieur du cercle unitaire de H(z) et assurer sa convergence. Or, on ne peut avoir des filtres à phase linéaire qu'avec des structures non récursives ou FIR.

Avec la notation employée ci-dessus et si l'ordre du filtre vaut M, la longueur de la réponse est L = M+1. Le comportement de la phase et du temps de groupe (ou de latence) sont exprimés respectivement par:

$$\Phi(w) = -\frac{L-1}{2}w$$

$$Tg(w) = -\frac{\partial\Phi(w)}{\partial w} = \frac{L-1}{2}$$

On peut déjà observer un éventuel problème dans les cas de filtres avec L pair, car alors, la relation entre les temps de latence et d'échantillonnage ne sera pas entière.

En fonction des valeurs de L et du type de symétrie des coefficients, on peut distinguer quatre sortes de filtres FIR à phase linéaire avec les caractéristiques suivantes:

Type 1.

$H_1(z)$ ne présente pas de zéros en $z = \pm 1$

L impaire

Symétrie paire des coefficients

$$h\left(\frac{L-1}{2}+j\right) = h\left(\frac{L-1}{2}-j\right); \qquad j = 0 \ldots \frac{L-1}{2}$$

$$h(n) = h(L-1-n); \qquad n = 0 \ldots L-1$$

La réponse de ce type de filtre est

$$H\left(e^{jw}\right) = e^{-j\frac{L-1}{2}w} H_R(w) \qquad \text{avec}$$

$$H_R(w) = \sum_{i=0}^{\frac{L-1}{2}} a_i \cos iw$$

On peut observer un temps de latence en relation entière avec la période d'échantillonnage. Le filtre présente aussi un zéro de transmission en fe/4 (fe: fréquence d'échantillonnage). Ceci peut nous empêcher de réaliser des filtres au comportement fréquentiel passe-bande.

Type 2.

Ajout d'un zéro de multiplicité impaire en z = -1
par rapport à la structure précédente.

$$H_2(z) = (1+1/z)\, H_1(z)$$

L paire
Symétrie paire des coefficients

$$h\left(\frac{L}{2}+j\right) = h\left(\frac{L}{2}-1-j\right); \qquad j = 0 \ldots \frac{L}{2}-1$$

$$h(n) = h(L-1-n); \qquad n = 0 \ldots L-1$$

$$H(e^{jw}) = e^{-j\frac{L-1}{2}w} H_R(w) \qquad \text{avec}$$

$$H_R(w) = 2\cos\frac{w}{2} \sum_{i=0}^{\frac{L}{2}-1} a_i \cos iw$$

Ce type de filtre ne permet pas la réalisation de filtres passe-haut et présente un temps de latence à la sortie en relation non entière avec fe.

Type 3.

Ajout d'un zéro de multiplicité impaire en z = 1 par
rapport à la structure de type 1.

$$H_3(z) = (1\text{-}1/z)\, H_1(z)$$

L paire
symétrie impaire des coefficients

$$h\left(\frac{L}{2} + j\right) = -h\left(\frac{L}{2} - 1 - j\right); \qquad j = 0 \ldots \frac{L}{2} - 1$$

$$h(n) = -h(L - 1 - n); \qquad n = 0 \ldots L - 1$$

$$H(e^{jw}) = je^{-j\frac{L-1}{2}w} H_R(w) \qquad \text{avec}$$

$$H_R(w) = 2\sin\frac{w}{2} \sum_{i=0}^{\frac{L}{2}-1} a_i \cos iw$$

Ce type de filtre ne peut pas conduire à un passe-bas et présente, d'autre part, un temps de groupe non entier. Il faut noter aussi un décalage constant de phase de π/2.

Type 4.

Ajout des zéros en multiplicité impaire en z = ± 1 par rapport au type 1.

$$H_4(z) = (1\text{-}1/z^2)\, H_1(z)$$

L impaire

Symétrie impaire

$$h\left(\frac{L-1}{2}+j\right) = -\,h\left(\frac{L-1}{2}-j\right); \qquad j = 0 \ldots \frac{L-1}{2}$$

$$h(n) = -\,h(L-1-n); \qquad n = 0 \ldots L-1$$

$$h\left(\frac{L-1}{2}\right) = 0$$

$$H\left(e^{jw}\right) = je^{-j\frac{L-1}{2}w}\, H_R(w) \qquad \text{avec}$$

$$H_R(w) = 2\sin w \sum_{i=0}^{\frac{L-3}{2}} a_i \cos iw$$

Comme dans le cas précédent, nous retrouvons un décalage constant de la phase de π/2. Cependant, le temps de latence est entier. Mais, ce type de filtre ne peut être employé que dans la réalisation de passe-bandes.

Le fait d'avoir un temps de latence de sortie en relation non entière avec la période d'échantillonnage peut être très gênant dans certaines applications. Cela nous conduit à nous limiter aux deux types 1 et 4 qui nous permettront malgré tout d'observer tous les comportements en fréquence. Le décalage de la phase des types 3 et 4 peut être utilisé pour le dessin des différentiateurs et des transformateurs de Hilbert dont les réponses seront :

$$H\left(e^{jw}\right) = 1 - e^{-jw} \qquad \text{et}$$

$$H(w) = \begin{cases} -j & 0 \le w \le \pi \\ +j & -\pi \le w < 0 \end{cases}$$

3.2. Etude des structures pour le filtrage FIR

Le projet a pour but l'étude de l'implantation automatique de structures de traitement du signal et doit aboutir à la réalisation d'un générateur de filtres FIR qui sera employé dans les applications de conception des circuits intégrés.

Le choix d'une structure pour les filtres doit être considéré comme un pas préliminaire à toute réalisation. Plusieurs degrés de liberté doivent être fixés : codage des opérandes, largeur du chemin de données, architecture interne des opérateurs, etc… Une fois que le choix est fixé, certaines contraintes apparaissent (débit maximum, complexité mal pondérée pour des applications plus simples, incompatibilité avec certains types d'opérandes, etc…). En conséquence, il est impossible d'établir un choix optimal et général pour toutes ou la plupart des applications surtout celles qui ne sont pas encore totalement définies, comme la TVHD. Un choix basé sur l'optimisation de certaines prestations ou caractéristiques est encore très difficile à cause des différences intrinsèques entre les structures (comparaison entre éléments de différente nature) et de la quantité de variations qu'on peut proposer pour chacune.

L'étude qui suit est en conséquence approximative et elle est basée sur la comparaison entre certaines structures connues et d'autres proposées dans la documentation spécialisée. Les résultats ne sont pas exhaustifs, c'est pourquoi l'étude des ordres de magnitude n'est pas faite au MOS près, ce qui ne serait d'ailleurs pas réaliste. Donc, dans le cas d'une étude structurelle comme celle-ci, ne seront pas pris en compte les parties périphériques telles que les amplis d'horloge, les buffers locaux, etc.

Une première approche consiste à considérer tous les multiplieurs du filtre comme des matrices régulières d'additionneurs complets (FA) en propagation série de retenue (Carry Ripple Adder, CRA) ou en représentation redondante du graphe de dépendance (Carry Save Adder, CSA). Ces multiplieurs auront une largeur du chemin de données de sortie constante et des opérandes codés en position et sans signe (binaire naturel). Par conséquent, l'utilisation d'un codage des coefficients en notation canonique signée (CSD) (voir chapitre 5.1) qui simplifie les multiplieurs est incorrecte parce que les opérateurs signés ne sont pas pris en compte. Cependant, on verra plus loin que l'on peut aboutir à une structure de multiplieur en complément à deux (CA2) et à chiffres signés (SD) comparable en complexité à celle du cas binaire-binaire et l'employer dans le cas CA2-CSD qui sera une particularisation du SD vers le CSD en coefficients fixés.

Pour la réalisation de cette étude, considérons un filtre générique d'ordre N avec N' coefficients distincts (FIR éventuellement symétrique) codés sur m digits (dont m' sont différents de zéro en CSD). Les données sont codées sur n bits et les résultats partiels sont conservés entiers jusqu'à la sortie du filtre où pourra être envisagé un arrondi ou une troncature. Les cas des multiplieurs basés sur des structures CRA et CSA sont traités séparément.

Le pipelinage des multiplieurs augmente le nombre de transistors de chaque structure de la même quantité à style d'additionneurs identique. Les valeurs absolues de ces structures augmenteront dans la même proportion, par contre, la différence entre les complexités des structures étudiées ne sera pas affectée par le pipelinage. Par conséquent, il ne sera pas tenu compte du fait que les multiplieurs sont pipelinés ou non.

Dans un premier temps, nous considèrerons les structures transversales classiques comme des variantes des structures directe et transposée, puis seront introduites des structures plus innovatrices comme la bit-plane, modified-bit-plane et la matrice-vecteur (traitement par blocs). La fonction de transfert que l'on espère atteindre est donnée par l'expression de la convolution :

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

La notation employée est la suivante :

N	Ordre du filtre
N'	Nombre de coefficients distincts du filtre
/log$_2$ N/	Le plus petit entier supérieur au log$_2$ de N
k:	Nombre de cellules entre étages de pipeline
Tadd	Temps pour additionner 2 chiffres de n bits
ts, tr	Temps de somme et retenue d'un additionneur
(N'/k)	Nombre réel de lignes de pipeline du filtre
VMA	Ajout supplémentaire d'une assimilation en sortie

On considère les données codées sur n bits et les coefficients codés sur m bits dont m' sont différents de zéro. Pour un codage CSD des coefficients, on peut supposer que m' = m/3 en moyenne.

3.2.1. Structure Transposée

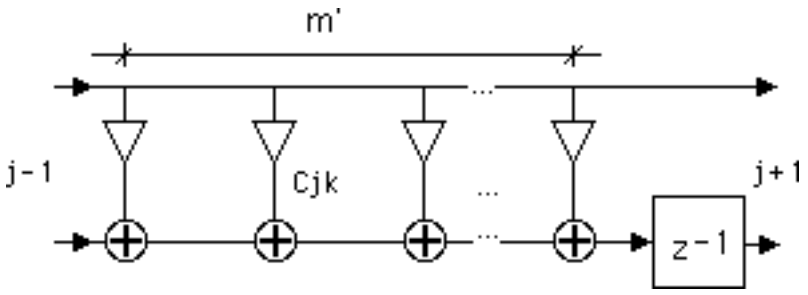


Figure 1. Structure transposée

1. Additionneurs en carry-ripple

1.1. Filtre non symétrique

largeur du chemin de sortie	n+m+/log2 N/ bits
multiplieur = m' add's de	n+m+/log2 N/ bits
nombre total registres/cellule	n+m+/log2 N/
nombre total de cellules	N
nb total d'additionneurs	m'N[n+m+/log2 N/]
nb total de registres	N[n+m+/log2 N/]
Temps total par opération	T = m'ts + (n+m+/log2 N/)tr

1.2. Cas symétrique (phase linéaire)

Maintenant, on peut considérer que l'on passe de N à N' étages dans le filtre et que l'on somme des données convenablement décalées avant le produit par le coefficient (Figure 2).

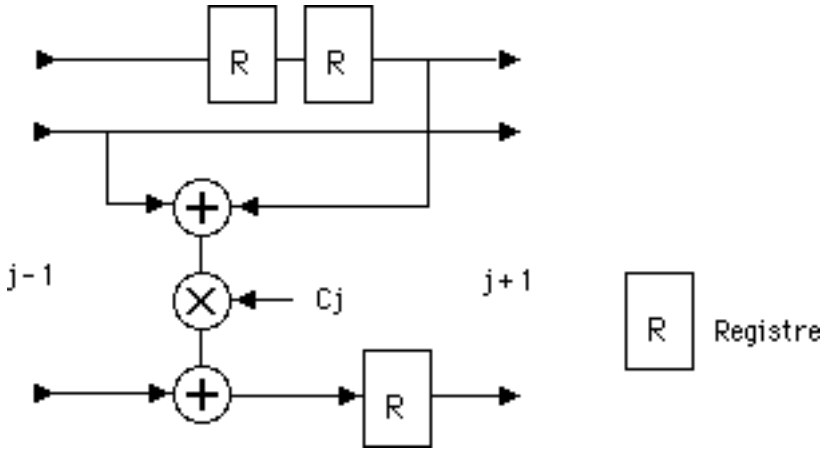


Figure 2. Etage d'un filtre symétrique (structure transposée)

nb de cellules	N'
nb total de registres/cellule	n+m+/log2 N/+2n
nb total d'additionneurs	m'N'[n+1+m+/log2N/]+(n+1)N'
nb total de registres	N'[n+m+/log2 N/+2n]
Temps tot. sans pipeline	T' = T + Tadd = T + ts

Temps tot. avec pipeline	T" = max (T,Tadd) = T

Le premier temps calculé correspond au cas non pipeliné entre l'addition des échantillons et le produit par le coefficient, le deuxième résultat considère une ligne de pipeline horizontale entre additionneur et multiplieur, avec l'ajout de (n+1)N' registres.

2. Carry-save

Il faut noter que l'arithmétique carry-save nous impose de doubler le nombre de registres du chemin de sortie parce que la redondance entre étages du filtre a été conservée. Le résultat sera assimilé à la sortie du filtre par un Vector Merge Adder (VMA).

2.1. Cas non symétrique

largeur du chemin de sortie	n+m+/log2 N/ bits
multiplieur = m' add's de	n+m+/log2 N/ bits
nombre total registres/cellule	2[n+m+/log2 N/]
nombre total de cellules	N
nb total d'additionneurs	m'N[n+m+/log2 N/] + VMA
nb total de registres	2N[n+m+/log2 N/] + VMA
Temps tot. par opération	T = m' max (ts',tr')

2.2. Filtre aux coefficients symétriques

nombre de cellules	N'
nombre total registres/cell	2[n+m+/log2 N/]+2n
nombre total add's	m'N'[n+1+m+ +/log2 N/]+(n+1)N'+VMA
nombre tot de registres	2N'[n+m+/log2 N/]+2nN'+VMA
Temps sans pipeline add/mult	T' = T + Tadd = T + ts + ntr
Temps avec pipeline	T" = max (T,Tadd)

3.2.2. Structure Directe

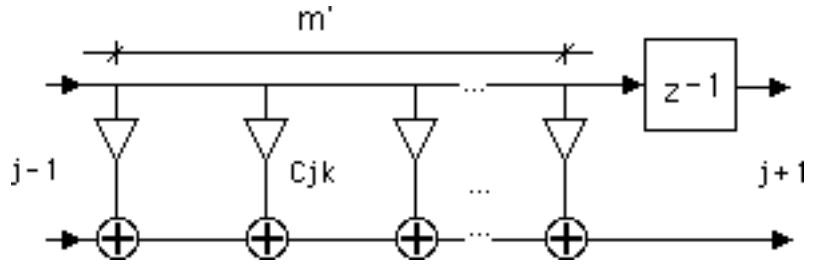


Figure 3. Structure directe

1. Carry-ripple

1.1. Cas non symétrique

largeur du chemin de sortie	n+m+/log2 N/ bits
multiplieur = m' add's de	n+m+/log2 N/ bits
nombre total registres/cellule	n
nombre total de cellules	N
nb total d'additionneurs	m'N[n+m+/log2 N/]
nb total de registres	nN+(2n+m+/log2 N/)(N'/k)
Temps par opération	T = km'ts + (n+m+/log2 N/)tr

1.2. Cas symétrique

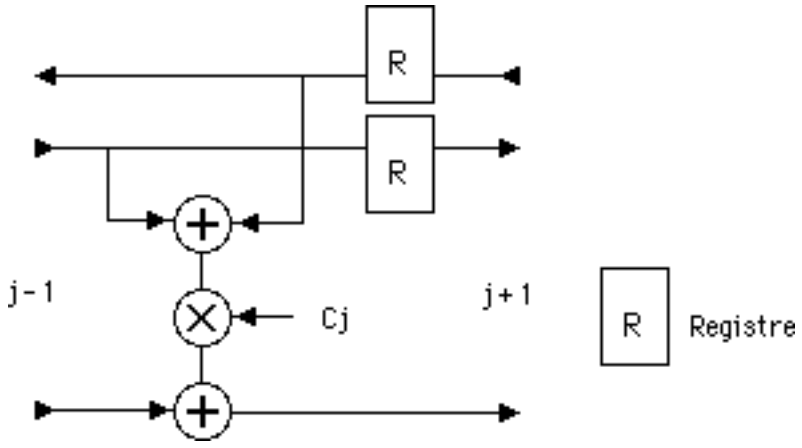


Figure 4. Etage d'un filtre symétrique (semi-systolique)

nb de cellules	N'
nb total de registres/cellule	2n
nb total d'additionneurs	m'N'[n+1+m+/log2 N/]+(n+1)N'
nb total de registres	2nN'+(N'/k)[n+m+/log2 N/]
Temps sans pipeline	T' = T + Tadd = T + ts
Temps avec pipeline	T'' = max (T,Tadd) = T

2. Carry-save

2.1. Cas non symétrique

largeur du chemin de sortie	n+m+/log2 N/ bits
multiplieur = m' add's de	n+m+/log2 N/ bits
nombre total registres/cellule	n
nombre total de cellules	N
nb total d'additionneurs	m'N[n+m+/log2 N/] + VMA
nb total registres	nN+(N'/k)(n+2[n+m+/log2 N/])+VMA
Temps par opération	T = km' max (ts',tr')

2.2. Cas symétrique

nombre de cellules	N'
nb total de registres/cellule	2n
nb total d'additionneurs	m'N'[n+1+m+/log2N/]+ +(n+1)N'+VMA
nombre total de registres	2nN'+2(N'/k)[n+m+/log2 N/] + VMA
Temps sans pipeline	T' = T + Tadd = T + ts + ntr
Temps avec pipeline	T'' = max (T,Tadd)

3.2.3. Structure Bit-Plane

L'implémentation directe des structures étudiées jusqu'à présent peut comporter des problèmes de débit quand la contrainte est élevée donc on est toujours limité par la

rapidité des opérateurs élémentaires comme additionneurs et multiplieurs. Une solution possible passe par l'augmentation du degré de pipeline (voir chapitre 4) mais certains auteurs /5,20,22/ ont montré qu'il est possible d'aboutir à des structures plus régulières et flexibles grâce à un développement différent décrit ci-dessous :

Si l'on prend l'expression du produit de convolution et que l'on représente le coefficient dans sa représentation codée en CA2, on voit que l'on peut décomposer l'équation en une somme d'expressions plus simples selon :

$$
\begin{aligned}
y(n) &= \sum_{k=0}^{N-1} x(n-k)h(k) \\
&= \sum_{k=0}^{N-1} x(n-k)\left[-2^{0}a_{k,m-1} + 2^{-1}a_{k,m-2} + \ldots 2^{-(m-1)}a_{k,0}\right] \\
&= -2^{0}\sum_{k=0}^{N-1} x(n-k)a_{k,m-1} + \ldots 2^{-(m-1)}\sum_{k=0}^{N-1} x(n-k)a_{k,0} \\
\text{avec:} \quad & h(k) = -2^{0}a_{k,m-1} + \ldots 2^{-(m-1)}a_{k,0}
\end{aligned}
$$

Les coefficients du filtre sont éclatés et regroupés en fonction de leur poids dans la nouvelle structure et les produits sont dégénérés pour devenir un simple fil. Généralement, on parle de filtres de m plans avec N lignes par plan (Figure 5).

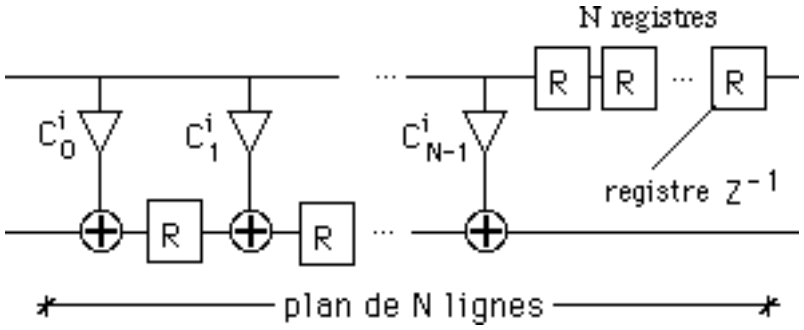


Figure 5. Un plan pour une structure bit-plane

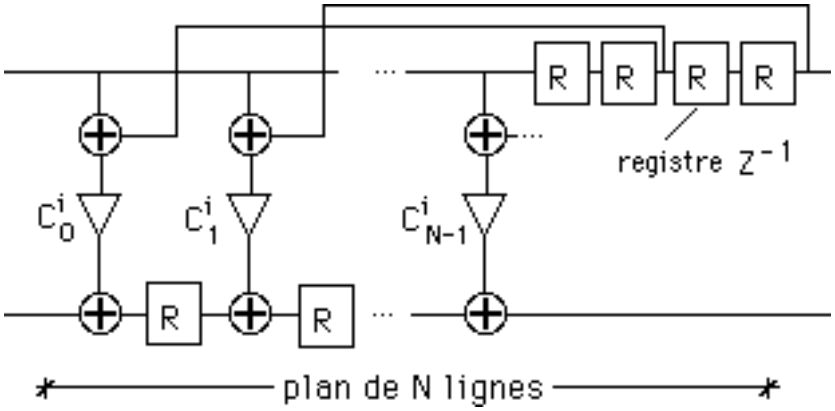


Figure 6. Un plan pour une structure bit-plane symétrique

1. Carry-ripple

Il faut souligner que ce cas ne présente en fait aucun intérêt. On a en effet une structure pipelinée au maximum mais la propagation de la retenue fait tout perdre. Une structure carry save partiellement pipelinée est plus efficace et moins couteuse.

1.1. Cas non symétrique

m plans

N lignes/plan et m'N lignes par filtre sur mN lignes en tout

Largeur du chemin de sortie	n+/log2 N/
nb d'additionneurs/ligne	n+/log2 N/
nb de registres/ligne	n+/log2 N/
nb de registres/plan	2nN+N/log2 N/
nb total d'additionneurs	m'N[n+/log2 N/]
nb total de registres	mN[2n+/log2 N/]
Temps par opération	T = ts + (n+/log2 N/)tr

1.2. Cas symétrique

m plans

N' lignes/plan et m'N' lignes/filtre

nb d'additionneurs/ligne	n+/log2 N/+n
nb total d'additionneurs	m'N'[2n+/log2 N/]
nb total de registres	mN'[n+/log2 N/]+mnN
temps sans pipeline	T' = T + Tadd = T + ts
temps avec pipeline	T" = max (T,Tadd) = T

2. Carry-save

2.1. Cas non symétrique

m plans

N lignes/plan et m'N lignes par filtre

Largeur du chemin de sortie	n+/log2 N/
nb d'additionneurs/ligne	n+/log2 N/
nb de registres/ligne	2n+2/log2 N/
nb de registres/plan	3nN+2N/log2 N/
nb total d'additionneurs	m'N[n+/log2 N/] + VMA
nb total de registres	3nmN+2Nm/log2 N/ + VMA
temps par opération	T = max (ts',tr')

2.2. Cas symétrique

m plans

N' lignes/plan et m'N' lignes/filtre

nb d'additionneurs/ligne	n+/log2 N/+n
nb total d'additionneurs	m'N'[2n+/log2 N/] + VMA
nb total de registres	mN'[2n+2/log2 N/]+mnN+VMA
temps sans pipeline	T' = T + Tadd = T + ts + ntr
temps avec pipeline	T" = max (T,Tadd)

Grâce à cette stratégie on travaille au niveau du bit ce qui permet un pipelinage maximal et très régulier. Les structures résultantes peuvent opérer aux débits les plus hauts. On verra plus tard que ces structures aboutissent à un coût matériel très important, désavantage encore plus évident pour les filtres d'ordre moyen et petit par rapport aux structures antérieures (voir tableaux comparatifs, annexe).

3.2.4. Structure Modified Bit-Plane

Le principal désavantage du bit-plane est l'énorme nombre de registres nécessaire. Ce facteur est très important parce qu'il influe directement sur la taille du circuit. Une nouvelle structure est proposée /20,22/ avec le regroupement de deux bits par coefficient (Figure 7) pour former les plans modifiés où les registres sont éclatés en demi-registres.

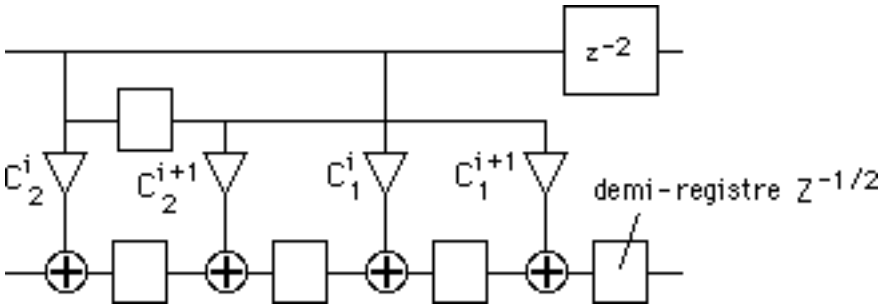


Figure 7. Le principe du modified bit-plane

Les tableaux confectionnés avec les expressions obtenues montrent que l'intérêt des structures bit-plane et modified bit-plane est de réaliser des filtres d'ordres élevés avec une contrainte très exigeante sur le débit. Les applications que l'on envisage de couvrir avec le générateur de filtres sont satisfaites dans la plus part des cas par des structures traditionnelles moins gourmandes en matériel. Une autre raison qui nous a fait abandonner les bit-plane est l'intérêt d'avoir un générateur de multiplieurs, partie intrinsèque ou niveau hiérarchique inférieur du générateur de filtres, qui puisse servir pour la génération de multiplieurs d'une façon générale (stand alone).

1. Carry-save. Cas non symétrique

m/2 plans, 2N lignes/plan et m'N lignes par filtre

Largeur du chemin de sortie	n+1+/log2 N+log2 3/
nb d'additionneurs/ligne	n+1+/log2 N+log2 3/
nb de demi-registres/ligne	n+1+/log2 N+log2 3/
nb total d'additionneurs	m'/2 2N(n+1+/log2 N+log2 3/)
nb total de registres	(n/2-1)nN+mN(n+1+/log2 N +log2 3/)+nm/4 + VMA
temps par opération	T = 2 max (ts',tr')

3.2.5. Matrice-vecteur, filtrage par blocs

Un filtre FIR, et en conséquence une convolution, peut être considéré comme un produit entre la matrice nxm des données et un vecteur mx1 des coefficients ou vice versa. L'équation du produit matrice-vecteur est la suivante :

$$\vec{C} = \vec{A} \times \vec{B} \rightarrow c_i = \sum_{j=1}^{n} a_{ij} b_j ; \qquad 1 \le i \le n$$

qui en forme matricielle apparaît comme :

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} a_{11} \cdots & a_{1n} \\ a_{21} & \vdots \\ \vdots & \\ a_{n1} \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Si l'on considère maintenant la relation entre le vecteur de sortie Y, le vecteur des données X et le vecteur des coefficients du filtre comme suit :

$$\vec{Y} = \vec{X} \times \vec{H}, \quad y_k = \vec{X}_k \times \vec{H}$$

et que l'on construit la matrice avec X_k vecteur formé par les n derniers échantillons à l'instant k, on est capables de calculer n échantillons du signal de sortie selon :

$$\begin{bmatrix} y_k \\ y_{k-1} \\ y_{k-2} \\ \vdots \\ y_{k-n+1} \end{bmatrix} = \begin{bmatrix} \vec{X}_k \\ \vec{X}_{k-1} \\ \vdots \\ \vec{X}_{k-n+1} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{n-1} \end{bmatrix}$$

Cette idée peut paraître très intéressante, mais le coût matériel qui nous amène ce degré de parallélisme est trop élevé pour les besoins des applications de télécom, qui sont en général plus modestes. Cependant, ces structures sont à considérer comme des structures dont les débits doivent être supérieurs à ceux obtenus avec un pipeline total et une structure classique (une seule donnée en entrée par cycle). Le parallélisme du réseau nous permet de travailler en interne avec des fréquences inférieures au débit grâce à une conversion série-paralléle (SIPO) en entrée et paralléle-série (PISO) à la sortie (Figure 8).

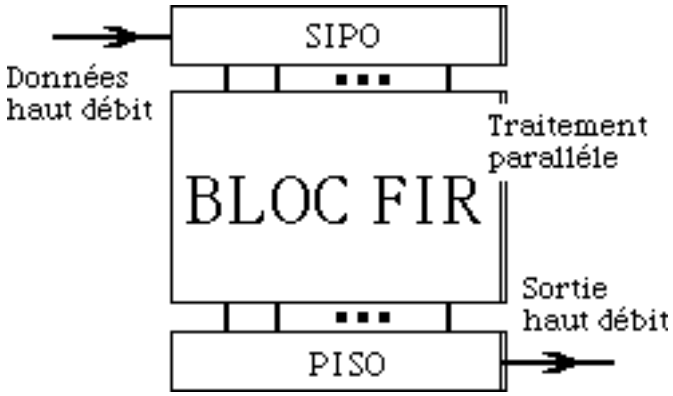


Figure 8. Le traitement par blocs à haut débit

Comme conclusion à ce chapitre on peut dire que la réalisation automatique de filtres FIR pour le traitement du signal dans le domaine des télécommunications est

possible dans la majorité des cas avec des structures traditionnelles. La contrainte de débit, fixée par l'application, peut être satisfaite avec une structure suffisamment régulière et simple pour envisager sa génération automatique.

4. Notions de Parallélisme, de Pipelinage et de Systolisme

Les chapitres suivants utilisent certains acquis sur les notions élémentaires de parallélisme, pipelinage et systolisme car ce sont des concepts fondamentaux dans toute étude sur les architectures de traitement du signal. C'est pour cette raison que l'inclusion de ce chapitre a été jugée nécessaire.

Parallélisme

La complexité croissante des circuits actuels implique l'impossibilité de concevoir les systèmes comme des éléments processeurs uniques. Alors, le système est vu comme un ensemble de processus qui, pour aboutir à un débit satisfaisant dans les applications, doivent travailler en parallèle. Le parallélisme peut être considéré à plusieurs niveaux /17/ dont la table suivante effectue un classement par la taille du processus :

Niveau	Opération	Temps	Interaction
travail	échange des données	1 sec	software du système
tâche	échange des données	1 ms	système et soft utlilisateurs
Instruction	échange et op's complexes	1 µs	soft utilisateur et hardware
µ-instruction	func. binaires et combinat.	10 ns	hardware

Table 1. Niveaux de parallélisme

Le parallélisme au niveau du travail est géré par le software dans un environnement multiprocesseur. Le disque dur ou des buffers RAM sont le moyen d'échange de fichiers. Parce que les constantes de temps sont assez longues, l'emploi de

communications asynchrones entre processus ne dégrade pas les performances du système et permet une certaine flexibilité.

Le parallélisme au niveau des tâches est caractérisé par une constante de temps très inférieure à la précédente et un échange de données de taille beaucoup plus petite, telle que des mots ou même des signaux binaires. Le "soft" du système et aussi celui de l'utilisateur doivent synchroniser les processus et la communication entre les tâches. Le DMA (Direct Memory Access) pour les ordinateurs personnels ainsi que les environnements multi-tâches des stations de travail en sont des exemples.

A l'étage au dessous, le niveau des instructions consiste à partager les données entrantes ou sortantes des processeurs pendant la même tâche. La synchronisation et le partage des ressources entre eux devient un vrai problème à résoudre par le "hard" quand les solutions apportées par le soft se montrent inefficaces. L'aspect le plus important du dessin à ce niveau est le contrôle de la communication entre processeurs.

Le niveau des micro-instructions, autrement dit sous-opérations, est caractérisé par une constante de temps proche du délai des portes logiques et, en conséquence, dépendante de la technologie employée. Ici, plusieurs processeurs élémentaires (PE) travaillent concurremment pour chaque instruction machine ou coup d'horloge. La technique qui permet une synchronisation entre ses courtes constantes de temps s'appelle pipeline. Le pipeline se traduit par une période opératoire (temps entre les résultats successifs) strictement inférieure au délai de calcul (temps nécessaire pour calculer la fonction). Cette technique n'est pas du tout nouvelle car elle est connue et employée en production depuis la fabrication de la Ford T.

D'une façon plus formelle, on peut "mesurer" le parallélisme à partir de relations simples du type :

$$\prod = N/I \qquad \text{avec } 1 \leq \prod \leq N \text{ et où}$$

$\prod$ est le degré de parallélisme

N est le nombre d'opérations élémentaires à effectuer pour le calcul de chaque valeur successive de la fonction

I est le nombre d'itérations de calcul de la fonction nécessaires pour effectuer ces N opérations

Par exemple, une chaîne de calcul en 4 étapes peut être décomposée en deux parties de deux étapes chacune en les séparant par une mémoire où la première place ses résultats partiels et la deuxième les prend. Ainsi les données peuvent entrer avec une période de deux itérations et les résultats sont sortis avec la même période bien que le temps entre une entrée et la sortie associée soit supérieur ou égal (si l'on considère aussi le temps de traversée de la mémoire) à celui de la chaîne complète. Le degré de parallélisme est évidemment de 0,5 et la mémoire centrale représente le pipelinage. Enfin, les valeurs extrêmes pour le degré de parallélisme sont $\prod = 1$ pour une architecture complètement série et $\prod = N$ pour une architecture complètement parallèle.

Pipelinage

Le pipelinage est l'application du parallélisme au niveau de la sous-opération. Cela signifie que les opérations sont effectuées de façon concurrente par des parties de "hard" isolées.

Soit $y = g(x)$ la fonction à réaliser. $g(x)$ peut être décomposée selon :

$$g(x) = h_f(h_{f-1}(\ldots h_2(h_1(x))\ldots))$$

qui peut s'implémenter "en ligne" de manière à ce que l'on calcule chaque fonction hi en disposant les registres entre ces sous-fonctions pour les effectuer simultanément. f évaluations de g peuvent donc être traitées en parallèle (Figure 1).

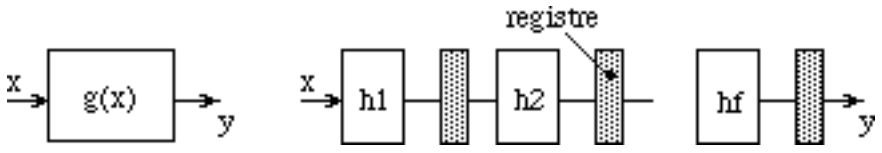


Figure 1. Le principe du pipelinage

Normalement la fonction g est complexe ou longue à calculer mais elle a une décomposition simple en termes de f sous-fonctions hi (parce qu'elle est une longue succession d'opérations simples, filtre, corrélateur, FFT). L'équilibre exigé pour une architecture pipelinée impose l'ensemble des fonctions hi avec des temps de calcul très similaires, parce que c'est le temps de propagation le long de la chaîne de logique combinatoire la plus longue, entre deux étapes de registres, qui déterminera le débit.

Si le délai de calcul est T alors, chaque hi devrait posséder un délai associé t pour vérifier l'égalité T = f x t. Soit une période d'horloge tp où tp = t + t1 avec t1 le délai introduit par un registre et les relations suivantes sont valides :

Le premier résultat sort en un temps de : $ts1 = f \times tp = f \times (t + t1)$

et les résultats subséquents en : $tsn = (f - 1 + n) \times (t + t1)$

en comparaison avec une structure non pipelinée : $tsn' = n \times f \times t = nT$

Il faut constater que pour un n important et pour des petites valeurs de t1 (ce qui est d'ailleurs très réaliste) l'avantage est significatif et représente une augmentation du débit. Cependant, le pipelinage ne peut pas être toujours utilisé car dans certaines structures (graphe dont tous les arcs ne sont pas orientés dans le même sens) dont les boucles sont un exemple important, l'ajout de délais implique une renormalisation de l'unité de temps pour maintenir le séquencement.

Systolisme

Une architecture systolique est constituée d'un réseau pipeliné de processeurs élémentaires possédant leur propre mémoire et connectés seulement à leur proches voisins. Le parallélisme résultant est maximal. Certaines caractéristiques ou propriétés peuvent être citées ici :

- Ce sont des architectures constituées de la répétition régulière, suivant une maille à une ou deux dimensions (ou n) (Figure 2) d'un certain nombre de processeurs élémentaires tous égaux à l'exception des cellules des bords.

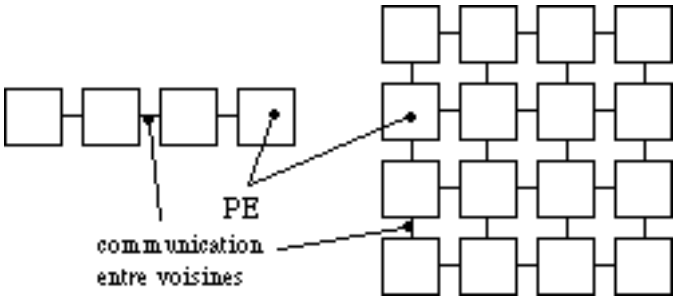


Figure 2. Réseaux systoliques en 1 et 2 dimensions

- A chaque cycle, d'une horloge qui peut être unique les cellules reçoivent des données de leur voisines et effectuent une opération spécifique. Le résultat est stocké et sera transmis aux cellules suivantes au prochain coup d'horloge. C'est un fonctionnement rythmique. Les données ne peuvent entrer et sortir que par les extrémités du réseau et toute sorte de branchement conditionnel ou de rebouclage est impossible.

- Les communications sont temporellement et spatialement locales entre processeurs élémentaires du réseau. Spatialement parce que toute diffusion d'une donnée à plusieurs PE ou d'un ensemble de données vers le même PE sont interdites. Temporellement parce que les calculs s'effectuent en cascade sur plusieurs PE toujours séparés par des barrières temporelles. Ceci implique que la fréquence de travail des PE et aussi le débit de la structure sont indépendants de la taille de cette dernière.

En fait, le seul signal qui est propagé dans tout le réseau (et qui viole le principe de la localité) est l'horloge, ce qui peut créer de sérieux problèmes pour des tableaux de PE à deux dimensions dont la distribution d'horloge représente un point critique à l'étape de conception. Plusieurs schémas peuvent être considérés dont les arbres en H /6/ minimisent le décalage (skew) en utilisant la régularité et répétitivité des réseaux systoliques.

La généralisation du systolisme peut nous conduire jusqu'au niveau des bits avec des structures très simples. Cependant, ces structures ont un coût matériel très important (structures gourmandes) et pour la plupart des applications en télécommunications un pipeline plus léger est suffisant (voir l'étude des structures Bit-Plane et Modified Bit-Plane dans le chapitre 3.2).

5. Etude des Multiplieurs

5.1. Codage numérique et opérateurs élémentaires

La génération automatique d'opérateurs pour les architectures de traitement du signal est d'autant plus intéressante que l'on cherche à élargir leur domaine d'application. Un générateur de multiplieurs sera beaucoup plus utile si les structures auxquelles l'on veut aboutir peuvent être employées dans un grand nombre d'applications telles que les filtres FIR et IIR, les convoluteurs, les calculateurs de la Transformée Discrète de Fourier etc.

Après plusieurs discussions, il a été mis en évidence l'intérêt de la réalisation de multiplieurs redondants. La redondance de la représentation nous permet d'effectuer des opérations en parallèle sans propagation de la retenue du bit de poids faible au bit de poids fort. Le temps employé, par exemple, pour réaliser une addition est en conséquence constant et il ne dépend pas de la largeur des opérandes. De plus, dans certaines situations les calculs peuvent être réalisés avec le poids fort en tête.

La conception d'architectures systoliques pour implanter, par exemple, des filtres IIR affronte le problème fondamental du temps de latence proportionnel au nombre d'étages de pipeline. Le parallélisme ne se traduit pas par une augmentation du débit et la structure devient inefficace. L'arithmétique redondante permet d'effectuer des calculs récursifs sans les limitations associées au temps de latence de la boucle car le poids fort est calculé et rebouclé en un temps constant et un délai indépendant de la taille des opérandes /15/.

En conséquence, l'objectif à atteindre est pour le multiplieur une architecture capable de travailler avec un opérande en arithmétique redondante signée (voir paragraphe suivant). Une solution complètement redondante a été écartée car elle conduit à une structure très lourde (paragraphe 5.1.2). Les cas suivants sont donc à considérer :

* Données en CA2 et coefficients en SD

* Données en CA2 et coefficients fixes en CSD

* Données en SD et coefficients en CA2

* Données en SD et coefficients fixes en CA2

Les cas variables seront utilisés dans les applications "stand-alone" et les fixes correspondent à des applications de filtrage à coefficients fixes. L'inversion des données et des coefficients est envisagée pour des application où la redondance est employée pour le produit du coefficient en CA2 par la somme de deux données également en CA2 comme ce sera expliqué plus loin.

Cette idée est à reprendre pour des structures générales de traitement du signal telles que les filtres à réponse impulsionnelle infinie et finie. Pour les premiers on obtient des boucles avec un temps de latence constant pour un séquencement des données avec le digit de poids fort en tête. Enfin, pour les deuxièmes, la possibilité d'un filtrage à phase linéaire est envisagée avec l'addition implicite de deux échantillons des données et de leur produit par le coefficient dans le multiplieur. La propagation liée à une somme extérieure des données est éliminée, le débit de la structure augmenté.

5.1.1. L'arithmétique redondante et les chiffres signés

Les représentations redondantes ont été introduites par Avizienis /2/ avec l'intention de réduire la propagation de la retenue dans les opérateurs parallèles de base comme l'addition, la soustraction, la multiplication et la division. La différence principale par rapport aux représentations conventionnelles est la possibilité pour les digits composants d'un chiffre de posséder un poids négatif.

Les représentations conventionnelles ont une cardinalité (nombre d'élements) qui est égale à la base /9,18/. Par exemple, en base-10 (décimale) les digits sont limités à l'ensemble {0,9} avec évidemment 10 éléments. Une représentation signée peut avoir

une cardinalité supérieure à la base et dans ce cas là, elle est dite redondante. Plusieurs représentations sont valides pour une valeur algébrique quelconque et, par exemple, une base-10 signé peut employer des digits dans {-9,9} avec une cardinalité de 19. Une valeur arbitraire comme 147 peut être représentée par 147, 15$\underline{3}$, 1$\underline{9}$5$\underline{3}$ etc, où le digit souligné a un poids négatif. L'unicité de la représentation est perdue par rapport à une notation codée en numération simple de position et plusieurs bits par digit sont nécessaires selon le niveau de redondance mais il est possible d'aboutir à des simplifications importantes des opérateurs et d'effectuer des opérations libres de propagation de retenue.

Il existe différents types de représentations redondantes comme les notations en retenue conservée où carry-save, la virgule flottante et les chiffres signés. Dans ce projet la notation redondante des chiffres signés a été choisie en base 2 car elle conduit à des opérateurs simples. Deux types de représentations des chiffres signés sont employées :

1. Notation en partie positive - partie négative

Chaque digit est représenté au moyen de deux bits avec un poids positif et négatif. Ainsi la valeur du digit vaut :

$$a = a^{+} - a^{-}$$

et la valeur du chiffre complet peut être calculée par :

$$C = \sum_{j=1}^{m} c_j^{+} 2^{-j} - \sum_{j=1}^{m} c_j^{-} 2^{-j}$$

Par exemple, la valeur décimale 7 peut être codée avec les formats équivalents suivants ($\underline{1}$ est équivalent à -1) :

p. pos p. neg digit SD

0111 0000 0111

1000 0001 111$\underline{1}$

1001 0010 10$\underline{1}$1

1101 0110 10$\underline{1}$1

etc …

2. Notation en signe et magnitude

Ici, chaque digit est éclaté en deux bits qui représentent le signe et la magnitude séparément.

$$a = \text{sgn}(a)|a|$$

$$C = \sum_{j=1}^{m} \text{sgn}(c_j)\left|c_j\right| 2^{-j}$$

D'une façon équivalente au cas précédent, le chiffre 7 peut également représenté de la façon suivante :

signe magn digit SD

0000 0111 0111

0001 1001 100$\underline{1}$

0010 1011 10$\underline{1}$1

0110 1011 10$\underline{1}$1

etc…

La redondance obtenue est évidente et elle nous permettra aboutir à des simplifications importantes pour les opérateurs. Cette simplification est possible car plusieurs représentations des opérandes d'entrée et de sortie sont permises et en conséquence différentes solutions seront valides dans les cases des tableaux de

Karnaugh. L'ordre des regroupements pourra augmenter par rapport à celui d'une notation en arithmétique ordinaire non redondante. Les fonctions logiques seront simplifiées.

On peut citer ici le cas particulier du codage canonique en chiffres signés, appelé Canonical Signed Digit (CSD), où la minimisation des digits différents de zéro est recherchée. Cette notation centre son intérêt sur la réalisation d'opérateurs fixes, notamment dans les filtres dont les coefficients sont constants pour une application donnée.

Plusieurs algorithmes essaient de minimiser les digits non nuls grâce à la redondance de la représentation signée. Ils sont basés sur la recherche d'une représentation canonique pour les groupements de uns consécutifs. Par exemple soit le coefficient :

$$0{,}875 = 2^{-1} + 2^{-2} + 2^{-3} \rightarrow 0.111$$

peut s'exprimer en CSD comme :

$$1 - 2^{-3} \rightarrow 1.00\underline{1}$$

L'augmentation du nombre de digits nuls pour le coefficient se traduit directement par une élimination équivalente de lignes d'additionneurs dans les multiplieurs. Pour gérer les chiffres signés, il faudra additionner ou soustraire comme il convient dans chaque étage, en conséquence, une représentation carry-save avec complément et somme ou une structure signée doit être envisagée.

5.1.2. Opérateurs en arithmétique redondante

La réalisation d'architectures basées sur l'arithmétique redondante prend en compte l'utilisation des opérateurs conçus pour une telle représentation. Plusieurs stratégies peuvent être proposées en fonction de l'application desirée et du type concret de codage auquel l'on veut aboutir /11/.

Notre cas d'étude est un multiplieur. Dans les paragraphes précédents la nécessité de traiter un opérande signé a été justifiée. Pour un graphe de dépendance quelconque, les opérateurs ou nœuds du réseau sont déterminés par l'arithmétique employée. Parmi l'infinité de cas possibles, nous pouvons citer :

1. Cas général. Données en SD sur n digits et coeff en SD sur m digits

- Nombre de couches add/bit coeff = 2
- Nombre de MOS/mult pp = 16 (0 pour coeff fixes)
- Nombre de reg/couche = 6n + LSB

La double redondance de ce cas nous oblige à avoir un produit partiel avec entrées et sorties redondantes et aussi, une addition complètement redondante (Figure 1).

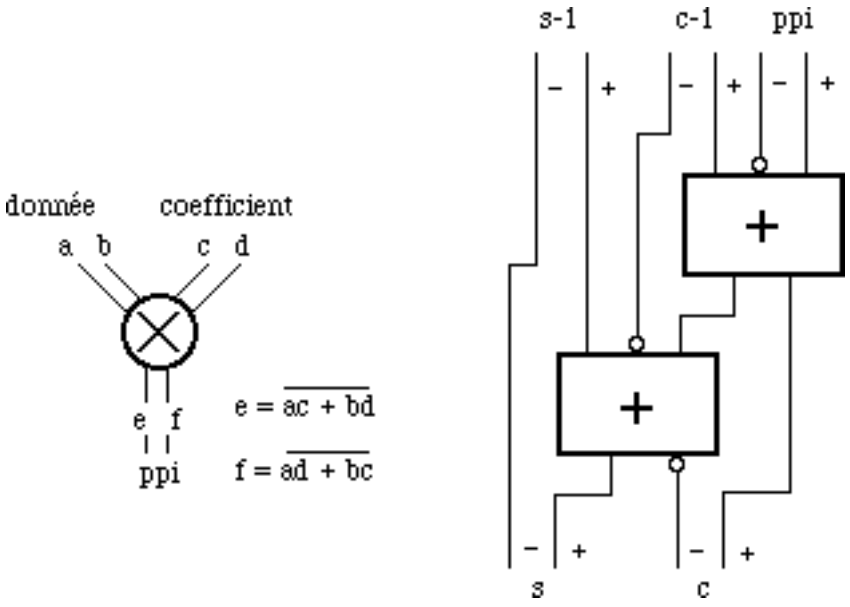


Figure 1. Opérateurs élémentaires SD-SD.

En conséquence, cette solution a été rapidement éliminée via à vis des applications envisagées. L'intérêt se centre vers une structure hybride en CA2 et SD moins couteuse.

2. Les données en SD sur n digits et les coefficients en CA2 sur m bits

La notation pour les chiffres signés est en bit positif-bit négatif

- Nombre de couches add/bit coeff = 2
- Nombre de MOS/mult pp = 12 (0 pour coeff fixes)
- Nombre de reg/couche = 6n + LSB

Dans ce cas la redondance des données nous oblige à propager les deux bits pour chaque digit puis à les calculer en fonction de leur poids et leur signe (Figure 2).

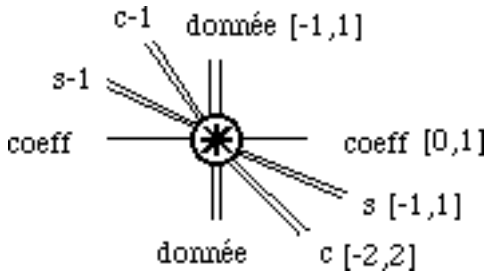


Figure 2. Opérateur élémentaire SD-CA2

Le principal désavantage de cette idée est la nécessité d'avoir dans chaque nœud deux couches d'additionneurs pour la somme des chiffres signés ce qui interdit toute application pratique de cette structure (Figure 3).

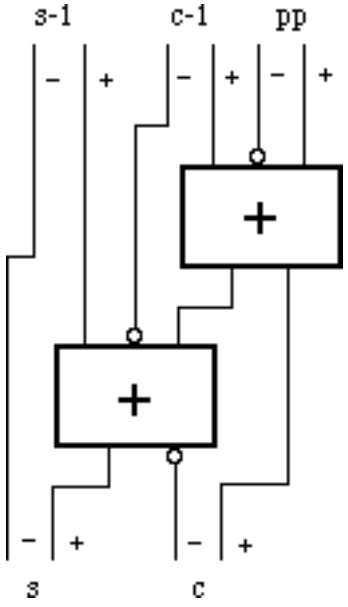


Figure 3. Additionneur redondant pour le cas SD-CA2

3. Les données d codées en CA2 sur n bits
le Coefficient c en SD sur m bits (m' pour le cas des coefficients fixes CSD)

C'est le cas classique Carry-Save où il nous faut complémenter les données pour effectuer les soustractions et ajouter une extension de signe au réseau pour traiter le bit de signe de la donnée en CA2 (Figure 4).

Une estimation de la complexité de chaque nœud :

- Nombre de couches add/digit coeff = 1
- Nombre de MOS/mult pp = 12 (0 pour le cas fixe)
- Nombre de reg/couche = 3n + LSB

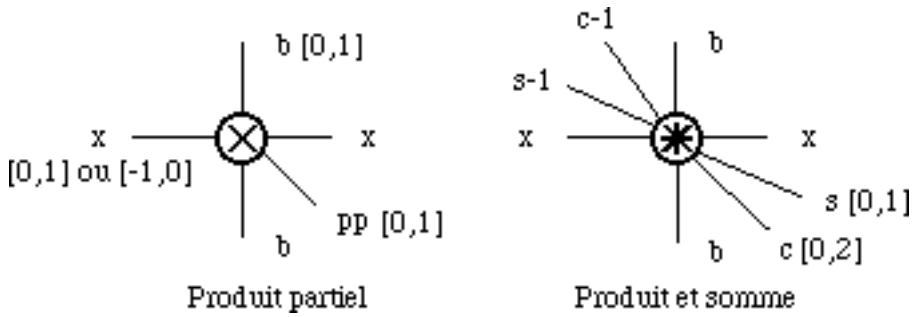


Figure 4. Opérateurs élémentaires CA2-SD
s : somme, c : retenue, pp : produit partiel, x : coeff.

Le produit partiel prend toujours les valeurs [0,1] et le traitement du signe est fait avec le complément et l'addition d'un poids faible aux données et l'ajout d'une colonne supplémentaire.

4. Les données en CA2 sur n bits les coefficients en SD sur m bits (m' pour CSD)

L'idée est d'exprimer le produit partiel sur un seul bit, dont le signe sera interprété différemment suivant le signe du digit entrant du coefficient. Ceci permet alors de n'avoir que trois nombres à additionner par étage. Un seul additionneur (FA) est alors requis. Par contre, les sorties s et c de l'additionneur ont un signe qui doit être interprété en fonction du signe du digit de la donnée entrante (Figure 5).

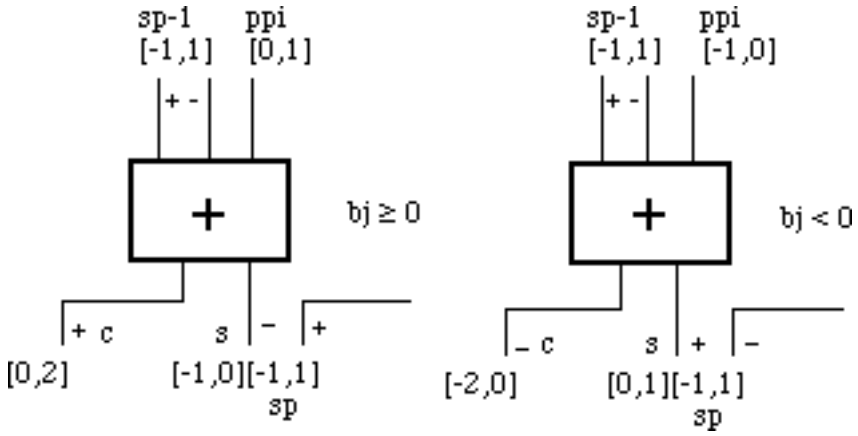


Figure 5. Addition redondante pour le cas CA2-SD

- Nombre de couches add/bit coeff = 1
- Nombre de MOS/mult pp = 12 (0 pour coeff fixes)
- Nombre de reg/couche = 3n + LSB

5.1.3. Assimilation, troncature et arrondi des résultats

L'assimilation

Les représentations redondantes sont employées dans les opérateurs pour éviter la propagation de la retenue et permettre d'atteindre des débits de travail importants. Cependant, le résultat de ces opérateurs conserve malheureusement cette redondance qui doit donc être assimilée (soit convertie) vers une sortie binaire ou en CA2. L'assimilation des résultats redondants peut comporter une nouvelle propagation si bien que l'objectif final n'est pas forcément atteint.

Les figures qui suivent, présentent les structures d'assimilation (VMA) qui ont été étudiées pour un cas générique de représentation interne en carry-save (digits [0,1,2]). La particularisation des chiffres signés est immédiate et il suffit de modifier les additionneurs complets ou les demi-additionneurs (FA ou HA) pour traiter correctement le signe. Les structures étant complètement pipelinées, leur complexité et leur débit maximum sont indiqués ci-dessous.

1. Structure VMA avec Additionneurs complets FA

Nombre d'additionneurs : L
Nombre de registres : L(3L+1)/2-1
Débit max. : 1/T(FA)

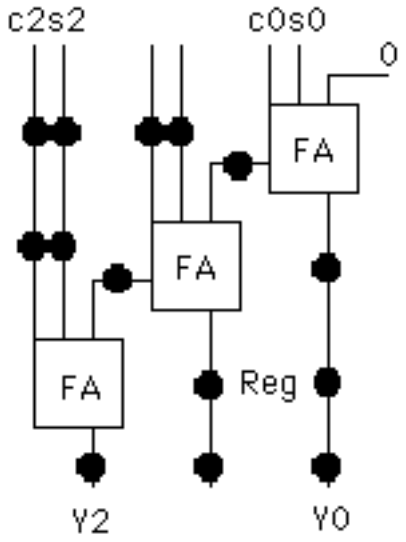


Figure 6. VMA avec Additionneurs Full Adder

2. Structure VMA avec Half Adders

(Structure avec un minimum de registres)

Nombre de demi-additionneurs : L(L+1)/2

Nombre de registres : L^2

Débit max. : 1/T(HA) [Skew des données]

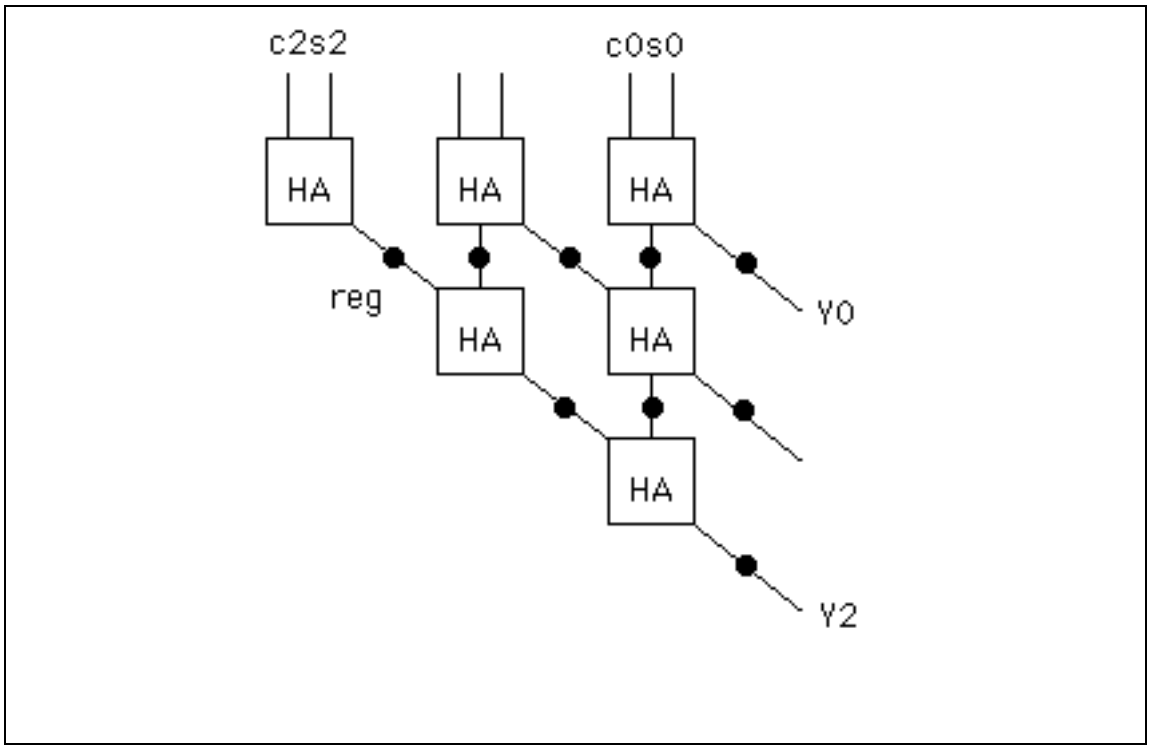


Figure 7. VMA avec demi-Additionneurs (min. registres)

On s'aperçoit de la présence d'un décalage (skew) des données à la sortie qui peut présenter un inconvénient pour certaines applications. L'élimination de ce décalage conduit à l'augmentation du nombre de registres comme il est montré ci-dessous.

Structure sans décalage en sortie

Nb additionneurs (HA):	L(L+1)/2
Nb registres:	$L^2 + L(L-1)/2$
T = T(HA)	[Il n'y a pas de skew]

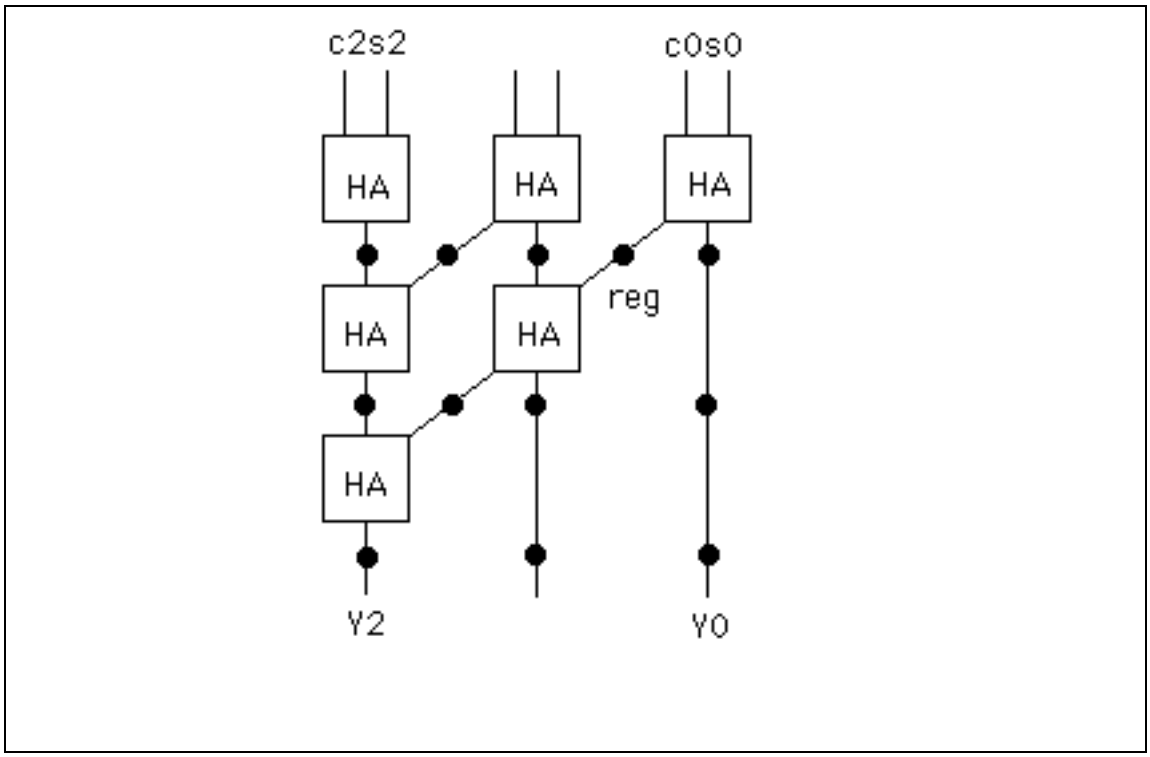


Figure 8. VMA avec demi-Additionneurs (sans skew)

Le choix d'une structure est évidemment lié à l'application envisagée. Les tableaux des annexes comparent la complexité de chaque structure en fonction du nombre de digits d'entrée. De plus, il est possible en pratique d'envisager un pipeline partiel de la structure afin de satisfaire au plus juste la contrainte de débit.

La troncature et le problème de l'arrondi

La précision complète à la sortie des opérateurs n'est généralement pas conservée car une cascade d'opérateurs impliquerait un chemin de données de largeur croissante. La troncature ou l'arrondi limite le nombre de bits (ou digits) nécessaires pour la représentation des résultats. L'effet direct est l'ajout de bruit à la sortie. En conséquence le compromis à respecter se situe entre le nombre de bits à conserver et le rapport signal à bruit minimum admis pour l'application donnée.

1. La troncature

Pour les chiffres, positifs la troncature produit toujours une erreur (valeur tronquée - valeur réelle) dans le même sens et qui, si l'on coupe au bit k, est située dans l'intervalle :

$$-2^{-k} < e < 0\,; \qquad q = 2^{-k}$$

La probabilité d'erreur p(e) vaut 1/q pour e dans]-q,0[les statistiques de moyenne et variance peuvent être calculées comme suit :

$$\bar{e} = \frac{-q}{2} \quad \text{et}$$

$$\sigma^2 = \int_{-} (e - \bar{e})^2 p(e)\,de = \frac{1}{q}\int_{-q}^{0} \left(e + \frac{q}{2}\right)^2 de = \frac{q^2}{12}$$

Dans le cas des chiffres en CA2 l'erreur de troncature est toujours négative. Les résultats antérieures sont valides.

Néanmoins, pour les chiffres signés l'erreur est placée dans :

$$-2^{-k} < e < 2^{-k}; \qquad q = 2^{-k}$$

avec les caractéristiques statistiques suivantes :

$$\bar{e} = 0 \quad \text{et}$$

$$\sigma^2 = \int_{-} (e - \bar{e})^2 p(e)\,de = \frac{1}{2q}\int_{-q}^{q} e^2 de = \frac{q^2}{3}$$

2. L'arrondi

Pour un résultat où k bits sont conservés, on ajoute au nombre :

$$2^{-k} \quad \text{si} \quad \text{bit}_{-(k+1)} = 1$$
$$0 \quad \text{si} \quad \text{bit}_{-(k+1)} = 0$$

Pour cette raison, l'erreur et les statistiques sont :

$$-2^{-k+1} < e < 2^{-k+1}; \qquad q = 2^{-k}$$

$$\bar{e} = 0 \quad \text{et}$$

$$\sigma^2 = \frac{1}{q}\int_{-q/2}^{q/2} e^2 de = \frac{q^2}{12}$$

Pour les chiffres en SD, il faut étudier séparément la partie positive et la partie négative ou arrondir la magnitude si la représentation est en sgn-mag. Les résultats obtenus préalablement sont encore valables car l'intervalle est doublée.

5.2. Conception descendante d'un multiplieur

5.2.1. Cas variable à partir d'un modèle câblé

Une étude de la complexité de certaines structures de filtrage a été réalisée précédemment pour des applications typiques du domaine du traitement du signal. Les résultats que l'on pourrait classer en deux grands blocs ou familles de structures pour les FIR ont été comparés : d'un côté les filtres dérivés des structures directe/transposée, de l'autre les topologies bit-plane et modified-bit-plane et en dernier lieu, les structures parallèles de traitement par blocs. Nous avons vu dans l'analyse effectuée que le pipelinage au niveau du bit pour les structures bit-plane et modified-bit-plane nous permet un débit de travail très élevé (qui est parfois difficile à gérer à l'extérieur du circuit) mais avec un coût matériel très lourd pour la plupart des applications. Les structures de traitement par blocs permettent de très hauts débits d'entrée avec des fréquences internes inférieures dans le réseau , mais les applications de télécommunications ont des exigences plus modestes en rapidité et sont plus exigeantes en surface.

Pour les applications de traitement du signal que l'on peut considérer a priori dans ce travail comme le filtrage vidéo, audio pour la TV et TVHD ou la téléphonie, il

serait intéressant de fixer un cahier des charges générique qui nous imposerait des filtres FIR à phase linéaire d'un maximum de 60 coefficients pour des fréquences bornées supérieurement aux alentours de 100 Mhz /14,25/. Ces caractéristiques peuvent être réalisées avec une structure simple du type directe/transposée avec des données en complément à deux (CA2) et des coefficients codés en notation canonique à chiffres signés (CSD) pour aboutir à une simplification maximale du multiplieur et du filtre.

Le Choix du Graphe

Il est évident que la structure interne au niveau logique de tout opérateur dépend du codage des entrées, ainsi, une codage donnée doit nous imposer le type des opérateurs. Cependant, le choix d'une structure pour le réseau de notre multiplieur est totalement indépendant du codage des opérandes. Le choix d'un graphe de dépendance pour le réseau doit être fait en préambule à toute étude.

Nous cherchons un multiplieur parallèle-parallèle qui puisse être pipeliné d'une façon efficace pour répondre aux contraintes de débit prévues pour ces applications. Différentes sortes de graphes de dépendance ont été étudiées dans la littérature spécialisée /23,24/, nous en citons trois types principaux, voir les figures ci-dessous. Ce sont des graphes canoniques où la retenue est propagée vers le plus proche voisin. Une solution de type arborescent, comme par exemple les arbres de Wallace /19/, a été écartée à cause des contraintes imposées par les générateurs qui exigent des structures très régulières. Dans les graphes suivants, les données sont codées en notation fractionnaire sur a1-a5 avec le poids fort sur a1. La même convention a été prise pour les coefficients qui sont codés sur b1-b5.

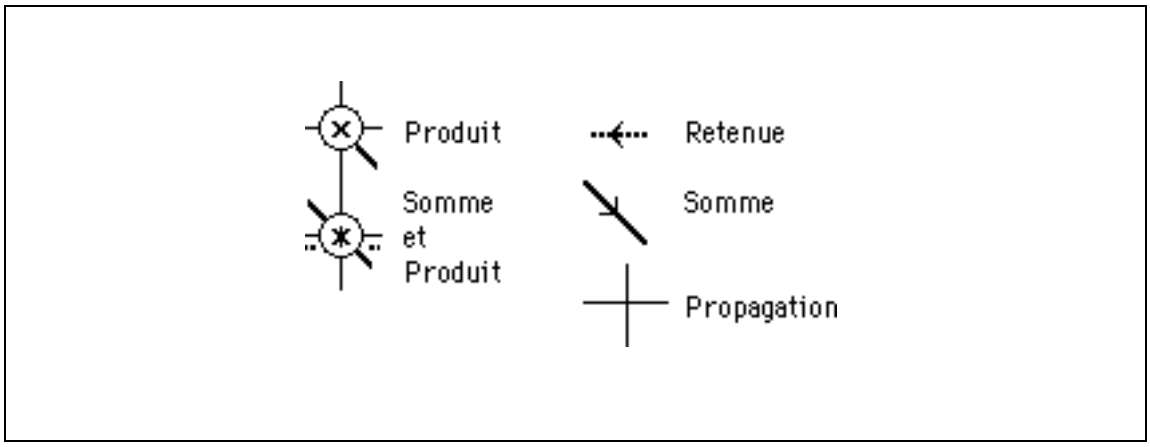


Figure 1 (a). Notation employée dans les graphes suivants

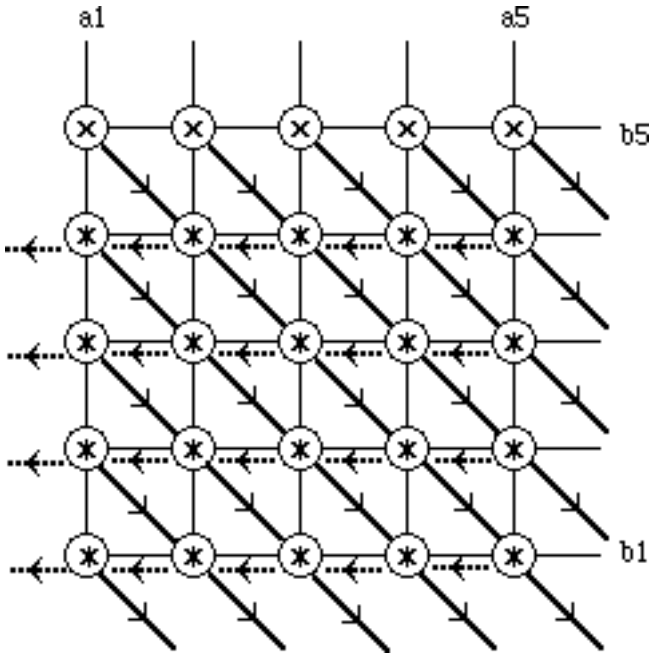


Figure 1 (b). Graphe 1

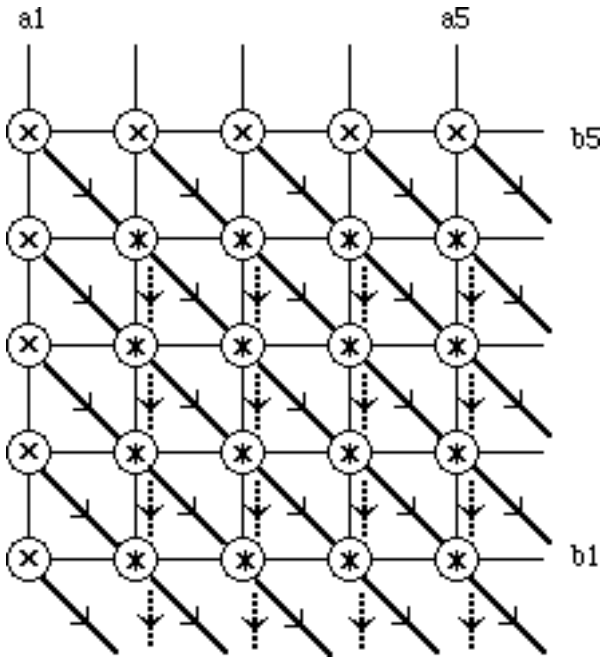


Figure 2. Graphe 2

Le graphe 1 correspond au cas de la propagation de retenue orthogonale à la propagation du résultat (Carry-Ripple) et pour cela nous sommes obligés de pipeliner selon les deux axes pour aboutir à une structure performante en ce qui concerne le débit. Par contre cette solution implique une complexité excessive au niveau matériel et logiquement un mauvais résultat au niveau de la surface.

Une amélioration immédiate de cette structure est présentée sur la figure 2. Ici, la retenue est envoyée au poids supérieur de la ligne suivante (Carry-Save) /21/ en introduisant une certaine redondance au résultat. Grâce à cette stratégie on peut pipeliner la structure d'une façon plus performante. Par contre, il est évident que l'on doit assimiler cette redondance extérieurement avec une architecture spécialisée (Voir structures VMA).

L'idée de propagation de la retenue aux poids supérieurs de la ligne suivante, employée dans le graphe 2, peut être généralisée vers une propagation à une ligne sur deux, trois etc. Evidemment, il y aura une redondance progressivement plus difficile à assimiler en sortie (compromis entre généralisation et complexité de la sortie). Dans le graphe 3 (Figure 3), on a pris une propagation sur deux lignes et il faut voir qu'en

comparaison avec le graphe précédent on doit assimiler les sorties latérales (dans le graphe 2 cette assimilation est faite implicitement), en conséquence il faut placer le VMA en bas et à droite du multiplieur.

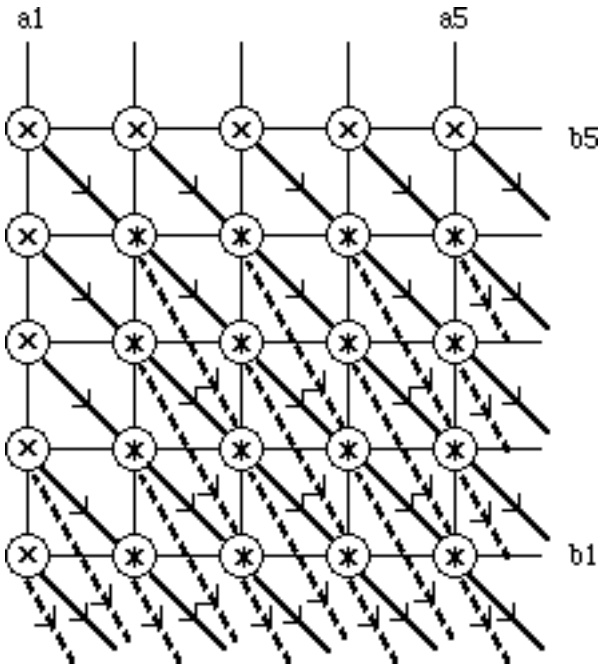


Figure 3. Graphe 3

A ce point de l'étude, on peut se demander l'intérêt d'une telle architecture basée sur le graphe 3 qui est plus complexe du point de vue du "hard" par rapport à une structure selon le graphe 2. En fait, une des caractéristiques du graphe 3 est qu'il nous permet, grâce à un pipelinage vertical, de travailler avec le poids fort du coefficient en tête (MSB first) sans décalage (skew) du résultat et en adoptant le sens -a- pour les coefficients et -b- pour les données. Cette propriété est tout à fait intéressante pour le dessin et la conception des filtres à réponse impulsionnelle infinie (IIR) /15/ où le temps de latence doit être réduit au minimum pour améliorer le débit du filtre. Cependant, pour l'étude et la réalisation future d'un filtre FIR, le but de ce travail, cette caractéristique n'est pas très intéressante et complique la structure logique de l'opérateur à cause d'une difficulté supplémentaire de routage. Celui-ci est alors plus complexe car

il doit être fait diagonalement entre des rangés de cellules. L'assimilation nécessaire du résultat est aussi plus compliquée parce qu'elle passe de deux bits à trois bits par digit.

Maintenant, on est dans un stade critique où il faut envisager un cas global pour un générateur de multiplieurs capable de générer des structures basées, à la fois, sur les graphes 2 et 3 et pour cela constituer une partie commune entre la conception de filtres FIR et IIR. Pour fixer les idées, dans un premier temps, on partira pour l'étude de notre multiplieur d'un choix préliminaire basé sur un graphe 2 et on laissera une porte ouverte pour un développement postérieur à partir du graphe 3.

Le Multiplieur

Tout d'abord, et pour le plus haut niveau de notre conception descendante, après le choix du graphe de dépendance, considérons un multiplieur général CA2-SD comme suit (Figure 4) :

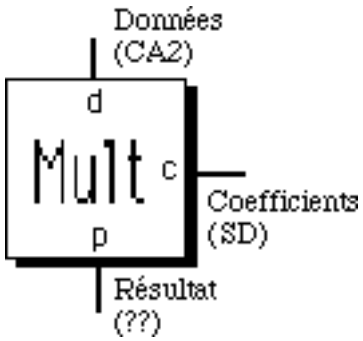


Figure 4. Deuxième étage de la conception descendante

Les données ont été prises arbitrairement en CA2 et les coefficients en SD mais on peut aussi considérer le cas inverse. Ceci impliquerait de prendre les coefficients en CA2 et les données seraient introduites de façon redondante comme la combinaison de deux chiffres également en CA2. On entrevoit, comme il a déjà été dit, que la redondance d'une entrée va nous permettre d'effectuer un produit par une somme dans la même durée qu'une multiplication simple.

Une alternative complètement redondante (coefficients et données codées en SD) a aussi été considerée dans une étape préliminaire de notre étude (chapitre des opérateurs redondants). Ici les capacités de l'arithmétique redondante conduisent à un coût matériel trop important, car deux couches d'additionneurs sont nécessaires pour chaque digit.

D'une façon générale, proposons les relations suivantes :

$$d = -a_{-1}2^{-1} + \sum_{i=2}^{n} a_i 2^{-i}$$
$$c = c^+ - c^-$$
$$c^+ = \sum_{i=1}^{m} c_i^+ 2^{-i}$$
$$c^- = \sum_{i=1}^{m} c_i^- 2^{-i}$$

Les données (d) sont codées en CA2 et les coefficients (c) en SD. Nous ne prenons pas de convention pour le produit (p) qui peut donc être a priori soit en SD soit en CA2. Une conversion à la sortie vers le codage choisi doit être envisagée.

Si le fait de conserver une entrée libre pour additionner un opérande auxiliaire selon p = cd + u est accepté, les relations suivantes peuvent être accomplies :

$$p = p^+ - p^- = d(c^+ - c^-) + u = dc + u$$
avec $u = -d; \quad c^- = \bar{e}$
$$p = d(c^+ - \bar{e}) - d = d(c^+ + e)$$
p en SD; $u, d, \bar{e}, c^+$ en CA 2

On peut donc réaliser, avec un tel multiplieur, le produit d'un coefficient en CA2 par la somme de deux données en CA2. Ceci est particulièrement intéressant dans le cas des filtres FIR à symétrie paire des coefficients.

réseau, on doit gérer à la fois des chiffres en CA2 et en SD. Ceci est équivalent à considérer un produit partiel dans un intervalle [0,1] ou [-1,0] selon le signe du digit Cj

et en conséquence un comportement de chaque ligne du multiplieur en fonction des signes. Il nous faut reprendre l'idée des opérateurs CA2-SD vue préalablement.

Dans une première approche, on peut considérer un produit binaire-SD et développer les cellules de base pour le réseau, puis étudier le traitement du signe du CA2 si une telle structure est satisfaisante pour le traitement du SD.

Pour déduire les cellules de base, on décompose le problème en deux parties suivant que le digit du coefficient est positif ou négatif. Ceci peut se traduire par l'obtention d'un premier multiplieur à coefficients fixes qui sera "câblé" avec des lignes d'un type ou d'un autre en fonction du signe de chaque digit. Chaque PE du graphe de dépendance contiendra un produit et une somme. On considère un produit toujours dans l'intervalle [0,1] indépendant du signe du coefficient et on le traite implicitement dans chaque ligne d'additionneurs du multiplieur. Ceci implique une interprétation intéressante du réseau en arithmétique redondante signée. Le signe est traité ligne par ligne en fonction du signe des digits du coefficient. Deux cas qui permettent d'aboutir au premier ensemble de cellules peuvent être envisagés :

Premier cas: Digit positif ou nul

Le produit partiel pp est positif [0,1], par conséquent pour une somme partielle entrante dans [-1,1] (soit -1,0,1) on aura une somme sortante entre [-1,2] (soit -1,0,1,2). Le poids 2 peut être décalé vers la gauche de façon à ce que l'on maintienne la somme entrante toujours entre [-1,1] comme on a indique ci-dessous sans justifier. La table logique et la déduction des fonctions sont représentés dans le tableau 1 et la figure 5, ci-dessus.

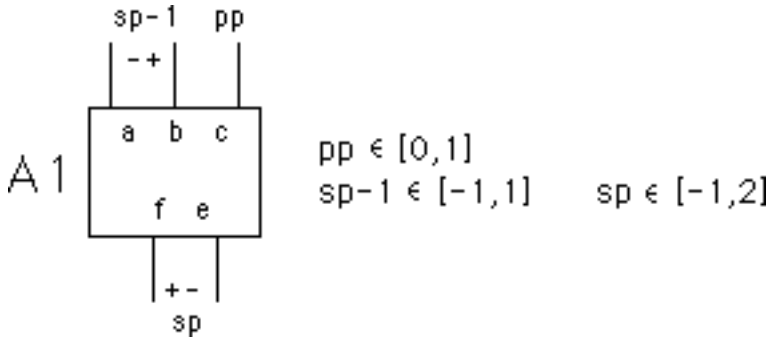


Figure 5. Cellule pour le cas du coefficient positif

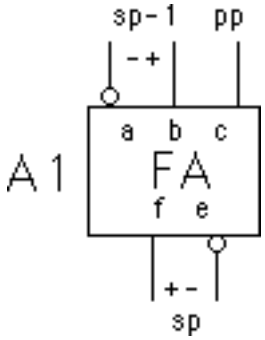
Les fonctions obtenues sont les suivantes:

$$e = \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc + a\bar{b}\bar{c} = a \oplus b \oplus c$$
$$f = \bar{a}b + \bar{a}c + bc = maj(\bar{a}, b, c)$$

sp-1(-)	sp-1(+)	pp(+)	sp	f(2)	e(-1)
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	2	1	0
1	0	0	-1	0	1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	1	1	1

Tableau 1. Comportement logique pour A1

En conséquence, notre cellule peut être implementée avec un additionneur complet et deux inverseurs (Figure 6).



$$\overline{e} = \overline{a} \oplus b \oplus c$$
$$f = maj(\overline{a}, b, c)$$

Figure 6. Cellule A1 à partir d'un FA

Deuxième cas: Digit négatif

Le produit pp appartient à [-1,0] et avec la même justification d'une somme entrante appartenant à [-1,1], on obtient à la sortie une somme contenue dans [-2,1]. Le poids -2 sera également décalé vers la gauche pour maintenir les sommes entrantes dans [-1,1]. Comme dans le cas précédent, on peut remplir le tableau 2 et la figure 7.

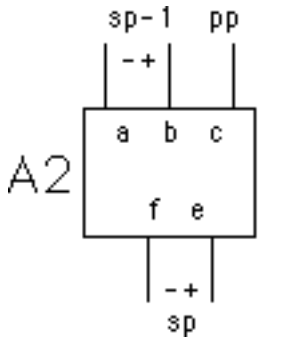


Figure 7. Cellule pour le cas du coefficient négatif

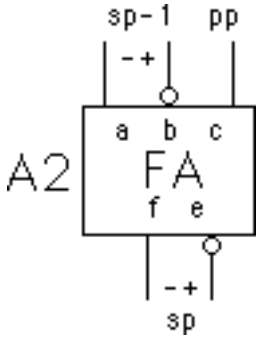
D'un façon équivalente au précédent cas, on peut arriver aux équations suivantes:

$$e = a \oplus b \oplus c$$
$$f = \overline{b}c + ac + a\overline{b} = maj(a, \overline{b}, c)$$

sp-1(-)	sp-1(+)	pp(-)	sp	f(-2)	e(1)
0	0	0	0	0	0
0	0	1	-1	1	1
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	-1	1	1
1	0	1	-2	1	0
1	1	0	0	0	0
1	1	1	-1	1	1

Tableau 2. Comportement logique pour A2

Les expressions obtenues peuvent aussi être synthétisées au moyen d'un additionneur complet et de deux inverseurs (Figure 8).



$$\overline{e} = a \oplus \overline{b} \oplus c$$
$$f = maj(a, \overline{b}, c)$$

Figure 8. Cellule A2 à partir d'un FA

De manière équivalente, on peut trouver des variantes à ces cellules en changeant par exemple les signes des entrées ou des sorties et des inverseurs peuvent être éventuellement placés aux entrées/sorties de poids négatif. Avec cette stratégie et un système similaire aux lignes précédentes, nous obtiendrons les cellules nommées A2 et A3 selon la figure 9 (noter l'inversion des signes des terminaux entre les deux cellules A2 bien qu'elles réalisent la même fonction logique) :

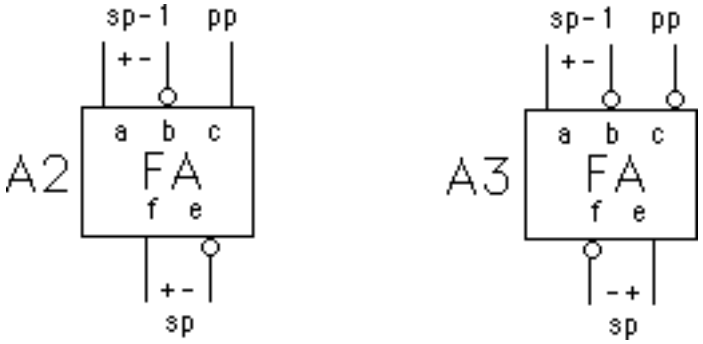


Figure 9. Nouvelles cellules A2 et A3

La raison pour laquelle les quatre cellules obtenues ont été conservées n'est pas évidente a priori car il serait plus logique de prendre la première paire A1 et A2 au lieu de la paire A2 et A3, ce qui est équivalent à simplifier A3 pour retrouver A1. Ce point sera traité plus loin et on verra que la non-simplification de A3 conduit à une simplification très importante de la structure globale pour le cas définitif CA2-SD avec des coefficients variables ou non câblés (voir schémas dans les annexes).

La structure obtenue permet de travailler en binaire-SD comme on a pu le vérifier avec les simulations fonctionnelles. Le pas suivant consiste à modifier ce résultat pour gérer le signe contenu dans la représentation du CA2. Ainsi les opérandes d'entrée (d,c) et le résultat (y) sont écrits avec la notation suivante :

$$d = -a_{-1}2^{-1} + \sum_{i=2}^{n} a_i 2^{-i}$$

$$c = c^+ - c^- = \sum_{j=1}^{m} c_j 2^{-j}; \quad c_j = c_j^+ - c_j^-$$

$$y = cd = \sum_{j=1}^{m} c_j 2^{-j}\left[-a_{-1}2^{-1} + \sum_{i=2}^{n} a_i 2^{-i}\right]$$

$$= -a_{-1}\sum_{j=1}^{m} c_j 2^{-j-1} + \sum_{j=1}^{m}\sum_{i=2}^{n} a_i c_j 2^{-i-j}$$

En comparant ce résultat avec celui obtenu dans le cas précédent, on constate facilement que la seule différence se trouve dans la dernière colonne du graphe où on

doit "simplement" inverser le signe du pp, autrement dit, on doit permuter A1/A2 ou A2/A3. Malheureusement cette inversion n'est pas possible à cause de l'incompabilité des signes d'entrée et de sortie entre les cellules comme il est indiqué sur la figure 10.

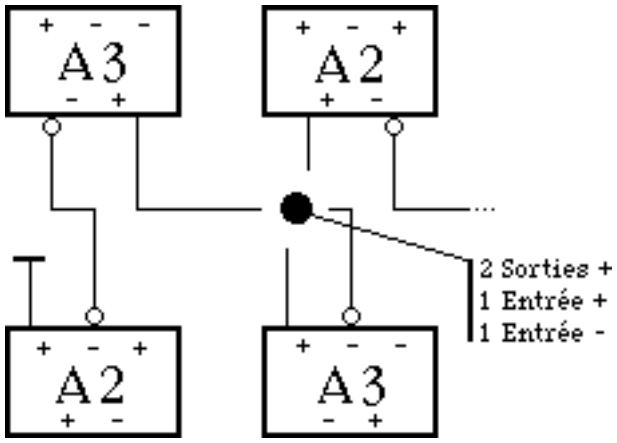


Figure 10. Incompatibilité des signes entre les cellules

Par conséquent, il est nécessaire de développer de nouvelles cellules qui seront placées en la dernière colonne de la structure en ajoutant un terminal de sortie et qui permettra de conserver le poids de chaque configuration. Ces cellules doivent être capables de gérer à la fois le signe du nombre en CA2 et le signe de chaque digit du coefficient en SD.

La même procédure a été suivie ici que pour le développement logique des cellules A1/A2 et A2/A3. Les tableaux 3 et 4 indiquent le fonctionnement de ces cellules avec le poids et le signe des entrées et des sorties.

Les figures 3 et 4 montrent les cellules et leurs équations logiques. Noter que la réduction à des simples additionneurs n'est pas possible.

* Cas digit du coefficient positif ou zéro ($Cj \geq 0$)

a(-)	b(+)	c(-)	f1(-2)	f2(+2)	e(-1)
0	0	0	x1	x1	0

0	0	1	x2	x2	1
0	1	0	0	1	1
0	1	1	x3	x3	0
1	0	0	x4	x4	1
1	0	1	1	0	0
1	1	0	x5	x5	0
1	1	1	x6	x6	1

Tableau 3. Comportement logique pour Cj ≥ 0

* Cas digit du coefficient négatif (Cj < 0)

a(-)	b(+)	c(+)	f1(-2)	f2(+2)	e(+1)
0	0	0	x1	x1	0
0	0	1	x2	x2	1
0	1	0	x3	x3	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	x4	x4	0
1	1	0	x5	x5	0
1	1	1	x6	x6	1

Tableau 4. Comportement logique pour Cj < 0

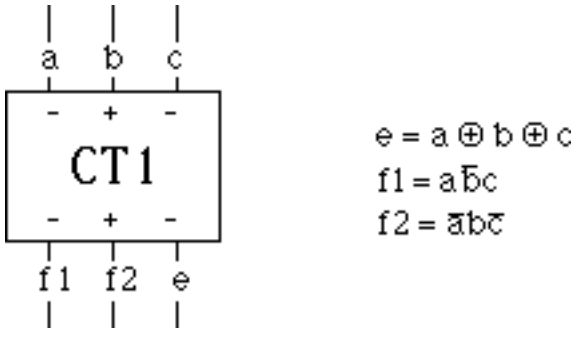


Figure 11. Cellule terminale pour Cj ≥ 0

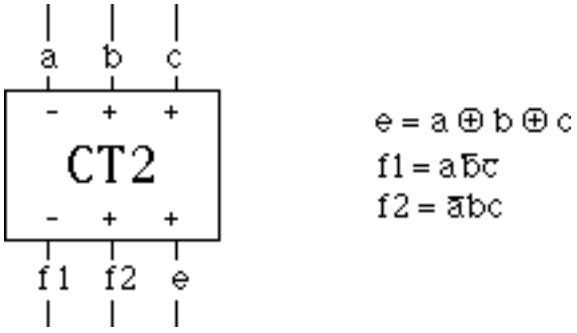


Figure 12. Cellule terminale pour Cj < 0

On peut vérifier que la nouvelle structure constituée par les cellules CT1, CT2 et A1/A2 ou A2/A3 nous permet de travailler en CA2-SD avec l'entrée de droite câblée et fixe. Les produits partiels sont calculés avec une représentation signe-magnitude du digit du coefficient. Par contre, les autres calculs sont faits avec une représentation partie positive - partie négative. Il nous reste à mettre au point une structure à coefficients variables grâce à des cellules dont le comportement sera dépendant du signe de Cj. Une première approche consisterait à gérer les inverseurs d'entrée et sortie avec des portes XOR mais cette solution apporte un coût très important en matériel à la matrice du réseau qui croît en complexité avec le produit du nombre de bits de la donnée par le nombre de digits du coefficient.

Comme solution à ce problème, peut être reprise la paire de cellules A2/A3 (on voit maintenant pourquoi elles n'ont pas été rejetées) avec les cellules terminales CT1 et CT2; le schéma suivant (Figure 13) montre que tous les inverseurs du réseau peuvent être simplifiés de façon à ce que ne soient conservées à l'intérieur du réseau que de simples additionneurs complets (FA) et qu'une colonne avec CT1 et CT2.

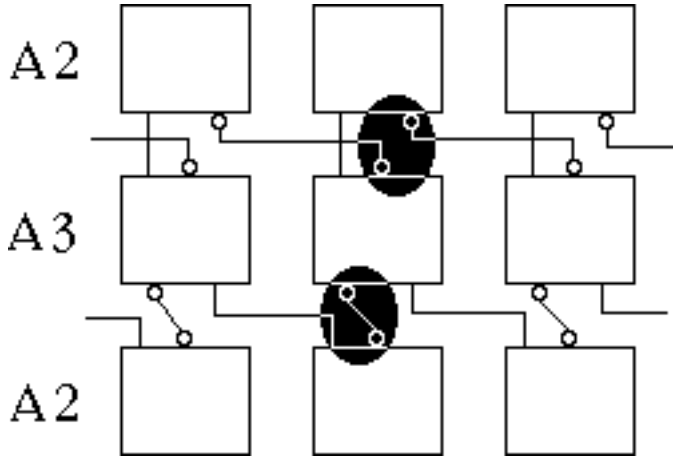


Figure 13. Simplification des inverseurs internes

Le cas des coefficients variables peut être envisagé facilement car il ne nous reste que des cellules FA à l'intérieur du tableau. Il faudra simplement implémenter une cellule terminale CT comme l'ensemble des cellules CT1 et CT2. La sortie (e) de la cellule (0,j) sera inversée si le digit du coefficient est positif, pour éliminer l'inverseur de la cellule (1,j+1) (voir schémas dans les annexes).

La seule différence entre connexions des FA est au pp qui doit être inversé ou non selon le signe de Cj. Ceci a pour conséquence d'éliminer tous les croisements de fils entre les cellules. Cependant, il faut maintenir les inverseurs des entrées et sorties des extrémités du tableau sur certaines positions et assimiler le résultat à partir de sorties de signe variable.

Le multiplieur obtenu, bien que très simple, parait un peu particulier, car ses sorties ont un signe variable et en fonction de ce signe il faut les inverser ou pas. Le signe de la sortie est connu, évidemment, à partir du signe de chaque ligne d'additionneurs. A la sortie les poids négatifs sont inversés et réordonnés pour maintenir les signes des digits du résultat au même ordre.

Une première approche consisterait à employer un multiplexeur pour chaque paire de sorties. Ce multiplexeur serait commandé par le signe entrant dans chaque ligne et corrigerait les résultats négatifs avec un inverseur. Cependant, le coût matériel ajouté va interdire toute application de cette solution à notre structure. Il parait donc

judicieux, de chercher une solution plus simple exploitant la redondance de la représentation employée.

La stratégie que l'on a suivie est détaillée ci-dessous (Figure 14) et consiste à considérer toutes les représentations valides possibles pour la sortie en SD. Pour une sortie quelconque du multiplieur, on regroupe les terminaux du même poids en formant un digit en représentation signée en parties positive et négative :

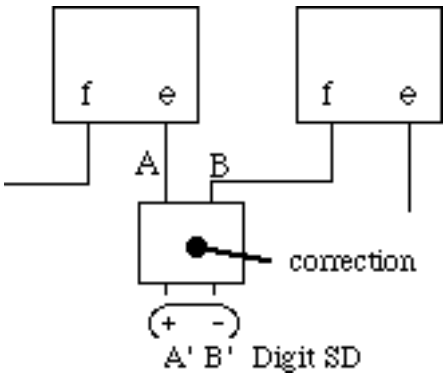


Figure 14. Assimilation de la partie MSB

Il est intéressant de chercher à corriger les sorties A et B pour maintenir le même ordre aux signes des digits de sortie. Les tableaux logiques suivants (Tableaux 5 et 6) nous conduisent à une solution particulièrement simple :

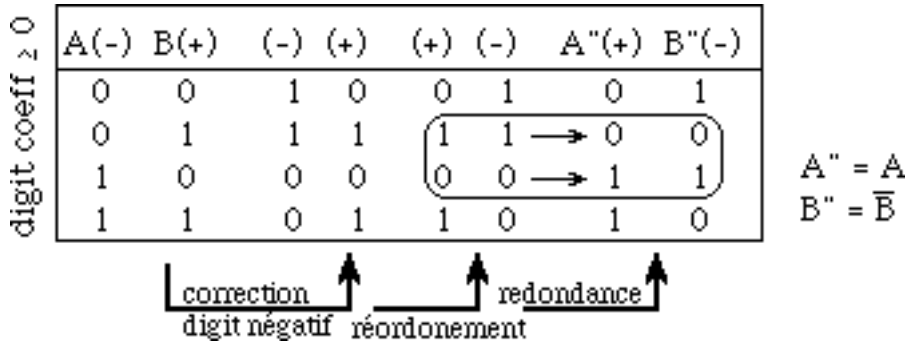
digit coeff < 0	A(+)	B(-)	A'(+)	B'(-)
	0	0	0	1
	0	1	0	0
	1	0	1	1
	1	1	1	0

$A' = A$

$B' = \overline{B}$

correction
digit negatif

Tableau 5. Solution redondante à l'assimilation MSB pour le cas négatif



Tableaux 6. Solution redondante à l'assimilation MSB pour le cas positif

Le résultat montre que grâce à la redondance de la représentation de sortie, il suffit d'ajouter un simple inverseur et non un multiplexeur (Figure 15).

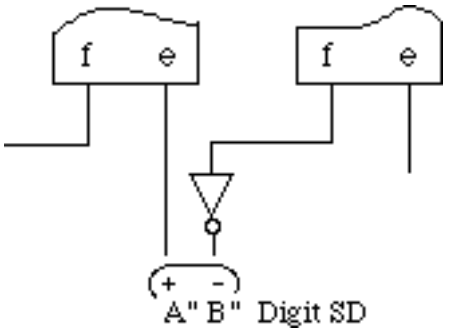
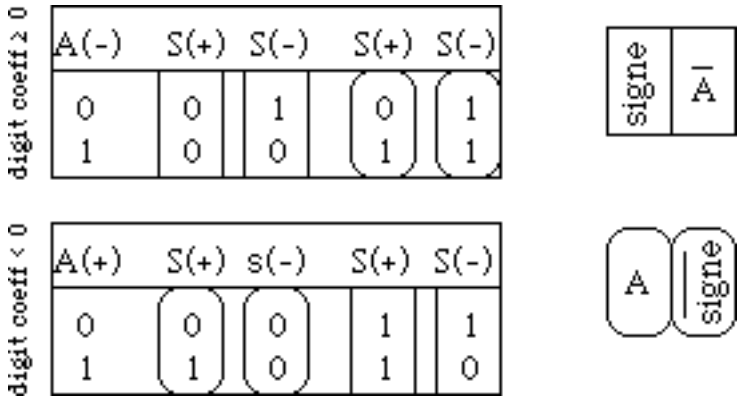


Figure 15. Solution employée pour les sorties MSB

Le même cas a été résolu pour les sorties de poids faible du multiplieur avec une procédure similaire et des résultats identiques (Tableaux 7, 8 et Figure 16). Il sera vu plus loin qu'une des solutions indiquées ci-dessous simplifie les cellules de produit partiel (Tableau 9 et Figure 17).



Tableaux 7, 8. Solution redondante à l'assimilation LSB

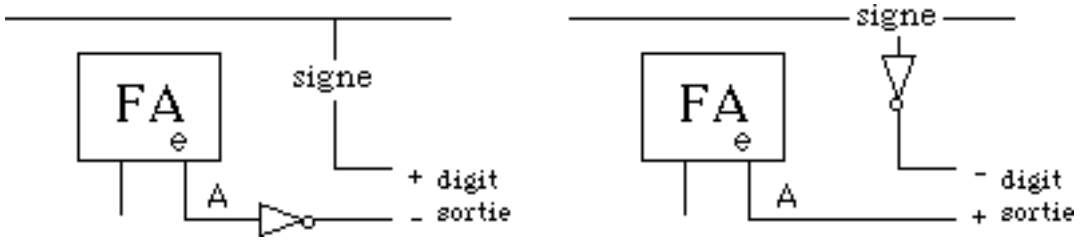


Figure 16. Deux solutions possibles pour les sorties LSB

Jusqu'à présent, dans le raisonnement théorique, les chiffres signés étaient représentés par leurs parties positive et négative mais dans le schéma du multiplieur les entrées et les produits partiels sont données en bit de signe et bit de magnitude (sgn-mag), les sorties sont, par contre, en pos-neg. Les cellules de produit partiel ont été conservées en notation sgn-mag car ceci nous permet de simplifier beaucoup le tableau.

1. $p = s \oplus dm$
2. $p = \overline{d}c_{-}\overline{c}_{+} + d\overline{c}_{-}c_{+}$

1 représente le cas sgn-mag avec la donnée en CA2 et le coefficient en signe et magnitude. Ceci est équivalent à 12 MOS.

2 représente le cas pos-neg avec la donnée en CA2 et le coefficient en partie positive et négative. 22 MOS!

Cependant, à l'extérieur du tableau, il serait mieux d'avoir une notation des SD en pos-neg car il faut maintenir une cohérence avec la sortie. De même, la possibilité d'inverser les données et les coefficients pour les filtres à phase linéaire doit être également considérée. La conversion des entrées de pos-neg à sgn-mag amène à une nouvelle simplification du réseau. Le tableau 9 montre que l'on peut déduire de façon simple les fonctions logiques pour cette conversion et dans la figure qui suit (Figure 17), la simplification des sorties de poids faible et des produits partiels est indiquée.

a(+)	a(-)	s	m
0	0	0/1	0/0
0	1	1	1
1	0	0	1
1	1	0/1	0/0

$$s = \overline{a}_{+}$$
$$m = a_{+} \oplus a_{-}$$

Tableau 9. Conversion pos-neg -> sgn-mag

Il est intéressant d'observer qu'il est possible de prendre le signe interne équivalent à l'entrée positive inversée en simplifiant les produits partiels obtenus et aussi éliminer l'inverseur de sortie pour les digits de poids faible.

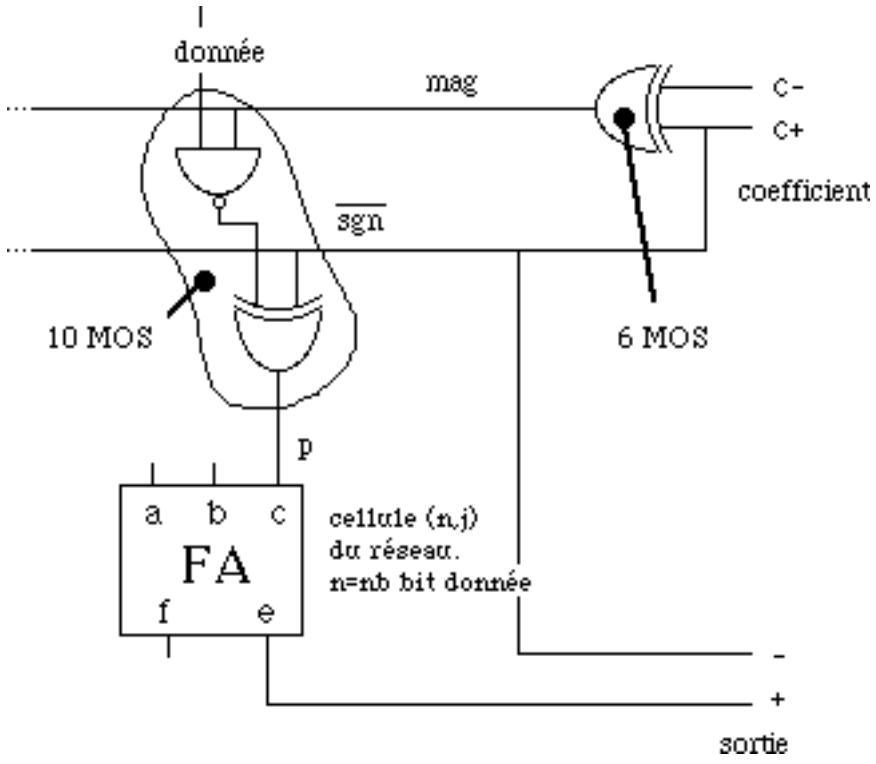


Figure 17. Conversion d'entrée, simplification du pp et sortie

Réalisation et vérification de la structure

Dans un premier temps, nous avons conçu un générateur de schémas avec l'aide du système GDT. Les structures générées sont un assemblage de cellules de base décrites soit par un schéma logique, soit par un modèle fonctionnel en langage M (qui fait partie de GDT). Le générateur de schémas a pour simple mission le placement et interconnexion des cellules. Les paramètres d'entrée sont initialement le nombre de bits de la donnée en CA2 et le nombre de digits du coefficient en SD.

Pour vérifier cette structure, un multiplieur nxm a été instancié avec un générateur de stimuli écrit en langage M et qui est responsable de tout le test. Ce modèle génère les signaux qui vont attaquer les entrées du multiplieur et il compare le

résultat obtenu avec le résultat attendu en activant, s'il est nécessaire, un signal d'erreur. Il suffit de suivre l'évolution de ce signal.

Cependant, le principal problème qui se pose pour la vérification complète d'un multiplieur nxm est la longueur de la simulation nécessaire (voir le chapitre dédié à la génération et à la simulation de la structure). Rappelons ici les équations régies par le multiplieur CA2-SD (Figure 18) :

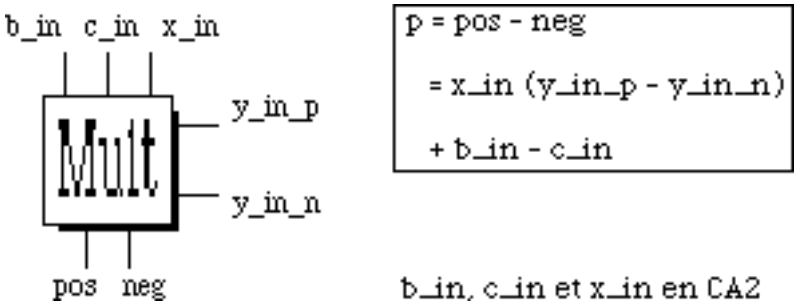


Figure 18. Nomenclature du multiplieur CA2-SD

Ainsi, il nous faudrait, pour un simple multiplieur 8x8 un nombre d'opérations de $(256)^5$ = 1,09951e+12. Pour un modèle dont le délai est de 2 ns, la simulation durerait donc 2.199 secondes. Or, la limitation en temps d'une simulation est de 200.000 ns, ce qui équivaudrait à répartir la simulation en 10,9951 millions d'étapes.

La nécessité d'automatiser et de simplifier le processus de simulation est bien évident. Les énormes quantités de temps CPU et de mémoire nécessaires pour une telle simulation nous obligent à chercher une autre solution plus performante et en conséquence, à considérer la possibilité d'employer, par exemple, des vecteurs de test aléatoires.

Dans le chapitre suivant l'écriture des générateurs schématiques, des icône et des layouts est commentée en partant des résultats obtenus ici et en développant les algorithmes correspondants. Nous plaçons l'accent sur le thème de la génération automatique, la vérification et la simulation de structures comme des multiplieurs parallèle-parallèle et les filtres FIR.

5.2.2. Cas fixe comme simplification du cas variable

Dans les applications, notamment dans les filtres, un des opérandes des multiplieurs est fixe. Ceci doit impliquer certaines simplifications sur la structure originale. Le travail à suivre est donc l'étude de ce cas pour aboutir à une nouvelle structure, particularisation du réseau original, beaucoup plus simple car le codage en CSD doit éliminer en moyenne deux tiers des lignes d'additionneurs correspondant aux digits nuls. Le cas des coefficients codés en CA2 doit être considéré séparément et la simplification possible du nombre de colonnes pour le multiplieur doit être étudiée.

La simplification pour les coefficients fixes entraîne le problème du respect des signes à l'intérieur et extérieur du réseau. L'élimination d'une ligne ou d'une colonne peut comporter une incompatibilité entre les signes des entrées et des sorties des cellules comme on l'a vu pour le traitement du signe du chiffre en CA2. L'élimination des inverseurs à l'intérieur du réseau (et l'utilisation des additionneurs FA) cache l'information originale du signe qui est maintenant traité aux extrémités et dans les produits partiels et en conséquence, implicitement dans le réseau. Pour vérifier la validité de la simplification il faudrait repartir les structures simples câblées où le signe est explicite (cellules CT1, CT2, A2 et A3) et évoluer vers la structure générale avec une simplification des produits partiels, des cellules terminales et des entrées.

1. Coefficients en CSD

Pour une application donnée où les coefficients sont fixés, le codage est fait en CSD ce qui nous permet d'éliminer les deux tiers des lignes du multiplieur. Maintenant, il faut étudier la manière de faire cette simplification et de générer la structure automatiquement.

La structure basée sur les cellules A2, A3, CT1 et CT2 est directement simplifiable car après avoir éliminé des lignes, on conserve toujours une concordance des signes et aussi des poids en nous déplaçant en bas et à droite de notre graphe de dépendance. Il faut chercher la première ligne non nulle (A) placée avant la ligne à

digits nuls (B). Ce sont les poids les plus faibles de cette ligne (A) qui seront envoyés vers la sortie, les poids restants seront connectés aux cellules de la ligne (C) de coefficient non nul qui suit la ligne (A).

L'étape suivante élimine les inverseurs. On s'aperçoit qu'il faut ajouter un inverseur à la sortie f2 de CT1 et CT2 pour propager le signal à l'intérieur du réseau. Les cellules du produit partiel sont, évidemment, supprimées dans la colonne de CT1, CT2 et dans les lignes correspondantes aux digits positifs du coefficient. Un inverseur doit être ajouté pour les lignes aux digits négatifs.

L'assimilation de la sortie et la correction des signes est faite comme pour le cas variable. Pourtant les portes ou-exclusifs sont éliminées et les inverseurs en bas du réseau sont conservés. Pour la sortie des poids faibles, trois cas peuvent être distingués :

(a) Digit du coefficient nul. Les sorties de la ligne antérieure (digit ≠ 0) sont groupées par poids et corrigées comme suit (Figure 19) :

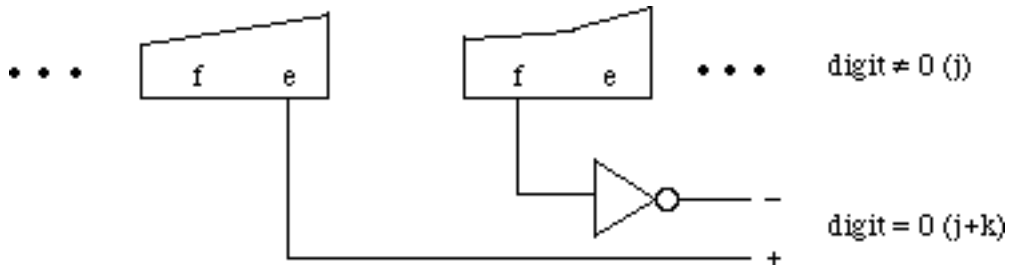


Figure 19. Assimilation LSB pour les digits nuls

(b) Digit du coefficient positif. Si l'on regarde le schéma général du multiplieur avec les portes ou-exclusifs nous constatons que le bit (-) du digit de sortie vaut toujours 1 (digit du coefficient positif). En conséquence cette sortie sera directement câblée à vdd (Figure 20 a).

(c) Digit du coefficient négatif. Le raisonnement est équivalent au cas (b) avec une connexion à vss (Figure 20 b).

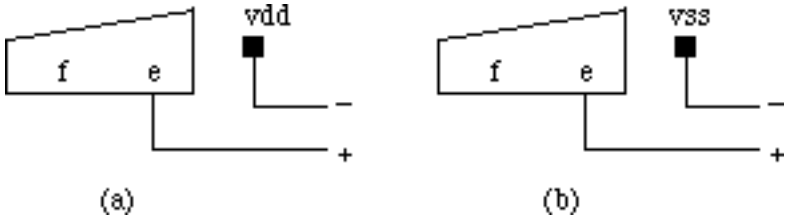


Figure 20. Câblage pour les cas avec le digit ≠ 0

2. Coefficients en CA2

Un raisonnement similaire a été réalisé pour le cas où les coefficients fixes sont codés en CA2 et les données sont codées de façon redondante. Malheureusement on s'aperçoit rapidement que la simplification d'une ou plusieurs colonnes peut entraîner des problèmes avec les signes associés aux terminaux des cellules. Dans certains cas, il existe des solutions comme le changement des cellules CT1, CT2 pour A3, A2 ou la permutation entre A2 et A3. Cependant, ces solutions ne sont pas générales et la simplification n'est pas toujours possible. Une première stratégie consisterait à rechercher de nouveau des cellules d'interface entre les colonnes correspondant aux bits non nuls. La pluralité des cas et le manque de régularité et de répétitivité que cela implique vis à vis d'une génération automatique ont été à l'origine de l'abandon de cette solution et ont conduit à envisager seulement le cas des coefficients en CA2 pour les filtres à des coefficients variables (filtres adaptatifs ou filtres programmables). Finalement, après l'évolution de cette étude, en partant d'un arithmétique redondante signée, on finit par retrouver les mêmes limitations qu'avec la structure classique Carry-Save. Nous avons donc trouvé une structure qui lui est équivalente.

Conclusion

Le résultat final obtenu est un peu décourageant car des quatre cas prévus à l'origine, nous ne sommes arrivés à développer des structures que pour seulement trois d'entre eux à savoir : les deux cas variables (CA2-SD et SD-CA2) et le cas fixe pour les coefficients en SD. La structure et les générateurs correspondants sont limités à ces trois cas comme s'il s'agissait d'une structure classique. Néanmoins, il faut noter que le cas des coefficients fixes en CSD reste très simple, régulier, facilement générable avec seulement trois types différents de cellules.

6. Algorithmes et Résultats

6.1. Générateurs avec GDT

Un générateur n'est qu'une cellule décrite à l'aide du langage textuel L avec des paramètres variables. La génération des structures régulières est assez simple car il s'agit simplement de placer et d'interconnecter des instances d'autres objets de plus bas niveau. Le manque de régularité de la structure se traduit directement par une perte de régularité et de structuration de l'algorithme du générateur.

Chaque cellule possède quatre vues ou représentations possibles, à savoir :

ICON : "Boîte" ou icône comprenant une enveloppe et des terminaux qui est destinée à être utilisée dans les schémas.

SCHEMATIC : Schéma logique de la cellule pour les simulations et la documentation.

LAYOUT : Dessin au micron de la cellule.

BBOX : "Boîte" au micron de la cellule avec ses ports.

Un générateur L complet doit être capable de générer à la demande les quatre vues ci-dessus dans différents niveaux de la hiérarchie.

Le niveau le plus haut d'un générateur est un programme L qui accepte des paramètres extérieurs et qui appelle d'autres programmes L pour générer les vues correspondantes. Les paramètres sont généralement calculés et traités à l'extérieur du générateur à cause des limitations algorithmiques du L. Normalement, l'interface avec l'utilisateur est aussi externe (-genie- pour la version 5 de GDT) pour compenser la pauvreté de communication liée au langage L. En conséquence, la partie en L ne s'occupe que des tâches de "bas niveau" telles que le placement, le routage, les petits

calculs arithmétiques et les blocs de décision et enfin, la génération des vues du circuit. Le schéma suivant essaie de synthétiser cette idée :

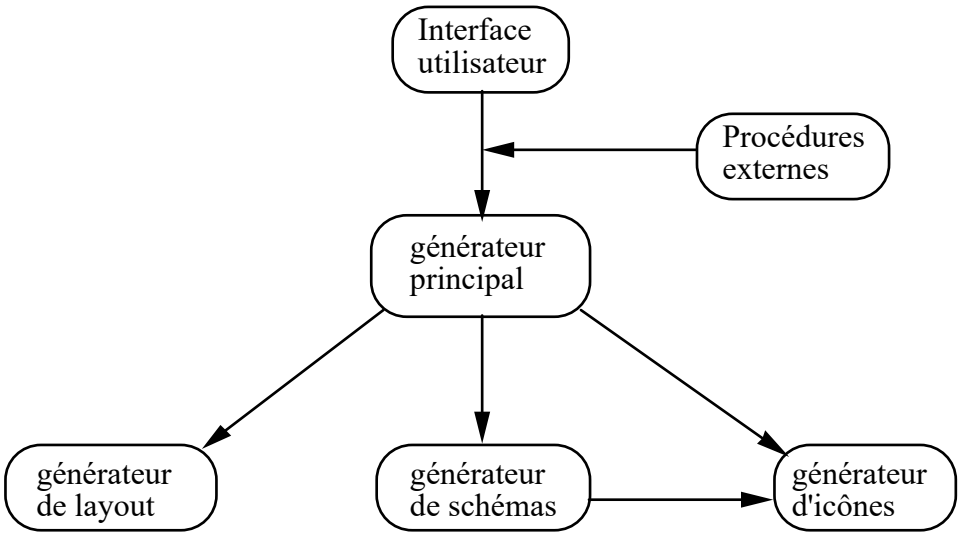


Figure 1. Structure typique d'un générateur

Les blocs <générateur> sont décrits en L, l'interface utilisateur en genie et enfin, les procédures externes en langage C, ADA etc.

6.2. Génération d'un multiplieur. Algorithmique

Le générateur de multiplieurs a été conçu avec l'arborescence décrite dans le paragraphe 6.1 en partant du générateur principal L pour devenir une partie d'un générateur plus global de blocs de traitement du signal tels que des filtres FIR, IIR ou même des multiplieurs "stand-alone". Un certain nombre de paramètres est passé au nœud principal et il appelle le générateur de la vue correspondante.

La philosophie suivie pour la programmation est décrite ci-dessous pour la branche de la vue Layout. Une organisation équivalente a été employée pour les schémas.

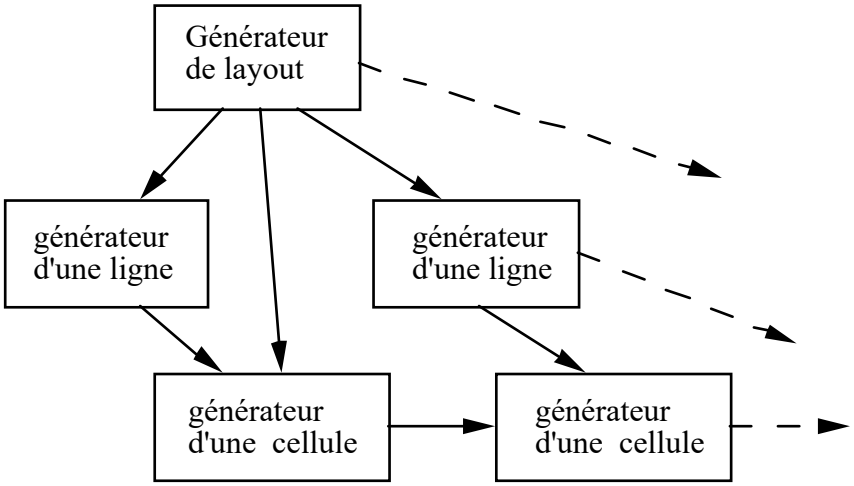


Figure 2. Structure d'un générateur de layout

Le layout d'un multiplieur parallèle-parallèle peut être envisagé comme la génération, l'aboutement et le routage de lignes de cellules identiques. Les lignes sont générées donc un niveau plus bas comme la juxtaposition de cellules générées dans un niveau hiérarchique inférieur (génération d'autres lignes ou des cellules). Les générateurs de cellules sont normalement de simples listes d'éléments. Cette arborescence permet de conserver la paramétrabilité jusqu'au dernier niveau de la hiérarchie et agir directement sur les valeurs des paramètres comme la largeur des alimentations et des fils de connexion, la taille de certains transistors, la séparation et le nombre de traversées à l'intérieur ou entre les cellules etc. La génération du BBox fait partie du générateur de layout et elle est gérée directement par le système GDT.

Le générateur d'icônes est un peu particulier par rapport aux types précédents car il s'agit seulement de construire des blocs avec des terminaux (connectés aux terminaux de la vue schéma) pour chaque niveau dans la hiérarchie. Cette partie est appelée et utilisée par le générateur schématique pour le placement des icônes dans les schémas logiques.

La vérification des circuits est directe, les ERC/DRC sont incorporés au système et des "netlists" peuvent être extraites pour la simulation avec Lsim (simulateur de GDT très performant) ou avec SPICE, Eldo, etc.

Les paragraphes suivants montrent d'une façon plus fine l'algorithmique du générateur de multiplieurs. Les "listings" les plus représentatifs ainsi que les figures correspondantes se trouvent dans les annexes.

L'organisation des générateurs est constituée de la manière suivante :

- un fichier d'appel qui commande les générateurs
  des vues correspondantes.
- les générateurs de ces vues.
- des générateurs de niveau hiérarchique inférieur jusqu'à la cellule.
- éventuellement des cellules "dures" (sans paramètres).

Il est conseillé, pour préserver la cohérence de l'ensemble, d'organiser la base de données avec une structure arborescence qui sépare les différentes vues et applications. Ainsi, dans le répertoire du générateur on aura les sous-répertoires suivants (structure recommandée au CNET et qui facilite l'échange entre utilisateurs) :

SCHEMATIC/ générateurs de schémas et cellules dures schématiques
LAYOUT/ idem pour le niveau du layout
ICON/ idem pour la génération des icônes (boites)
fmods/ modèles de haut niveau pour les simulations fonctionnelles
generator/ fichiers généraux d'appel aux générateurs de vues

Le générateur de multiplieurs démarre par un fichier d'appel où l'on fait les vérifications des paramètres d'entrée et sont lues des variables globales liées à l'environnement (répertoires d'autres cellules et générateurs à employer) et à la technologie.

```
CELL mult_G
```

```
(
        <paramètres du générateur>
)
{
        <vérification des paramètres>
        <appel de la vue correspondante>
}
```

Les paramètres des générateurs sont une liste de variables telles que les largeurs des opérandes, la taille des alimentations, le nombre de traversées entre cellules etc. qui sont passées par l'utilisateur via l'éditeur graphique de GDT, par l'intermédiaire d'un générateur de niveau supérieur. La vérification des paramètres empêche des erreurs sur la valeur des variables (notamment introduites directement par l'utilisateur) et parce que le système ne nous permet pas un retour à l'éditeur (limitation importante), des valeurs par défaut sont prises. L'appel de la vue correspondante se déroule comme suit :

```
(…)

IF(!EXISTS(VIEW))
        STR VIEW = "ICON";

ELSE IF (STREQ(VIEW, "ICON"))
{
        PUSHDIR STRCAT (mult_dir, "/ICON");
        USE "mult.icon";
        POPDIR;
}

ELSE IF (STREQ(VIEW, "SCHEMATIC"))
{

(…)

ELSE PERROR STRCAT("Vue ", VIEW, "inconnue…");
```

L'existence de la variable globale VIEW (qui doit être modifiée par l'utilisateur qui choisit la vue désirée) est vérifiée. Si elle n'existe pas, elle est créée et positionnée à ICON. La première lecture du générateur affiche son icône à l'écran. Pour une valeur différente de VIEW le code L correspondant est appelé dans son propre répertoire.

6.2.1. Icônes

Un générateur d'icônes est typiquement la définition d'un ou de plusieurs polygones (des enveloppes rectangulaires) avec des terminaux correspondants au niveau schématique et éventuellement du texte et des commentaires. Les icônes sont employées dans les schémas logiques soit comme une vue externe d'une cellule instanciée, soit comme un bloc instancié qui possède un modèle fonctionnel associé mais pas de schéma propre (conception descendante du fonctionnel au schéma).

```
(…)

STR le_titre;
SPRINT (le_titre, "Multiplieur");
TEXT (-15,-5) W=2 le_titre;
POLYGON ICONLEV (-20,-10) (0,20):

(…)

VDD vdd (-20,-10);
GND vss (-20,-10);

(…)

IN x_in[nb_bit_donnee-1:0] R270 (-15,10);
VIEWNAME x_in[nb_bit_donnee-1:0] (-2,-15) W=0.5;

(…)
```

6.2.2. Schémas

Le générateur de la vue schématique démarre avec l'appel des générateurs des cellules à employer, des lignes de cellules et/ou des cellules "dures". Ensuite, il y a une étape de connexion des cellules et des terminaux.

```
(…)

IF(!EXISTS(addS))
CALL STRCAT(mult_dir, "/generator/facell_G") CELL addS (power_width);
```

```
(…)

WHILE (++j < nb_dig_coeff)
{
        WHILE (++i < nb_bit_donnee)
        {
                INST addS add[i][j] AT (i*50, -j*50);
                IF (j == 0) (…)
                ELSE (…)

                IN MET1 x_in[nb_bit_donnee-1:0] R270 AT (-50,0);
                VIEWNAME x_in[nb_bit_donnee-1:0] (-2,-1) W=0.5;

                NODE node_x_in[i] AT pp[nb_bit_donnee-i-1][0].d + (0,10);
                SIG node_x_in[i] (-2,1) x_in[i];

                (…)
```

A la fin du code, le générateur de la vue ICON peut être appelé pour avoir une icône associée au schéma complet qui sera affichée si l'on instancie le multiplieur dans une autre cellule (voir paragraphe 6.2.4).

```
CALL "../generator/mult_G";
CELL CELLNAME (nb_bit_donnee, nb_dig_coeff …);
ICON CELLNAME;
```

6.2.3. Layout

Le générateur de la vue layout est globalement très similaire au générateur schématique. Des générateurs de cellules et de lignes de cellules sont appelés et les résultats instanciés. Une première différence par rapport au cas schématique vient de la façon d'effectuer le placement, car normalement la mise bout à bout des cellules est employée pour optimiser l'occupation de la surface. Les terminaux coïncidents sont connectés automatiquement.

```
(…)

NUM right_move = n_feed * MET1W_MET1W + 1;
```

```
IF (i  == nb_bit-1)
        INST inv inv`(i) AT (0,0);
ELSE
{
        INST inv inv`(i);
        inv`(i).vdd_l AT inv`(i+1).vdd_r;
        MOVE inv`(i) RIGHT = right_move;
}

(…)
```

Comme deuxième différence fondamentale, il faut mentionner la partie de routage entre les cellules et les lignes (ou colonnes) de cellules. Le routeur fait partie des outils GDT et permet une programmation aisée en L. Néanmoins, les résultats obtenus ne sont pas toujours extraordinaires.

```
ROUTE VER (inv0.URX,inv0.URY)(pp`(j).LLX,pp`(j).LLY)
        HOR MET2 VER MET1;
        MOVE DOWN;
        WHILE (--k > -1)
        {
                NET inv0.out[k], pp`(j).d[k];
        }
REND
```

6.2.4. Simulations et vérification de la structure

Le test exhaustif du multiplieur comporte un nombre de calculs qui augmente très rapidement avec la largeur des opérandes d'entrée. Ceci implique, comme on l'a déjà vu dans le chapitre 5, que pour vérifier tous les cas possibles d'un multiplieur 8x8 il nous faut 10,9951 millions de simulations à 200.000 ns chacune.

Une étude en profondeur de la structure peut nous conduire à la génération de vecteurs de test aléatoires capables d'assurer des résultats statistiquement corrects et qui vont simplifier le processus de vérification. Cependant, à cause de la longueur limitée de ce travail, une procédure plus systématique a été employée.

Il s'agit, effectivement, de tester toutes les opérations possibles pour un multiplieur nxm. Le résultat obtenu à la sortie du multiplieur est comparé avec le résultat calculé par le générateur de stimuli et s'ils ne coïncident pas, un signal d'erreur est généré. A la fin de la simulation, on doit suivre les évolutions du signal d'erreur pour vérifier qu'il reste toujours à zéro. En fait, les structures testées n'étaient pas de très grande taille (un multiplieur 4x4 avec entrées complémentaires b_in et c_in, un multiplieur 2x4, un multiplieur 4x2 et un multiplieurs 6x6 sans les entrées b_in et c_in). C'était suffisant car la structure est simple et régulière pour pouvoir valider les résultats pour tous les cas.

Dans l'intention d'automatiser le processus, un générateur de stimuli a été écrit. Celui-ci possède seulement la vue icône et un modèle de haut niveau en langage M responsable du contrôle de toute la vérification. Après avoir instancié le multiplieur et la boîte de stimuli, on déclare équivalents les terminaux qui portent le même nom. La simulation peut alors démarrer (Figure 3).

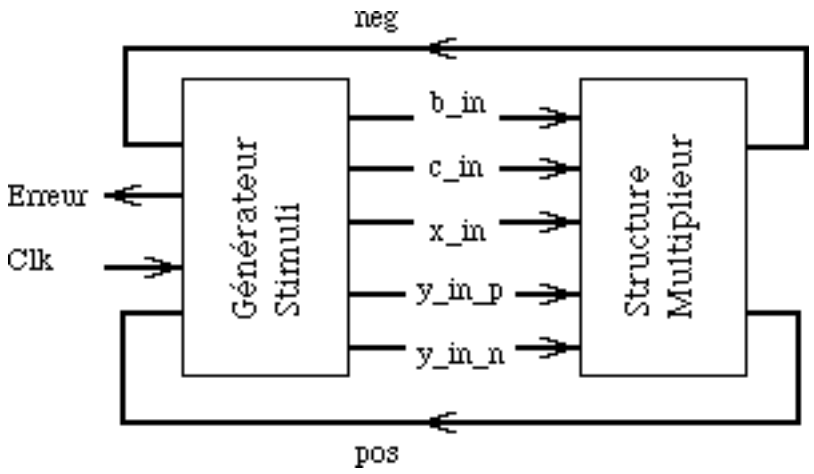


Figure 3. Schéma de principe pour la vérification de la structure

Les parties du générateur global et du générateur de l'icône pour la boîte de stimuli sont tout à fait classiques et la méthode a été décrite auparavant. Par contre, ici l'intérêt est centré sur le programme ou la description fonctionnelle du générateur de stimuli.

La boucle de simulation s'exécute sur chaque front montant d'une horloge extérieure. Ceci nous permet une référence du temps et de travailler aussi avec des registres :

```
coprocess()
if (RISE(clk))
{

(…)

WAIT (FALL(clk))
}
```

Toutes les valeurs possibles pour les variables de sortie sont prises et assignées aux bus de sortie de la boîte de stimuli (et en conséquence aux entrées du multiplieur) :

```
if (i < (1 << nb_bit_donnée))
{
        if (j < (1 << nb_dig_coeff))
        {

        (…)

        x_in[nb_bit_donnee-1:0] = i;
        c_in[nb_bit_donnee-1:0] = c;

        (…)

        y_in_p[nb_dig_coeff-1:0] = j:
        y_in_n[nb_dig_coeff-1:0] = k;

}
```

Ensuite, le résultat atteint est calculé et comparé avec la sortie du multiplieur en activant s'il est nécessaire le signal d'erreur. Noter le traitement du CA2 :

```
if (i >= (1 << nb_bit_donnee-1))
{
        mult = (i - (1 << nb_bit_donnee)) * (j - k);
}
```

```
else
{
        mult = i * (j - k);
}

(…)

ipos = pos[nb_bit_donnee+nb_dig_coeff-1:0];
ineg = neg[nb_bit_donnee+nb_dig_coeff-1:0];

res = ipos - ineg;

if (res != mult)
{
        erreur = 1;
        ERROR (RUN,NULL,"Erreur en t = %d",t);
}

else    erreur = 0;
```

7. Conclusions

Ce stage s'est donc déroulé en trois grandes étapes. La première a été l'étude et la documentation des structures de filtrage à réponse impulsionnelle finie et leur possible génération automatique.

Cette phase a permis d'effectuer le choix de la structure à employer vis à vis de la génération des filtres et de centrer le problème du dessin et de la génération du multiplieur.

Dans une deuxième partie, a pu être effectuée l'étude des algorithmes et des architectures pour la réalisation du multiplieur. Cette étape déroulée parallèlement à un approfondissement des connaissances des outils de conception et l'amélioration à la procédure de travail.

La dernière phase a eu comme but la réalisation du générateur de multiplieurs et la vérification de la structure avec l'aide du système GDT.

Lignes à suivre

La suite de ce travail peut être développée selon les lignes proposées ci-dessous :

1. Dessin des cellules de base pour la partie du générateur de layout avec son optimisation et caractérisation électrique. Minimisation des temps et optimisation des additionneurs pour un cas Carry-Save.

2. Insertion automatique des cellules de pipeline.

3. Etude et adjonction du VMA.

4. Etude de la simplification des colonnes du multiplieur pour le cas SD-CA2 fixe avec la résolution du problème des incompatibilités de signes.

5. Intégration des résultats dans le générateur de filtres FIR.

8. Annexes

Etude des structures de filtrage, des VMA et des multiplieurs

Etude des structures pour un multiplieur CA2-SD

Générateurs

9. Bibliographie et Références

/1/ Asato C. *et al.* : "A Data-Path multiplier with Automatic Insertion of Pipeline Stages", IEEE Journal of Solid-State Circuits, Vol. 25, N° 2, April 1990, pp 383-387.

/2/ Avizienis. A. : "Signed Digit Number Representations for Fast Parallel Arithmetic", IRE Transactions on Electronic Computers, Vol. EC-10, N° 3, September 1961, pp 389-400.

/3/ Cappello P. : "Optimal Choice of Intermediate Latching to Maximize Throughput in VLSI Circuits", IEEE Transactions on Acoustics Speech and Signal Processing, Vol. ASSP-32, N° 1, February 1984, pp 28-33.

/4/ Cappello P. Wu C. : "Computer Aided Design of VLSI FIR Filters", Proceedings of the IEEE, Vol. 75, N° 9, September 1987, pp 1260-1271.

/5/ Denyer P. B. Myers D. J. : "Carry-Save Arrays for VLSI Signal Processing", pp 151-160.

/6/ Hatamian M. Cash G. L. : "Parallel Bit-Level Pipelined VLSI Designs for High Speed Signal Processing", Procs of IEEE, Vol. 75, N° 9, September 1987, pp 1192-1202.

/7/ Hatamian M. Rao S. K. : "A 100 MHz 40-Tap Programmable FIR Filter Chip", ISCAS 90, pp 3053-3056.

/8/ Henlin D. A. *et al.* : "A 16x16 Bit Pipelined Multiplier Macrocell", IEEE Journal of Solid-State, Vol. SC-20, April 1985, pp 542-547.

/9/ Hwang K. : "Computer Arithmetic Principles, Architecture and Design", John Wiley and sons 1979.

/10/ Hwang. K. : "Global and Modular Two's Complement Cellular Array Multipliers". IEEE Transactions on Computers, Vol. C-28, N° 4, April 1979, pp300-306.

/11/ Irwin M. J. Owens R. M. : "Digit Pipelined Arithmetic as Illustrated by the Paste-Up System : A Tutorial", IEEE Computer, April 1987, pp 61-73.

/12/ Ishikawa M. *et al.* : "Automatic Layout Synthesis for FIR Filters using a Silicon Compiler", ISCAS 90, pp 2588-2591.

/13/ Jain R. *et al.* : "FIRGEN : a CAD System for Automatic Layout Generation of High Performance Filters", IEEE 1990 Custom Integrated Circuits Conference, pp 14.6.1-14.6.4.

/14/ Joanblanq C. *et al.* : "A 54 MHz CMOS Programmable Video Signal Processor for HDTV Applications", IEEE Journal of Solid-State Circuits, Vol. 25, N° 3, June 1990, pp 730-734.

/15/ Knowles *et al.* : "Bit Level Systolic Architectures for High Performance IIR Filtering", Journal of VLSI Signal Processing, February 1989, pp 9-24.

/16/ Kwan H. K. : "Systolic Realisation of Linear Phase FIR Digital Filters", IEEE Transactions on Circuits and Systems, Vol. CAS-34, N° 12, December 1987, pp 1604-1605.

/17/ McCanny J. V. White J. C. : "VLSI Technology and Design", Microelectronics and Signal Processing series, Academic Press 1987.

/18/ Muller J. M. : "Arithmétique des Ordinateurs, Opérateurs et Fonctions Elémentaires", série Etudes et Recherches en Informatique, Ed. Masson 1989.

/19/ Nagamatsu M. : "A 15 ns 32x32 Bit CMOS Multiplier with an Improved Parallel Structure", IEEE 1989 Custom Integrated Circuits Conference, pp 10.3.1-10.3.4.

/20/ Noll T. G. : "A 40 MHz Programmable Semi-Systolic Transversal Filter", IEEE International Solid-State Circuits Conference, February 1987, pp 180-181; 390-391.

/21/ Noll T. G. : "Carry-Save Arithmetic for High-Speed Digital Signal Processing", ISCAS 90, pp 982-986.

/22/ Noll T. G. : "Semi-Systolic Maximum Rate Transversal Filters with Programmable Coefficients", Systolic Array Applications, pp 103-112.

/23/ Privat G. : "A Novel Class of Serial-Parallel Redundant Signal Digit Multipliers"

/24/ Privat G. : "Architectures Spécialisées de Circuits VLSI pour le Traitement Numérique du Signal", Note Technique NT/CNS/CCI/65 CNET Grenoble, Mai 1987.

/25/ Privat G. Paris L. : "Design of Digital Filters for Video Circuits", IEEE Journal of Solid-State, Vol SC-21, N° 3, June 1986, pp 441-445.

/26/ Yoshino T. *et al.* : "A 100 MHz 64-Tap FIR Digital Filter in 0,8µm BiCMOS Gate Array", IEEE Journal of Solid-State Circuits, Vol. 25, N° 6, December 1990, pp 1494-1501.