UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES
ÉCOLE D'INGÉNIERIE

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
À LA MAÎTRISE EN GÉNIE ÉLECTRIQUE (M.Sc.A)

PAR
FRÉDÉRIC MORIN

PIPELINE PAR VAGUES D'UNITÉS ARITHMÉTIQUES POUR LA COMMUNICATION À TRÈS HAUT DÉBIT

AVRIL 2004

« À Jennifer et à mes enfants Tristan et Xavier...

...à la mémoire de ma mère, Raymonde Élisabeth Béland...»

Remerciements

J'aimerais, en premier lieu rendre hommage à ma douce moitié, Jennifer Anne Russett, qui m'a soutenu tout au long de mes études et durant cette quasi-interminable période de rédaction. Sa persévérance, sa patience, son amour et ces nombreux conseils inestimables m'ont permis enfin de terminer ce mémoire. Elle m'a donné les deux plus beaux cadeaux qu'un homme aurait pu espérer, mes enfants Tristan et Xavier. Je leurs dois à tous une grande partie des nombreuses heures investies dans ce travail.

Je tiens à remercier chaleureusement mon directeur de mémoire, le professeur Daniel Massicotte, pour son dévouement à la recherche scientifique et à ces connaissances qui m'ont permis de développer en moi un sens critique de la recherche scientifique, et surtout mon perfectionnement à la recherche en microélectronique. Je tiens aussi à le remercier pour l'intérêt qu'il a accordé à ce travail et pour son aide précieuse à la rédaction/correction de ce mémoire. Qu'il trouve ici l'expression de mon entière gratitude et le témoignage de mon amitié.

Je tiens à remercier de tout cœur les membres du jury, Professeur Tanguy Risset et Professeur Mansour Dahmane pour avoir évalué cette thèse et pour leurs commentaires qui

ont contribués à la qualité de ce mémoire.

Je remercie le corps professoral, le personnel du département et toutes les personnes, qui de près ou de loin ont su m'aider dans mon cheminement et à la réalisation de ce travail, qu'ils trouvent ici l'expression de ma gratitude et de ma sympathie.

Je remercie aussi les professeurs Daniel Massicotte, Kodjo Agbossou et Yves Dubé pour leurs supports financiers durant mes études de maîtrise. Sachez qu'ils m'ont été d'un grand soutient sans quoi j'aurais été forcé de mettre un terme à mes études.

Je voudrais aussi remercier tous mes coéquipiers du Laboratoire de signaux et systèmes intégrés de l'Université du Québec à Trois-Rivières pour leur soutients, leurs conseils et pour avoir entretenu une atmosphère chaleureuse au sein du groupe.

Je voudrais également remercier mon collègue Martin Demontigny pour son aide et support moral, pour qui la robotique et l'asservissement non plus de secrets. Son amitié et ses précieux conseils m'ont permis de ne pas lâcher lors de moment difficile.

Finalement, je ne saurais passer sous silence mon entière gratitude à mon fidèle compagnon Martin Vidal pour les moments mémorables en sa compagnie. Ses conseils et ses connaissances inestimables furent très enrichissants et me serviront tout au long de ma vie professionnelle. Son dynamisme et son enthousiasme communicatif ont apporté d'heureux débats philosophiques et de fameux travaux d'équipes.

Résumé

Ce mémoire propose une architecture et des unités arithmétiques utilisant la technique du pipeline par vagues comme méthodes d'intégration à très grande échelle (ITGE) sur silicium. Améliorer la qualité, augmenter la distance entre les stations de régénération du signal et si possible obtenir un gain en vitesse de transmission auquel l'information à travers un canal de communication numérique sera transmis sont tous possible si un algorithme adaptatif de reconstitution des données transmises est utilisé.

L'égalisation de canaux considérés comme algorithme adaptatif consiste à annuler les effets du canal afin de réduire les ISI et CCI. Il s'agit d'un traitement numérique qui consiste à filtrer le signal reçu avec la réponse impulsionnelle inverse du canal. Dans la plupart des cas comme la réponse impulsionnelle du canal est souvent inconnue, l'égaliseur sera adaptatif utilisant ainsi des données d'apprentissages.

Dans le cadre de cette recherche l'implantation d'une architecture systolique à base d'un réseau de neurone artificiel sera effectuée. Elle aura comme avantage d'être régulière au niveau structural et de simplifier les communications locales entre les processeurs élémentaires.

L'architecture sera basée sur une topologie multicouche utilisant une fonction de

décision de type canonique linéaire par morceaux nommée PL-MNN (*Piecewise Linear Multilayer Neural Network*). Chaque processeur aura une quantité de calcules assez élevés qui nécessite donc une étude sur les différentes techniques d'augmentation du débit et ainsi améliorer la vitesse auquelle notre architecture peut procédée les symboles à corriger

La validation de l'architecture a été effectuée grâce à un modèle VHDL et un modèle analogique utilisant Pspice incluant les capacitées parasite fût utilisée pour la validation et l'évaluation des performances Analog Artist® et Virtuoso® de Cadence®. La technologie CMOS de 0.5 µm (CMOSIS5, SPTM) offert par la société canadienne de microélectronique (CMC) et Hewlett Packard fût utilisée.

Finalement, l'objectif principal du travail de recherche est l'étude du pipeline par vagues pour le développement d'une architecture ITGE à très haut débit dédié à l'égalisation de canaux. Le pipeline par vagues que nous allons étudier a comme particularité d'être implantée à partir d'une cellule nommée NPCPL, réalisant les fonctions logiques de base et possédant des délais de propagation équilibrés. Le développement et la réalisation de cette cellule sont la base de ce travail de recherche. Cette méthode de pipeline a été proposée à un algorithme de reconstitution de signaux basé sur le filtre de Kalman [ELO98b], [MAS98]. Notre projet est une suite allant vers la réalisation des cellules de base en vue d'une fabrication.

Table des matières

Liste des Figures

Liste des tableaux

Liste des abréviations

ALU	Arithmetic and Logic Unit
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
CAO	Conception Assistée par Ordinateur
CCI	Co-Channel Interference
CLA	Carry Locking Adder
CLB	Combinatory Logic Block
CMC	Canadian Micro-electronic Corporation
CMOS	Complementary Metal Oxide Semiconductor
CPL	Complementary Pass Logic
CSA	Carry Save Adder
DPTL	Differential Pass Transistor Logic
DPTL	Double Pass Transistor Logic
FA	Full Adder
IP	Intellectual Property
ISI	Inter Symbol Interferences

ITGE	Intégration à Très Grande Échelle
LSB	Least Significant Bit
LSSI	Laboratoire de Signaux et Systèmes Intégrés.
MSB	Most Significant Bit
NPCPL	Normal Process Complementary Pass transistor Logic
PAM	Pulse Amplitude Modulation
PE	Processeur Élémentaire
PL-MNN	Piecewise Linear Multi-layer Neural Network
PL-RNN	Piecewise Linear Recursive Neural Network
QAM	Quadratic Phase Amplitude Modulation
RCA	Ripple Carry Adder
SCM	Société Canadienne de Microélectronique
SOC	System On Chip
TPL	Tree Pass Logic
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integrated
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WTGL	Wave pipeline Transmission Gate Logic

1 Introduction

Les communications numériques se sont développées de façon phénoménale au cours des dernières années. Que l'on parle de transmissions par câble ou sans fil, les applications développées demandent de plus en plus de bandes passantes ce qui amène plusieurs problèmes dans la chaîne de transmission.

Tout d'abord, les récepteurs et les émetteurs doivent être en mesure de traiter un débit élevé d'information. À l'heure actuelle, il est facile de parler de transmissions approchant les 10Gbps sur les fibres optiques et les communications à 40Gbps sont presque une réalité sur silicium. Le traitement de l'information à ces débits contitue un véritable défi.

Pour les communications sans fils, les débits standard de transmission de la nouvelle génération seront de 2-3 Mbps. Les problèmes alors rencontrés sont différents. Lorsque les signaux transitent dans l'air, la durée des symboles transmis est allongée ce qui crée un chevauchement ou interférence entre les symboles (ISI : *Inter Symbol Interference*) [PRO95]. Un phénomène semblable se produit sur les fibres optiques; il s'agit de la diffusion de la lumière qui amène les signaux à utiliser un spectre de fréquence plus large que celui qui leur est attribué et à empiéter sur les canaux de communication de fréquences voisines que l'on nomme interférence entre les canaux (CCI:*Co-Channel*

Interference).[CAM98]

Afin de combattre ces altérations des transmissions et pour permettre une qualité de service acceptable, les principales méthodes de corrections d'erreur sont le codage et l'égalisation de canaux.

Le codage (Reed-Solomon, Viterbi, BCH, etc...) consiste à créer une redondance dans le message transmis afin de détecter et corriger les éventuelles erreurs à la réception. De façon générale, les codes qui possèdent un taux de redondance élevé, c'est-à-dire que le rapport entre le nombre de bits d'information et le nombre de bits transmis est faible, sont aptes à détecter et à corriger un plus grand nombre d'erreurs donc ayant une distance de Hamming élevée. Comme nous l'avons spécifié, l'information est diluée dans le message transmis, se traduisant donc par une perte de la bande passante allouée.

L'égalisation de canaux, quant à elle, consiste à annuler les effets du canal afin de réduire les ISI et CCI. Il s'agit d'un traitement numérique qui consiste à filtrer le signal reçu avec la réponse impulsionnelle inverse du canal. Dans la plupart des cas, comme la réponse impulsionnelle du canal est souvent inconnue, l'égaliseur est adaptatif.

L'implantation d'algorithme de traitement de signaux nécessite une plus grande quantité de calculs justifiés d'abord par la classe du problème ce devant d'être résolue. C'est-à-dire, quel médium et quel type de transmission aura-t-il lieu. Et d'autre part, la complexité des calculs devant être effectués par l'algorithme à être implanté. L'implantion d'une architecture au niveau ITGE pour le traitement numérique apporte de nouveaux problèmes qui ne sont pas rencontrés lorsque le traitement est effectué au niveau logiciel (*software*) *ou*

utilisant des architectures préfabriquées (DSP, µ-contrôleur).

Dans cette étude, nous évaluerons d'une part la capacité des différents égaliseurs pouvant être implantés pour des applications à haut débit tel les communications par fibres optiques. Et d'autre part nous évaluerons les techniques d'augmentation du débit pouvant être implantées sur nos unités arithmétiques et logiques.

1.1 Problématique

La méthode du pipeline est une méthode d'augmentation du débit à l'intérieur même d'un bloc de logique combinatoire (CLB). Cette méthode devient nécessaire lorsque les techniques conventionnelles, comme la parallélisation où la modification de la structure de l'algorithme ne permet plus d'augmenter le débit de l'architecture proposée. L'idée principale derrière cette étude est l'application du pipeline par vague pour la conception d'une architecture dédiée à l'égalisation de canaux.

L'utilisation d'une technique comme le pipeline conventionnel permet d'augmenter le débit d'une architecture sans nécessairement devoir diminuer l'échelle de la technologie qui sera choisie. Cependant, son utilisation a comme conséquence néfaste d'augmenter énormément les problèmes de synchronisations de l'horloge et la surface d'intégration. Ainsi, l'utilisation d'une technique d'augmentation de débits ne s'applique pas toujours. On doit d'abord vérifier s'il est réellement nécessaire d'augmenter la vitesse de traitement. Il existe des domaines non axés sur la rapidité de traitement des données, comme par

exemple, le domaine de la spectrométrie. Préférant ainsi des traitements portés sur la qualité (précision) des données échantillonnées que sur la quantité (rapidité). Ceci dit, il existe d'autres domaines par exemple, la télécommunication où la demande de vitesse est en constante évolution. Dès lors, nous tentons constamment d'augmenter le débit des architectures dédiées au traitement du signal en utilisant des algorithmes de plus en plus complexes par conséquent une augmentation assurée du nombre de calculs. Nous croyons fortement que l'utilisation d'algorithmes complexes sera de plus en plus nécessaire. Cette demande aura comme objectif d'augmenter substantiellement le débit de l'information transmise tout en compensant pour les limites de transmissions reliés au milieu physique du canal. Néanmoins, cette complexité algorithmique aura un lien direct avec l'augmentation de la latence au niveau architectural, plus particulièrement lors d'une intégration à très grande échelle, causée en général par les unités arithmétiques (multiplieurs, diviseurs, etc.) devant être utilisées. Alors, l'utilité de développer une architecture intégrée à très grande échelle devient une nécessité en contraste avec une implantation à l'intérieur d'architecture généralisée comme les DSP (*Digital Signal Processing*).

Les méthodes utilisant les techniques du pipeline ont comme rôle principal d'augmenter le débit à l'intérieur même de l'architecture lors d'intégration sur silicium. En effet, cette optimisation pointe directement au problème principal de limitation du débit en modifiant principalement les blocs de logique combinatoire constituant le berceau de l'engorgement des données. L'étude portera sur plusieurs types de pipeline, pour permettre de faire un choix éclairé d'une technique d'augmentation du débit pouvant être appliquée à notre

architecture.

Les types de pipeline les plus connus sont : le pipeline conventionnel [ELO98], [GHO94], [JOU97], le micro-pipeline [BAR97], [DAY95], [NOO97], [SUT89], [TAY98], le pipeline Domino [HSI95], [LIE92], [WEI95], et le pipeline par vagues (synchrone et asynchrone) [BUR98], [CHU96], [COT96], [DAE98], [GHO95], [GRA92], [GRA93], [HAU98], [HUG91], [MAR97], [MAS98], [NOW94], [NOW94b], [NOW95], [NOW95b], [WON93]. En premier lieu, le pipeline conventionnel est constitué de plusieurs étages de registres à l'intérieur du bloc de logique combinatoire (CLB), ceci permet de stabiliser les données en introduisant des éléments synchrones (Temporisateur ou registre) à l'intérieur du CLB. En effet, la méthode du pipeline divise par un nombre N d'étages le bloc combinatoire et par le fait même augmente le débit par un facteur N. Cependant, l'utilisation du pipeline conventionnel emmène aussi une augmentation de surface et de puissance. De plus, il augmente le nombre de cycle total et le réseau d'horloge peut devenir une contrainte significative. Malgré ces défauts, il constitue toujours une excellente technique de maximisation du débit.

Le micro-pipeline a pour but la transmission de donnée asynchrone. C'est-à-dire, il est conçu sur le principe du receveur/transmetteur. L'avantage de ce type d'augmentation du débit réside dans l'application de transmission de donnée sans transmettre l'horloge, ce qui permet d'éviter une diminution de la pente et du déphasage temporel (*clock skew)* de l'horloge pendant la distribution de celle-ci. Le nombre d'étages transmis de cette façon permet des vitesses quasi comparables au pipeline conventionnel, sans les problèmes reliés

à la distribution de l'horloge.

Le pipeline Domino est constitué de portes logiques dynamiques de type Domino. Ce type de portes permet d'utiliser N+4 transistors contrairement à une porte statique CMOS utilisant 2N transistors où N représente le nombre de portes statiques de type P ou N qui seraient normalement utilisées [SAV88]. Sa consommation de puissance est semblable aux portes logiques statiques. Cependant, le principe de fonctionnement reste très complexe.

Le pipeline par vagues, a comme principe d'introduire à l'intérieur du CLB, un nombre de vagues de données supérieur au délai de propagation à travers la logique combinatoire ceci en diminuant la période de l'horloge mère entre les deux registres principaux sans nécessairement utiliser de registre interne. Ainsi, une diminution concrète de surface d'intégration et une augmentation du débit liée aux nombres de vagues sont introduites. Cependant, cette méthode d'implantation demeure toutefois complexe et demande d'aller aux limites de la technologie d'intégration. Plus spécifiquement, au niveau des délais des transistors lors d'ajustements de précision (dimension des transistors) et lors des ajustements grossiers (longueurs des chemins). La figure 1.1 représente grossièrement les différences principales entre une logique combinatoire RTL conventionnelles vis-à-vis aux logiques combinatoires RTL pipeline conventionnel et pipeline par vagues.

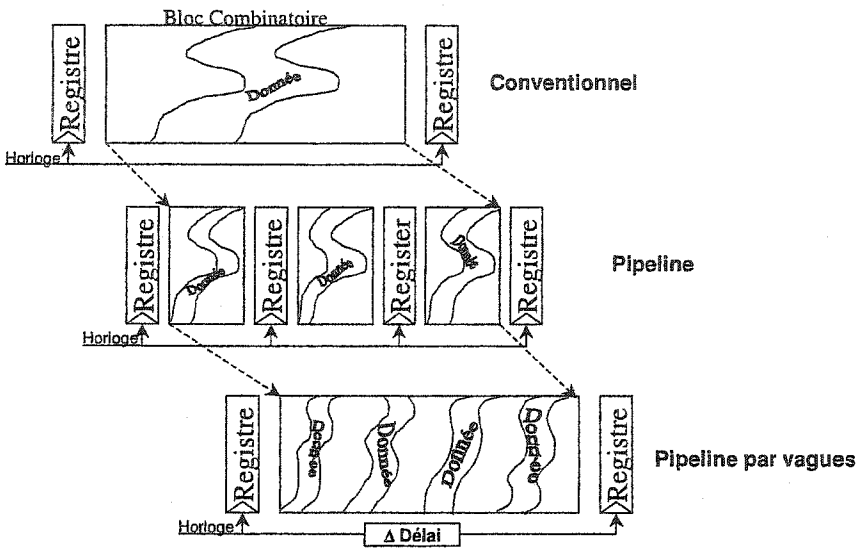


Figure 1.1) Concept du pipeline par vagues

En bref, les deux types (conventionnel et micro) cherchent à diminuer le trajet de la donnée la plus longue du bloc de logique combinatoire, ce qui pourrait être considéré comme un affinement grossier (*corse tuning*) dans le jargon. Alors que les types de pipelines, Domino et « par vagues », ont un ajustement de précision (*fine tuning)* en plus d'un affinement grossier, ce qui pousse vers une diminution du délai dynamique des transistors, qu'on appelle généralement ajustement de précision. Les deux derniers types de pipeline (Domino et par vagues) permettent d'utiliser l'intégration de portes logiques axées sur une conception niveau transistor. Nous discuterons en profondeur de l'optimisation utilisant l'affinement grossier et l'affinement de précision au Chapitre 4.

L'implication d'une nouvelle technique emmène nécessairement des questions encore non répondues. L'une d'elle est l'utilisation d'outils CAO. Très rare, ils constituent

toujours un obstacle puisque leurs maniements sont encore très peu utilisés. Comme exemple, l'utilisation d'un logiciel pouvant introduire de façon automatique des temporisateurs (*buffer*) aux chemins les plus courts pour équilibrer les chemins n'est pas encore disponible sur le marché. Il faut des lors, concevoir manuellement les pipelines par vagues en utilisant principalement des outils schématiques. En plus, l'utilisation de langages de haut niveau comme le VHDL ne peut simplement pas être utilisé seul pour l'implantation d'un pipeline par vagues. Le VHDL est utilisé souvent pour développer des architectures de type conventionnel ou pipeline conventionnel puisqu'il existe déjà plusieurs librairies CMOS offrant plusieurs combinaisons de logique booléenne. Cette recherche emmènera des réponses partielles ou complètes aux questions suivantes :

- Quelles sont les techniques récentes appliquées à la résolution d'une augmentation de débit?
- Quels sont les problèmes techniques qui devront être affrontés, comme par exemple le transport d'une horloge rapide et l'implantation d'une technologie d'intégration à très grande échelle (ITGE).
- Quels sont les paramètres externes qui peuvent influencer le débit d'une architecture lors de l'utilisation du pipeline par vagues?

1.2 Objectifs

L'objectif principal de cette recherche est d'apporter une contribution à l'avancement d'une récente technique peu connue et d'emmener le développement d'une nouvelle architecture dédiée à l'égalisation de canaux utilisant la méthode du pipeline par vagues comme élément principal. L'apprentissage des différentes méthodes d'augmentation du débit en comparant l'avantage principal d'utiliser le pipeline par vagues comme méthode pour permettre une plus grande transmission est notre premier objectif. Ceci dit, pour permettre à un algorithme comme le réseau de neurones d'être utilisé à son plein potentiel, nous devons absolument l'implanter en utilisant une méthode d'intégration avancée en VLSI (*very low scale of integration*) ou en français ITGE (intégration à très grande échelle).

Ainsi l'étude portera plus particulièrement sur les techniques d'implantation pipeline par vagues. La particularité de ce projet sera l'implantation d'une cellule nommée NPCPL qui permettra de réaliser des fonctions logiques de base avec la particularité d'avoir des délais de propagation équilibrés et d'obtenir de meilleurs débits architecturals. Le développement et la réalisation de cette cellule sont la base de ce travail de recherche. Cette méthode de pipeline a été proposée à un algorithme de reconstitution de signaux basé sur le filtre de Kalman [B. Elouafay, mémoire de maîtrise, UQTR], [MAS98] et le présent projet est une suite allant vers la réalisation des cellules de base en vue d'une fabrication.

Enfin, il ne faut pas oublier que la création et le développement de dessin de masque d'unités arithmétiques emmèneront une certaine expérience positive à l'école d'Ingénierie

de Trois-Rivières.

1.3 Méthodologie

La première étape est une recherche bibliographique permettant de sélectionner les différents algorithmes appliqués à l'égalisation de canaux, ainsi que des références pertinentes sur la méthode d'implantation du pipeline par vagues en ITGE. Ensuite, vient l'étude des architectures dédiées à l'algorithme retenu, où l'implantation de la technique du pipeline par vagues est possible. Une des solutions retenues pour traiter le problème engendré par la transmission de la logique est d'utiliser une cellule NPCPL (Normal Process Complementary Pass Transitor Logic). La cellule NPCPL permet en effet d'équilibrer les délais accordés aux portes logique tel qu'expliqué dans [GOS95]. Finalement, la proposition d'un processeur spécialisé utilisant le pipeline par vagues dédié à l'égalisation de canaux permettra une contribution à la recherche dans le développement d'architectures à très haut débit pour la communication. Il conviendra de concevoir les principaux blocs à partir de la cellule NPCPL à la base du processeur spécifique développé. Par ailleurs, l'étude d'une implantation de la distribution d'horloges à très haute vitesse en utilisant un PLL (Phase Locked Loop) sera conçue en vue d'une utilisation comme multiplieur de fréquence et verrouilleur de phase. Le tout sera conçu puis simulé à l'aide de plusieurs outils CAO (Matlab, Mentor Graphics, Synopsys et Cadence).

En bref, la réalisation de ce projet est justifiée par le cheminement des étapes qui suivent:

- Une recherche approfondie des nouvelles techniques d'augmentation du débit.
- Une étude reliée au pipeline par vagues.
- Un apprentissage des algorithmes dédié à l'égalisation de canaux
- Un apprentissage d'outils et méthodes pour la conception assistée par ordinateur (CAD) des circuits ITGE.
- Un apprentissage de la conception d'unités arithmétiques sur dessin de masque.
- Un développement de l'architecture dédié à l'égalisation de canaux

1.4 Organisation de ce mémoire

En premier lieu, une étude axée sur les techniques de base des algorithmes d'égalisation de canaux sera présentée au Chapitre 2. Plus spécifiquement, nous discuterons des principes à la section 2.1, des différences entre hors-ligne et en-ligne à la section 2.2, des algorithmes adaptatifs d'égalisation de canaux linéaires à la section 2.3, basés sur la logique floue à la section 2.4, basés sur les réseaux de neurones à la section 2.5 et enfin, du choix de l'égaliseur à être implanté à la section 2.6.

Ensuite, les unités arithmétiques entières seront abordées au chapitre 3 en commençant par la représentation binaire comme fondement de base à la section 3.1. Puis, nous présenterons les additionneurs binaires connus à la section 3.2. En passant à travers les additionneurs partiels (section 3.2.1), entiers (section 3.2.2), à retenue propagée (section 3.2.3), à retenue anticipée (section 3.2.4), à conservation de la retenue (section 3.2.5), à saut de la retenue (section 3.2.6), additionneurs sériel (section 3.2.7) et autres additionneurs (section 3.2.8). Après avoir discuté des additionneurs, nous présenterons les multiplicateurs connus à la section 3.3 pour passer ensuite au choix des unités arithmétique choisies à la

section 3.4.

Le chapitre 4 porte sur les principales techniques pipeline. Tout d'abord, il est question du pipeline conventionnel à la section 4.1, du pipeline asynchrone à la section 4.2 pour ensuite approfondir plus en détail sur le pipeline par vagues à la section 4.3. En passant d'abord à travers les équations de base (section 4.3.1), de l'optimisation des délais de propagation (section 4.3.2), des types de cellules disponibles en vue d'une implantation du pipeline par vagues (section 4.3.3), de la cellule choisie nommée NPCPL (section 4.3.4), ainsi que la structure ITGE qui utilisera le pipeline par vagues comme méthode d'optimisation de débit (section 4.3.5). Et finalement, le choix de la technique la plus approprié à l'égalisation sera présenté à la section 4.4.

Le chapitre 5 porte sur la réalisation des unités qui seront utilisées pour notre implantation pipeline par vagues. En premier lieu, l'intégration en technologie CMOS de 0.5 µm de notre cellule NPCPL sera présentée à la section 5.1. Ensuite nous présenterons les résultats individuels obtenus sur l'additionneur et le multiplicateur pipeline par vagues à la section 5.3. Par la suite, le multiplieur-accumulateur complet sera présenté à la section 5.3. Pour enfin évaluer les performances et les comparaisons finales proposées sur le multiplieur-accumulateur choisi, à la section 5.4.

Le chapitre 6 porte sur l'architecture axée sur l'égalisation de canaux hors-ligne à base d'un réseau de neurone. En passant par les contraintes et critères architecturals ITGE. Pour ensuite présenter le choix et la description de l'architecture à la section 6.2. Une estimation de la surface que notre architecture sera finalement présentée à la section 6.3

2 Algorithmes d'égalisation de canaux de communication

L'utilisation toujours croissante de communication analogique ou numérique prend une grande part d'importance dans le domaine du génie électrique car plusieurs branches se rejoignent. En effet, les domaines reliés à celui de la communication sont entre autres; le traitement du signal, la conception de circuit analogique et numérique, l'asservissement, et bien entendu l'intégration à très grande échelle (ITGE). Ceci dit, l'idée principale derrière une communication est de transmettre et de recevoir une information avec une certaine célérité et une fiabilité accrue. Puisque le sujet est vaste, nous limiterons ce chapitre à l'importance de l'égalisation de canaux.

De nos jours, le numérique a un grand avantage sur la transmission analogique. Il est beaucoup moins sensible aux paramètres externes et permet d'être régénéré périodiquement lorsqu'il est transmis sur une longue distance, éliminant ainsi les effets néfastes reliés au bruit [PRO94]. En effet, durant une transmission analogue, le bruit additionné au signal sera négativement amplifié lors de sa régénération périodique aux endroits spécifiques. De plus, le numérique peut éliminer la redondance appliquée au signal avant la démodulation et ainsi conserver la largeur de bande du canal de transmission. Finalement, la transmission

numérique coûte souvent moins chère comparativement à la communication analogique au niveau matériel [LEE94].

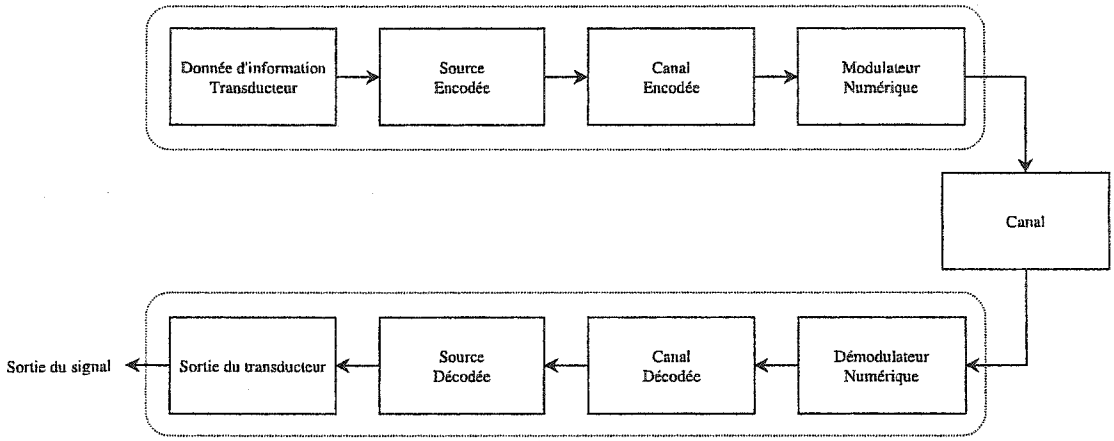


Figure 2.1) Communication numérique élémentaire

Le canal de transmission est d'abord et avant tout, le médian dans l'échange de données, Il sert de lien entre le transmetteur et le receveur. Il existe plusieurs milieux physiques classiques reliés au canal, comme:

- Une paire de fils de cuivre conduit par un courant électrique.
- Une fibre optique qui transmet un flux lumineux modulé.
- Une transmission sonore acoustique sous l'eau.
- Un Champ électromagnétique radié par une antenne électrique.

D'autres milieux peuvent aussi être considérés, comme la transmission magnétique ou optique de données emmagasinées par un lecteur de disque rigide d'un ordinateur

personnel.

Ceci dit, le milieu force le canal en limitant sa bande de transmission et caractérise sa fonction de transfert ou réponse impulsionnelle. Les problèmes rencontrés lors de transmissions sont souvent caractérisés par un bruit additif blanc de type Gaussienne créée en général par le bruit thermique. L'agitation des électrons à l'intérieur des résistances et autres circuits plus spécifiques au domaine des semi-conducteurs est une source de bruit. D'autres types de bruits et interférences externes peuvent être rencontrés. Une façon de résoudre ces problèmes est d'augmenter la puissance du signal transmis, par contre le type de canal peut limiter ce facteur. Le canal peut avoir d'autre dégradation du signal produit par l'atténuation, l'amplitude, la distorsion de phase et l'empiètement entre symbole communément appelé *intersymbol interference (ISI)* d'une même source emmène une distorsion de l'information [PRO95]. Le canal peut être décrit mathématiquement par une simple équation, où

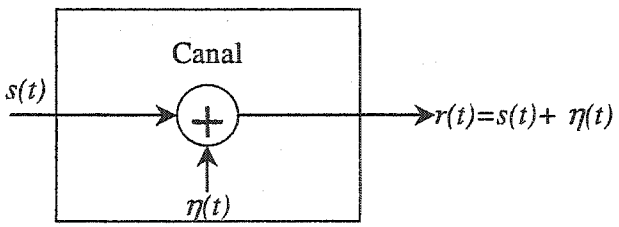


Figure 2.2) Canal avec bruit additif

$r(t) = s(t) + n(t)$ Où n représente le facteur d'atténuation (2.1)

Cependant, il existe quatre types de canaux généralement rencontrés, le premier le plus simple et le moins fréquent représenté par la figure 2.2, est un canal avec bruit blanc additif de type Gaussienne. En second, nous avons un canal avec filtre linéaire invariant, limité par sa largeur de bande, ce type de canal atténu l'énergie spectrale du signal pour permettre de ne pas empiéter sur d'autre données transmises (figure 2.2), souvent rencontré par les communications par les paires de fils de cuivre utilisés en téléphonie. Le modèle mathématique (2.2) de ce signal est une convolution entre la réponse impulsionnelle h(t) et les données transmises s(t).

$$r(t) = s(t) * h(t) + n(t) = \int_{-\infty}^{+\infty} h(\tau)s(t-\tau)d\tau + n(t) \qquad (2.2)$$

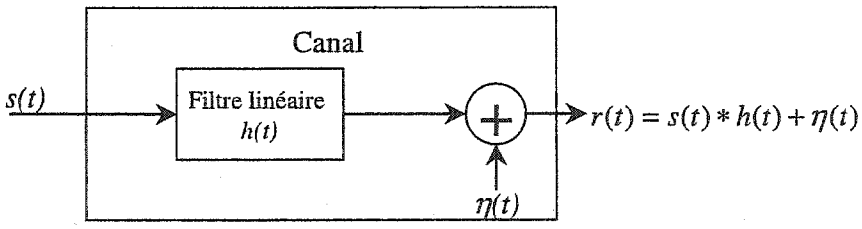


Figure 2.3) Canal linéaire invariant avec bruit additif

Comme troisième type de canal, nous avons le canal linéaire variant. Ce type de canal peut la plupart du temps être rencontré lors de transmission acoustique et lors de transmission de signaux à travers la couche ionosphérique, comme les transmissions satellites et cellulaires, où l'environnement change avec le temps. La réponse impulsionnelle du canal est $h(\tau;t)$, où $h(\tau;t)$ représente la réponse du canal à un temps t

causé par une impulsion au temps t-τ, alors que τ représente le délai entre les deux événements. La figure 2.4 est donc un exemple d'un canal linéaire variant. Ce canal sert souvent aux communication multiplex, c'est-à-dire lorsque plusieurs données sont transmises par le même canal de communication [LEE94].

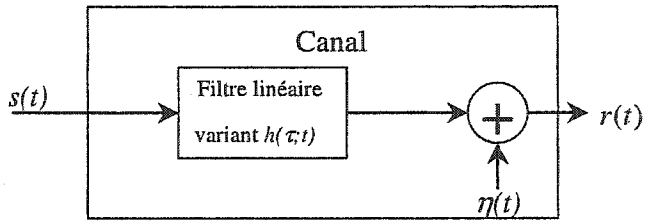


Figure 2.4) Canal linéaire variant avec bruit additif

Les trois types de canaux que nous venons de voir, représentent d'après [PRO94] une très grande majorité de canaux auquels nous devrons faire face. Cependant, il existe d'autres types de canaux, l'un d'eux est un canal non-linéaire. Le canal non-linéaire peut en majorité être estimé par sa linéarisation lorsque sa réponse est faiblement divergente. Cependant, les distorsions non-linéaires sur un canal de communication augmentent de plus en plus avec l'augmentation de la vitesse de transmission des données [WAN93]. Le modèle mathématique généralement rencontré d'un canal non-linéaire discret est :

$$\{r(t)\} = \Im[\{s(t)\}, \Theta] + n(t) \qquad (2.3)$$

Où Θ est un vecteur de paramètres de l'opérateur $\Im$. La reconstitution consiste à régulariser l'inversion de l'opérateur $\Im$, c'est-à-dire l'opérateur $\Re$ [VID99].

$$\{s(t)\} = \Re[r\{(t)\}, \Theta] \tag{2.4}$$

Un paramètre fréquemment utilisé pour mesurer la qualité de transmission est la quantité de bloc d'information (bit) corrigé sur le nombre de donnée transmis c'est-à-dire le nombre de bit sans erreur reçu communément nommée le BER (*bit error rate*). Plusieurs méthodes existent pour corriger les erreurs générées par les bruits créés sur le canal. Les méthodes proposées sur ce projet permettent un excellent BER. Ainsi l'égalisation de canaux, plus particulièrement les algorithmes : linéaires, adaptatifs et de boucle de retour (DFE), permettent de corriger le canal de transmission en utilisant le modèle inverse du canal. L'avantage d'utiliser l'égalisation de canaux c'est qu'elle est axée principalement sur la diminution du nombre de bit erroné lors de transmission (BER). Permettant ainsi d'obtenir des vitesses supérieures aux vitesses normalement atteintes lorsque nous nous limitons aux vitesses régies par le théorème de Nyquist [LEE94]. En plus, l'utilisation d'égalisation de canaux permet non seulement de filtrer le bruit blanc causé principalement par le récepteur, mais surtout de diminuer les **ISI** (l'empiètement entre symbole d'un même signal).

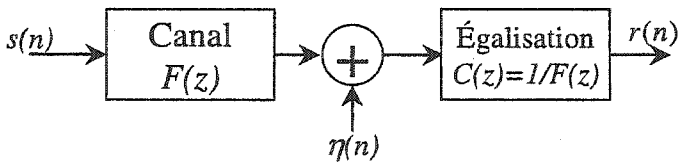


Figure 2.5) Exemple d'un égalisateur de canaux

Il existe trois grandes catégories d'égalisation de canaux [PRO95]. La première doit

prendre l'entrée et la sortie du canal avant le traitement pour identifier le canal et son inverse. Ces types de filtres, peu pratique, se nomment égalisateur linéaire. Ils utilisent des filtres sous optimum où on doit faire un apprentissage hors-ligne des poids (gains) utilisés. Les filtres adaptatifs doivent eux utiliser constamment l'erreur entre l'entrée et la sortie pour estimer le modèle de la fonction de transfert peu importe l'apprentissage (en-ligne ou hors-ligne). Il existe aussi des égalisateurs optimaux, où l'inverse du canal est estimé sans connaître son entrée, nommé auto-égalisateur (*Blind Equalisation*). Dès lors, il existe plusieurs approches à un problème et l'égalisation n'est pas une exception. Nous verrons plus loin différents types d'égalisateur de type optimum et sous optimum.

En bref, la transmission de données numériques est constituée principalement de données comprises entre -1 et 1. Ce qui facilite en quelque sorte la probabilité lors de l'adaptation des poids durant l'apprentissage. Un exemple de transmission de données additionnées d'un bruit blanc est représenté par figure 2.6.a. Alors que, la figure 2.6.b représente la sortie de l'égalisateur [VID99].

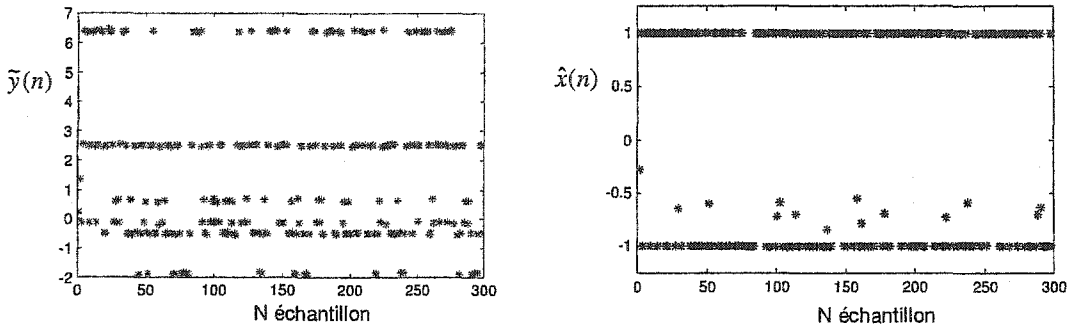


Figure 2.6) Exemple d'une transmission avant l'égalisateur (a), et après (b)

Les sous-sections suivantes serviront d'introduction au problème d'égalisation de canaux et des algorithmes proposés.

2.1 Principe de l'égalisation de canaux

Pour ce mémoire, nous allons plus particulièrement nous concentrer sur les algorithmes adaptatifs pour l'égalisation puisqu'ils ont souvent prouvés leur efficacité et leut robustesse au bruit. Ainsi, puisqu'ils sont polymorphes, ils varient automatiquement selon l'erreur fournie lors de sa validation. L'égalisation adaptative est un excellent candidat pour le choix de notre algorithme, puisqu'il est quasi impossible d'obtenir un canal qui ne varie pas avec le temps. Un exemple simple serait le transfert de données par modem téléphonique. Le canal varie uniquement lors de la connexion initiale, pour ensuite être stable tout au long de l'appel. Ceci peut facilement être considéré non variant, cependant le filtre devra être flexible initialement [PRO95].

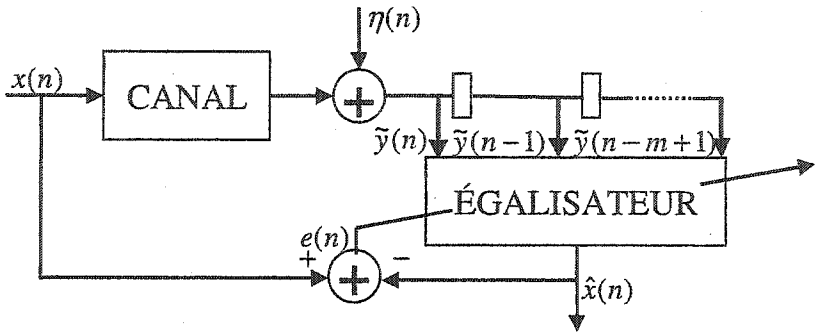


Figure 2.7) Exemple d'un égalisateur adaptatif

Ceci dit, l'égalisation adaptatif pour les communications numériques à été initialement développé par R.W. Lucky (1965, 1966). Aidé par les recherches de filtre adaptatif par B. Windrow et M.E. Hoff, Jr en 1960. Son algorithme a été initialement développé pour le critère de distorsion maximum (*peak distortion criterion*), ce qui l'emmena à développer l'algorithme *zero-forcing*. Ces études furent révolutionnaires et emmenèrent par la suite au développement de modem à haute vitesse, où il a été publié sur le sujet durant cinq ans. Ces études permirent le développement commercial de modem à très hauts débits de 9600 à 56000 bits/sec sur un canal téléphonique. Parallèlement, l'algorithme de Wiener de moyenne des moindre carrées (LMS) qui a été ensuite modifié par Windrow (1966), et ainsi que son utilisation à l'égalisation adaptative pour des signaux de valeurs complexes a été premièrement mise-à-jour, décrit et étudié par Proakis et Miller (1969). La percé révolutionnaire des travaux par Lucky en 1965, emmena aussi le développement de modulation de code de treillis plus poussé par Undergerboeck et Csajka (1976). Par la suite, d'autres études ont suivi, comme les recherches de Godard (1974), Picinbono (1978),

Morf et al. (1977.1979), Makhoul (1978), Satorius et Pack (1981), Satorius et Alexander (1979), Ling et Proakis (1982, 1984, 1985), de grand savant. L'algorithme RLS Kalman pour l'égalisation de canaux adaptatif a été premièrement introduit par Falconer et Liung (1978). De plus, les premières recherches sur l'auto-égalisation (*Blind equalisation*) furent introduites par Sato (1975). Pour ensuite, finir sur les communications multiplex par constellation (compression du signal utilisant la phase). [HAY96], [LEE94], [PRO94], [PRO95].

2.2 Adaptation hors-ligne et en-ligne

L'objectif principal de l'égalisation adaptatif est d'adapter les coefficients d'un filtre adaptatif quelconque en vue de minimiser le bruit et l'empiètement entre les symboles d'un même signal (*Inter-Symbol Interference*) ISI émis à la sortie, pour ensuite être saturé par la fonction de décision (-1,1). L'adaptation des coefficients est alors fonction de l'erreur du signal. Cette erreur, par la suite, détermine la direction du coefficient en fonction du symbole auquel son erreur tend vers un minimum stable. Avant tout, l'algorithme choisi est défini par ces propres règles initiales. Il faut donc définir en premier des poids qui pourront converger le plus près et le plus rapidement possible au point de stabilité asymptotique de la fonction de transfert du canal régis par le type de minimum choisi. Généralement, le MSE (*Mean Square Error*) ou l'erreur minimum de la moyenne des carrés est choisie. Par la suite, il faut déterminer si une adaptation en-ligne ou hors-ligne est absolument

nécessaire. En effet, le choix de l'adaptation en-ligne ou hors-ligne est principalement déterminé par le type de canal auquel l'égalisation adaptative sera appliquée.

- Adaptation en-ligne:

  Les poids de l'algorithme sont constamment modifiés en fonction du canal variant.

- Adaptation hors-ligne:

  Les poids sont introduits antérieurement au filtrage, et doivent donc estimer le canal qu'ils seront appliqués.

Ce qui détermine la différence entre une adaptation en-ligne ou hors-ligne est d'abord la charge de calculs. L'évolution toujours constante du filtre lors de l'adaptation en-ligne, permet une convergence plus rapide et une plus grande flexibilité. Donc, une adaptation en-ligne peut se permettre de changer de canal, contrairement à un apprentissage hors-ligne. Cependant, la charge de calcul et l'utilisation d'une plus grande surface d'intégration au niveau ITGE peuvent désavantager ce type d'apprentissage. À l'opposé, l'utilisation d'une adaptation hors-ligne simplifie l'architecture à implanter. Sa convergence est plus lente si les poids ont mal été choisis par rapport au canal appliqué. Donc, un signal beaucoup moins filtré, avec un BER faible [PRO94], [LEE94], [PRO95].

2.3 Algorithmes adaptatifs d'égalisation de canaux linéaires

En générale, tout types de canaux physiques seront initialement non-linéaire. Cependant, comme expliqué plutôt, il est souvent possible de linéariser le canal et donc de simplifier le nombre nécessaire de retard sur la mesure et la complexité de calculs. En effet, il existe des algorithmes simples qui peuvent être aussi bon qu'un autre filtre, comme le RLS. Le récepteur employé pour la démodulation du signal est souvent caractérisé par deux types. Soit, les receveurs optimums et sous-optimums. Les receveurs optimums sont en principe basés sur l'estimation de probabilité maximum (*maximum likelihood sequence*) pour détecter la séquence d'information échantillonnée à la sortie du démodulateur. Les receveurs sous-optimums emploient généralement un égalisateur linéaire ou un égalisateur avec retour de décision (DFE). Pour ce qui est d'un égalisateur linéaire, il faut déduire que la réponse impulsionnelle ou la réponse en fréquence est initialement connue à la reception. Dans la plupart des cas, l'égalisation adaptative est utilisée. Ce choix est justifié par la fonction de transfert du canal qui est inconnue au départ, et serait plus souvent du type variant. Pour de tel cas, l'égalisation de canaux adaptatifs permet alors d'être ajusté en fonction du changement du canal puisqu'il existe un asservissement entre l'entrée idéale et la sortie du canal. S'il s'averrait que le canal variant serait du type variant dans le temps, l'égalisateur doit changer avec le canal. Et donc, préférablement utiliser un apprentissage en-ligne. Plus précisément, un canal variant dans le temps n'est pas régit par une gaussienne. Il est donc difficile à faire converger sans un apprentissage en-ligne.

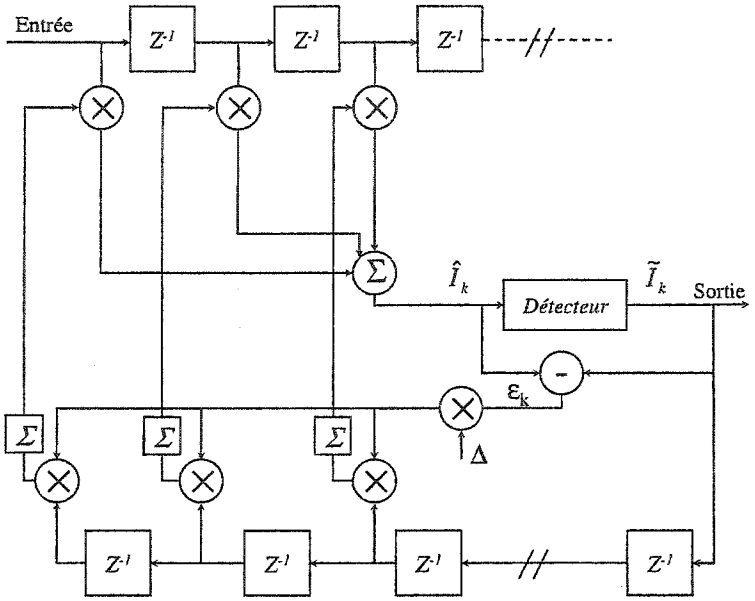


Figure 2.8) Exemple d'une égalisation adaptative *zero-forcing*

La figure 2.8, représente un type d'algorithme classé dans les algorithmes des égalisateurs de canaux adaptatifs linéaires. Cet algorithme utilise le critère de distorsion maximum $\wp(\mathbf{c})$ (*peak distortion criterion*) qui déterminera si une saturation est appliquée lorsque $\wp(\mathbf{c})<1$. Ce qui force la réponse de l'égalisateur $q_n = 0$ pour $1 \le |n| \le K$ et $q_0 = 1$. L'algorithme *zero-forcing* impose un ensemble de moyennes d''erreur $\varepsilon_k = I_k - \hat{I}_k$ et la séquence d'information désirée $\{I_k\}$ pour qu'elle diverge vers zéro en utilisant des paramètres se situant entre $0 \le |n| \le K$. Où Z^{-1} est la valeur échantillonnée discrète alors que Σ représente la sommation des multiplications. Ceci dit, si la réponse du canal est inconnue, l'injection de coefficients utilisés lors de l'entraînement pourrait être utilisé et donc la convergence serait plus rapide [PRO95], [LEE94].

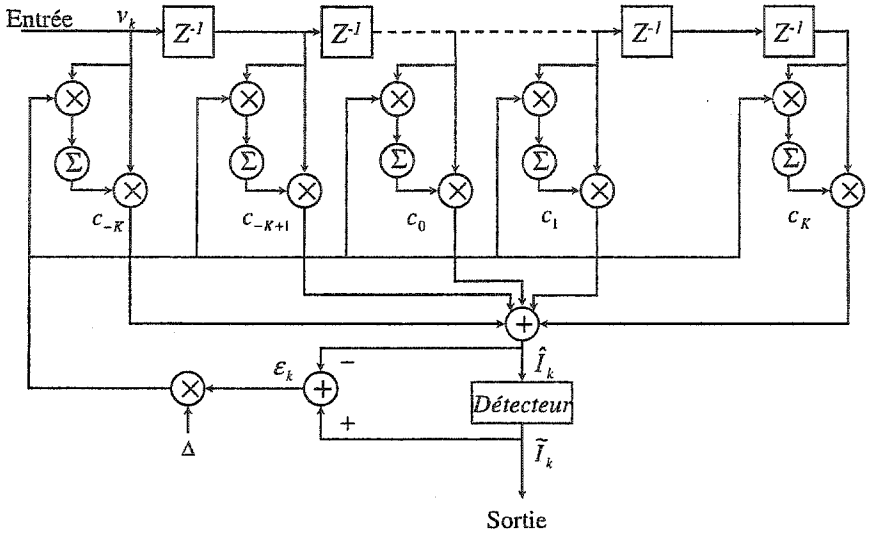


Figure 2.9) Architecture élémentaire de l'algorithme LMS

Un autre filtre connu dans la catégorie égalisateur adaptatif est l'algorithme LMS *(Least Mean Square)* ou la moyenne des moindres carrés de Wiener [HAY96]. Elle peut aussi être appelée algorithme du gradient stochastique. L'équation principale du filtre est

$$\hat{C}_{k+1} = \hat{C}_k + \Delta \varepsilon_k V_k \qquad (2.5)$$

Où C_k représente une série de coefficients de $k^{ième}$ itération, alors que l'erreur du signal à la $k^{ième}$ itération $\varepsilon_k = I_k - \hat{I}_k$. V_k représente le vecteur d'entrée du signal bruité qui crée l'estimation $\hat{I}_k$, et où $V_k = [v_{k+K} \ldots v_k \ldots v_{k-K}]^t$. Ainsi Δ est un nombre positif assez faible pour assurer une certaine convergence lors de l'itération. Puisque l'algorithme est basé sur le MSE (*Minimum square error*) ou l'erreur minimum au carrée, il cherche à converger vers

le minimum MSE ce qui permet de stabiliser la valeur des poids. Pour aider la convergence des poids, le récepteur doit connaître la covariance du signal à être envoyé et le niveau du bruit additif. Cependant, l'algorithme LMS peut quand même être efficace si nous estimons le vecteur du gradient $\hat{G}_k$. Ce qui permettra alors, d'obtenir des poids du coefficient $\hat{C}_{k+1}$ de meilleure qualité. [PRO95], [MIC92] Où :

$$\hat{C}_{k+1} = \hat{C}_k - \Delta\hat{G}_k \qquad (2.6)$$

Alors que l'estimé du gradient ($\hat{G}_k$) est:

$$\hat{G}_k = -\varepsilon_k V_k \qquad (2.7)$$

Ceci nous emmène donc à l'équation principale (2.5).

Après avoir vu les égalisateurs adaptatifs linéaires, nous avons une deuxième catégorie bien connue sous le nom d'égalisateur à retour de décision (*Decision Feedback equaliser*) ou DFE. Ce type d'égalisateur effectue non seulement une boucle de retour (*feedback*), comme les filtres adaptatifs linéaires, mais effectue aussi une boucle d'action (*feedforward*). La figure 2.10 représenté ci-dessous est un excellent exemple d'égalisateur à retour de décision. Comme vu précédemment, les filtres linéaires adaptatifs sont très efficaces pour un canal de type téléphonique, où les ISI ne sont pas trop élevés. Ainsi, si la

sévérité d'un taux d'empiètement est assez élevée entre les symboles d'un canal, elle peut donc causer des erreurs. Car, les ISI ne varient pas seulement au niveau temporel mais aussi au niveau de la caractéristique spectral. Selon les dires de J.G Proakis [PRO94], ce type de problème est souvent rencontré lors de communication radio, où le canal se situe spécialement dans la couche ionosphérique à des fréquences inférieures à 30 MHz. Donc, l'égalisateur à retour de décision est un égalisateur de type non-linéaire qui utilise les décisions passées pour ainsi éliminer les ISI.

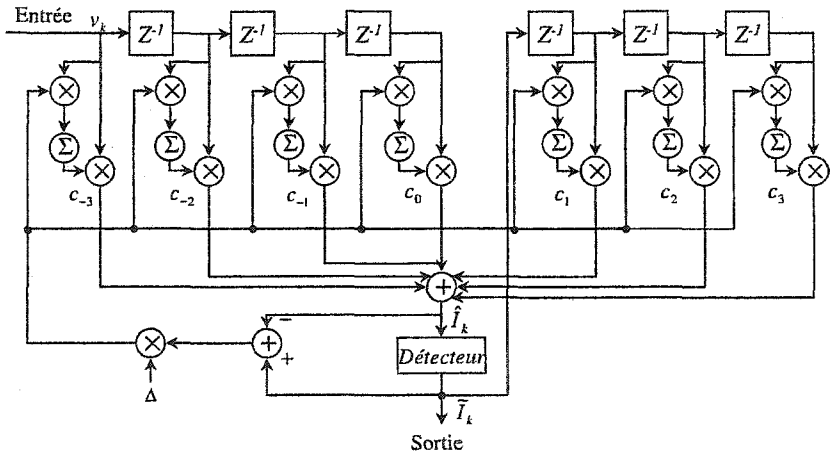


Figure 2.10) Architecture de l'égalisateur à retour de décision (DFE)

L'égalisateur DFE consiste à utiliser deux filtres. Le premier filtre est appelé « filtre à boucle de retour ». Le filtre généralement utilisé est souvent le RIF (*réponse impulsionnelle finie*) avec les poids variables. Par ailleurs, dans la boucle de retour, le RIF est aussi utilisé. Donc, nous avons deux algorithmes très simples à implanter qui emmènent un taux de rejet ISI supérieur. Cependant, il est important de mentionner que le

DFE donne de meilleurs résultats que les filtres adaptatifs au niveau des ISI, mais son taux de rejet au bruit ne surpasse pas les filtres adaptatifs. Pour simplifier la compréhension, l'architecture de la figure 2.10 est conçue avec deux filtres utilisant l'algorithme LMS.

Comme troisième catégorie, nous avons les auto-égalisateurs. Contrairement aux égalisateurs adaptatifs linéaires, les auto-égalisateurs reçoivent la correction d'une chaîne de données préalablement définie pour ajuster les poids des coefficients. Les communications réseaux cellulaires sont des canaux qui changent avec l'endroit de la communication. D'où l'importance d'une grande vitesse de transmission est primordiale. Le besoin de transmettre avant une chaîne d'information est excessivement coûteux. Alors, une égalisation qui aurait initialement appris un type de canal sans devoir redemander une chaîne d'information à chaque changement devient extrêmement pratique. Ce type d'égalisation est du type auto-égalisation. Par conséquent, l'erreur entre l'entrée et la sortie du canal n'est plus nécessaire. La figure 2.11 représente ce type d'égalisation.

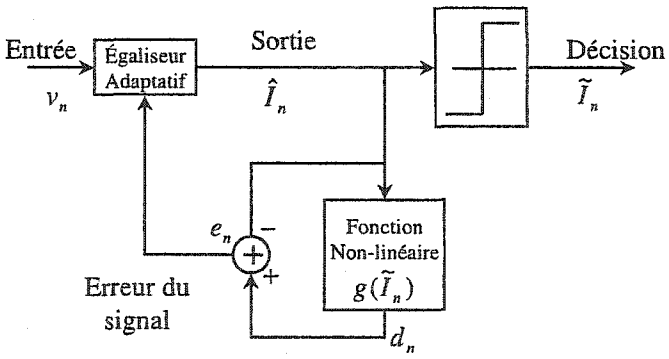


Figure 2.11) Égalisation aveugle (*Blind equalisation*)

Ceci dit, il existe d'autre type d'algorithmes adaptatifs utilisés comme le RLS (*Recursive Least Square*) communément appelé algorithme des moindres carrés récursifs, nous avons aussi le filtre de Kalman. Nous verrons dans les sections subséquentes des exemples d'égalisation de canaux optimum utilisant la logique floue et les réseaux de neurones.

2.4 Algorithmes basés sur la logique floue

L'utilisation de la logique floue comme méthode d'égalisation de canaux est encore tout nouveau. Peu reconnu au début de sa découverte, elle fût grandement appréciée par l'industrie japonaise vers les années 90. Ce boum permis de développer de nouveaux horizons [KOS97]. Des lors, l'arrivée de l'utilisation de la logique floue comme méthode d'égalisation de canaux permis de pousser l'égalisation pour les canaux non-linéaire.

Trois étapes principales régissent le concept de la logique floue:

1. La fuzzification
2. L'optimisation des lois d'inférences
3. La Défuzzification

La description proposée ici fait surtout référence aux articles, [WAN93], [MOU99]. L'égalisation de canaux non-linéaires utilisant la logique floue est représenté par la figure 2.12.

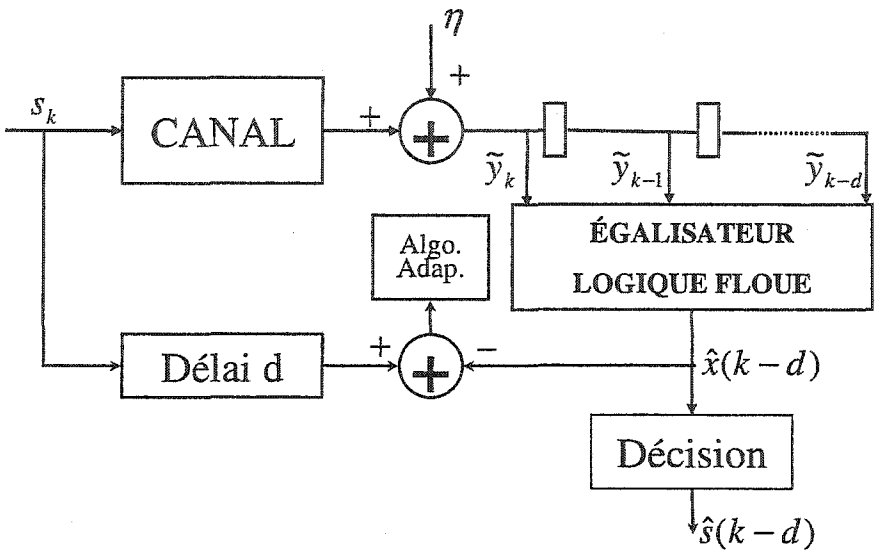


Figure 2.12) Modèle de la logique floue dans un égalisateur de canaux

Pour obtenir un filtre optimum basé sur la logique floue, il est important de bien choisir sa fonction d'appartenance et le nombre m d'appartenances auquelles nous voulons que notre système utilise comme référence. Généralement, la fonction d'appartenance utilisée est de type Gaussienne. Cependant, l'intégration d'une fonction de type Gaussienne n'est pas facilement implantable et lors de la défuzzification des problèmes reliés au calcul du moment d'inertie ou de centre de gravité pourraient avoir lieu. Il est difficile d'approximer la Gaussienne sans une trop grande complexité de calculs. L'utilisation d'une fonction canonique par morceau permet alors de simplifier grandement le nombre nécessaire de calculs, ce qui est directement relié à la surface d'intégration et à la consommation de puissance. La figure 2.13, présente un exemple d'une fonction d'appartenance de type canonique par morceau.

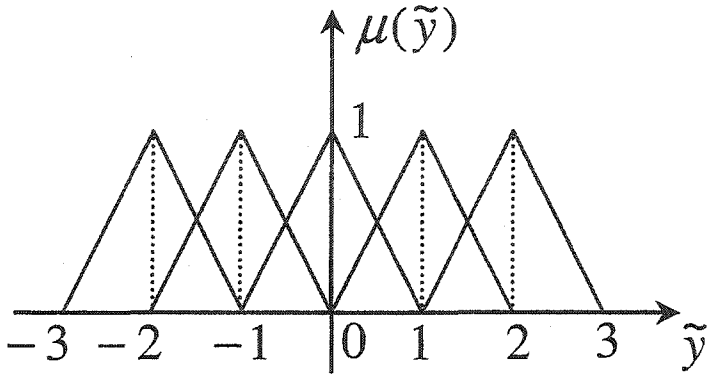


Figure 2.13) Fonction d'appartenance canonique linéaire

Après avoir choisi notre fonction d'appartenance, il faut assigner à l'entrée $\tilde{y}(n)$ la bonne valeur identifiée par sa fonction d'appartenance. La sortie de l'égalisation est représentée par l'équation 2.8, ci-dessous.

$$\hat{x}_k = p'(\tilde{y}) \cdot \Theta \tag{2.8}$$

Où $\hat{x}_k$, représente la sortie de l'égalisateur à logique floue à l'entrée du bloc de décision.

$p'(\tilde{y})$ est le vecteur d'entrée temposé, alors que Θ représente le vecteur des lois d'inférences choisies.

$$p^{l_1,l_2,\ldots,l_n}(\tilde{y}) = \frac{\mu(\tilde{y}_n, l_1) \cdot \mu(\tilde{y}_{n-1}, l_2) \cdot \mu(\tilde{y}_{n-m+1}, l_n)}{c_n(\tilde{y})} \tag{2.9}$$

Alors que, $c_k(\tilde{y})$ est le dénominateur de la somme des possibilités de la fonction

d'inférence. Alors que $\mu(\tilde{y}_{k-i+1})$ est la réponse de la fuzzification.

$$\mu(\tilde{y}_{k-m+1}) = \exp\left[-\frac{1}{2}\left((\tilde{y}_{k=n+1} - \bar{y}_{k-n+1})/\sigma_n{}'\right)^2\right] \tag{2.10}$$

Ceci dit, on utilise l'erreur entre le signal $\hat{x}(n)$ et le signal idéal transmis $s(n)$. Pour ensuite introduire cette erreur à un algorithme adaptatif (RLS, LMS, etc). Ceci permet donc de développer des lois d'interférences qui sont représentées par le vecteur des paramètres Θ. La logique floue permet la transformation du domaine temporel au domaine discret basé sur les lois d'inférences communément appelé la fuzzification. Après avoir filtré, le signal sera re-transformé en utilisant la défuzzification pour revenir au domaine temporel en se servant des lois d'inférences par le bais du moment d'inertie ou le centre de graviter.

Les résultats donnés par cet égalisateur de canaux nécessitent présentement une trop grande charge de calculs. En plus d'effectuer une multitude de multiplication et de divisions, l'égalisateur doit nécessairement utiliser un filtre adaptatif, ce qui alourdie la méthode. D'après une étude effectuée par M. Zakhama [ZAK99], cette technique ne peut pour l'instant égaliser un canal non-linéaire de plus de trois points d'après.

2.5 Algorithmes basés sur les réseaux de neurones

Grandement appréciés les réseaux de neurones ont énormément évolué durant les deux dernières décennies [MEH97]. Extrêmement robustes au bruit, et au non-linéarité, à l'origine, ils sont excellents pour l'égalisation de canaux, spécialement lorsque les canaux sont non-linéaires. Leur simplicité d'implantation leur donne un grand avantage comparativement à la méthode vue précédemment utilisant la logique floue.

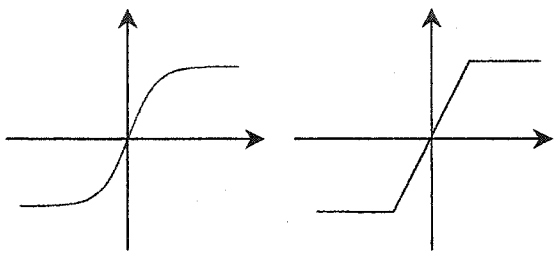


Figure 2.14) Exemple de fonction d'activation

a) Sigmoïde B) Canonique linéaire

Il existe plusieurs types de réseaux de neurones. Cependant, nous allons spécialement introduire deux types de réseaux qui sont à notre avis plus spécifiquement dédiée à l'égalisation de canaux non-linéaire. Ce sont les réseaux à multicouches récursifs et non récursifs [KEC94], [VID99] et [VID99b]. Les réseaux de neurones sont à l'origine non-linéaire. La cause principale vient de la fonction d'activation introduite à tous les niveaux. C'est-à-dire que chaque neurone en principe effectue une saturation. Cette fonction d'activation est généralement de type sigmoïde. Elle permet ainsi de contrôler le débit à

l'intérieur du réseau. La figure 2.14, représente deux types de fonctions d'activation bien connus. L'implantation ITGE numérique de la fonction sigmoïde constitue un problème d'implantation, mais l'utilisation d'une fonction canonique linéaire par morceau pourrait remplacer la sigmoïde [VID99]. Le réseau multicouche récursif ou PL-RNN (*Piecewise linear recursive multilayer neural network*) représenté ci-dessous démontre bien la complexité du réseau [KEC94] puisque chaque donnée est retournée aux neurones précédents. Cet algorithme constitue un excellent égalisateur adaptatif, cependant il nécessite énormément de calculs.

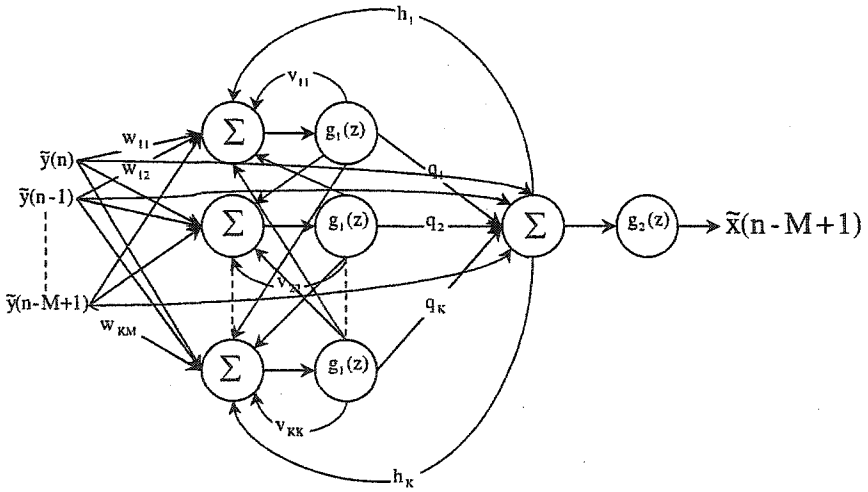


Figure 2.15) Réseau multicouche récursif (PL-RNN)

Par cette figure on peut remarquer qu'il utilise la valeur présente ainsi que les précédentes. Ce nombre dépend de la qualité de recouvrement du signal bruité à recevoir. C'est-à-dire que le nombre M de retard sur le signal reçu soit estimé, ce qui permet d'être représenté par un nombre minimum. Il est primordial cependant que les données reçues

soient sans erreur. En vérifiant son BER (*Bit error rate*), on peut ainsi déterminer le nombre de retard et de couche cachée qui seront nécessaires. Il existe différents autres facteurs qui déterminent la qualité du signal. Nous avons le nombre de neurone nécessaire, le nombre de retard vérifiable, ainsi que la valeur des poids, et bien sûr le type d'architecture. Jusqu'à maintenant, il n'existe pas d'outils permettant de déterminer ces facteurs en évaluant le signal devant être traité. Pour ce qui est de la qualité des signaux égalisés, il ne faut pas oublier qu'il est important lors de l'apprentissage du réseau de fournir des signaux de type qualitatif et quantitatif. Cependant, un nombre trop important de signaux peut emmener un sur-apprentissage du réseau et ainsi détériorer le niveau BER de la sortie. Pour ce qui est de l'apprentissage, généralement on utilise un niveau de bruit supérieur de donnée réelle. Par conséquent, le réseau augmente sa robustesse au bruit blanc. Pour ce qui est du choix de la fonction d'appartenance, M. Vidal [VID99] démontre bien qu'il n'est pas toujours quantifiable d'utiliser une sigmoïde. L'utilisation d'une fonction canonique par morceau s'avère suffisante spécialement lorsque l'implantation ITGE est visée. En plus de démontrer qualitativement les résultats utilisant une fonction d'appartenance simplifiée, M. Vidal [VID99] propose une architecture simple basée sur le réseau récursif multicouche. La topologie de la figure ci-dessous présente une architecture multicouche non récursive, pouvant être utilisée comme égalisateur de canaux non-linéaires (*PL-MNN*).

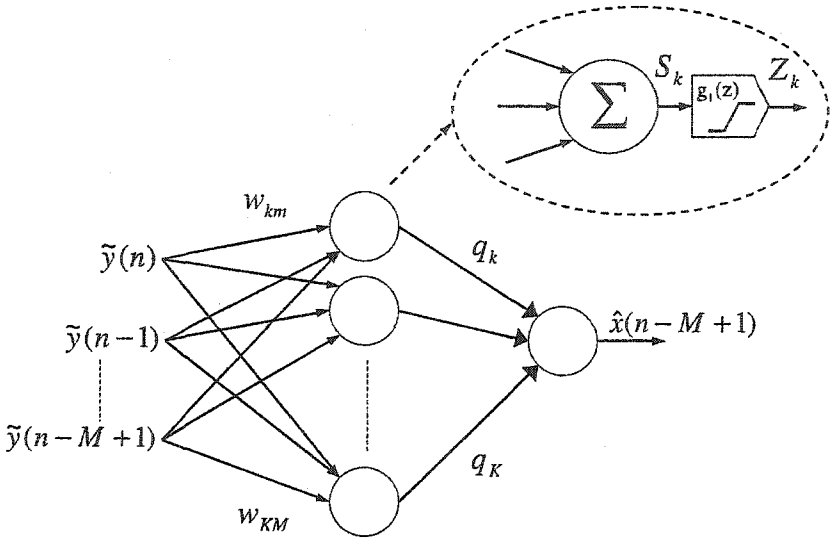


Figure 2.16) Réseau de neurone multicouche non récursif (PL-MNN)

L'algorithme nécessite M retard déterminé par la réponse impulsionnelle du canal et de la quantité de bruit additif (niveau signal/bruit). Chaque retard est multiplié par un vecteur W_{KM} identifié lors de l'apprentissage. Où K est le nombre de neurone dans la couche cachée et M est le nombre de retard. Après avoir accumulé dans chaque neurone la multiplication des signaux, $\tilde{y}(n-M+1)\times W_{KM}$, la fonction d'activation canonique linéaire *g(s)* suit. Ensuite, la sortie des neurones de la couche cachée est multipliée par un second poids (q_K), pour ensuite être additionnée. Pour ensuite passer à travers la fonction d'activation, ce qui permettra d'obtenir un échantillon du signal transmis, $\hat{x}(n-M+1)$. L'équation 2.11 représente le signal estimé du réseau PL-MNN.

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{L} q_k g\left(\sum_{j=1}^{M} w_{km}\tilde{y}(n-m+1)\right)\right) \tag{2.11}$$

$$s_k = \sum_{m=1}^{M} w_{km}\tilde{y}(n-m+1) \qquad \text{pour } k = 1,2,...,K \tag{2.12}$$

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{L} q_k g(s_k)\right) \tag{2.12) + (2.11)= (2.13}$$

Où s_K est la multiplication des signaux bruités et leurs retards par le poids de la première couche de neurone. L'équation de g(s) donne:

$$g(s) = \frac{|s+4| - |s-4|}{8} \tag{2.14}$$

Où la dérivée de g(s) est:

$$g'(s) = \begin{cases} 0.25 & \text{si } -4 < s < 4 \\ 0 & \text{ailleurs} \end{cases} \tag{2.15}$$

Pour ce qui est de la phase d'apprentissage, elle est basée sur la propagation d'une boucle de retour (*backpropagation*). L'erreur *e(n)* entre le signal idéal et l'estimé est ensuite introduite pour le calcul des poids de la couche cachée et de la sortie.

$$e(n) = \mu(x(n) - \hat{x}(n)) \tag{2.16}$$

Des lors, équation 2.17 et 2.18 emmènent le calcul des poids W_{KM} *et* q_K *:*

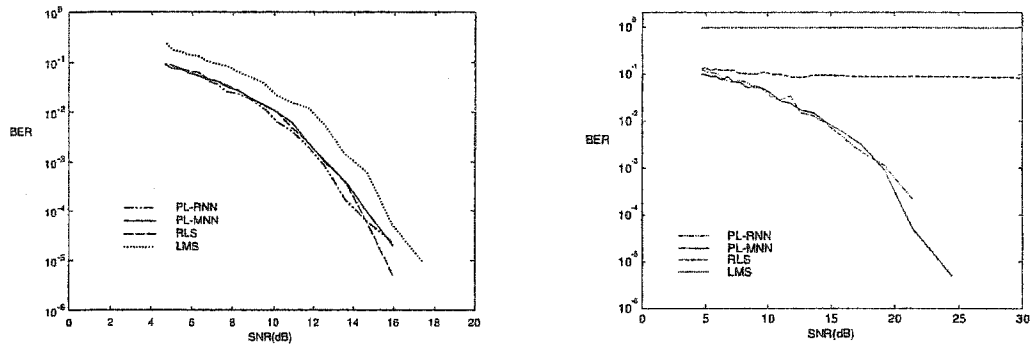
$$w_{km}(n+1) = w_{km}(n) + e(n)q_k(n)g'(s_k(n))y_m(n) \quad \textbf{(2.17)}$$

$$q_k(n+1) = q_k(n) + e(n)z_k(n), \quad q_k(1) = 0 \quad \textbf{(2.18)}$$

Pour le choix de l'implantation ITGE d'une architecture pour l'égalisation de canaux, notre choix s'arrête sur l'algorithme PL-MNN. Simple, régulier, et nécessitant beaucoup moins d'opérations comparativement aux deux algorithmes précédents. Le réseau de neurone multicouche PL-MNN obtient d'excellents résultats. Puisque l'architecture utilisera comme augmentation de débits la technique du pipeline par vagues, l'utilisation de l'algorithme PL-MNN devra utiliser un apprentissage hors-ligne. Ce qui permettra de stabiliser l'architecture. Comme nous verrons plus loin, il est primordial qu'une architecture utilisant le pipeline par vagues ait le moins possible de variations de débit. En plus, il devient extrêmement difficile d'équilibrer les chemins de l'architecture lorsque l'architecture est récursive. L'ajout d'une phase d'apprentissage crée cette récursivité dans l'architecture. Par conséquent, la fréquence de traitement de l'égalisateur de canaux devra être plus faible qu'une architecture continue. Ainsi en éliminant la phase d'apprentissage nous obtenons une architecture extrêmement linéaire, ce qui contribue à l'augmentation du débit et par le fait même celle de la latence.

2.6 Choix de l'Égaliseur

Le choix de l'égaliseur s'arrête sur l'utilisation d'un réseau de neurone, plus particulièrement le PL-MNN (*Piecewise Linear Multilayer Neural Network)*, présenté dans [VID99], [VID99b] et [VID99c]. L'avantage d'utiliser un réseau de neurone lors de communication PAM (*Pulse Amplitude Modulation)* est la facilité d'intégration comparativement à d'autres types de réseaux de neurone comme le PL-RNN. La récurrence utilisée par le PL-RNN emmène une plus grande difficulté d'intégration puisque chaque neurone est entièrement connecté avec les autres. Si nous comparons aux algorithmes plus populaires comme le LMS et le RLS décrit par S. Haykin [HAY93], le PL-MNN obtient de bien meilleur résultat. Notre principal objectif lorsque vient le choix de notre algorithme est sa robustesse face aux bruits. Alors, nous utiliserons le BER en fonction SNR (*Signal-Noise Ratio*) comme méthode de sélection de l'algorithme choisi.



a) b)

Figure 2.17) Robustesse au bruit

a) Canal linéaire, b) Canal non-linéaire

Lorsque nous comparons le BER en fonction du SNR pour un canal linéaire, nous obtenons sensiblement de bon résultat si nous considérons la simplicité de l'algorithme comparativement aux autres. Mais, pour un canal non-linéaire les performances de l'algorithme PL-MNN sont encore plus impressionnantes, Elles démontrent une supériorité face au RLS et LMS. Mais, ce qui est remarquable c'est que les résultats de la figure 2,17.b présente le réseau PL-RNN comme étant moins robuste au bruit pour un canal non-linéaire qu'un réseau PL-MNN. Ce qui est excellent étant donné que le canal aura toujours des caractéristiques non-linéaires.

Un autre facteur élémentaire au choix de l'algorithme est le débit pouvant être atteint par l'architecture choisie lors de son implantation ITGE. Ainsi, l'architecture proposée est du type systolique. Elle est donc constituée de plusieurs processeurs élémentaires de même configurations reliées localement entre eux. Ceci permet une communication locale et

régulière entre chacun des bloques élémentaires. Ainsi, les débits pouvant être atteint sont nécessairement supérieurs à une architecture non régulière (non-systolique). Autre facteur déterminant à notre choix présenté au chapitre 2.5, plus spécifiquement la figure 2.16, est la fonction sigmoïde approximé par sa linéarisation, communément appelé, fonction canonique par morceau. Celle-ci sera d'une valeur de 0.25, ce qui représente simplement de décaler la valeur vers la droite pour les valeurs faisant parti des frontières déterminées, lors de son implantation numérique.

2.7 Conclusion

Une communication à travers un milieu est d'une importance capitale dans notre société du 21^{ième} siècle. Que ce soit pour une communication par téléphone cellulaire ou par internet, de nouvelles techniques doivent sans cesse naître. Ainsi, l'idée derrière l'égalisation de canaux est d'améliorer le débit et la qualité du canal auquel un échange de données sera communiqué. Ce chapitre permet une vue d'ensemble sur les différents choix qui nous sont offerts avant de déterminer l'algorithme idéal qui sera par la suite intégré en ITGE.

Le chapitre est constitué de sept parties. D'abord, nous introduisons sur le principe de l'égalisation de canaux (section 2.1), ce qui permettra un apprentissage entre les avantages et inconvénients. Par la suite, la section 2.2 détaille les deux grandes branches qui divisent l'égalisation de canaux, elles sont l'apprentissage en-ligne et hors-ligne. Ce qui amène à

une brève description des choix qui nous sont offers. Pour ainsi en venir à la description plus détaillée de l'architecture présentement en recherche dans nos laboratoires universitaire (LSSI). Ces algorithmes sont d'une part le développement d'une architecture utilisant la logique floue [ZAK99] et d'une autre utilisant les réseaux de neurones [VID99] (section 2.4 et 2.5.). En terminant, la section 2.6 présente l'algorithme qui a été choisi pour ce projet.

Finalement, ce chapitre constitue en soi une introduction au chapitre 6. Il permet de comprendre l'idée derrière l'architecture présentée au chapitre futur, et ainsi de comprendre le fonctionnement des équations qui y sont rattachées.

3 Unités Arithmétiques entières

La fonction d'une unité arithmétique est de permettre l'opération logique de fonctions mathématiques nécessaire à l'opérateur. Ces opérations mathématiques sont généralement constituées de fonctions élémentaires mathématiques comme: l'addition, la soustraction, la multiplication et la division. Ceci dit, les opérations mathématiques plus spécifiques sont une constitution d'opération élémentaire (+/-, ×, ÷) connue. Ce chapitre servira principalement à l'introduction des différents types de structures arithmétiques élémentaires pouvant être développées. Ainsi, nous pourrons nous arrêter sur un choix idéal, pour le développement d'unités à haut débit.

Il existe principalement deux types d'opérations arithmétiques binaires. Ils sont de type série ou parallèle. Leurs différences reste principalement au niveau de leurs rapidités de traitement et de surface d'intégration utilisée. Nous verrons plus loin un exemple concret. Avant de faire un choix sur la structure qui nous sera nécessaire, on se doit de savoir quelles sont les représentations binaires qui nous sont offert avant de prendre une décision. Ainsi, la section suivante sera destinée aux différents types de représentations binaires fréquemment utilisées.

3.1 Représentation des nombres binaires

Lors de la conception d'architecture ITGE, il est important de concevoir en fonction de la représentation que nous voulons utiliser, de la surface qu'elle devra utiliser, de la vitesse de calcul ou de la précision qui devra être nécessaire. Ainsi, nous pourrons arrêter notre choix parmi les différentes représentations qui nous sont offertes. Un nombre de type signé sera aussi une considération lors de la conception.

Pour cette recherche, les nombres signés sont un atout. Ils sont considérés tout au long de ce chapitre. Ainsi, puisque la communication sera du type PAM, elle est constituée de -1 et 1, elle aura une partie positive et une partie négative. La représentation d'un nombre signé représente la partie négative d'un nombre de m bits en ajoutant après le bit le plus significatif, un bit de signe. Ce qui résulte en une longueur de m+1, le nombre qui sera nécessaire au calcul.

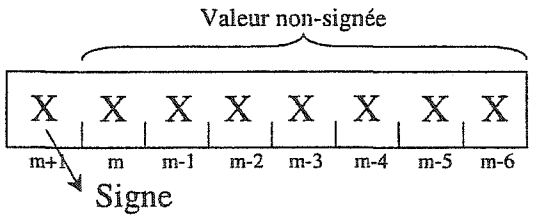


Figure 3.1) Représentation signée

Il existe quatre types de représentations binaires signées, la représentation:

- Signe-Module (virgule fixe)
- Complément à un (virgule fixe)
- Complément à deux (virgule fixe)
- Virgule flottante (IEEE)

Dans toutes les représentations, le nombre binaire positif sera égal à:

$$X = 0 * 2^n + \sum_{i=0}^{n-1} x_i * 2^i \qquad (3.1)$$

Pour ce qui est des valeurs signées à virgule fixe, chacune à une méthode différente de représentation. Pour ce qui est du signe-module, la représentation reste semblable à la représentation positive. Le seul changement est l'ajout du bit de signe auquel nous identifierons la valeur binaire '1' pour indiquer que l'information sera négative, l'opposé représente une valeur positive située plus précisément à gauche de la valeur binaire. Donc, la valeur la plus significative à l'extrême gauche sera la valeur signe sous forme négative ou positive. Où la représentation de complément à un est :

$$X = 1 * 2^n + \sum_{i=0}^{n-1} x_i * 2^i \qquad (3.2)$$

Pour ce qui est de la représentation complément à un, lorsque la valeur doit être

négative on inverse tous les bits à l'exception du bit de signe (celui-ci sera imposé à '1'). L'équation 3.3 est la valeur finale de cette conversion.

$$X = 1 * 2^n + \sum_{i=0}^{n-1} \overline{x}_i * 2^i \qquad (3.3)$$

Pour ce qui est de la représentation complément à deux, il s'agit simplement du même traitement que le complément à un pour ensuite additionner au bit le moins significatif (la valeur à l'extrême droite) la valeur '1' lorsque la valeur doit être négative. Ce concept est représenté par l'équation 3.4, où le nombre zéro a une seule expression contrairement aux deux autres, ce qui facilite la tâche. En plus, les frontières du complément à deux lors d'une représentation sur 5 bits, sont de [-16⇔15], ce qui lui apporte un avantage certain.

$$X = (1 * 2^n + \sum_{i=0}^{n-1} \overline{x}_i * 2^i) + 1 * 2^0 \qquad (3.4)$$

Le décalage des expressions peut être nécessaire, que se soit pour un principe de normalisation pour corriger la quantification ou pour simplement multiplier par deux lors d'un décalage d'une position à gauche ou d'une division par deux lors d'un décalage à droite. Les représentations à virgule fixe négative agissent différemment. Cependant, pour des nombres positifs n'importe quelle des trois formes de représentation agit de la même méthode. Que le décalage soit de la gauche ou de la droite, ils doivent insérer aux positions libres des '0' par la suite. Bien sûr, ce décalage n'agit pas sur le bit de signe, mais

uniquement sur la valeur. Ainsi, le bit reste fixe peut importe que le nombre soit négatif ou positif.

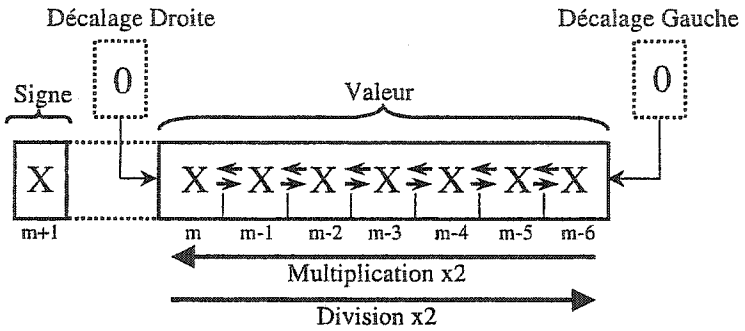


Figure 3.2) Décalage d'une valeur positive et signe-module

Pour ce qui est des nombres négatifs, l'unique changement se fait sur l'introduction de bit '1' ou '0' aux positions libres. Ainsi, pour la représentation signe-module, le décalage à gauche ou droite permet l'introduction de bit '0' aux positions libres. À l'opposé, la représentation complément à un, force le remplacement de bit de valeur '1' pour toute direction de décalage.

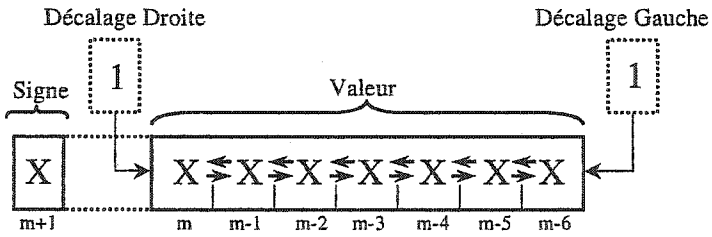


Figure 3.3) Décalage de valeur négative en complément à un

Pour ce qui est du nombre négatif en complément à deux, la direction du décalage

détermine la valeur introduite aux positions manquantes. Ainsi, un déplacement vers la droite permet le remplacement de bit par des bits de valeur '1', alors que son complément remplace les positions libres lors du déplacement de la gauche. Comme indiqué à la figure 3.4.

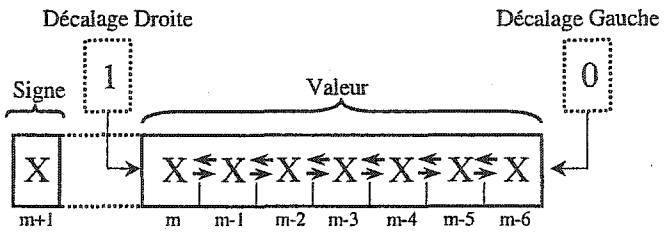


Figure 3.4) Décalage de valeur négative en complément à deux

Ainsi, il est facilement observable que la représentation en complément à deux est la plus efficace et la plus utilisée à l'intérieur des structures. Puisque la soustraction, la multiplication et la division sont en générale effectuée par l'entremise d'addition partielle, l'utilisation de structure à valeur positive peut être ainsi utilisée. Voilà pourquoi la méthode complément sera choisie. En comparant les trois représentations [DAN92] conclus que les nombres utilisant l'expression du complément à deux permettent:

- Le nombre zéro a une seule expression;
- L'opération d'addition s'effectue plus rapidement, car il n'existe pas de propagation cyclique de la retenue;

Ceci dit, il peut arriver quelque fois, lors d'opérations arithmétiques quelconque, qu'il y est un dépassement de la réponse supérieur au nombre total de bits pouvant représenter ce nombre. Ce débordement peut être observé lors d'utilisation d'une expression en complément à deux. Il existe alors deux façons de reconnaître ce dépassement. D'abord comme première solution nous avons l'équation 3.5, l'utilisation du signe et du bit le plus significatif disponible en sortie est utilisée, le symbole $\oplus$ représente la fonction « ou-exclusif » et $\otimes$ est son complément:

$$E = T_{m+1} \oplus T_m \tag{3.5}$$

Où T_{m+1} et T_m sont le signe et le bit le plus significatif obtenue en sortie.

$$E = (X^*_{m+1} \oplus Y_{m+1}) \bullet (X_{m+1} \otimes Y_{m+1}) \tag{3.6}$$

La seconde équation est utilisée lorsqu'on utilise un accumulateur, alors que X_{m+1} est la valeur du signe avant l'opération et X^*_{m+1} est après l'opération. Y_{m+1} est la valeur auquelle on additionne X_{m+1} initialement. La figure ci-dessous représente l'équation 3.6, où on utilise un additionneur série pour simplifier le dessin. On peut observer que les données sont fournies une à la suite de l'autre pour finir par le signe. Celui-ci sera vérifié avant et après en utilisant l'équation 3.6. Lorsque le niveau de sortie est de valeur '1', alors ceci indiquera un dépassement. La vérification se fait en comparant les signes avant l'opération, s'ils sont semblables et par la suite de valeur différente après l'opération. Alors, un débordement a été identifié. Le résultat est donc erroné lors de l'opération

utilisant l'accumulation.

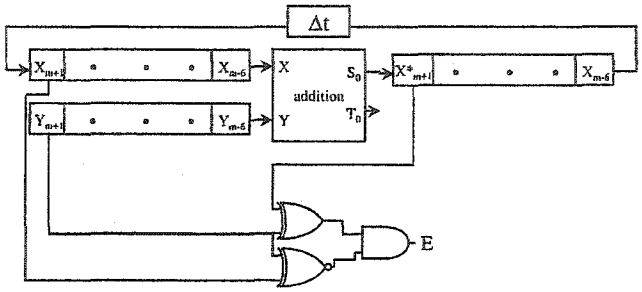


Figure 3.5) Vérification du dépassement lors d'une accumulation en série

3.2 Additionneurs binaires

Les additionneurs binaires rapides sont la clef des calculs arithmétiques (ALU). Ils servent comme structure générale pour la plupart des opérations, comme la soustraction, la multiplication et la division. Bien sûr, quelques-uns sont plus faciles à générer que d'autres. Mais, dans l'ensemble l'additionneur reste la base de l'arithmétique binaire. En effet, plus l'additionneur est rapide plus l'unité arithmétique et logique (ALU) sera rapide ainsi les opérations et le traitement sont inévitablement plus rapide. Cependant, il existe plusieurs types d'additionneurs du plus simple au plus complexe, et plus la vitesse est importante plus il sera dispendieux en surface et en complexité. Cette section sert d'introduction aux différentes additions principales utilisées lors d'une conception architecturale. Les additionneurs qui seront vus sont : entier, à propagation de la retenue, à conservation de la retenue, à retenue anticipée, à sélection de la retenue, et en série.

3.2.1 Additionneur partiel (HA)

L'additionneur partiel (HA) exécute l'addition de deux bits, et retourne le somme et la retenue. Représenté ci-dessous, elle est très peu utilisée. Elle sert d'intermédiaire à l'utilisation d'additionneur entier (section 3.2.2.). Étant donné quelle est constituée nécessairement de nombre inférieur de portes logiques, et servira principalement à réduire la quantité de surface utilisée lors d'une conception de multiplieurs.

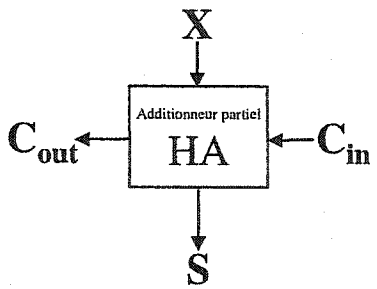


Figure 3.6) Principe de l'additionneur partiel

3.2.2 Additionneur entier (FA)

L'additionneur entier (FA) exécute l'addition d'un bit. Il possède trois entrées et deux sorties. L'addition se fait sur les bits X, Y et considère la retenue précédente C_{en}. En sortie, nous avons S qui est la somme et C_{so} pour la retenue présente. La figure 3.7, représente l'entité de la cellule. La plupart des arithmétiques complexes et plus élaborées utilisent l'addition entière comme principale composante.

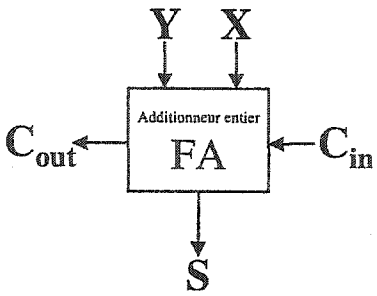


Figure 3.7) Principe de l'additionneur entier

La cellule FA est régie par deux simples équations qui sont:

$$S = X \oplus Y \oplus C_{en} \tag{3.7}$$

$$C_{so} = XY + XC_{en} + YC_{en} \tag{3.8}$$

La figure 3.8, représente en porte logique une façon où la cellule FA peut être conçu tout en équilibrant les délais de propagation de la retenue avec la somme. Cette façon d'équilibrer les délais les plus courts avec les plus longs sera approfondie au prochain chapitre.

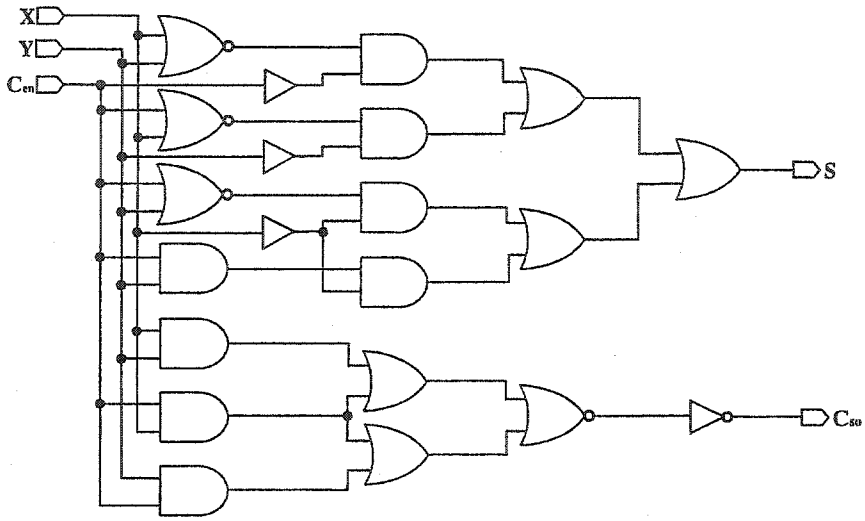


Figure 3.8) Schématique d'une cellule FA avec délais équilibrés.

3.2.3 Additionneur à retenue propagée (RCA)

L'additionneur à retenue propagée représenté à la figure 3.9 est conçu d'un nombre FA égale au nombre de bit voulant être additionné. Le RCA est dans la catégorie d'additionneur parallèle, car les nombres additionnés ou soustraits sont introduits dans un même temps. Cependant, comme son nom l'indique, la retenue précédente est propagée au bit suivant. Ainsi, la somme sera complète seulement à l'arrivée du dernier complément qui sera considéré comme bit le plus significatif (*MSB*). Ainsi, en plus de considérer un délai minimum pour chaque porte logique à l'intérieur de chaque cellule FA, nous devons considérer le délai de la transmission d'une retenue par le nombre de bit total à l'addition. Ainsi, l'addition peut réellement être considérée sérielle. Ce type d'arithmétique sérielle peut d'une part être résolue en augmentant la période de l'horloge qui contrôle les registres

d'entrées et sorties, et d'autre part, introduire des temporisateurs sous forme triangulaire pour dénormaliser les bits avant le RCA pour finalement normaliser la somme en introduisant des temporisateurs triangulaires inverses, comme proposé dans [JOU97].

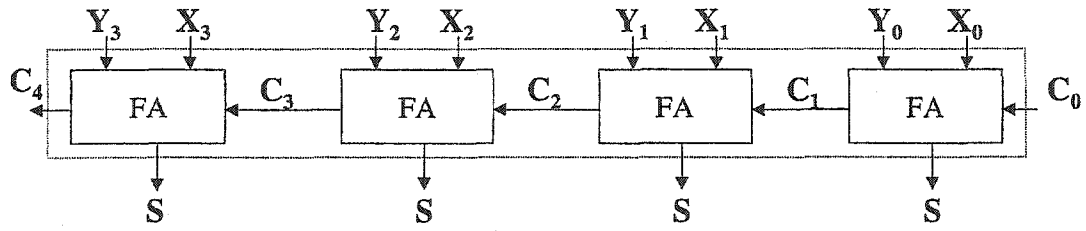


Figure 3.9) L'additionneur à retenue propagée (RCA)

La figure 3.10 représente la dénormalisation des bits avant d'être introduite dans le RCA. La somme est normalisée en inversant le model.

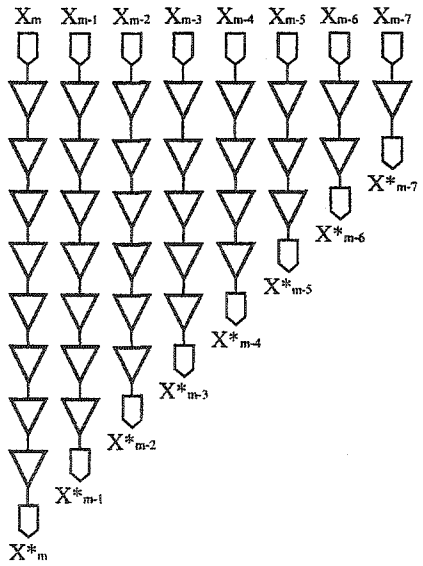


Figure 3.10) Délais en forme triangulaire (dénormalisation)

3.2.4 Additionneur à retenue anticipée (CLA)

L'additionneur à retenue anticipée (figure 3.11) utilise deux blocs principaux. L'un pour la sommation des bits, l'autre pour le calcul de la retenue par anticipation. Ainsi, la retenue est calculée en utilisant uniquement comme dépendance les bits d'entrées. Ceci permet, comparativement au RCA (vu à la section 3.2.3), une accélération substantiellement plus élevée. Dès lors, il n'est plus nécessaire d'attendre le trajet complet de la retenue à l'intérieur de chaque FA pour obtenir la somme résultant de l'addition. L'équation booléenne suivante représente la fonction principale qu'aura le bloc de retenu. [ZAR95]

$$C_{i+1} = x_i y_i + C_i(x_i + y_i) \qquad (3.9)$$

Où C_{i+1} est la fonction utilisée par l'additionneur complet pour un bit. Il suffit ensuite d'introduire la retenue redondante passée pour permettre le calcul de la retenue présente.

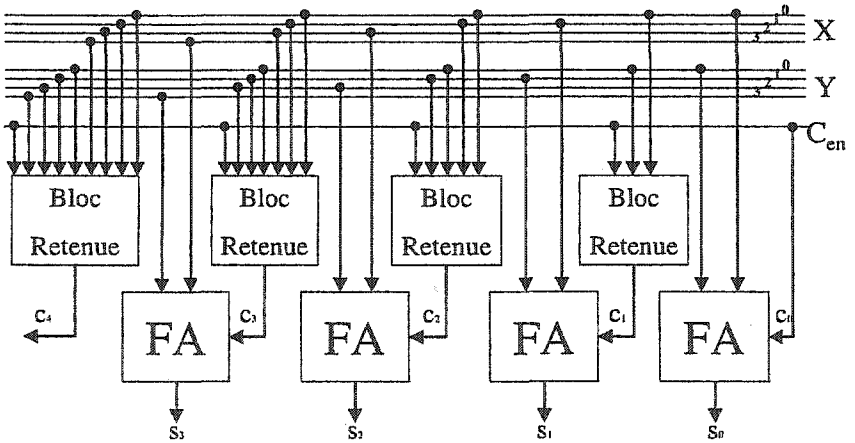


Figure 3.11) Structure du CLA

Ainsi, la figure ci-dessous représente le diagramme en porte logique du calcul de la retenue du $4^{ième}$ bit.

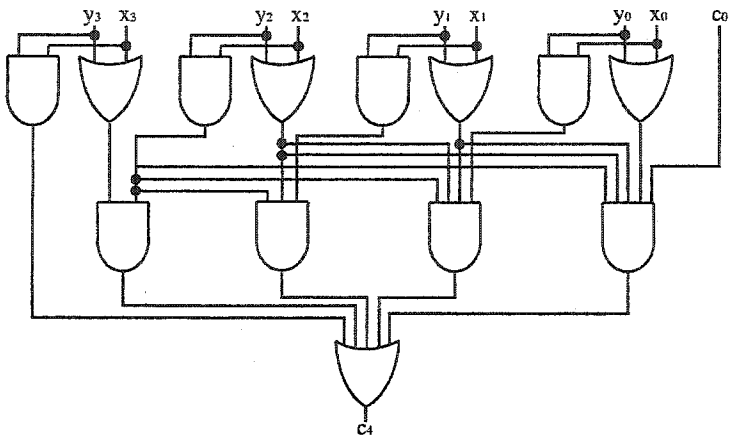


Figure 3.12) Structure du bloc de retenue C_4

3.2.5 Additionneur à conservation de la retenue (CSA)

Le CSA a plusieurs avantages. Il est surtout remarquable pour l'addition de plus de deux nombres. L'addition ce fait simultanément, et la somme peut par la suite être additionnée à un quatrième, à un cinquième, etc. Généralement utilisée comme structure principale d'un multiplieur, elle permet d'être combinée avec un second CSA par la suite et ainsi instaurer la fonction principale d'un multiplieur-accumulateur ou MAC.

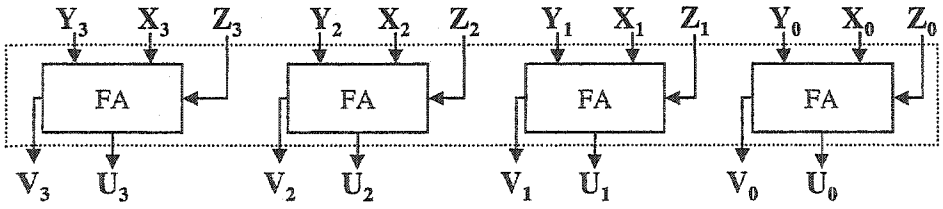


Figure 3.13) Additionneur à conservation de la retenue (CSA)

Dans les articles [GHO94] et [GHO95], on présente un nouveau type de cellule CSA. Celui-ci sert à combiner l'avantage d'une addition cellulaire comme le RCA avec la rapidité d'addition partielle du CSA. Ceci est principalement instauré à l'intérieur d'un multiplieur. Nous verrons plus loin quel type de multiplieur pourra utiliser ce type de cellule pour l'addition partielle, puisqu'une multiplication est généralement constituée d'addition. La figure 3.14, représente la structure en porte logique, où *u* et *v* sont les sommes partielles.

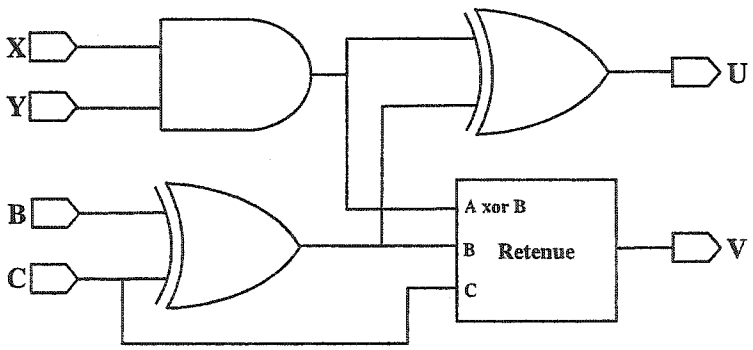


Figure 3.14) Schématique d'une cellule CSA modifiée (CSA 1-bit)

3.2.6 Additionneur à saut de la retenue

D'après [HEN92], l'additionneur à saut de la retenue se situe à mi-chemin entre un additionneur à retenue propagée (section 3.2.3) et l'additionneur à retenue anticipée (section 3.2.4), à la fois en terme de vitesse et de coût. Comme démontré à la figure 3.15, l'additionneur utilise ainsi plusieurs FA et sélectionne entre la retenue présente, et la retenue précédente. La retenue initiale se propage donc de sous-unité à sous-unité. Cet exemple représente un saut de la retenue par 4-bits additionnés. L'addition ce fait avec une latence de plus en plus faible avenant l'utilisation d'une sous-unité FA de plus grande capacité. Il faut cependant mentionner que la vitesse de calculs peut être dépendante au temps d'arrivée. Puisqu'il y a une grande dépendance entre la retenue et la somme totale, la retenue arrivera nécessairement avant la somme finale (*Hold time*). La retenue devra nécessairement être temporisée ou cette limite affectera par lien direct la vitesse de l'horloge utilisée.

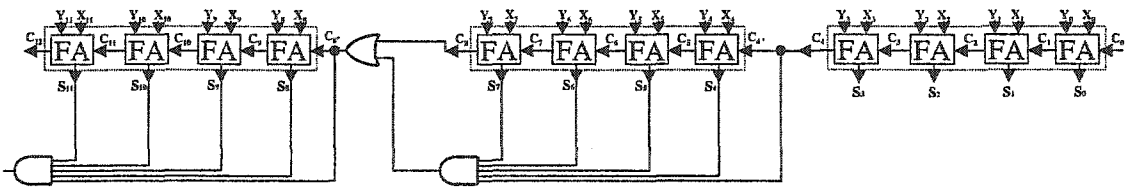


Figure 3.15) Structure de l'additionneur à saut de la retenue (4 bit blocs)

3.2.7 Additionneur sériel

Comme son nom l'indique, l'additionneur sériel exécute des additions bit par bit en utilisant qu'un seul additionneur entier. Idéal lorsque la surface requise est plus importante que la vitesse de calculs. Puisque sériel, chaque entrée dois être nécessairement décalée pour permettre l'addition du moins significatif (LSB) au plus significatif (MSB). Sa structure (figure 3.16) est constituée principalement de deux portes logiques ET, d'une bascule D et comme mentionné plus tôt, d'un additionneur entier (FA). La bascule sert à transmettre la retenue au temps *t-1* pour le calcul de la donnée future au temps *t*.

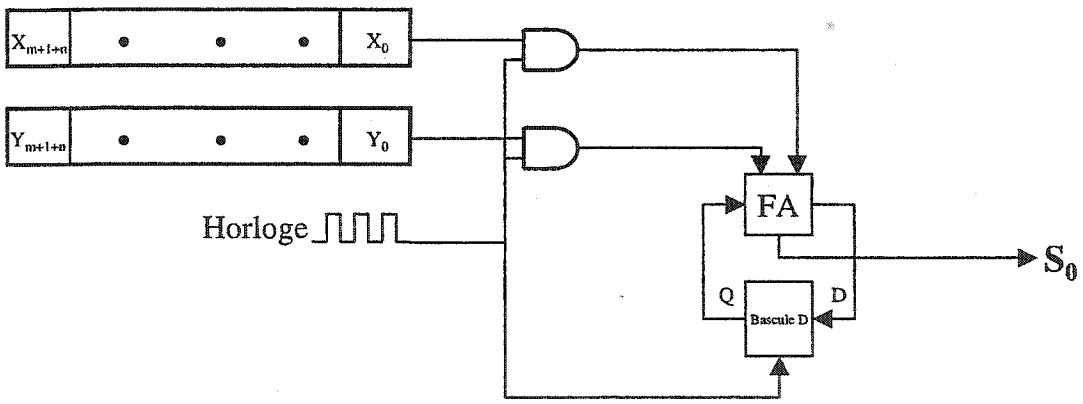


Figure 3.16) Structure de l'additionneur sériel

3.2.8 Autres additionneurs

D'autres additionneurs existent, tel que l'additionneur à sélection de la retenue, et l'additionneur BCD (*Binary-Coded decimal*). Moins utilisés, ils permettent une vue

globale, et peuvent être utilisés pour une conception spécifique. L'additionneur à sélection de la retenue peut être pratique uniquement si la surface utilisée n'a pas d'importance. Il utilise deux additionneurs RCA pour concevoir la réponse d'une addition en introduisant à chacun la valeur que pourrait avoir la retenue précédente. Ainsi, en utilisant doublement de surface de silicium, le calcul des valeurs les plus significatives est effectué en même temps que les valeurs moins significatives. Ceci permet finalement de choisir la somme MSB qui sera transigée par la retenue précédente réelle de la partie la moins significative (LSB).

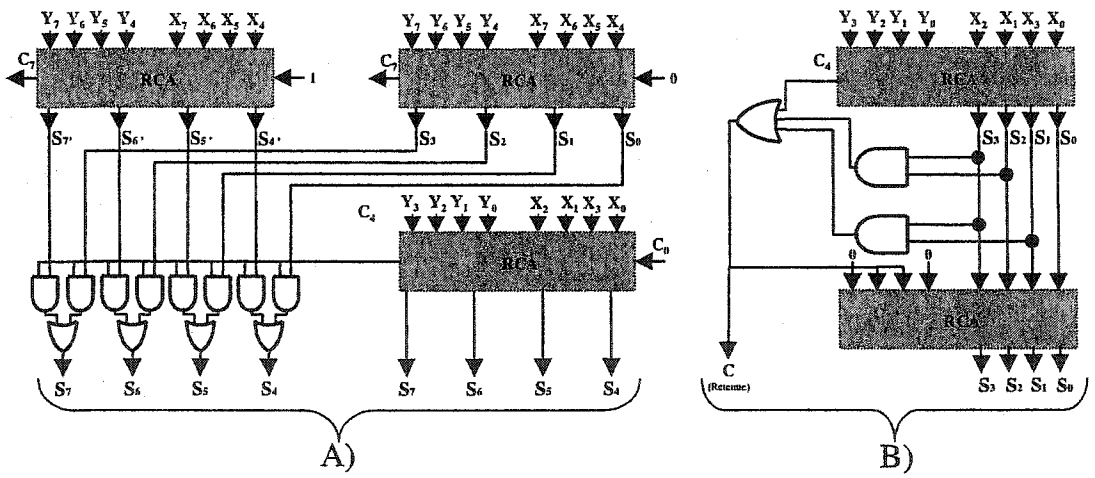


Figure 3.17) Structure de L'additionneur : a) à saut de retenue, b) BCD

Finalement, nous ne pouvons terminer cette section sans présenter l'additionneur BCD. Il est très pratique lorsque deux nombres devant être additionnés sont de format BCD. Ainsi, il n'est plus nécessaire de faire une conversion en nombre binaire pour ensuite additionner, et enfin retourner en format BCD. Puisque le format BCD peut comporter un nombre entre 0 et 9. Les nombres supérieurs à 9 seront additionnés au nombre 6. La figure ci-dessus présente les structures de nos deux additionneurs.

3.3 Multiplicateurs

La multiplication est généralement conçue d'une succession élémentaire d'addition, qu'on utilise un additionneur qui exécute une série d'accumulations ou plusieurs additions partielles branchées en chaînes, le choix de l'architecture implantée tiens sur le type d'utilisation. Après l'addition et la soustraction, la multiplication est considérée la troisième en importance lors d'implantation ITGE et DSP. La multiplication implique plusieurs types de structures. Cette section servira d'introduction aux multiples architectures proposées dans plusieurs références. Dans l'ensemble, il existe deux grandes branches auxquelles les multiplieurs font parti. Nous verrons ainsi, les multiplieurs à accumulation partielle (Signe-module, Booth, etc..) et les multiplieurs parallèles (HA/FA, CSA, RCA, Baugh-Wooley, Booth, etc...).

3.3.1 Multiplieur à accumulation partielle

Très peu utilisé de nos jours, la multiplication utilisant un additionneur est l'une des façons la plus simple de multiplier des nombres binaires. Contrairement à l'addition et à la soustraction d'un nombre de format binaire complément à deux, où il suffisait d'additionner les deux nombres pour obtenir le bon résultat toujours de format complément à deux. Ceci est maintenant complètement faux pour la multiplication, et doit être considéré lors d'une implantation. Ce qui ramène à la même conclusion mentionnée

précédemment au début de ce chapitre, sur le choix du format binaire devant préférablement être choisi. Ceci dit, l'architecture suivante sera simplement de format signe-module. Cependant, nous verrons plus loin le même type d'architecture légèrement modifiée utilisant le fameux algorithme Booth, conçu pour l'utilisation de nombres binaires de format complément à deux. Pour l'instant restons avec le format signe-module, ce qui permettra de comprendre la logique derrière la multiplication par accumulation partielle. [DAN92], [HEN92]

L'idée derrière cette multiplication est d'accélérer notre multiplication lorsque le bit le moins significatif du multiplicateur est 0. Alors, l'addition n'est pas nécessaire puisqu'elle sera de valeur nul. Pour sauver du temps de calculs précieux, nous n'additionnerons pas la multiplicande à l'accumulateur, nous effectuerons seulement un décalage de la droite des bits des registres de l'accumulation et du multiplicateur. La structure est donc constituée d'un registre pour la multiplicande d'une largeur de M. D'un registre pour le multiplicateur N et d'un accumulateur de M largeur. Le résultat sera l'union entre le registre de l'accumulateur et le registre du multiplicateur. Le résultat final sera défini sur un nombre de bit égal à (M+N). Ainsi, le résultat de deux nombres de 4-bits sera de 8-bits de largeur. Par la suite, si nécessaire, le résultat peut être tronqué en gardant intacte les bits les plus significatifs.

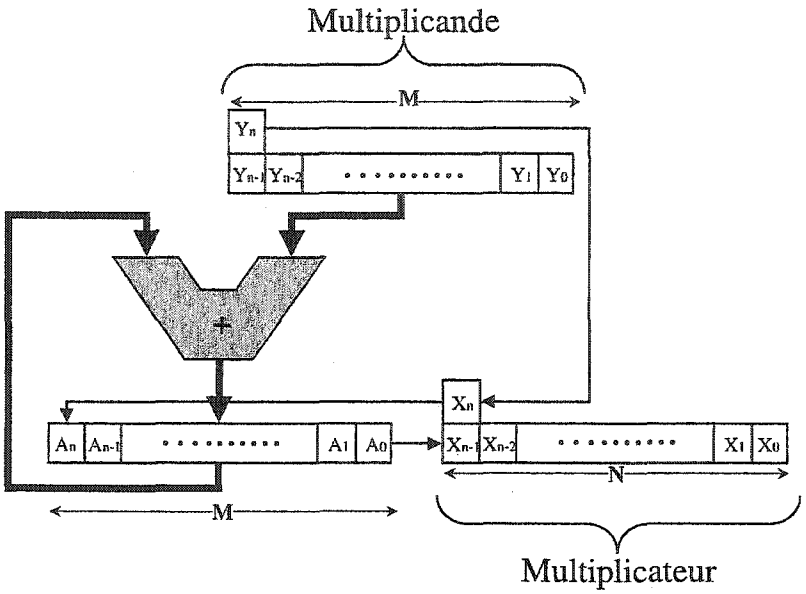


Figure 3.18) Multiplication par accumulation partielle (Signe-module).

Cette multiplication se fait en additionnant le multiplicande d'un nombre x de fois, équivalent à la valeur du multiplicateur. Alors, après chaque accumulation nous décalons les registres de l'accumulateur et du multiplicateur d'un bit par la droite. Ce type de multiplicateur peut très bien être modifié pour exécuter la division, simplement en exécutant un décalage vers la gauche au lieu de la droite en principe. Mais, puisque la division ne fait pas parti de notre recherche, nous n'irons pas plus loin dans ce qui constitue la division. Autres étapes à exécuter constitue le chiffre signe du résultat, comme présenté sur la prochaine figure. Le signe du résultat est constitué du signe du multiplicande et du multiplicateur. Où $\oplus$ représente la fonction logique ou-exclusif de base. [DAN92], [ZAR95]

$$x_n = y_n \oplus x_n \tag{3.1}$$

On peut observer la relation entre les étapes qui amènent à la multiplication par accumulation partielle signe-module en regardant la figure 3.19. Tirée de [DAN92], elle constitue une excellente façon de comprendre les étapes.

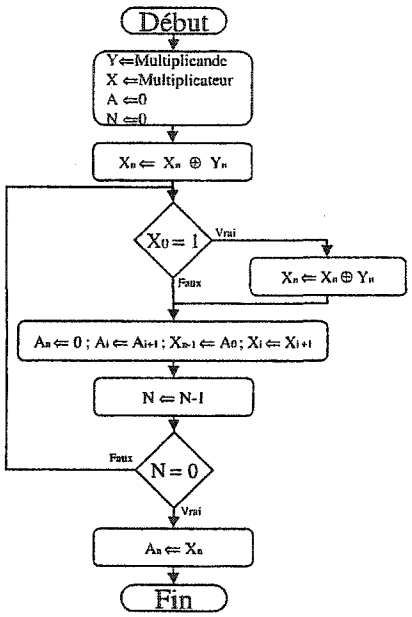


Figure 3.19) Étapes d'une mult. par addition successif (signe-module).

Puisque l'utilisation de nombres signés utilisant la représentation complément à deux a plusieurs avantages et constitue une quasi-norme lors de calculs à point fixe. Il est primordial de présenter l'algorithme de Booth. Celui-ci consiste à utiliser plus d'un bit pour vérifier s'il est nécessaire d'additionner ou soustraire un nombre lorsque le bit le moins significatif n'est pas nul. Ainsi, pour le cas d'un multiplieur par addition partielle

utilisant l'algorithme de Booth on doit rajouter une case de plus pour conserver le bit x_{-1}. Le tableau suivant représente chaque état rencontré et la fonction devant être exécuté. Alors, si les bits x_0 et x_{-1} sont de valeur semblable aucune opération n'est exécutée excepté le décalage obligatoire de droite. Dans le cas contraire, une addition ou soustraction sera exécutée. Ainsi en observant x_i et x_{i-1} lors d'une soustraction, il suffit d'inverser le multiplicande utilisant la méthode de complément à deux. Cette opération est un peu encombrante mais nécessaire. Nous verrons plus loin comment l'exécuter d'une façon simple. [ZAR95]

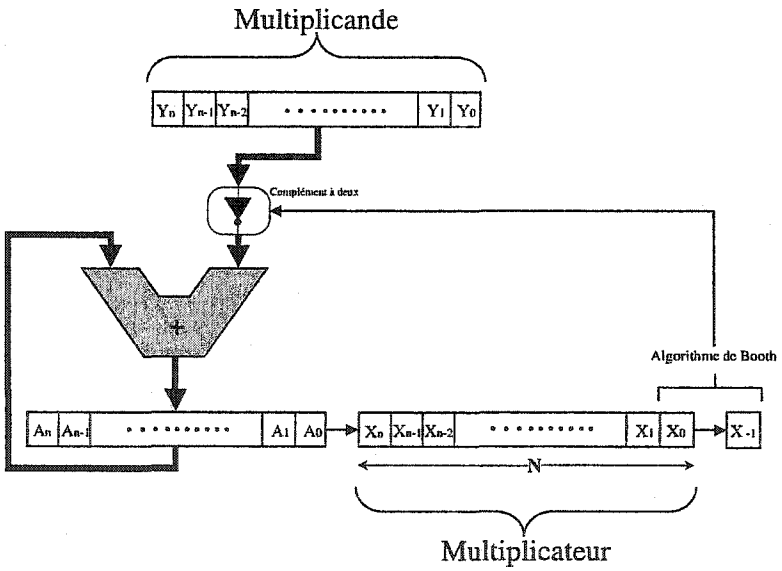


Figure 3.20) Multiplication par accumulation partielle (Algorithme Booth).

X_i	X_{i-1}	Opération	Multiplicande
0	0	Décalage à droite d'une position.	0
0	1	Additionné la multiplicande avec l'accumulateur et Décalage à droite d'une position.	+x
1	0	Soustraction du multiplicande avec l'accumulateur et Décalage à droite d'une position.	-x
1	1	Décalage à droite d'une position.	0

Table 3.1) Opération reliée à l'algorithme de Booth (non-modifiée)

On peut aussi observer, la relation entre chacune des étapes qui amènent à la multiplication par accumulation partielle utilisant l'algorithme de Booth (figure 3.21). [DAN95]

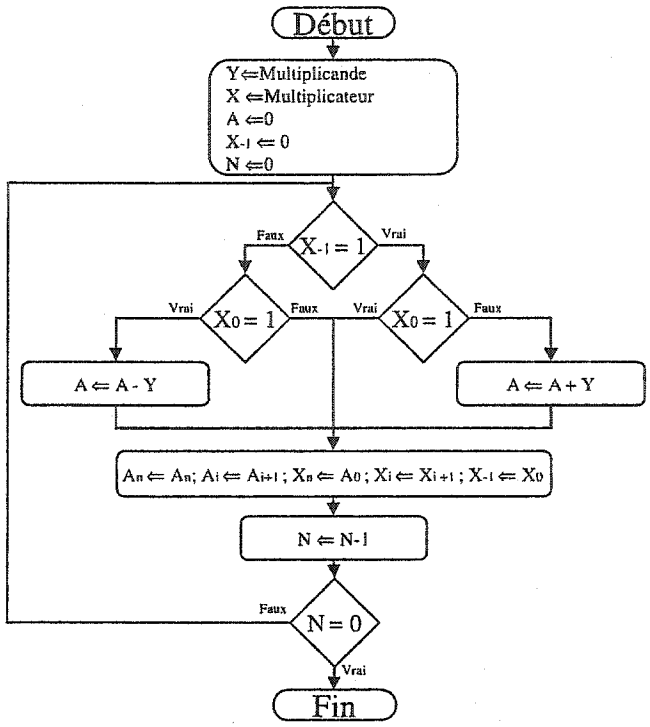


Figure 3.21) Étapes d'une mult. par addition successif (Algorithme Booth).

3.3.2 Multiplieur parallèle

La multiplication séquentielle vu précédemment ne permet pas d'optimiser la rapidité des multiplications. Nous devons malheureusement attendre plusieurs additions pour obtenir la réponse. Le résultat final obtenu par ce type d'architecture prendra alors plusieurs coups d'horloge pour exécuter une seule multiplication comparativement au multiplicateur utilisant une architecture à plusieurs additionneurs. Un autre défaut pour le multiplieur séquentiel serait le temps de calcul d'une multiplication. Puisqu'on additionne uniquement les valeurs non-nuls, la latence varie en fonction du nombre introduit dans le

registre du multiplicateur. Ce qui est très imprévisible, et très nuisible lors d'une implantation où chaque étape nécessite un temps bien précis. On gagne donc énormément lorsque le calcul se fait en un seul coup d'horloge.

Ceci dit, puisque le multiplicateur parallèle est considéré comme bloc de logique combinatoire, chaque opération se fait en un coup d'horloge. L'avantage d'avoir une logique arithmétique combinatoire permet une certaine flexibilité aux modifications impliquant des stratégies structurelles qui pourraient améliorer le débit. Comme par exemple, introduire des étages pipeline à l'intérieur même du circuit combinatoire. Ceci dit, on gagne en rapidité de calcul lorsqu'on utilise plusieurs additionneurs en parallèle sur plusieurs couches pour exécuter la même tâche, bien sûr la quantité de surface d'intégration en soufre.

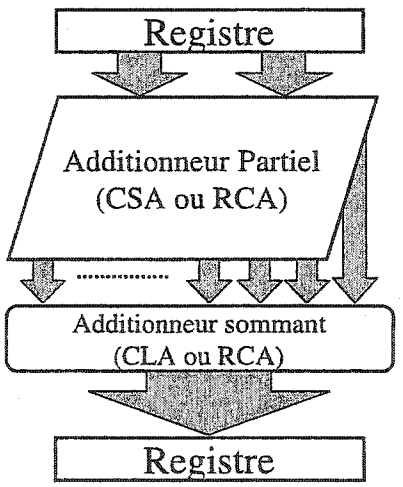


Figure 3.22) Structure du Multiplieur Parallèle

Deux blocs principaux constituent le multiplieur parallèle, nous avons en premier le

bloc d'addition partielle (CSA ou RCA) et en second l'additionneur sommant les additions partielles (CLA ou RCA). C'est deux blocs sont constitués de deux types d'additionneurs, ils sont l'additionneur partiel ou demi-additionneur (figure 3.6) et l'additionneur entier (figure 3.7) présenté à la section subséquente.

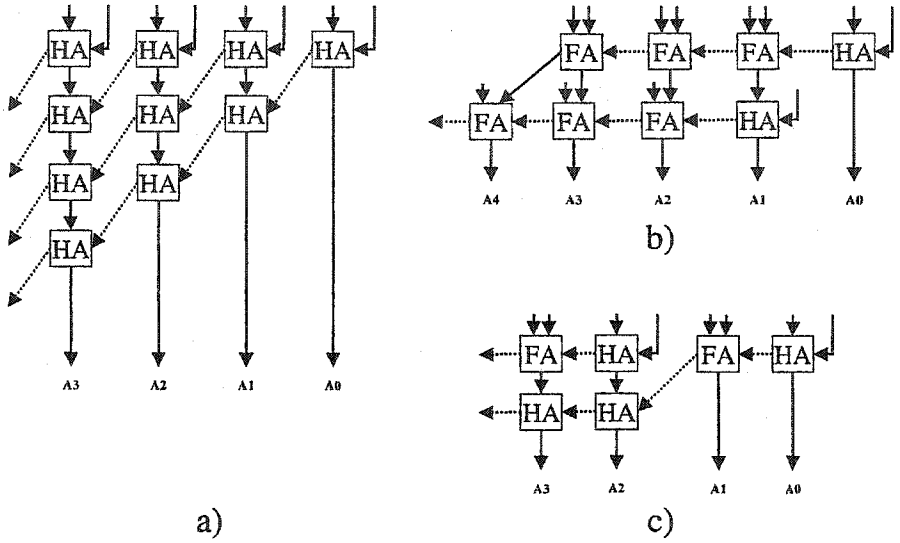


Figure 3.23) Divers Type d'additionneur partiel

a) CSA, b) RCA, c)CSA & RCA

Le bloc le plus important du multiplieur est l'arbre d'addition partielle, présenté ci-dessus. Cet arbre binaire peut avoir une structure de type CSA (a), une structure de type RCA (b), ou une structure de type mixte, c'est-à-dire un mélange CSA et RCA dans le même arbre (c). Dans [PAR99], on explique comment l'utilisation d'une structure constituée d'additionneurs à retenues propagées (RCA) ralentirait le débit du bloc combinatoire. Puisque la retenue doit être propagée à travers son niveau (étage), la donnée

doit constamment attendre la retenue de sortie pour passer à un second niveau. Alors que, pour l'additionneur à conservation de la retenue (CSA), un plus grand parallélisme peut être atteint, car la retenue du bit précédent est propagée au niveau de logique suivant, ce qui cependant augment le nombre de niveaux (étages) et bien entendu la surface d'intégration. Ceci dit, l'arbre binaire présenté en c) utilise les deux types de structures, où un compromis est atteint entre le débit et la surface d'intégration nécessaire.

Lorsque les sommations partielles sont terminées, nous terminerons par le bloc de sommation. Celui-ci, pourra être choisi en fonction de son diagramme temporel, c'est-à-dire qu'il dépend du type d'arrivée binaire, « Arrivent-ils avec un certain délais entre chaque bits ou arrivent-ils tous en même temps ». Ainsi, par cette dépendance nous pouvons choisir notre type d'additionneur. Dans le premier cas (ils arrivent un bit à la fois), nous choisirons plus particulièrement un additionneur à propagation de la retenue (RCA), tirant ainsi avantage d'une structure un peu boiteuse, mais simple et nécessitant peu de surface d'intégration. Le second cas (ils arrivent tous en même temps), utilisant un additionneur à retenue anticipée (CLA), dans ce cas l'addition sera plus rapide vis-à-vis l'utilisation des additionneurs CSA et RCA. La figure 3.24, présente un exemple d'un multiplieur parallèle utilisant un arbre binaire de type CSA, et d'un additionneur RCA, sommant les additions partielles. Cet exemple constitue un exemple idéal d'un multiplieur parallèle très rapide. Cependant, comme mentionné au début de cette section, le multiplieur de ce type de structure multiplie uniquement des nombres binaires de format signe-module.

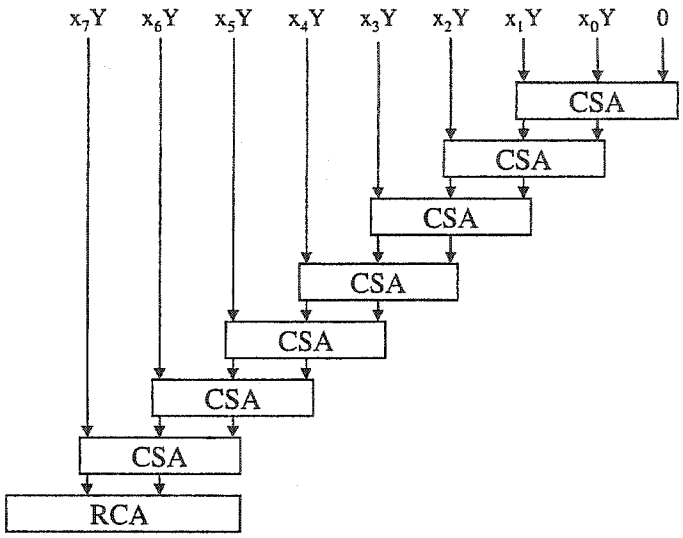


Figure 3.24) Structure du multiplieur cellulaire

L'algorithme Booth vu précédemment lors d'addition séquentielle peut être associée à l'arbre de Wallace permettant ainsi de replacer le format signe-module de l'additionneur parallèle, par le format complément à deux. Où l'arbre de wallace constitue simplement une sommation de résultats partiels. Constitué de plusieurs blocs à conservation de la retenue (CSA), l'arbre de Wallace additionne les résultats partiels en utilisant une sorte de multiplexage en sommant n nombres de bits à $n/2$ et de $n/2$ à $n/4$ ainsi de suite jusqu'à l'obtention d'un résultat. Un point néfaste est tiré de ce type de structure, l'utilisation de bloc CSA ne permet pas de terminer par l'addition de deux nombres (2 entrées) à un nombre (1 sortie), alors l'utilisation d'un additionneur de type deux entréees à une sortie sera utilisé, comme par exemple un RCA ou CLA.

X_{i+1}	X_i	X_{i-1}	Multiplicande
00	0	0	0
0	0	1	+x
0	1	0	+x
0	1	1	+2x
1	0	0	-2x
1	0	1	-x
1	1	0	-x
1	1	1	0

Table 3.2) Opération reliée à l'algorithme de Booth (modifiée)

L'algorithme de Booth et Booth modifiée (utilisant un groupe de bit plus grand comme contrôle pour la multiplicande) permettent de concevoir une architecture arithmétique rapide et utilisant le format complément à deux. Ainsi, la figure 3.24 pourrait utiliser l'algorithme de Booth en modifiant la structure cellulaire passant d'entrées de type x_iY à des entrées $x_ix_{i-1}Y$, et évitant en même temps le calcul de 3x, ce qui diminue la profondeur de l'arbre cellulaire. En effet, en réduisant le nombre de produits partiels, nous réduisons la latence totale d'une multiplication. [PAR97]

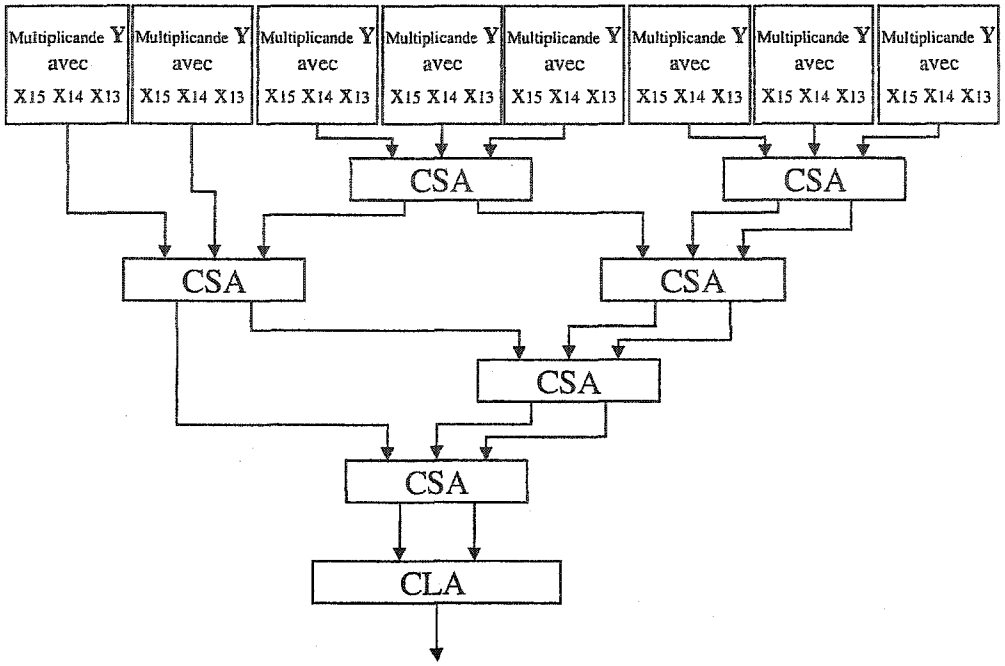


Figure 3.25) Multiplieur Booth Modifié utilisant un arbre de Wallace

L'utilisation de l'algorithme de Booth modifiée permet de diminuer encore plus la profondeur de l'arbre de Wallace ou cellulaire en exécutant des additions partielles sur un nombre de bits plus élevé. Le Tableau de la page précédente indique l'opération à exécuter lorsqu'un type de format binaire entre x_i et $[x_{i+1}, x_{i-1}]$ sont comparés. L'opération la plus élevée est l'addition double du multiplicande. Un exemple de l'algorithme de Booth modifiée est présenté à la figure 3.25, où nous utilisons l'arbre de Wallace comme sommant.

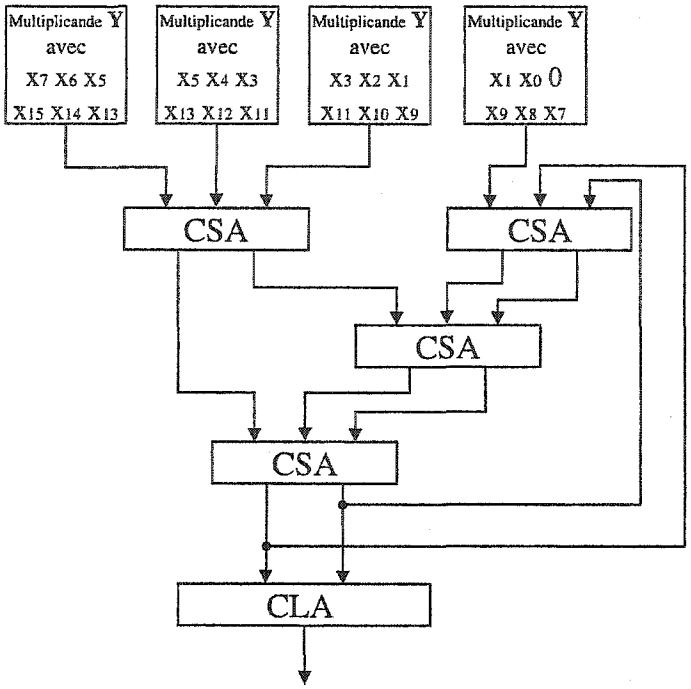


Figure 3.26) Multiplieur Booth Modifié utilisant un arbre de Wallace récursif

Nous pouvons tirés avantage de l'algorithme de Booth en structurant la multiplication utilisant deux fois en une même opération l'algorithme de Booth. Utilisant ainsi plusieurs fois l'arbre de Wallace pour exécuter la sommation. Très efficace lorsque la surface d'intégration devient un élément important. Cependant, le contrôle d'architectures récursives à l'intérieur même d'un ensemble combinatoire peut limiter le débit. Car il est fort probable que la boucle de retour devra être contrôlée par l'horloge, limitant ainsi les délais de propagation à la fréquence de l'horloge. La boucle de retour peut aussi ne pas nécessité de registre dans de très rare occasion. Cette condition sera uniquement atteinte lorsque les chemins (*paths*) à l'intérieur d'une logique combinatoire ont le même temps de

propagation. Ce qui n'ai pas le cas avec l'arbre de Wallace présenté. Nous verrons au chapitre suivant quelles sont les conditions idéales auquelles une structure combinatoire pourrait être récursive. Pour ce qui est de l'implantation de l'algorithme de Booth en ITGE, une intégration au niveau ITGE est extrêmement difficile et nécessite une machine à états pouvant ralentir son débit. [ZAR95], [PAR99]

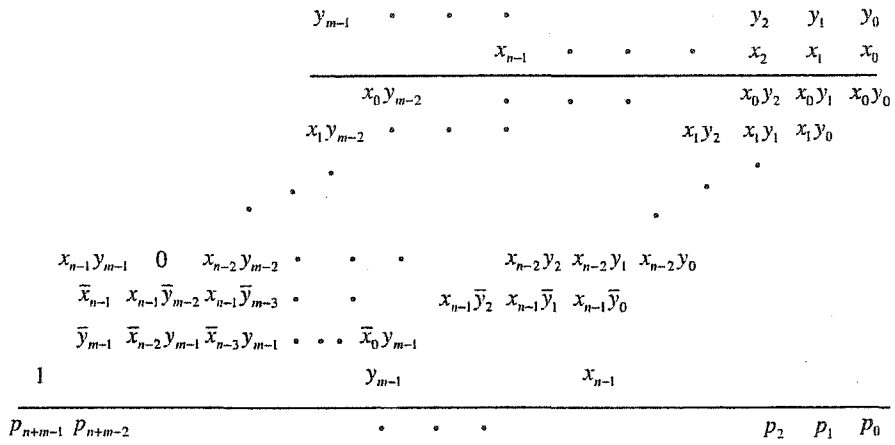


Figure 3.27) L'algorithme de Baugh et Wooley pour un (N•M) bits.

D'autre multiplieur utilisant le complément à deux est proposé, utilisant celui-ci l'algorithme Baugh et Wooley. Ce type de multiplieur est d'après moi le plus efficace. En plus d'avoir une architecture régulière utilisant uniquement des additionneurs entiers, il permet entre autre d'utiliser tout au long de l'architecture des additions uniquement positives. Comparativement aux autres multiplieurs utilisant le format binaire complément à deux, qui utilisent des sommes partielles négatives et positives pour effectuer le produit. Ainsi nous avons une structure où les délais des chemins sont de type déterministes c'est-à-

dire, son temps de propagation est connu de par la structure qu'elle utilise (FA), et tous délais ajoutés par des effets externes seront imputable à tous les étages. Ceci permet une plus grande flexibilité lors d'optimisation du débit comme par exemple l'utilisation d'étages pipeline. Une représentation générale d'une multiplication 4x4 (figure 3.28) de l'algorithme est présentée par M. Zakhama [ZAR95]. Elle permet ainsi d'observer que la structure est du même type de la structure d'addition partielle CSA. Cependant, il n'est plus nécessaire de terminer la multiplication par un étage de sommation des additions partielles.

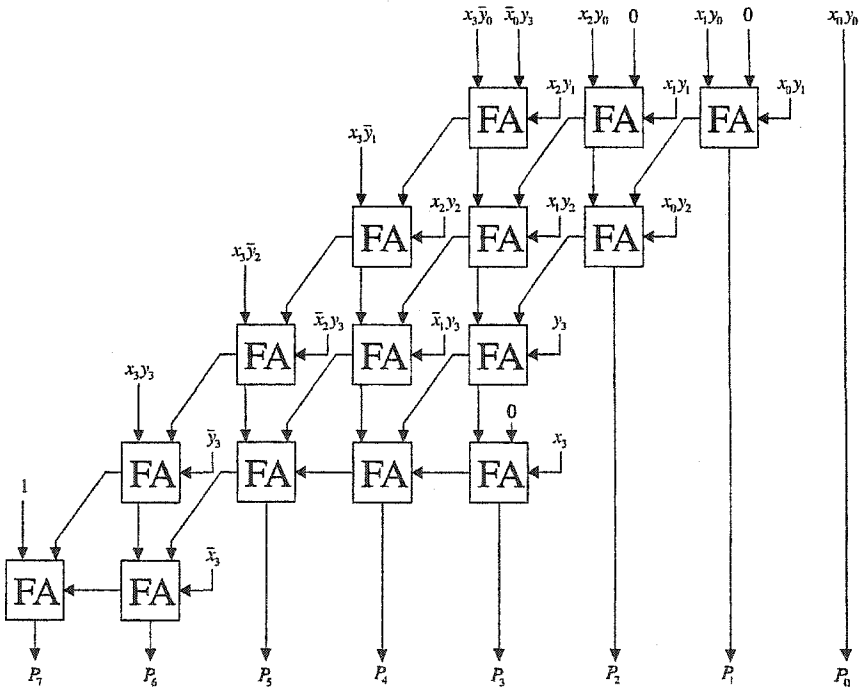


Figure 3.28) Multiplieur complément à deux utilisant Baugh et Wooley.

3.3.3 Multiplieur-Accumulateur (MAC)

Le multiplieur-accumulateur est un élément important d'une implantation architecturale auquelle une de ses fonctions principales est le traitement numérique des signaux (DSP). Par exemple, l'utilisation de filtres numériques demande généralement une multiplication entre les poids du filtre et le signal à traiter, pour ensuite les accumuler entre elles. Ainsi une structure récursive incluant un multiplicateur et un additionneur permettrait d'augmenter le débit du traitement à condition qu'il soit tous deux inclus dans la même structure combinatoire.

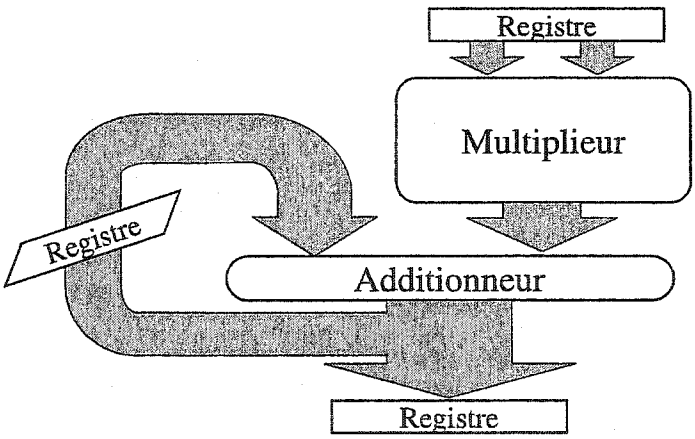


Figure 3.29) Exemple de structure d'un multiplieur-accumulateur

Puisque le résultat de la multiplication précédente doit arriver après que le résultat est été stable lors de la multiplication présente, il est nécessaire de synchroniser la boucle de retour avec une horloge qui elle sera régie (ou limitée) par le temps de propagation du multiplieur et de l'accumulateur. Cette horloge contrôlera le registre tampon pour garder la

donnée précédant suffisamment longtemps avant d'être accumulée avec la donnée présente. Ainsi on respecte le *set-up time* (temps nécessaire à la stabilisation de la donnée) et le *hold time* (temps nécessaire avant d'utiliser une nouvelle donnée).

3.4 Choix des unités arithmétiques

Pour ce projet l'idée principale est l'utilisation d'une architecture de traitement numérique des signaux optimisés par l'utilisation d'arithmétiques utilisant le pipeline par vague. Ainsi, il faut tenir compte de deux choses principales. D'abord, quelles sont les arithmétiques nécessaires à l'implantation de l'architecture utilisant les réseaux de neurones. Et en second, parmis quels types de structures pourrons-nous implanter sur silicium la technique d'augmentation de débits qui est le pipeline par vague. Ainsi, comme vue au chapitre 2, la structure aura besoin d'un multiplieur-accumulateur de 8x8 bits utilisant le format binaire signe-module en entrée et le format complément à deux en sortie. Ceci emmène déjà en soi une restriction assez grande sur le choix des arithmétiques. Puis, nous aurons besoin d'un multiplieur utilisant le format complément à deux et d'un additionneur, tous deux de largeur 8x8 bits. Toute unité devra être rapide, ce qui emmène le choix d'une arithmétique parallèle. Nous avons instauré un type de pipeline, alors l'architecture recommandée pour ce type d'implantation nécessitera une structure régulière pouvant facilement estimer le temps de propagation entre l'entrée des données et la sortie. En plus, l'implantation nécessitera des délais de propagation entre la transmission d'un '1' et celui d'un '0' quasi-semblable ce qui ne permet pas l'utilisation de portes logiques

CMOS et donc emmènera au développement d'une toute nouvelle librairie avec un courant de sortie ne pouvant malheureusement supporter que trois porte logique en sortie, ce qui élimine sans aucun doute l'utilisation d'additionneur CLA, où chaque entrée se décharge à plusieurs autres portes.

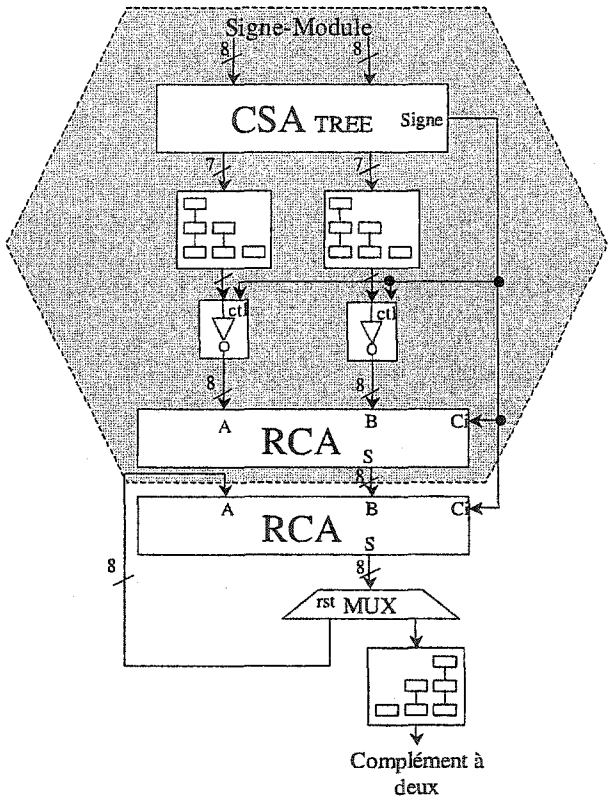


Figure 3.30) Concept d'un multiplieur-accumulateur (MAC)

Alors nous proposons ici l'utilisation de bloc arithmétique conçu généralement d'additionneur FA. Celle-ci servira à la conception de base à toute arithmétique choisie. Elle est simple et facile de balancer les délais. Nous discuterons plus sur ce sujet au

chapitre suivant. En second, puisque la rapidité de calcul est primordiale, nous utiliserons des unités arithmétiques parallèles utilisant cependant plus de surface mais pouvant atteindre de plus grands débits. Alors, nous avons choisi le multiplieur-accumulateur conçu d'une addition partielle utilisant l'arbre CSA et d'additionneurs RCA. Puisque chaque donnée devra être décalée lors de son calcul et transformée en complément à deux, nous introduirons des triangles de temporisation et de dé-temporisation. Mais d'abord, nous conceverons la première partie du MAC représenté par la section grise de la figure 3.30. Celle-ci inclus, l'additionneur et l'arbre CSA exécutant l'addition partielle. La conception de l'additionneur utilisera la structure RCA. Et comme troisième étape, la conception complète d'un MAC sera effectuée. Pour terminer, nous conceverons une unité arithmétique basée sur l'algorithme Baugh et Wooley présenté à la figure 3.28. Le multiplieur Baugh et Wooley utilisera le même type de structure que l'arbre CSA après quelques modifications. Nous verrons au chapitre 5, une description plus détaillée des unités arithmétiques conçues et leurs performances.

3.5 Conclusion

Lors d'une conception nécessitant l'implantation ITGE d'unités arithmétiques, il est primordial de connaître quels sont les choix qui sont offerts et quelles caractéristiques ont-ils. Il en dépend de l'application à laquelle l'architecture fera face pour déterminer le niveau associé à la rapidité, aux coûts et à la fiabilité de la structure voulant être implantée.

Dans ce chapitre, nous avons présenté trois grandes sections primordiales à la conception et aux choix arithmétiques, en plus du format binaire voulant être utilisé. D'abords, comme première section (section 3.1), nous avons fait un survol des formats binaires à point fixe. Comme par exemple, le format signe-module ou complément à deux. Par la suite, nous avons discuté des éléments de base de toute conception arithmétique qui sont les additionneurs. Dans cette section, nous expliquons en bref tout additionneur généralement utilisé à l'intérieur de processeur spécifique. La section 3.3, présente différents types d'algorithmes utilisant une succession d'additions pour permettre la multiplication de deux nombres binaires de format spécifique. Finalement, la dernière section présente le choix des différentes arithmétiques choisies. Le chapitre prochain présentera les types de méthodes d'augmentation du débit basés sur l'implantation de l'architecture et l'utilisation du type de pipeline choisi pour notre projet.

4 Les principales techniques pipeline

Le pipeline par vagues est l'une des méthodes utilisées pour maximiser le débit d'une architecture ITGE (Intégration à Très Grande Échelle). La particularité de cette méthode c'est quelle touche l'analyse et la conception sur plusieurs niveaux (procédé, dessin de masque, circuit, logique (temporel et architectural)).

L'idée principale derrière la méthode du pipeline par vagues a été introduite par Cotten dans les années 1980. Il l'appela *maximum rate pipelining*. Cotten observa que le niveau de flux logique pouvant être propagé à travers un circuit ne dépendait pas du délai du chemin le plus long, mais sur la différence des délais entre le chemin le plus long et le plus court. Il eut la vision d'un nombre de vagues qui se propagent à l'intérieur d'une même logique sans qu'il y ait collision entre elles. Le pipeline par vagues peut même être considéré comme un pipeline virtuel où chaque porte logique représente un registre virtuel.

Ce chapitre a pour but d'introduire les différents types de pipeline connus à ce jour et d'approfondir sur celui choisi. Nous discuterons d'abord sur le pipeline conventionnel de type synchrone où nous verrons ses avantages et inconvénients. Pour ensuite poursuivre sur une technique peu utilisée appelée micro-pipeline ou pipeline asynchrone. Ensuite, nous enchaînerons sur le pipeline par vagues. Finalement, nous approfondirons sur cette

dernière tout en acheminant les étapes à la conception d'unités arithmétiques utilisant principalement des demi-portes de transmission.

4.1 Pipeline conventionnel

Le pipeline synchrone est une technique utilisée depuis longtemps et qui a déjà fait ses preuves dans l'industrie. La technique de pipeline est une méthode d'optimisation du débit. Ce débit peut être dans un premier temps de type programmation, réfèrent le pipeline comme codage d'instruction de haut niveau. Pour nous, la technique du pipeline sert à l'optimisation du débit au niveau ITGE. Elle sert donc à optimiser notre architecture en diminuant la profondeur du circuit combinatoire et ceci en divisant les délais les plus longs en plusieurs étages pipelines.

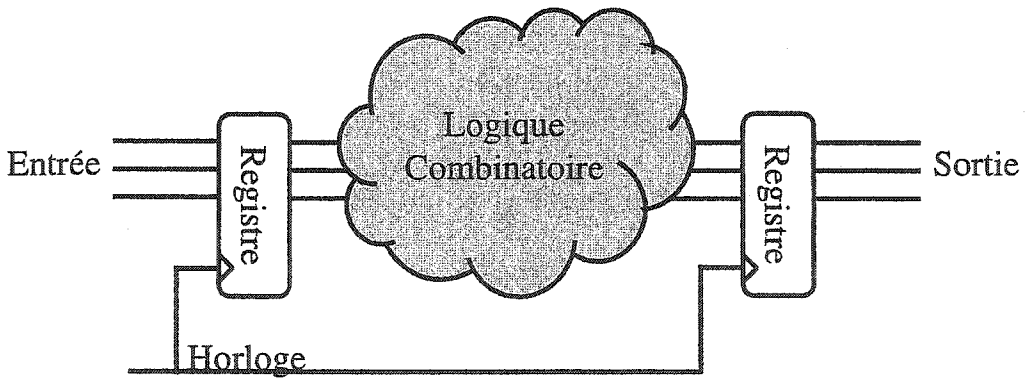


Figure 4.1) Logique combinatoire synchrone

D'abord qu'est-ce qu'une logique combinatoire? Une logique combinatoire est un nombre de portes élémentaires (ET, OU, OU-EX) utilisées pour un but précis à l'intérieur même de son architecture. Cette logique est bornée (en entrée et en sortie) par deux registres séquentiels. La figure 4.1, représente grossièrement une logique combinatoire synchrone. On peut observer que l'horloge sert au transfert de données entre l'entrée et la sortie. Dans ce cas, l'exactitude des données en sortie est dépendante du délai non-idéal introduit sur l'horloge mère généralement appelé en anglais *jitter*. Le *jitter* ou le *wander* (l'ajout de fréquence est inférieur à 10KHz), sont touts deux des conséquences non-voulues à long terme d'une variation dans le temps d'un signal électrique de sa position d'origine. Ce délai, peut être non-voulu, créé soit par son environnement extérieur (température, etc.) ou par la résistance et la capacité interne au guide d'onde (ligne) et porte logique. La figure ci-dessous présente le concept de *jitter* comparativement à une insertion d'un délai contrôlé.

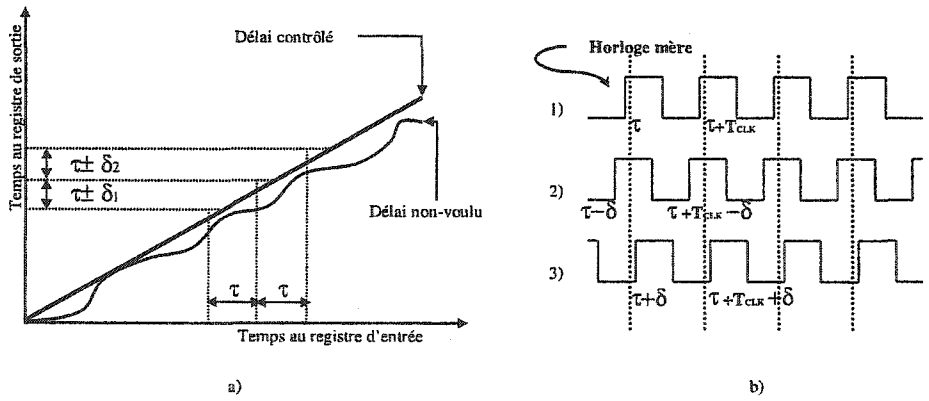


Figure 4.2 Comparaison entre délai voulu et non-voulu,

1) Clk du registre d'entrée 2) Clk du registre de sortie avec délai négatif 3) avec délai positif.

Cependant, puisque nous utilisons une technologie d'intégration de plus en plus petite les problèmes rencontrés sont difficilement prévisibles. Car en plus de considérer le délai de transition pour chaque porte logique ainsi que pour la charge de sortie (*fanout*) auquelle la porte logique devra faire face, il est primordial de considérer les délais créés par la ligne ou la connexion entre chaque porte. Ceci dit, si le débit maximum d'une logique combinatoire est de peu inférieur à la période de l'horloge. Il se peut que la donnée introduite dans le registre de sortie ne soit plus valable. Un bon moyen de contrôler ces délais est d'introduire ou d'ajouter des délais temporels (*buffer*) prédéterminés sous forme de portes logiques. Ceci est considéré dans le jargon technique anglais comme un ajout de *skew*. Comme exemple, l'association de deux inverseurs, l'un à la suite de l'autre sur le chemin où la ligne critique est une insertion contrôlée. Cette horloge temporisée représentée à la figure 4.2.b sert à décaler la phase de l'horloge mère de façon positive contrairement à un décalage pouvant être négatif qui serait néfaste sur un circuit combinatoire utilisant la technique du pipeline d'ordre élevé. Pour ce qui est des circuits conventionnels ou pipelines, tout type de décalage est néfaste sur la performance du circuit, diminuant ainsi la robustesse du circuit. Un délai positif est très utile lorsqu'on utilise la technique du pipeline par vagues. Nous verrons dans une section subséquente, qu'il sera important d'introduire des délais de contrôle (*buffer*) pour limiter les effets néfastes d'une technologie à haute intégration, de procédés de fabrication ou des paramètres externes comme la température lorsqu'on utilise la technique du pipeline par vagues.

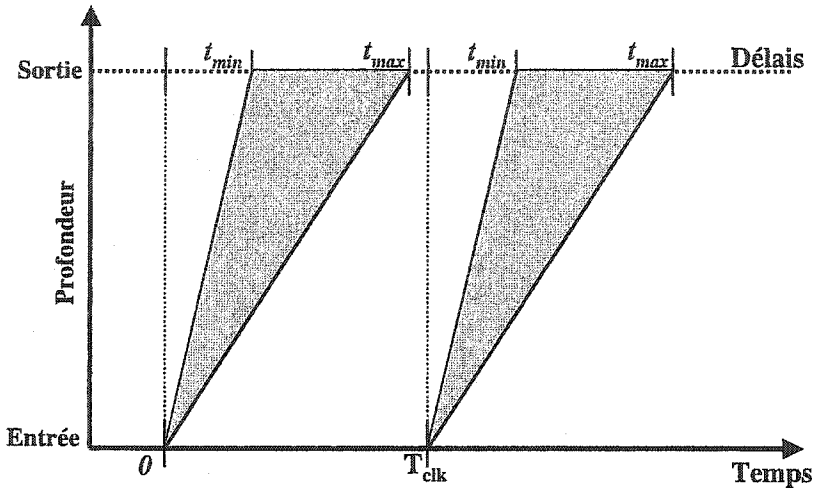


Figure 4.3) Diagramme temporel d'un circuit combinatoire synchrone

Il est donc primordial pour un système synchrone comprenant plusieurs millions de transistors d'obtenir un délai quasis idéal sur la ligne de l'horloge. Une des techniques utilisée est l'implantation d'horloges réseaux en forme d'arbre H. Ce qui a pour bût d'affronter les problèmes de surcharge qui s'appliquent sur l'horloge mère et de limiter les délais registre de circuit voisin ou d'un seul circuit répandu sur une grande surface.

Le diagramme temporel du circuit combinatoire (figure 4.3) représente assez bien ce que nous voulons expliquer lorsque nous discutons du chemin le plus long du circuit comparativement au chemin le plus court. Il servira aussi comme modèle de comparaison avec les autres techniques de pipeline. Ainsi, l'échelle verticale représente la profondeur du circuit combinatoire, de l'entrer à la sortie du circuit. L'échelle horizontale représente le temps. Où T_{clk} est la période de l'horloge. La variable t_{min} représente le temps nécessaire pour traverser le chemin le plus court. La variable t_{max} est le temps nécessaire pour

traverser le chemin le plus long.

Ainsi, la fréquence maximum pouvant être atteinte par l'horloge est :

Freq. max= $1/T_{clk} = 1/t_{max} + 1/t_{setup} + 1/t_{jitter}$

Où t_{setup} est le temps minimum nécessaire au registre pour acquérir une donnée lors du front montant de l'horloge. Et t_{jitter} est le délai non-voulu introduit entre l'horloge d'entrée et de sortie.

Ceci dit, il est assez facile après avoir vu l'équation précédente qu'il suffit de diminuer t_{max} pour augmenter notre circuit en fréquence. Ainsi, en divisant notre circuit par un registre nous augmentons en théorie notre fréquence par deux. Cette technique d'optimisation se nomme pipeline et son ordre est 1. L'ordre est en direct relation avec le nombre d'étages pipeline introduit dans le circuit. Ainsi, pour chaque registre introduit, nous augmentons la fréquence du système. La figure suivante représente un circuit pipeliné synchrone d'ordre 2.

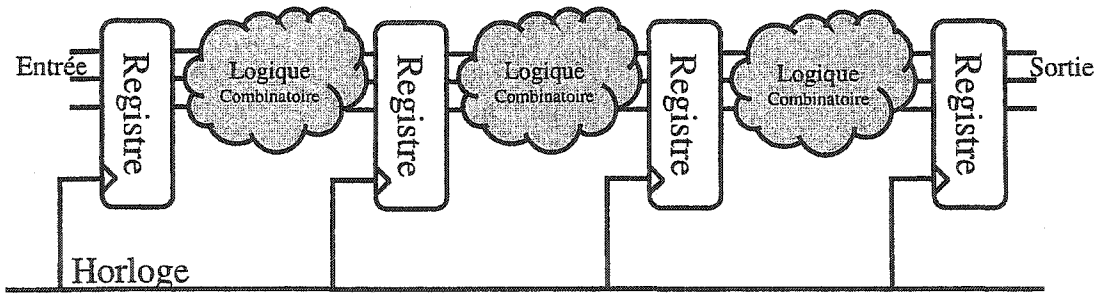


Figure 4.4) Circuit utilisant le pipeline synchrone.

La technique du pipeline synchrone permet de diviser le délai maximum en plusieurs plus petits délais, augmentant ainsi le débit. Cependant, plus le débit augmente plus la synchronisation devient importante entre registres. Ainsi, la technique du pipeline a ses avantages, mais elle a aussi ses inconvénients. L'un de ses inconvénients est que la consommation de surface d'intégration du pipeline augmente rapidement avec l'ordre du pipeline. En effet, chaque bascule maître-esclave nécessite en moyenne 8 portes élémentaires (i.e. 32 transistors). Alors pour chaque étage rajouté nous multiplions ce nombre par le nombre de bits en parallèle, ainsi que par le nombre d'étages pipeline, ce qui ne peut être ignoré lors d'une estimation en surface que le circuit nécessitera. Un autre inconvénient serait une augmentation substantielle de la latence du circuit. Car, chaque nouvelle étape nécessite un coup d'horloge de plus. Donc, ce facteur sera la période de l'horloge utilisée multipliée par l'ordre du pipeline implanté. En plus, il ne faut pas oublier de rajouter le délai de propagation du registre en soi.

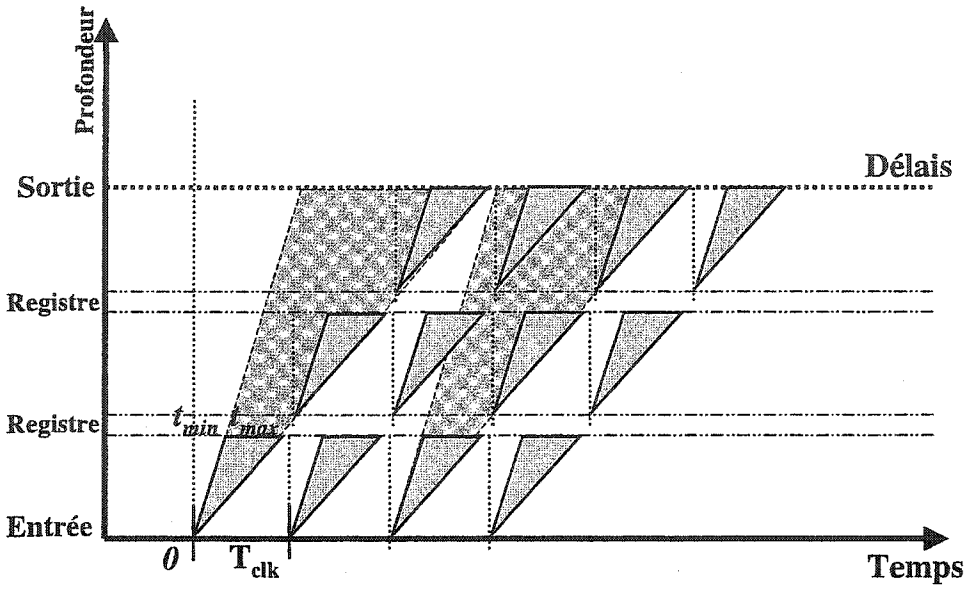


Figure 4.5) Diagramme temporel d'un circuit combinatoire pipeline.

Comme troisième inconvénient néfaste, nous avons le niveau de surcharge crée sur ligne d'horloge qui synchronise chaque registre entre eux. Ceci dit, une surcharge sur la même ligne tant à rallonger les transitions sur la ligne. Ainsi, puisque le délai diminue et que l'horloge à une période plus courte il devient important d'obtenir des transitions franches, mais l'utilisation d'un trop grand nombre d'étages pipeline amène une surcharge de l'horloge. En plus, puisque la transition est lente plus la consommation de puissance en courant de fuite augmente, puisque le niveau de consommation de puissance dynamique est très élevé lorsque la tension d'entrée est sensiblement égale au seuil logique. Sans compter bien sûr que la consommation de puissance normale est proportionnelle à la fréquence d'utilisation pour tout type d'optimisation CMOS [SAV88]. Mais, comme discuté plutôt, il existe des techniques limitant la surcharge de l'horloge en introduisant un arbre temporisé,

permettant ainsi de diviser la charge en sous section et de limité la quantité en *jitter*. D'autre part, puisque l'idée principale derrière la technique du pipeline est de diminuer le chemin critique en ajoutant des registres, nous augmentons d'une part la fréquence de l'horloge, mais d'un autre point vu, nous augmentons aussi le délai de propagation total de chaque registre ainsi que le délai de mémorisation (*setup time*) et le délai de retenu (*hold time*) de chaque registre. Comme on peut observer, la technique du pipeline synchrone a ses limites. Voilà pourquoi nous présenterons à la section suivante une alternative au pipeline synchrone, en introduisant le micro-pipeline (pipeline asynchrone).

En conclusion, la technique du pipeline conventionnel constitue une excellente optimisation du débit d'une architecture à court et moyen terme, c'est-à-dire lorsque nous voulons optimiser notre circuit en vitesse mais sans nécessairement rechercher une intégration optimale (fréquence et surface). Cette technique constitue la plus simple à intégrer mais ne présente aucun défis technique à part lorsque l'ordre du pipeline est élevé.

Les points positifs sont :

1. Facilité d'application et d'intégration aux logiques combinatoires.
2. Type de structure semblable aux conceptions conventionnelles synchrones.

Les points négatifs sont :

1. Relation directe entre l'ordre du pipeline et l'utilisation de réseau d'horloge, causant une augmentation de puissance et des *jitters* néfastes.
2. Relation directe entre l'ordre du pipeline et la surface utilisée.

3. Relation néfaste entre la diminution du délai critique et l'augmentation des délais causée par l'augmentation croissante de registre.

4.2 Pipeline Asynchrone

Le micro-pipeline est une technique développée pour contrer les effets néfastes qu'une horloge centrale surchargée par un nombre toujours croissant d'étages pipeline peut apporter. Dans cette section, nous parlerons principalement du micro pipeline Sutherland, ainsi que des principales communications entre blocs logiques (mono et duo donnée). Pour terminer avec la mise en valeur d'une porte spécialement conçue pour le pipeline asynchrone communément appelé porte-C (*C-gates*) de Muller.

Comme vu précédemment, l'un des points néfastes d'une implantation pipeline est la quasi-impossibilité d'obtenir une synchronisation idéale entre les registres sans l'ajout de portes temporisatrices pour contrer la surcharge d'impédance d'entrée. Ce qui a pour conséquence, une augmentation substantielle de la surface d'utilisation, ainsi qu'une augmentation de la puissance dynamique consommée. Le micro-pipeline ou pipeline asynchrone, quand à eux, n'utilisent aucune horloge centrale, leurs communications reste locale. Puisqu'elle reste locale, la consommation de puissance dynamique est d'autant plus faible car seul les transistors en transition (passant d'un 0 à 1 ou 1 à 0) consommeront de l'énergie. Contrairement aux méthodes conventionnelles qui activent tous transistors

dominés par l'horloge centrale.

Une architecture asynchrone permet le transport des signaux sans devoir synchroniser les valeurs avec l'horloge mère, ce qui emmène une simplification du transport de l'horloge en haute fréquence. Le micro-pipeline fonctionne selon le principe du transmetteur/receveur. Ils existent deux principales méthodes de communication asynchrone généralement utilisées. La première méthode et la plus simple est une communication du type *Bundle-data protocol*, utilisant le protocole « mono-donnée ». Elle fait l'objet de trois types de signaux : la transmission des données à transmettre ($x_0 : x_n$) entre les deux structures, le signal indiquant que les données sont disponibles (*Demande*) à l'entrée et le signal indiquant au système précédent qu'il y a eu réception (*Acceptation*). Les articles sur le micro-pipeline proposent des différentes techniques basées sur ce principe.

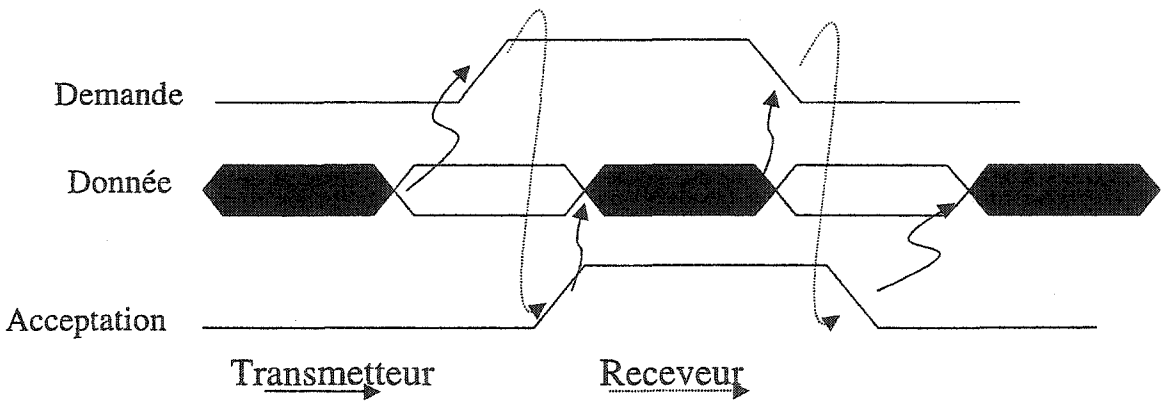


Figure 4.6) Exemple de transmission à deux phases.

La figure ci-dessus représente le modèle de communication à deux-phase utilisant

comme protocole le système mono-donné. La communication à deux phases utilise les fronts montant et descendant du demandeur pour communiquer. Pour diminuer la complexité de concevoir une machine à états à deux fronts, l'utilisation d'une communication à quatre phases a été développée. Celle-ci utilise uniquement les fronts montants comme transition mais le principe reste le même.

Le protocole à deux ou quatre phases représentées à la figure 4.7 n'est pas sans faille. Car, ce qui permet d'avoir une structure asynchrone idéale est une structure indépendante aux délais de connexion. Comme on a déjà vu, les délais de connexion entre deux structures comportent un certain risque, surtout lorsque la technologie d'intégration se fait de plus en plus petite. Alors, même si le signal de demande a été envoyé un certain temps après les données, on sera toujours dans une condition où le signal de demande peut arriver avant les données. Le jargon anglais utilisé pour ce type de problème est un *racing time condition*. Ainsi, nous ne pouvons considérer le protocole mono-donné comme un protocole aux délais insensibles.

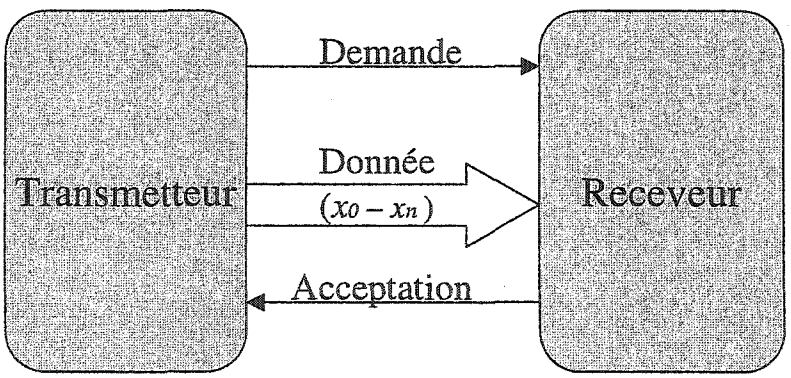


Figure 4.7) Protocole mono-donné (micro-pipeline)

Contrairement au protocole mono-donnée, notre second protocole ne possède pas de signaux indiquant que les données sont disponibles (*Demande*). Il utilise plutôt ce concept, mais combiné avec le signal des données. C'est-à-dire, que nous transmettons deux signaux pour chaque bit transmit ($x_{i,0}$ et $x_{i,1}$). Où, $x_{i,0}$ indique que nous transmettons un '0', et $x_{i,1}$ indique que nous transmettons un '1'. Ce protocole se nomme duo-donnée (*Dual-rail protocol*). Il se sert donc de la donnée pour la capturer par une bascule. Ceci contribue à désensibiliser le micro-pipeline des délais d'interconnexion, puisque la demande arrive en même temps que la donnée. Un point néfaste de l'utilisation du protocole duo-donnée est que la surface d'utilisation pour les interconnexions tend à augmenter par un facteur N^2 comparativement à la surface d'utilisation du protocole mono-donnée. En effet, puisque le nombre de bits est un facteur important, ceci peut nous causer problème lorsque vient le temps de la conception. Comme pour P. Day et J. Viv. Woods [DAY95], ce problème a été un choix déterminant lors de la sélection de leur architecture.

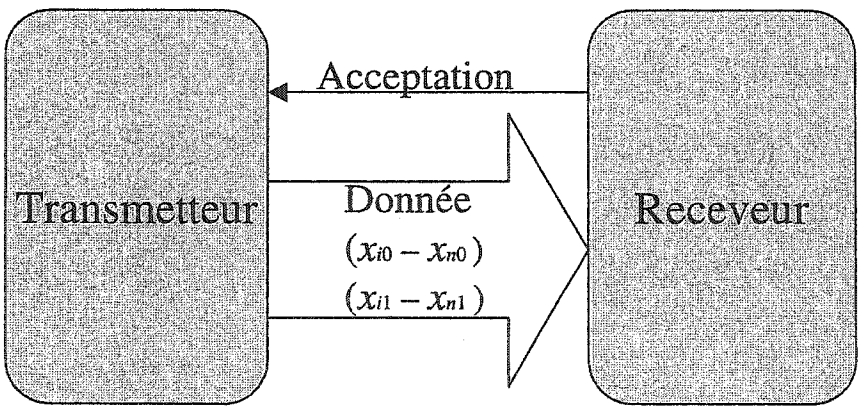


Figure 4.8) Protocole duo-donné (micro-pipeline)

Le micro-pipeline communément appelé pipeline asynchrone, d'après plusieurs articles ([DAY95], [PAR96], [PAR97], [HAU98], [JOH98], [TAY98]), choisit en principe l'architecture proposée par Sutherland [SUT89]. Cette architecture présenté par P. Day [DAY95] serait la plus pratique. Elle est extrêmement bien conçue au niveau performance et du coût. Fonctionnant sur le principe de deux-phases et utilisant le protocole mono-donnée, cette architecture a pour avantage d'être extrêmement simple et pouvant aussi être utilisée comme structure **FIFO** (*first in, first out*). Le FIFO est un circuit temporisateur séparant deux domaines utilisant des horloges maîtres de vitesse différente. Comme par exemple, une interface qui fonctionne à une horloge 104 MHz alors que l'architecture interne fonctionne à 77.76 MHz. Puisque les débits sont de différente vitesse, il nous faut absorber une quantité prédéterminée de donnée avant l'échange. La figure ci-dessous représente l'architecture de Sutherland pour un FIFO.

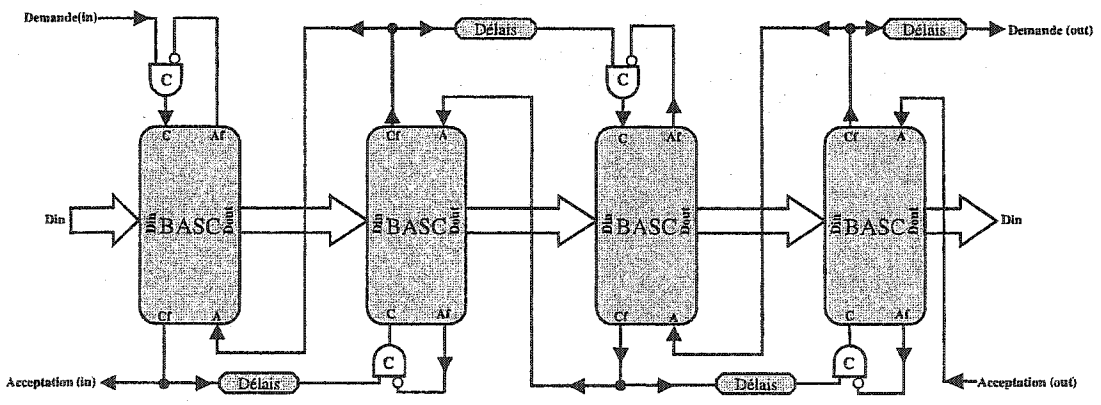


Figure 4.9) FIFO asynchrone de type Sutherland.

Il suffit ensuite d'introduire les blocs combinatoires (*CLB*) entre les registres

asynchrones pour obtenir le micro-pipeline. La figure 4.10.a présente grossièrement à quoi ressemble chaque bascule proposée par Sutherland. Plusieurs propositions ont été faites pour la conception de ce registre asynchrone, mais la meilleure a été proposée par P. Day [DAY95]. Elle est constituée de portes de transmission logique et d'un élément essentiel appelé une porte C (*C-gates*) proposé par Muller (figure 4.10.b). Cette porte logique fonctionne principalement comme une porte ET. Cependant, lorsque l'une des deux entrées est de différente valeur comparativement à l'autre, la sortie gardera la valeur précédente en mémoire. La figure représente deux façons de concevoir cette porte l'une en utilisant un procédé CMOS et l'un en intégrant un multiplexeur comme lors d'une conception par FPGA (*Field-Programmable Gate Arrays*). Ceci dit, on peut remarquer que le signal de demande ou le signal d'acceptation sont directement reliés à son prochain, de même pour son inverse. Ainsi, comme discuté auparavant nous devons minutieusement calculer les délais en conséquence de la distance entre chaque structure pour s'assurer que les données on eu suffisamment de temps pour arriver avant que le signal de contrôle (Demande) ne soit activé.

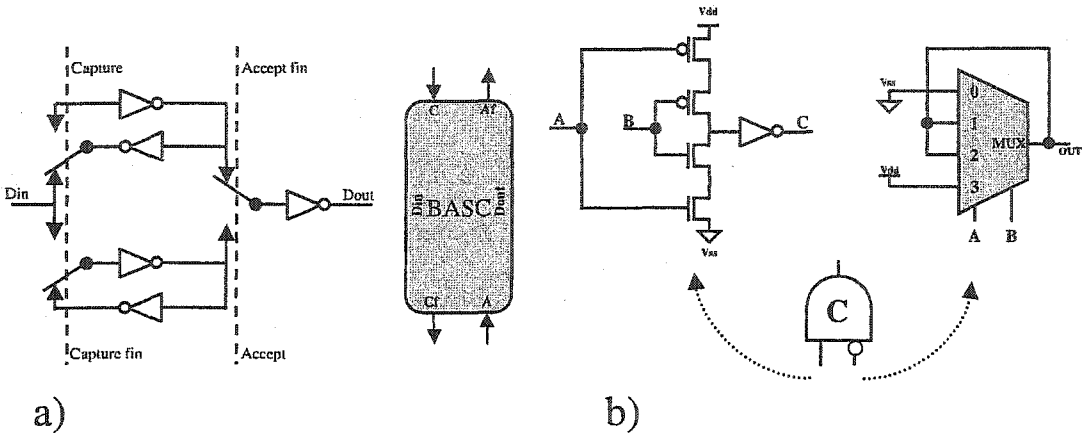


Figure 4.10) Bascule Sutherland (a) Porte C de Muller (b)

En conclusion, la technique du pipeline asynchrone constitue une excellente optimisation du débit d'une architecture à long terme. C'est-à-dire, lorsque nous voulons optimiser notre circuit en vitesse et que tout autre avenue ne fonctionne pas. Cette technique constitue la plus complexe à intégrer mais peut apporter énormément au circuit à plusieurs millions de transistors. M. Keating [KEA99] prédit que nos circuits intégrés seront de l'ordre d'environ 100 Millions de transistors dans la prochaine décennie régie par la loi de Moore. Alors, nous devons penser à optimiser nos structures puisqu'il est improbable qu'une seule horloge puisse fournir une synchronisation à tout le circuit en même temps même avec l'ajout d'un réseau temporisateur. La figure ci-dessous représente le circuit miro-pipeline Sutherland.

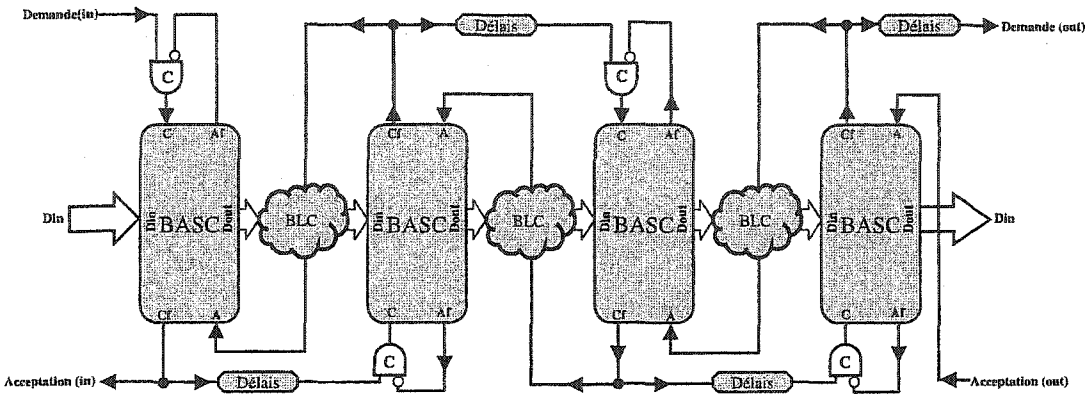


Figure 4.11) Structure principale du micro-pipeline Sutherland

Les points positifs sont :

- Robuste aux délais d'interconnexions (insensible aux délais).
- Implantation d'un réseau d'horloges temporisées non-nécessaire.
- Consommation de puissance très faible puisque les communications restent locales.

Les points négatifs sont :

- Implantation complexe au niveau des contrôles entre bascules.
- Utilisation d'interconnexions supplémentaires lors de l'intégration de la technique duo-donnée, ce qui amène une utilisation de surface.
- Très peu utile pour des débits très élevés (la vitesse est limitée par les retours trop fréquents de contrôle des données.)

4.3 Pipeline par vagues

Le pipeline par vague (*maximum rate pipelining*) est une technique d'optimisation du débit à l'intérieur d'architectures synchrones. Cette technique a pour caractéristique d'introduire des données dans un bloc combinatoire de façon plus rapide que le temps que prend une valeur pour traverser la structure (latence) et être mémorisée par le registre de sortie. Ainsi, chaque transistor sert d'élément mémoire virtuel du même type que lorsqu'une architecture utilise le pipeline conventionnel. Les registres sont utilisés pour diviser la logique en plus petite partie. D'après [NOW95], nous pouvons atteindre des vitesses de 2 à 7 fois supérieures en utilisant le pipeline par vague en théorie comparativement au pipeline conventionnel.

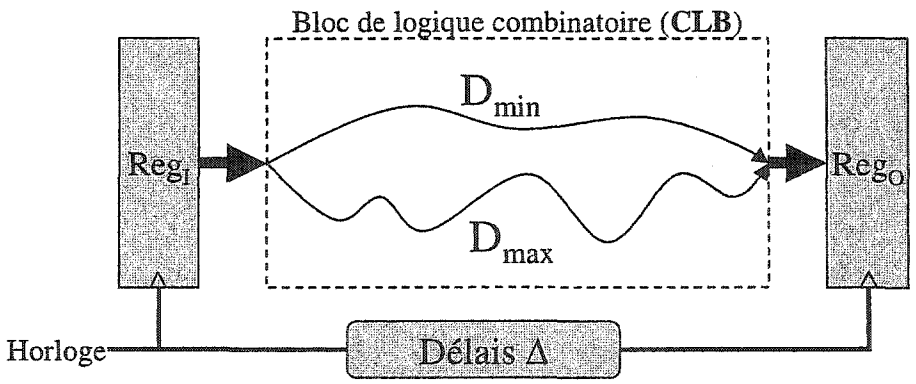


Figure 4.12) Technique du pipeline par vagues.

Ceci dit, chaque donnée est transmise dans un circuit de logique combinatoire (*CLB*) à l'instant où nous pouvons garantir quelle ne chevauchera pas la donnée transmise

précédemment. Voilà d'où vient le terme "vague", c'est-à-dire chaque donnée transmise serait une sorte de vague qui est régie non par le chemin le plus long comme le pipeline conventionnel, mais d'une optimisation entre le délai le plus court *Dmin* et le plus long *Dmax*. Grosso Modo, si nous ne tenons pas compte des autres caractéristiques moins importantes pour augmenter le nombre de vague à l'intérieure d'un CLB, nous augmenterons par le fait même le débit de notre architecture. Nous pouvons alors prendre en considération qu'il faut atteindre une différence quasis nulle entre le délai *Dmax* et *Dmin* pour obtenir une architecture idéale à l'implantation pipeline par vague.

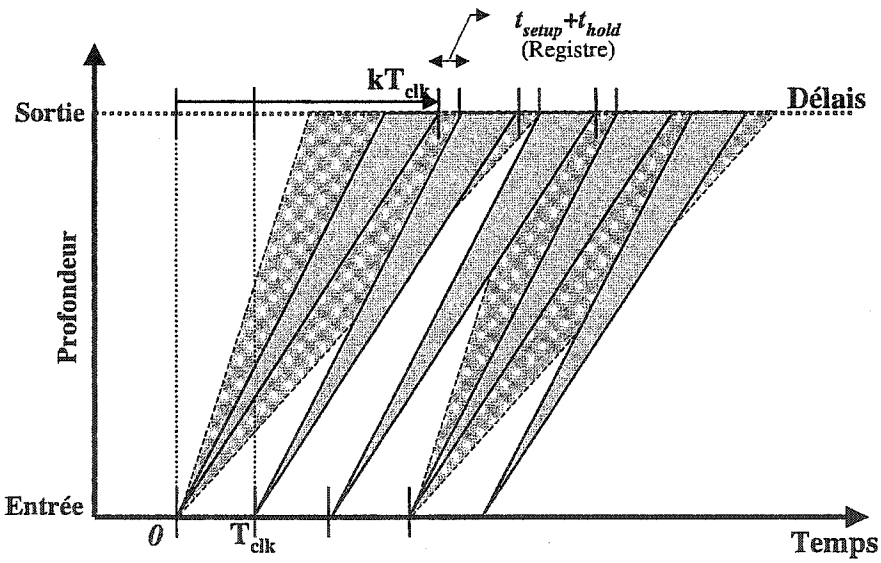


Figure 4.13) Diagramme temporel d'un CLB pipeline par vagues.

Le pipeline par vague fût originalement proposé par [COT69] en 1969 lors de recherche pour IBM. Cependant, le terme proposé était "maximal rate pipelining" (pipeline au débit maximum) au lieu de pipeline par vague. Ce concept fût malheureusement mis de côté

puisque les autres techniques étaient amplement suffisantes et malheureusement les technologies utilisées ne permettaient pas son implantation. Ces dernières années, cette technique a pris en importance puisque la technologie et la conception nécessitent qu'on développe des architectures de plus en plus rapide, consommant moins d'énergie et de surface.

Il existe plusieurs facteurs qu'il faut prendre en considération lors d'une implantation par vagues. Nous avons la technologie utilisée, le procédé de fabrication, la source de tension, la température, la technique d'implantation, la structure ITGE à implanter. Tous ont leur importance, cependant il n'y a que deux qui seront sous notre contrôle. C'est-à-dire, la technique d'implantation et la structure ITGE idéale au pipeline par vagues. Dans un premier temps, nous avons la technique d'implantation. Celle-ci peut être constituée de plusieurs sous-éléments, comme le type de porte logique utilisée, l'ajustement de type grossier et précis et même la temporisation contrôlée de l'horloge maître (Clock Skew). Dans un deuxième temps, nous avons la structure ITGE idéale pour la fonction voulant être optimisée. La structure doit être choisie avec précaution. Plusieurs caractéristiques comme la latence, la dépendance entre les données et le type de structure. Le type de struture (forme mono ou poly-harmonique), aura-t-elle une boucle de retour (Feed-Back) avec ou sans registre, etc.. Ainsi, l'implantation peut devenir assez compliqué si nous voulons obtenir une structure idéale. Les sections suivantes permettront d'approfondir notre connaissance sur le pipeline par vagues mais tout en restant bref et concis.

4.3.1 Calcul fondamental

Cette section permet de relier mathématiquement le débit maximum possible et le nombre de vagues devant être introduites avant qu'il en résulte un chevauchement. Les calculs reliés au nombre maximum de vagues introduites seront considérés pour un bloc de logique combinatoire quit est bordée par deux registres principaux (l'un à l'entrée et l'autre à la sortie), cette structure sera de type mono-harmonique. C'est-à-dire, les données auront qu'une fonction. La figure ci-dessous présente les différences entre mono-harmonique et poly-harmonique. Pour simplifier encore plus notre modèle, notre structure n'aura aucune boucle de retour, contrairement à la figure 4.14.c. Ceci permet de simplifier la compréhension du système. En plus, nous introduirons le principe du contrôle de la phase temporelle (self skew), introduit pour permettre de prendre un certain pouvoir sur un déphasage temporel créé par la ligne qui transmet l'horloge au registre de sortie.

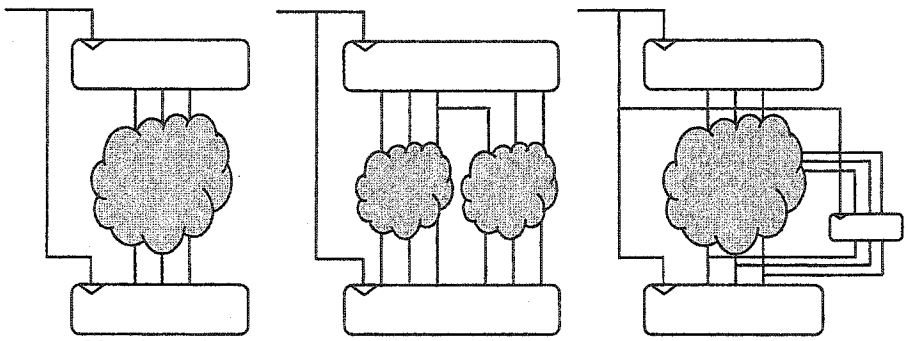


Figure 4.14) Type de structure rencontrée.

La représentation des variables principales du système est indiquée ci-dessous par une liste de symboles:

- D_{MIN} représente le délai de propagation minimum du bloc combinatoire. (CLB)
- D_{MAX} représente le délai de propagation maximum du bloc combinatoire.
- T_{CK} La période de l'horloge
- T_S, T_H Le temps de stabilité des registres (*setup, hold time*)
- D_R Le délai de propagation du registre
- Δ Délai contrôlé de l'horloge en introduisant des temporisateurs.
- Δ_{CK} Délai non contrôlé de l'horloge (cause des registres et surcharge)

La figure 4.12, représente le bloc de logique combinatoire où Dmax et Dmin sont représentés. Un exemple est présenté pour comprendre les étapes de calculs nécessaires à l'égalisation des chemins logiques. Prenons comme exemple un simple circuit logique comprenant un inverseur et une porte de type ET. Nous pouvons observer que l'entrée X sera le chemin le plus court traversant uniquement la porte ET. Alors que le chemin le plus long nécessite un trajet où l'inverseur n'a malheureusement pas les mêmes délais que la porte logique ET. Un diagramme temporel présente Dmin et Dmax. [GRA93], [PAR99]

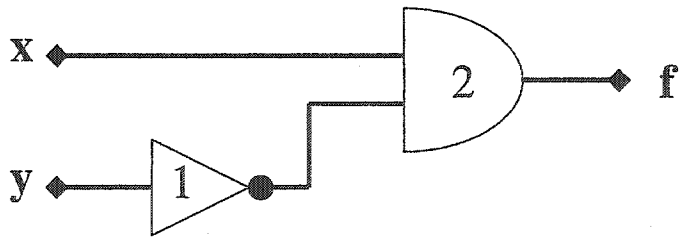


Figure 4.15) Exemple de logique combinatoire

Puisque le délai traversant un inverseur est plus court pour une porte CMOS qu'une porte élémentaire ET ont remarque sur la Figure 4.16 une non-linéarité exercée par des chemins de dimension non-égale, ainsi en égalisant nos chemins notre délai le plus court aura une représentation temporel plus linéaire. Et donc, ceci permet d'optimiser le nombre de vagues. Bien sûr, la profondeur ou la latence permet aussi d'augmenter le nombre de vagues, mais nous en reparlerons plus loin.

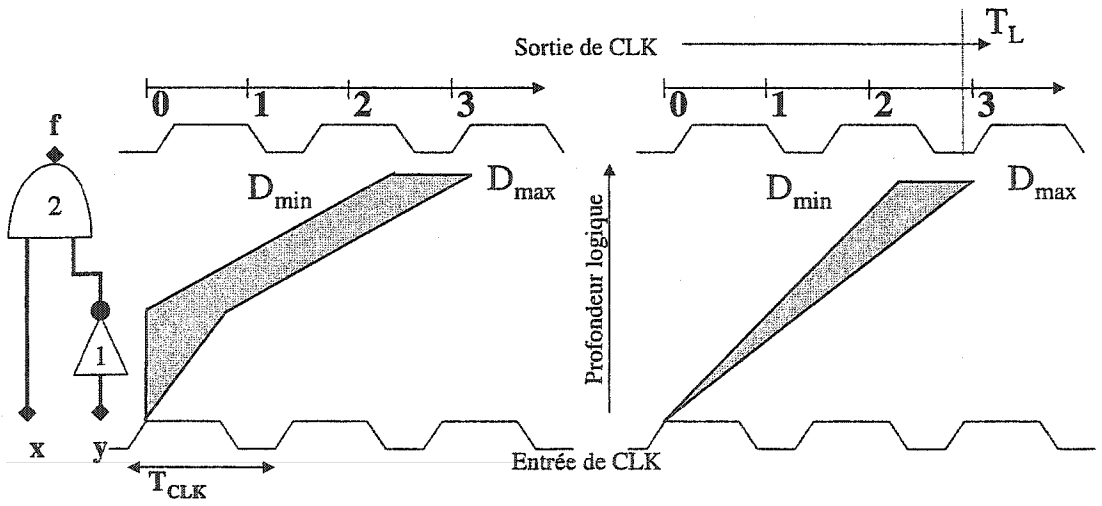


Figure 4.16) Linéarisation temporel des chemins

Pour simplifier nos équations nous ferons appel à un circuit combinatoire linéaire. La figure 4.17 permet alors une vue globale d'une structure combinatoire à plusieurs vagues, il prend en considération les pires conditions néfastes, (où la température est élevée et la source de tension est au-dessous des normales). La variable N est le degré du pipeline par vague, un peu comme le nombre de registre représente l'ordre du pipeline conventionnel. Comme on peut l'observer, durant t_{ref} nous avons plusieurs vagues transmises et chaque vague à une distance T_{sx} de son prochain. Puisque l'horloge n'est pas idéale nous inclurons les effets néfastes qui peuvent être causés sur le réseau de l'horloge mère dépendant des conditions externes (tension, température, procédées, etc....) [BUR98].

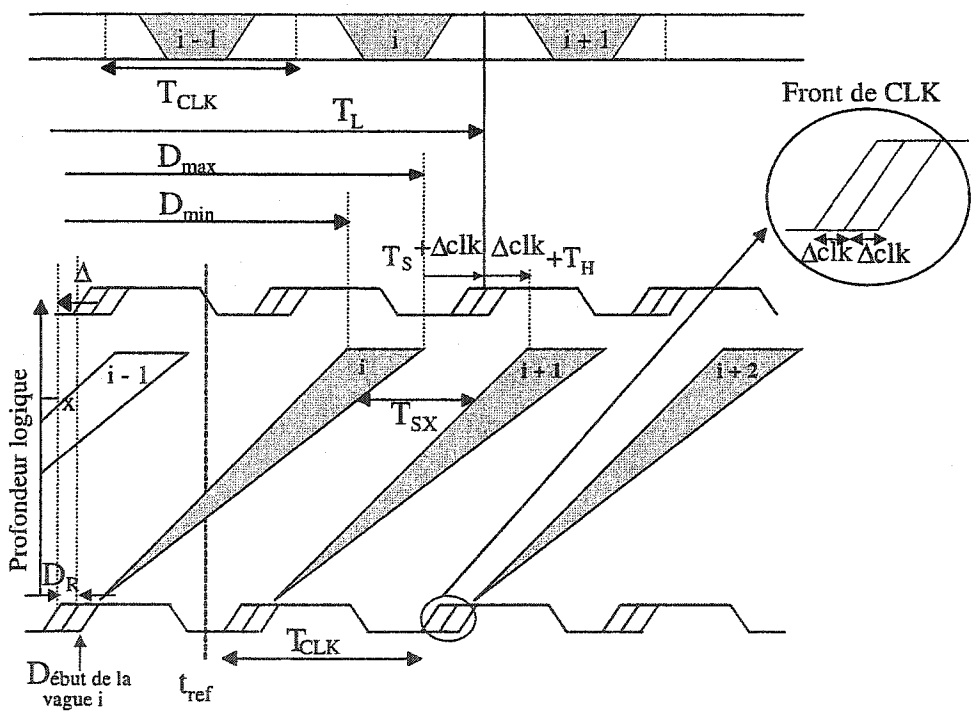


Figure 4.17) Schéma temporel.

La capture de chaque vague de donnée doit être échantillonnée à tout les T_L, c'est-à-dire dans la zone non grisâtre (lorsque les données sont stables). Donc

$$T_L = (NT_{CLK} + \Delta) \quad (4.1)$$

Tsx représente la séparation temporelle entre deux vagues au même endroit dans le circuit logique ou de même profondeur x. Alors que Δ est le temps de propagation de l'horloge à travers la logique combinatoire. C'est-à-dire, les délai entre l'horloge au registre d'entrée et de l'horloge en sortie. Δ_{CLK} est le *jitter* causé les variations externes (Température, Source de Tension, etc)

A) Contrainte des registres:

temps d'arrivée de la dernière donnée:

$$T_L > D_R+D_{MAX}+T_S+\Delta_{CLK} \quad (4.2)$$

temps d'arrivée de la plus récente donnée:

$$T_L < T_{CLK}+ D_R + D_{MIN}-(\Delta_{CLK}+T_H) \quad (4.3)$$

Le maximum rate pipelining (Cotten):

$$T_{CLK} > (D_{MAX} - D_{MIN})+T_S+T_H+2\Delta_{CLK} \quad (4.4) \quad \leftarrow (4.2)+(4.3)$$

Ainsi on peut observer que la période de l'horloge dépend d'une part, de la différence entre (D_{MAX}-D_{MIN}) et d'autre part du dépassement relié à l'ajout de registre synchrone (T_S+T_H + 2Δ_{CLK}).

L'équation (4.4), permet de confirmer qu'une période minimum doit être respecté pour ne pas avoir de chevauchement entre les vagues à la sortie du bloc combinatoire. Cependant, il ne garantit pas le chevauchement des vagues à l'intérieur du bloc CLB.

B) Contrainte liée au nœud interne du CLB.

Ainsi la période ne peut être inférieure à la différence de délais au même nœud pouvant créer un chevauchement à l'intérieur même du bloc CLB, c'est-à-dire à un nœud précis:

$$T_{CLK} \geq (d_{MAX}(x) - d_{MIN}(x)) + T_{SX} + \Delta_{CLK} \quad \textbf{(4.5)}$$

L'équation (4.5) est quasi semblable à l'équation (4.4) sauf que la contrainte interne n'a pas de registres externes donc le décalage temporal est moins élevé. T_{SX} remplace (T_S + T_H) qui est le temps minimum qu'une valeur doit être stable avant d'être propagée.

En utilisant les équations (4.1), (4.2), (4.3), nous obtenons:

$$D_R+D_{MAX}+T_S+\Delta_{CLK} < NT_{CLK}+\Delta < T_{CLK}+D_R+D_{MIN} - (\Delta_{CLK} + T_H) \quad \textbf{(4.6)}$$

En utilisant T_{MAX} et T_{MIN} dans (4.6):

$$T_{MAX} = D_R + D_{MAX} + (T_S + \Delta_{CLK}) - \Delta \quad (4.7)$$

$$T_{MIN} = T_{CLK} + D_R + D_{MIN} - (\Delta_{CLK} + T_H) - \Delta \quad (4.8)$$

Ainsi nous avons une équation régie par le nombre de vagues,

$$\frac{T_{MAX}}{N} < T_{CK} < \frac{T_{MIN}}{N-1} \quad (4.9) \quad \leftarrow (4.7) + (4.8) + (4.4)$$

4.3.2 Optimisation des délais de propagation

Ils existent trois facteurs principaux qui limitent la fréquence de l'horloge. Premièrement, nous avons le déphasage temporel (*clock skew*) de l'horloge non contrôlé. En second, nous avons l'échantillonnage des registres et en troisième nous avons le délai maximum de transmission à l'intérieur des portes logiques. Ces trois facteurs existent pour tous types de pipeline. Cependant voilà qu'une nouvelle étape doit être réalisée pour obtenir le maximum de vagues à l'intérieur de notre architecture lors d'une implantation pipeline par vagues. En équilibrant la latence de chaque chemin, nous optimisons la fréquence et l'ordre du pipeline.

Voici quelques problèmes statiques et dynamiques reliés à la limitation de fréquence:

1) Délai introduit par la dépendance des données au changement d'une porte logique.
2) L'effet des capacités de couplage.
3) Le bruit induit par la source de tension et variation de tension.
4) Variation dans les procédés de masque.
5) Délai introduit par une variation de température induite.

Comme premier point, la dépendance des données à l'intérieur d'une porte logique peut être résolue en utilisant une logique Pseudo-NMOS. Celle-là permet ainsi d'éliminer la dépendance des données au détriment d'une augmentation de puissance dissipée. Nous verrons plus loin quels sont les choix offerts à nous en vu d'une implantation architecturale utilisant le pipeline par vagues.

Les paramètres externes, comme le changement en température, la variation de tension ou le procédé de fabrication ont des équations sensiblement semblables.

Donc si d'après ((4.9)←(4.7)+(4.8))

$$T_{MAX} < NT_{CLK} < T_{MIN} + T_{CLK} \qquad \textbf{(4.10)←(4.9)}$$

Si nous ajoutons un facteur $\beta_S > 1$ pour des températures élevées, alors (4.10)←(4.9)←(4.7)+(4.8) → (4.11).

$$\beta_S T_{MAX} < NT_{CLK} < \beta_S T_{MIN} + T_{CLK} \quad (4.11)$$

Et pour des températures inférieures, ajoutons $\beta_F < 1$ alors (4.12) → (4.13)

$$\beta_F T_{MAX} < NT_{CLK} < \beta_F T_{MIN} + T_{CLK} \quad (4.12)$$

Ainsi en utilisant les équations (4.12), (4.13) et (4.14), nous obtenons

$$\beta_S T_{MAX} < NT_{CK} < \beta_F T_{MIN} + T_{CLK} \quad (4.13)$$

Si on isole T_{CK}

$$T_{CLK} > \beta_S T_{MAX} - \beta_F T_{MIN} \quad (4.14)$$

Finalement, on peut en déduire que la période de l'horloge est limitée par la différence de température maximum par délai maximum (T_{MAX}), contre la différence de température minimum et le délai minimum (T_{MIN}). Si nous comparons avec le pipeline conventionnel, la variation des paramètres externes sera toujours plus importante car le pipeline ne nécessite que l'optimisation des délais les plus élevés et en conséquence les variations de faible température peuvent être négligeables. Ceci dit, le pipeline par vagues est régit par

deux extrêmes (T_{max}, T_{min}). Même si leur différence était quasi nulle, la différence entre eux sera toujours dépendante des éléments externes. Le pipeline par vagues aura donc une dégradation plus rapide en performance comparativement au pipeline.

Une excellente façon de négliger les effets néfastes des paramètres externes est l'introduction de délai temporel sur l'horloge mère utilisé comme décalage de phase [GRAY93]. Cependant, les délais intentionnels limitent le nombre de vague pouvant être introduite. Ainsi, en variant le délai temporel de l'horloge de $\Delta \approx 0$ à T_{max} , on varie la latence qui sera égale au minimum le nombre de coup d'horloge nécessaire au transfert du nombre maximum de vague N_{max} pouvant être introduit jusqu'à $\Delta \approx NT_{max}$, c'est à dire la pente des données entre l'entrée et la sortie de chaque vagues variera avec Δ.

Où Wd, est l'ordre du pipeline par vagues à travers les données :

$$Wd = (k \times T_{clk} + \Delta) / T_{clk} \qquad (4.15)$$

Où Wc, est l'ordre du pipeline par vagues à travers l'horloge :

$$Wc = \Delta / T_{clk} \qquad (4.16)$$

Alors, la latence globale de l'horloge est :

$$k = Wd - Wc \qquad (4.17)$$

La figure suivante représente le diagramme spatial temporel du Δ en fonction de l'horloge mère. Où k, représente la latence globale de l'horloge. On peut observer, que l'ordre du pipeline est directement proportionnel à la latence globale.

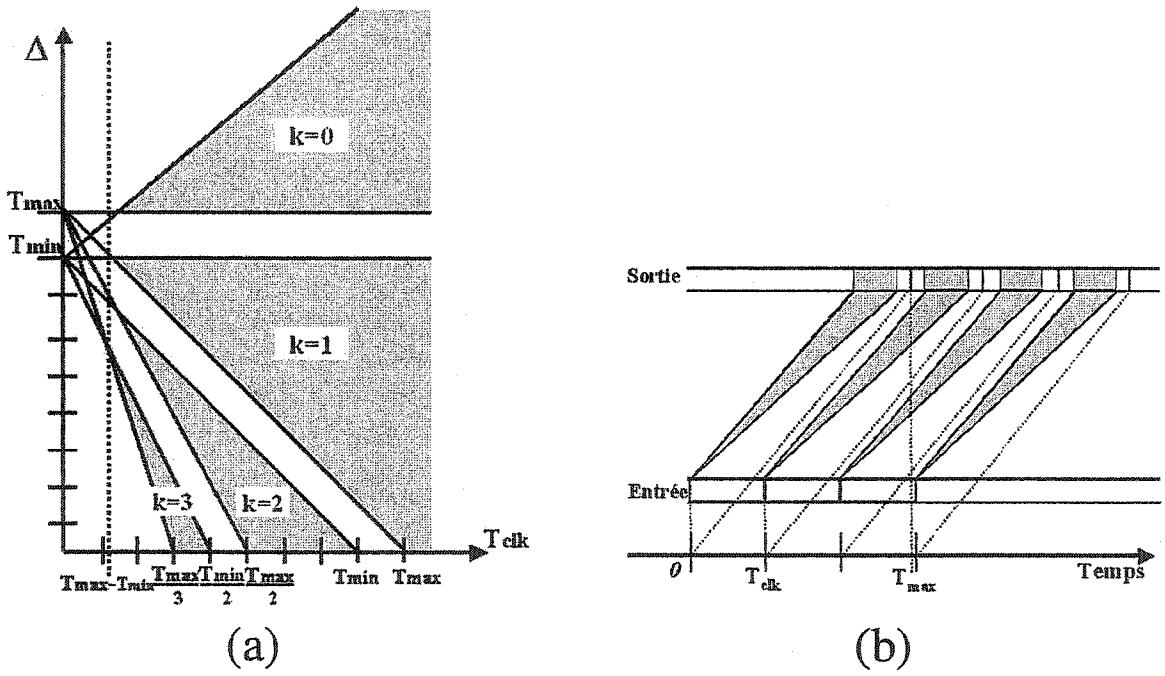


Figure 4.18) Schéma Spatial temporel d'une CLB pipeline par vagues.

(a) Spatial temporel en fct du Δclk, (b) Temporel en fct N vagues

4.3.3 Types de cellules disponibles

Pour ce qui est du choix du type de portes logiques qui peut être implanté, plusieurs choix s'offrent à nous. Comme par exemple, une logique CMOS (Complementary Metal

Oxide Semiconductor), CPL (Complementary Pass Logic), NPCPL (Normal Process Complementary Pass Logic) [GHO93], Domino, WTGL (Wave Pipelined Transmission Gate Logic) [SRI96], DPTL (Double Pass Transistor Logic) [PAR97].

Principalement, les premiers développements apportés à la technique du pipeline conventionnel et pipeline par vagues étaient une logique statique simple, où les procédés utilisent des portes statiques élémentaires de type CMOS comme l'inverseur ou NAND, etc. Le problème avec ces portes était d'abord l'utilisation d'une trop grande surface puisqu'ils utilisent une double logique (PMOS et NMOS). Ce qui provoque donc l'utilisation d'une double surface et d'une plus grande dissipation. En plus, puisque la mobilité des électrons à l'intérieur d'une diffusion P est inférieure au type N [SAV88], l'utilisation des transistors de type P a un délai supérieur au type N.

Dépendant du type de transition effectué à l'entrée d'une porte statique, les délais peuvent même être du double. Par exemple, les délais d'une porte Non-ET, la propagation d'un '1' par rapport à un '0' prend plus de temps que l'inverse [NOW95]. Comme nous le savons déjà, le degré du pipeline par vague est jugé en conséquence du minimum de différence entre le délai le plus long et le plus court de notre architecture. Cependant, pour obtenir le minimum il faut avoir une architecture utilisant des portes logiques élémentaires qui ne seront pas dépendantes des données qu'on leurs transmettent. Ceci nous permettra d'obtenir une structure ayant une zone de sécurité entre les vagues plus petite et ainsi la possibilité d'introduire un plus grand nombre de vagues sans obtenir de chevauchement

entre-elles. Puisque les portes élémentaires CMOS ont déjà une grande dépendance aux données introduites, il est difficile d'imaginer une conception utilisant le pipeline par vagues sans devoir calibrer chaque porte au niveau transistor (*fine tuning*). Un calibrage au niveau transistor est exécuté lorsque nous varions la longueur et largeur du canal du transitore pour obtenir un délai plus ou moins grand, équilibrant ainsi nos chemins. Il existe une autre méthode appelée ajustement grossier des chemins (*coarse tuning*). Il s'agit d'ajouter des délais temporels tout le long des chemins les plus courts pour obtenir le minimum de différence avec le plus long chemin.

Il faudra aussi prendre en considération la sensibilité aux paramètres externes comme la variation de température ou de la source de tension pour estimer quelle sera la variation du délai que notre porte CMOS aura. Dans [NOW95], une étude exhaustive a été faite sur la variabilité qu'une porte CMOS peut avoir dépendant des facteurs. Comme les données transmises, la température, la charge ou l'impédance de sortie qui peuvent à eux tous modifier son délai.

Cette discussion nous emmènes sur le sujet, "Quel type de portes logiques serait idéal à une implantation pipeline par vagues?". Premièrement, il est primordial d'avoir une famille de circuit logique n'ayant pas de dépendance avec le type d'entrée exercé. Ainsi, le temps qui devra être consacrée à équilibrer les chemins (*paths*) à l'intérieur de l'architecture sera beaucoup plus simple. Il existe principalement deux types d'ajustement ou d'équilibrage des chemins ou trajets à l'intérieur de l'architecture que nous avons discutée précédemment. Nous avons l'ajustement de précision au niveau des transistors (*fine tuning*

) et l'ajustement grossier ou l'ajustement des chemins par l'entremise d'insertion de portes logiques temporelles pouvant ainsi diminuer la différence entre Dmin et Dmax (*coarse tuning*).

Alors pour ce qui était du développement de la technique du pipeline par vagues, l'égalisation des chemins avait une limite interne puisqu'il devenait impossible d'ajuster le délai des portes statiques fournies par la librairie du fabricant. La recherche axée sur le développement d'une logique idéale à l'intégration au niveau pipeline par vagues introduit plusieurs types de logique permettant de concevoir une architecture utilisant des portes élémentaires. Celle retenue pour ce projet a été développée à partir d'une logique pseudo-NMOS et de portes de transmission. Ce choix fût motivé d'abord par l'économie de surface qu'une porte Pseudo-NMOS emmène comparativement à une logique CMOS, puisqu'elle n'utilise que des transistors de type N. En second, nous obtenons une consommation d'énergie plus faible puisque la logique Pseudo-NMOS exerce que la moitié des transitions normalement exercées par une logique utilisant deux fois plus de transistors. Et comme troisième argument, une porte qui n'utilise pas de transistor PMOS diminue sa latence. Cependant l'utilisation d'énergie statique consommée à travers une porte Pseudo-NMOS est un point néfaste. Voilà pourquoi l'utilisation de demi-porte de transmission (*Complementary Pass Transistor*) devient alors plus alléchante. La demi-porte de transmission consomme très peu de surface et d'énergie. Très flexible, elle permet d'être utilisée pour tout type de fonction logique. Par contre, elle supporte peu sa charge capacitive reliée aux grilles des transistors "*Cg*" et doit être généralement régénérée

dynamiquement et constamment. Son niveau de retenue est faible puisqu'elle n'est pas chargée statiquement comme une logique CMOS. Voilà pourquoi [GHO93-95] proposent une nouvelle sorte de demi-porte de transmission qu'ils nomment NPCPL (*Normal Process Complementary Pass Logic*)

4.3.4 Cellule NPCPL

La cellule NPCPL a pour avantage d'être très flexible, c'est-à-dire qu'elle permet d'exécuter toutes fonctions logiques de base, comme OU, NonOU, ET, NonET, OU-EX, NonOU-EX, Inverseur, sommation et calcul de retenue. L'utilisation de la cellule permet d'économiser la quantité de surface d'intégration ITGE. Si nous comparons l'apport de transistors utilisés en CMOS sur une porte OU-EX comparativement à la cellule NPCPL nous avons le double de surface consommée par la technologie CMOS, c'est-à-dire 16 transistors contre 8 pour la cellule NPCPL. Et puisque les opérations arithmétiques utilisent beaucoup de Ou-exclusif, nous économisons alors encore plus au niveau de la surface d'intégration.

Fonction	A_i	A_j	B_i	*Sortie (Q)*	*Sortie ($\overline{Q}$)*
ET/$\overline{\text{ET}}$	A	B	B	AB	$\overline{\text{AB}}$
OU/$\overline{\text{OU}}$	A	B	$\overline{\text{B}}$	A+B	$\overline{\text{A+B}}$
OU-EX/$\overline{\text{OU-EX}}$	A	$\overline{\text{A}}$	$\overline{\text{B}}$	A⊕B	$\overline{\text{A⊕B}}$
Retenue	C	B	A⊕B	Retenue	$\overline{\text{Retenue}}$
Somme	C	$\overline{\text{C}}$	A⊕B	A⊕B⊕C	$\overline{\text{A⊕B⊕C}}$
Tampon	B(t)	B(t)	B(t)	B(t+1)	$\overline{\text{B(t+1)}}$

Table 4.1) Fonction élémentaire de la cellule NPCPL.

Spécialement conçu pour l'implantation du pipeline par vague, la porte NPCPL fonctionne par l'entremise de son signal et de son complément. Ainsi, pour chaque signal fourni à son entrée, nous devons introduire son complément. Nous aurons alors en sortie le complément de l'opération booléenne. Ceci permet alors d'exécuter une inversion sans qu'il ne soit nécessaire d'introduire d'inverseur. Car l'inverseur fait partie intégrant de la structure NPCPL. En plus, le signal de sortie ne se dégrade pas aussi rapidement face à une trop grande capacitée de sortie et donc permet de fournir en courant un plus grand nombre de portes logiques comparativement aux portes de transmission, cependant beaucoup moins qu'une porte CMOS. En plus, l'introduction d'inverseurs CMOS permet de restaurer le niveau de tension en sortie. La figure suivante présente la cellule NPCPL. Chaque fonction booléenne a le même délai peut importe qu'on transmette un '1' ou '0', ce qui facilite énormément l'implantation pipeline par vagues. En plus, l'ajustement de précision n'est plus nécessaire, si nous choisissons comme structure principale la cellule NPCPL. Cependant, l'ajustement grossier constitue une nécessité puisque le choix de l'architecture détermine la quantité d'ajustement qu'il sera nécessaire d'accomplir.

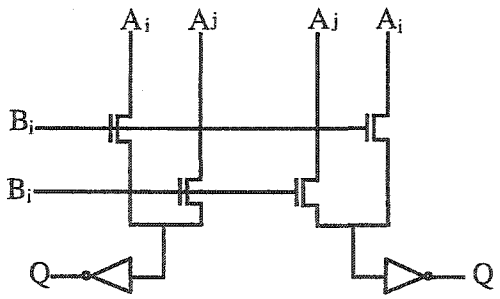


Figure 4.19) Cellule NPCPL.

D'autres types de cellules ont aussi été proposées, ses cellules utilisent des portes logiques dynamiques comme la porte Domino, TPL (*Tree Pass Transistor*) et la DPTL (*Differential Pass Transistor*). Cependant, ces types de portes ne permettent pas de concevoir tous les types de portes élémentaires comme la cellule NPCPL et donc elles ne permettent pas d'obtenir des délais équilibrés. D'autant plus que leur fonctionnement serait assez complexe. Par contre, une nouvelle cellule appelée « NPCPL modifiée » permet de corriger le problème de charge et de courant et peut donc transmettre fidèlement un très grand nombre de vagues à plusieurs sorties sans qu'il ne soit nécessaire d'introduire des étages d'amplification améliorant ainsi les transitions.

4.3.5 Structure ITGE proposée pour le pipeline par vagues.

Notre analyse sur une implantation dédiée au pipeline par vagues est généralement axée sur un circuit de type mono-harmonique. Ce qui par le fait même facilite de beaucoup les calculs présentés. Cependant, puisque nous devons implanter deux types d'architectures, l'un sera du type mono-harmonique simple où nous implantons un additionneur RCA et un multiplieur utilisant l'arbre CSA. En soi, sa plus grande complexité est le déphasage des données en insérant un délai triangulaire. Nous verrons au chapitre suivant les détails des architectures à implanter. Le second type est une architecture mono-harmonique avec boucle de retour, où nous implantons un multiplieur-accumulateur (MAC) dédié à une architecture systolique pour le traitement des signaux.

Après une recherche complète, aucun article ne fait mention d'une architecture de ce type. Un seul article implante une architecture qui pourrait être considérer, l'article [BUR94]. Cependant, utilisé pour la conception d'un filtre numérique adaptatif de type LMS, le type de MAC utilisé n'a aucune bouclee de retour puisqu'ils multiplient son signal d'entrée avec un poids pour ensuite l'additionner à un signal externe. Ceci dit, ce type d'architecture serait du type Poly-Harmonique au lieu de Mono-harmonique avec boucle de retour mémorisée (avec registre). Contrairement à cette architecture, le MAC voulant être implanté nécessite une retransmission des valeurs accumulées avec le résultat passé. Cette boucle de retour nécessite donc une excellente synchronisation entre le résultat présent et l'addition future. Ceci dit, pour compliquer la tâche le tout sera déphasé. Heureusement, notre MAC ne nécessite aucun ordre d'accumulation entre les données transmisses, ce qui nous permettra d'ajouter un certain nombre de délais temporels voulus à l'horloge qui synchronise la boucle de retour du MAC. Ce retour sera ensuite mémorisé à travers un registre triangulaire. Voilà pourquoi il est important de temporiser notre horloge spécialement lors d'architecture utilisant une boucle de retour.

Une architecture à boucle de retour est une architecture complexe à implanter. Elle ne peut fonctionner que sur une plage restreinte de fréquences à laquelle elle a été originalement destinée à fonctionner. Ainsi, nous devrons imposer une fréquence minimum et maximum pour un fonctionnement normal. Ce qui est très délicat, puisque plusieurs effets peuvent débalancer notre synchronisation à l'intérieur de la boucle de retour. L'unique méthode pour augmenter le débit est en augmentant k (la latence de l'horloge)

dans la boucle de retour, proposés par [GRA93], [NOW95b]. Ce qui peut ainsi limiter les fréquences d'opérations à une valeur plus faible que Tmax-Tmin.

L'équation suivante présente la période minimum pouvant être obtenue lors qu'une architecture comprend une boucle de retour.

$$T_{clk} \geq (k+1) \times (P_{max} - P_{min}) + k \times (P^{b}_{max} - P^{b}_{min}) \qquad (4.18)$$

Une architecture utilisant une boucle de retour sans registre intermédiaire en générale ne peut fonctionner lors d'une implantation pipeline par vagues. Car, d'après [NOW95b] la période minimale de l'horloge à l'intérieur d'une architecture avec boucle de retour n'utilisant aucun registre ne peut être prédéterminée. Deux cas peuvent permettre qu'une architecture fonctionne sans registre de retour. Dans le premier cas, aucune variation de délai entre les signaux à travers la boucle ne doit exister. En second, lorsque la latence k à travers la boucle devra être semblable pour toute information faisant partie de la boucle de retour.

Les points positifs sont :

1. Économie de surface d'intégration et de puissance consommée.
2. Pouvant atteindre de plus grande vitesse au niveau du débit de l'architecture
3. Innovateur et permettant d'être implanté avec le pipeline conventionnel pour optimiser le débit.

Les points négatifs sont

1. Implantation complexe nécessitant un équilibre parfait entre les délais.
2. Sensibilité au procédée de fabrication, au type de technologie et paramètre externe (Bruit induit par la source de tension, variation de la température, capacité induite).
3. Aucuns outils de conception ne sont disponibles au niveau commercial.

4.4 Choix de la technique la plus appropriée à l'égalisation

Parmi les trois techniques que nous venons de présenter, une sera choisie. Plusieurs objectifs doivent être respectés durant notre choix, comme la fréquence minimale de calculs lors d'un traitement de signaux numériques, le coût de fabrication, le projet doit être innovateur, la consommation de puissance doit être faible, etc. Pour notre projet, utilisant l'égalisation de canaux, l'unitée la plus importante et la plus utilisée est l'arithmétique. Ceci dit, à l'intérieur de notre architecture chaque bloc combinatoire arithmétique constituera le goulot d'étranglement. Il est donc primordial d'utiliser une implatation à laquelle nous aurons beaucoup plus de chance d'obtenir une rapidité de calcul accrue, tout en économisant la consommation de puissance utilisée. Le tout doit aussi être innovateur, et par le fait même permettre de relever des défis.

Le pipeline conventionnel serait une alternative simple, mais elle constitue très peu d'innovation puisqu'elle a été depuis longtemps apprivoisée par plusieurs chercheurs. En plus, le pipeline utilise énormément de surface et il nécessitera l'implantation d'un réseau en arbre de temporisateur pour l'horloge mère. Le pipeline asynchrone est une excellente

alternative, mais nécessite qu'on conçoive une architecture complexe purement analogue et spécifique. En plus, le micro-pipeline ne permet pas d'atteindre des vitesses très élevées mais amène plutôt une robustesse au design. Le pipeline par vague constitu la technique relevant le plus de défi. Elle offre de grandes possibilités puisqu'elle peut être implantée seule ou avec les deux autres méthodes, comme proposé par O. Hauck [HAU98]. En plus, nous proposons une cellule (NPCPL) permettant d'énormes avantages.Offrant une logique booléenne de base et aussi propage le même délai de transmission, optimisant ainsi le rapport ($T_{max} - T_{min}$) et donc le degré du pipeline par vagues. Il faut cependant mentionner que l'utilisation de cellule NPCPL n'est malheureusement pas préférable pour n'importe quelle type d'implantation. Très peu robuste au bruit et ayant une grande zone grise de saturation, elle n'est pas conçue pour tous les types de circuits, contrairement au logique booléenne préconçu de type CMOS. En plus, la cellule NPCPL a un faible niveau de charge et nécessite alors quelle sois recordé à une entrée de logique P et N ce qui la rend très encombrante. Mais elle est parfaitement adaptée au pipeline par vagues car l'important pour nous est d'obtenir des délais équilibrés. Finalement, le pipeline à débits maximum (pipeline par vagues) offre une économie de surface d'intégration substantielle puisque nous n'utilisons très peu de registres.

4.5 Conclusion

Le pipeline par vagues est l'une des méthodes utilisées pour maximiser le débit d'une

architecture ITGE (Intégration à Très Grande Échelle). La particularité de cette méthode est qu'elle touche l'analyse et la conception sur plusieurs niveaux (procédé, dessin de masque, circuit, logique (temporel et architectural)). Ainsi, nous avons vu pourquoi nous choisissons cette méthode pour notre conception d'unités arithmétiques à haut débit implantée plus particulièrement à l'intérieur d'architectures effectuant l'égalisation de canaux numériques.

Ce chapitre avait comme objectif de présenter trois grands principes ITGE améliorant le débit d'une architecture. À la section 4.1 nous présentions la technique du pipeline synchrone. La méthode d'optimisation du débit la plus utilisée a été présentée incluant ses avantages et ses inconvénients. Par la suite, nous avons présenté à la section 4.2, le pipeline asynchrone. Très pratique pour des architectures de grande surface. Elle permet de simplifier la transmission d'horloges principales à tout le circuit sans introduire de déphasage. Ceci permet de modifier la structure en créant plusieurs régions ayant leurs propres communications locales. Le micro-pipeline (pipeline asynchrone) peut aussi simplifier la communication entre deux domaines régie par deux horloges de différente fréquence ou de phase. Finalement à la section 4.3, nous présentions brièvement la technique du pipeline par vagues. Dans cette dernière section, quelques équations fondamentales ont été présentées décrivant quel est le nombre maximum de vagues pouvant être inséré, et quelle est la période minimale pouvant être atteinte. Dans cette présente section, nous discutions des différentes architectures et leurs concepts vis-à-vis du pipeline par vagues (mono-harmonique, poly-harmonique, boucle de retour avec ou sans registre,

etc.). Pour ensuite, suivre avec quel type de logique devrait-on utiliser. Finalement, nous terminions en incluant comme dernière section (section 4.4), le choix de la technique la plus appropriée à l'égalisation. Puisque nous utiliserons plusieurs différentes architectures arithmétiques, les résultats de cette technique seront vus au chapitre suivant.

5 Réalisation d'U.A. pipeline par vagues

Ce chapitre portera sur la conception et l'intégration d'unités combinatoires sur CMOS, permettant d'évaluer les performances du pipeline par vagues comme méthode d'optimisation du débit. La conception d'une toute nouvelle logique sera développée pour faciliter l'implantation du pipeline par vagues en vue d'obtenir des performances optimales. Ce chapitre présente les résultats de cette logique qui est basée sur la cellule NPCPL (*Normal process complementary pass transistor logic*). Ainsi, la conception d'une famille Booléenne complète (ET, Non-ET, OU, Non-OU, XOR, Non-XOR, Temporisateur, Bascule-J) sera conçue et utilisée comme élément de base aux unités arithmétiques. Cette étude portera principalement sur les effets néfastes de la tension et de la température sur notre logique NPCPL versus CMOS. Ce qui confirmera les bienfaits d'utiliser la cellule NPCPL, au lieu de la porte logique standard CMOS, en vue d'une implantation axée sur le pipeline par vagues. Par la suite, nous présenterons les résultats et les dessins du Multiplieur et du MAC conçus et fabriqués par la CMC (*Canadian Micro-Electronic Corporation*), ainsi que les résultats synthétiques obtenus en utilisant P-Spice (pré-layout et post-layout).

Ce projet a été réalisé grâce à la participation de la CMC, et a permis la création de

deux circuits sur silicium, d'un multiplieur 8x8 bits et d'un Multipleur-Accumulateur 8x8 bits. Les deux ont été fabriqués en utilisant une technologie de 0.5µm CMOS.

5.1 Intégration de la cellule NPCPL

La cellule NPCPL vue au chapitre subséquent a été implantée utilisant comme outils CAD, les logiciels de CADENCE® (Analog Artist®, Virtuoso®). Utilisant d'abord l'outil Analog Artist®, nous avons simulé, en utilisant l'interface P-Spice, notre cellule sous plusieurs configurations évaluant ainsi la longueur (L) et la largeur (W) optimale de chaque transistors. Ainsi, les transistors utilisés comme demi-porte de transmission ont comme dimension L/W= (0.6/2.5). Alors que les inverseurs utilisés en sortie de charges ont un rapport L/W = (0.6/10). Ce qui permet de fournir assez de courant à un certain nombre de portes en avale. Ce nombre a été déterminé à quatre portes NPCPL maximum. Ainsi chaque logique combinatoire devra nécessairement être équilibrée en fonction d'un nombre maximum de portes logiques.

La Figure 5.1.a présente la structure schématique P-Spice (ou H-Spice sur UNIX) créée pour la cellule NPCPL. Après avoir simulé la version pré-layout (version P-Spice sans capacité parasite), nous avons conçu chaque fonction booléenne élémentaire basée sur le tableau 5.1. Ce tableau présente les configurations que nous devons y apporter pour obtenir une opération booléenne spécifique [GHO95]. Le temps de propagation en moyenne est

aussi présenté. Comme dernière étape nous avons généré les masques de chaque cellule conformes aux règles de dessin fournies par le fabriquant de la technologie, c'est-à-dire Hewlett-Packard. La conception de chaque cellule a été optimisée au niveau de la surface et de la rapidité d'où la largeur λ est de 0.6µm (suggéré par la CMC). Puisque la technologie n'avait pas encore atteint une maturité au moment de la fabrication, nous devions respecter les règles conseillées.

5.1.1 Dimension de la cellule

La cellule NPCPL, a une dimension physique de : 21.6x22.8 μm^2.

La Cellule (figure 5.1.a) comporte une rampe supérieure VDD et inférieure VSS. Les inverseurs de sortie sont de chaque côté. Ainsi, nous pouvons observer qu'il y a deux transistors de type P situés sur chaque coin supérieur et deux transistors de type N aux coins inférieurs. Ensuite, le centre inclus nos quatre transistors de type N utilisés comme demi-porte de transmission. Le tableau 5.1, présente en détail comment chaque entrée devra être configurée pour obtenir en avale la fonction désirée.

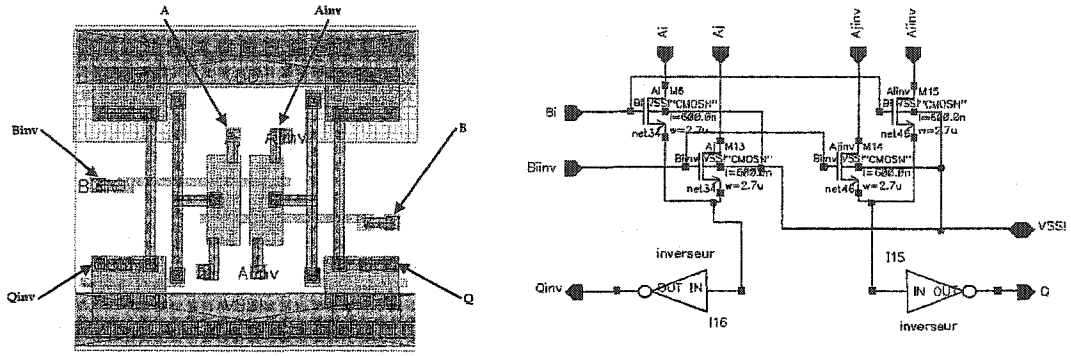


Figure 5.1) Dessin de masque d'une cellule NPCPL et Schéma P-Spice®.

5.1.2 Résultats de simulation

Nous avons vérifié l'intégrité de chaque fonction conçue par la cellule NPCPL. Pour un procédé normal et fonctionnant à température ambiante (27°C et 3.3 Volts) tous on obtenu les même délais de propagation permettant ainsi d'obtenir des pipelines virtuels semblables et de simplifier l'implantation du pipeline par vague puisque la propagation de chaque signal interne aura, si bien équilibré, un temps moyen semblable.

Une comparaison entre la cellule NPCPL et CMOS a été effectuée. Nous avons décidée de présenter une comparaison de deux types de portes logiques. La première est la porte Non-ET. Elle est équivalente à un grand nombre de portes utilisant 4 transistors, et elle est utilisée comme élément de base pour mesurer en portes logiques la surface et la puissance que le circuit utilisera. Comme par exemple, le logiciel de synthèse automatique conçu par la compagnie Synopsys®. Nous utilisons donc la porte Non-ET comme source de

comparaison entre CMOS et NPCPL. En seconde partie, nous présentons les résultats de comparaison d'une porte qui présente des caractéristiques qui permettent de démarquer notre cellule (pour ce qui est de propagation de délai), puisqu'une porte OU-Exclusif standard CMOS utilise en générale 4 portes Non-ET, c'est-à-dire 16 transistors comparativement à 8 transistors pour la NPCPL. Nous pourrions considérer la même analogie si nous comparons un registre CMOS à un registre que nous avons conçu utilisant la cellule NPCPL.

Fonction	A_i	A_j	B_i	*Sortie (Q)*	*Sortie ($\overline{Q}$)*	Délai(0.5µm)
ET/$\overline{\text{ET}}$	A	B	B	AB	$\overline{\text{AB}}$	0.3 ns
OU/$\overline{\text{OU}}$	A	B	$\overline{\text{B}}$	A+B	$\overline{\text{A+B}}$	0.3 ns
OU-EX/$\overline{\text{OU-EX}}$	A	$\overline{\text{A}}$	$\overline{\text{B}}$	A⊕B	$\overline{\text{A⊕B}}$	0.3 ns
Retenue	C	B	A⊕B	Retenue	$\overline{\text{Retenue}}$	0.3 ns
Somme	C	$\overline{\text{C}}$	A⊕B	A⊕B⊕C	$\overline{\text{A⊕B⊕C}}$	0.3 ns
Tampon	B(t)	B(t)	B(t)	B(t+1)	$\overline{\text{B(t+1)}}$	0.3 ns
Bascule D	D	Q	Clk	Q	$\overline{\text{Q}}$	0.3 ns

Table 5.1) Résultat obtenu à Température ambiante et Procédé Normal.

Notre première comparaison est le délai de propagation d'une porte Non-ET en fonction de la température. Une porte Non-ET régulière à 2 fois moins de transistors (CMOS=4, NPCL=8). Cependant, les délais de propagation suivent la même tendance. La seule grande comparaison serait la dépendance des données. Cette dépendance est le temps qu'un '1' traverse notre porte comparativement à la transmission d'un '0', ce qui est assez élevée du côté CMOS. En conséquence, nos étages pipelines par vagues auraient été débalancés si l'usage de portes CMOS auraient été utilisées. On peut aussi remarquer

l'écart de dépendance d'une porte CMOS avec l'augmentation de la température.

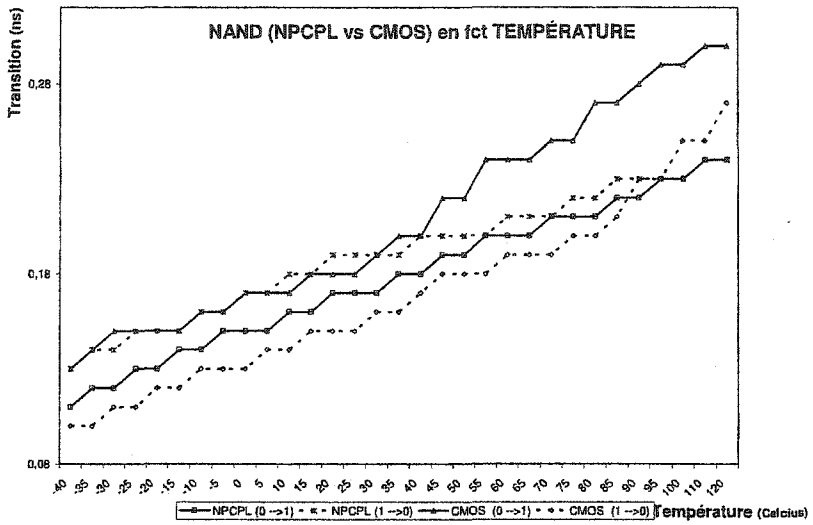


Figure 5.2) Propagation en fonction de la température (Non-ET).

La figure ci-dessous présente les résultats lors d'une variation de tension. On peut observer deux choses. La première est la sensibilité du procédé en ce concerne la cellule NPCPL. C'est-à-dire qu'on observe que les transitions ne suivent pas la même courbe. Cela entraîne des équidistances entre le signal au temps de propagation le plus long versus le signal ayant le temps de propagation le plus court comparativement au CMOS, où leurs courbes suivent au moins la même tangente à faible tension. La deuxième, nous remarquons une certaine rigueur de la part de la porte CMOS, elle apparaît très fragile en haute tension. Comparativement à la cellule NPCPL, où son procédé aurait en effet le même problème qu'il soit en faible et haute tension.

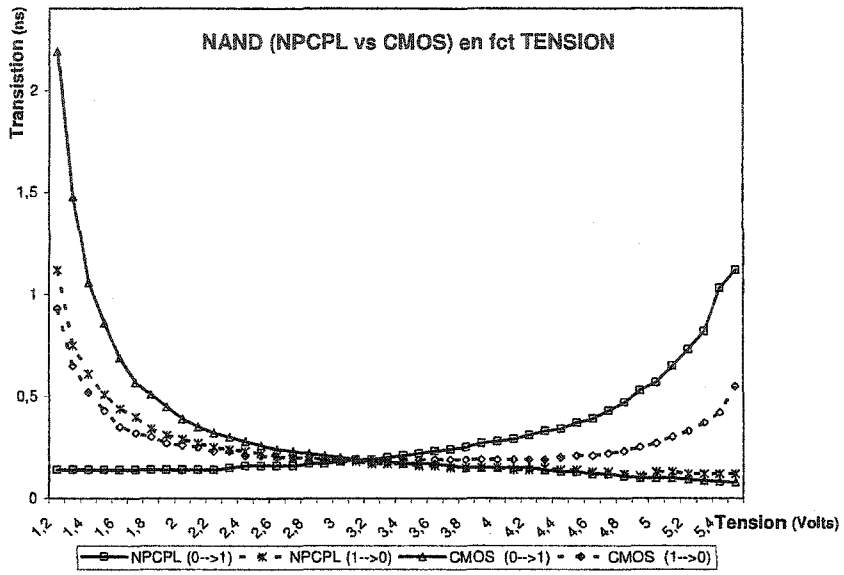


Figure 5.3) Propagation en fonction de la tension (Non-ET).

La figure ci-dessous représente la dépendance des données face à la température lors d'une comparaison entre NPCPL et CMOS pour notre porte logique OU-EX. À première vue, la figure représente assez bien les qualités qu'une cellule NPCPL fournie à l'intégration du pipeline par vagues. D'abord, nous avons la vitesse de propagation qui est de beaucoup supérieur à une porte standard CMOS. En second, nous avons la dépendance des données qui permet l'intégration simplifiée du pipeline par vagues. Comme dernier élément, la surface d'intégration est d'autant plus optimale puisqu'elle n'utilise que 8 transistors comparativement à 16 transistors en CMOS. Nous pouvons respecter la même analogie pour ce qui est du registre NPCPL.

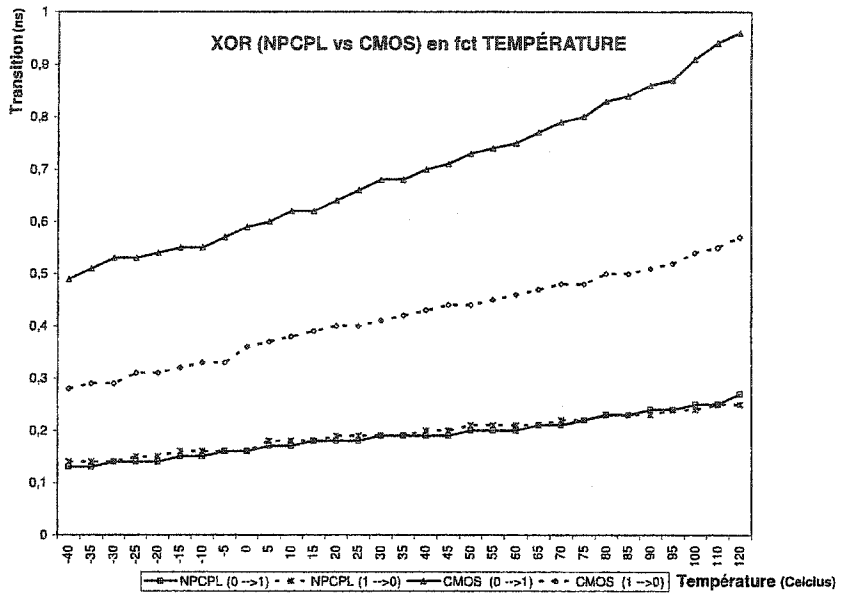


Figure 5.4) Propagation en fonction de la température (OU-EX).

La propagation en fonction de la tension emmène encore aux mêmes conclusions, c'est-à-dire que la cellule NPCPL (OU-EX) présente d'excellent résultats lorsque notre circuit fonctionne dans une plage de tension de faible à normal, où une tension de 3.3 volts serait la tension approuvée par le fabricant de la technologie.

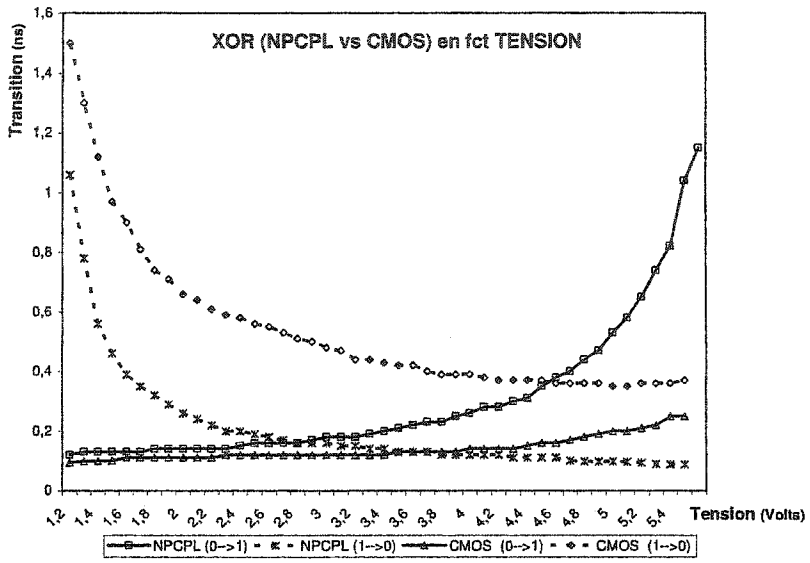


Figure 5.5) Propagation en fonction de la tension (OU-EX).

Nos résultats démontrent que notre cellule NPCPL apporte d'excellents résultats. L'utilisation d'une logique qui simplifie l'implantation et améliore la quantité de surface et de puissance utilisée, permet d'obtenir des gains substantiels important à une intégration ITGE. En plus de fournir une logique double, c'est-à-dire quelle fourni l'inverse (en tout temps) de sa fonction (ET & Non-ET, etc...), elle permet d'obtenir des fonctions plus complexe, ainsi d'excellent gain en vitesse. Notre seul point néfaste serait le faible gain de courant en sortie. Ceci limite la quantité de portes logiques qui sera utilisée en parallèle, puisque la porte en amont ne peut fournir de courant qu'à une quantité limitée de portes NPCPL. Ce résultat aurait dû faire part de notre étude. C'est-à-dire de comparer notre pente en tension en sortie en fonction de la surcharge. En conclusion, la suite de notre

recherche portera sur d'autres alternatives ou d'autres modifications pouvant améliorer la charge qu'une porte NPCPL fournira. L'article [CHO96] en est un excellent exemple de cellule NPCPL modifiée.

5.2 Pipeline par vagues de l'additionneur et du multiplieur

L'objectif principal de ce projet a été l'implantation de structures arithmétiques permettant le traitement rapide de signaux de communication lors d'une transmission à haut débit. Ceci dit, deux éléments sont en partie responsables du goulot d'étranglement dans toute architecture et limitent également la vitesse de notre architecture. Le premier serait tous les types d'architectures arithmétiques, en particulier les multiplieurs et MAC. En second, serait les RAM ou mémoire dynamique (Read-Access Memory).

Ainsi, notre objectif pour cette section est de présenter nos travaux en ce qui concerne le développement arithmétique. L'étude des mémoires ne sera pas discutée. Il existe un troisième facteur aussi non mentionné, qui ne fait pas partie d'un élément architectural mais bien d'un élément physique, ce sont les circuits d'entrée/sortie (*I/O Pads*). Ils sont généralement de grandes charges capacitives. Nous discuterons plus loin des limites en fréquences créées par ces éléments.

L'importance d'une technique nouvelle en arithmétique permettant de très haut débit constitue le berceau de nos recherches. Dans un premier temps, nous présentons l'additionneur et notre multiplieur développés à partir de cellule NPCPL. Tous deux

pouvant servir comme élément de calcul à eux seuls. Ils seront donc jumelés à la section suivante pour le développement d'une nouvelle structure multiplieur-accumulation (MAC) proposée.

5.2.1 Description

Au chapitre 3.4, nous avons discuté du choix de l'additionneur et multiplieur utilisé pour notre implantation pipeline par vagues. Notre choix a été déterminé par plusieurs facteurs, l'un d'eux fût la simplicité d'implantation. Voilà ce qui nous emmène à implanter un additionneur qui pourra être utilisé seul et être en même temps compatible à notre Multiplicateur-Accumulateur. Nous avons donc choisi l'additionneur à propagation de la retenue ou RCA (*Ripple carry adder*) comme unité à implanter sous forme ITGE (*intégration à très grande échelle*).

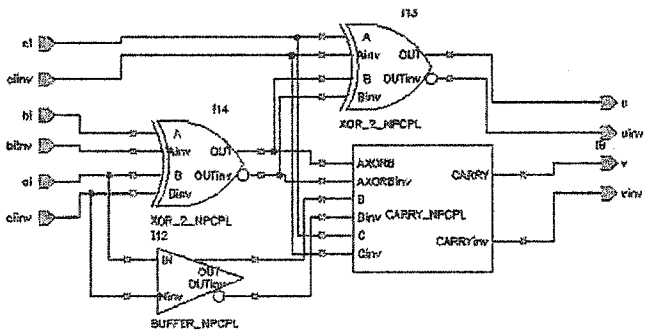


Figure 5.6) Dessin de l'additionneur Entier (FA).

Puisque l'additionneur RCA est conçu uniquement d'additions entières (FA), nous

avons conçu notre propre macro FA. Une macro est un dessin masque principalement conçu pour une opération spécifique, limitant la quantité de surface à utiliser et permettant de meilleurs résultats au point de vue de la capacité parasite et autres procédés intrinsèques. Cette macro a donc été complètement dessinée manuellement. Elle n'a pas été dessinée automatiquement par l'outil CAO. Permettant ainsi d'améliorer les délais de propagation et d'obtenir des résultats de simulations plus précis. La figure 5.6, présent la structure FA dessinée à l'aide Analog Artist®. Le chapitre 3 explique en détail le fonctionnement de cette cellule. Notre macro conçu en utilisant Virtuoso® est présenté par la figure suivante. Précisons, que la cellule FA a été utilisée dans la construction de notre additionneur RCA. Nous avons simplement relié un à un chaque FA, propageant donc la retenue à son prochain.

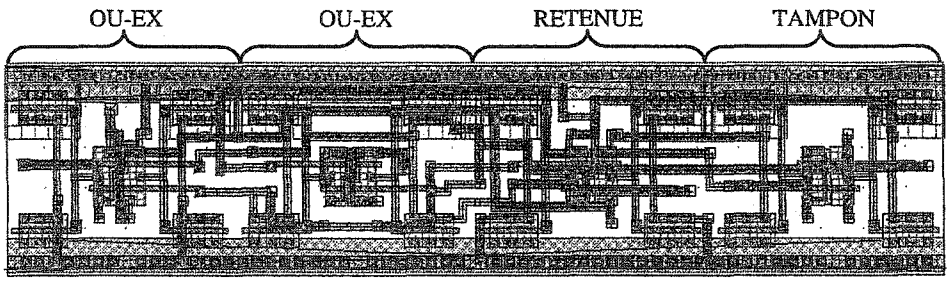


Figure 5.7) Dessin de masque de notre addition entière.

La figure 5.8, représente notre additionneur à propagation de la retenue pour une addition 8-bits. L'additionneur est finalement inclus à l'intérieur de notre multiplieur 8x8 qui comprend un arbre d'additions partielles CSA+ et termine en sommant la multiplication partielle en utilisant notre additionneur RCA (figure 5.9).

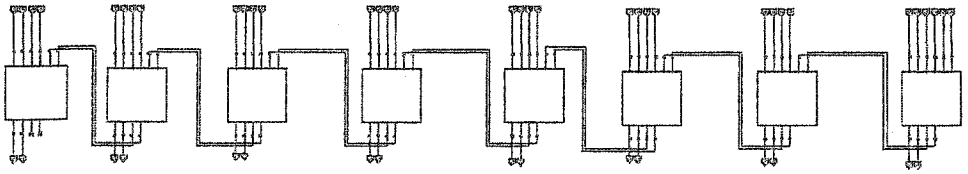


Figure 5.8) Schématique de l'additionneur RCA.

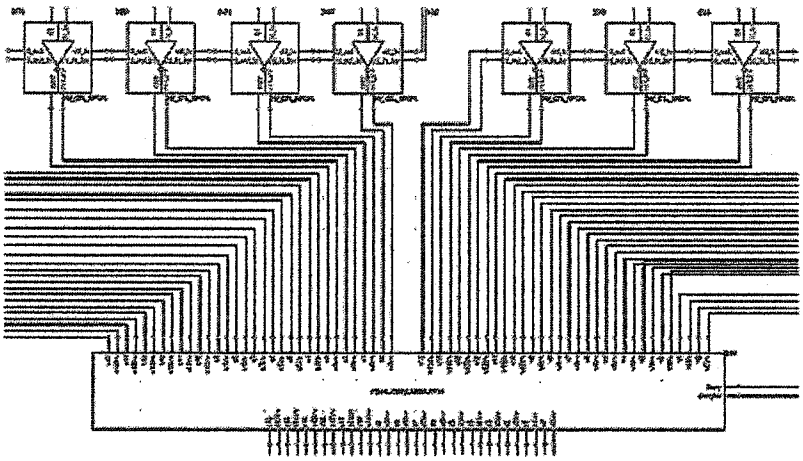


Figure 5.9) L'additionneur RCA à l'intérieur de notre multiplieur (8x8).

Après la conception et la vérification de notre additionneur à propagation de la retenue, l'arbre CSA (*Carry Save Adder*) se devait de respecter les mêmes étapes. Ceci a donc nécessité le développement d'une cellule CSA+ présenté par Ghosh et Nandy dans [GHO93, 94, 95] dans le cadre de leur recherche sur l'utilisation de la cellule NPCPL. Cette unitée a été développée sous forme macro pour simplifier notre conception. Le dessin de masque est lui aussi optimisé en fonction d'une réduction de surface. Environ 70% de notre logique repose sur ce bloc. Comme présenté au chapitre 3, la cellule a été

conçue à partir de portes NPCPL. Ceci permet donc d'obtenir une logique double et une propagation indépendante aux données transmises.

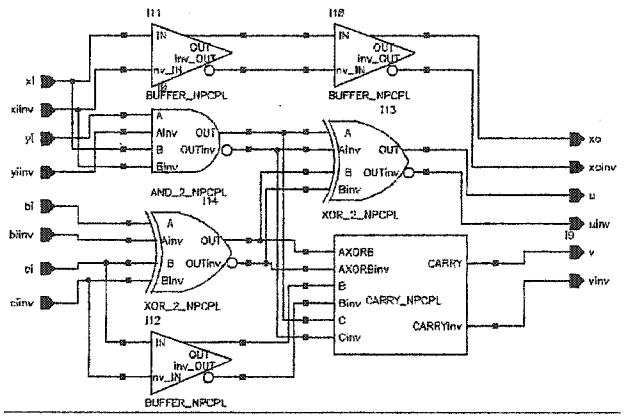


Figure 5.10) Structure CSA+ Dessiné dans Analog Artist®.

La cellule a été conçue pour minimiser l'impédance en amont, car la pluspart des impédances d'entrée de nos portes configurées affrontent l'entrée drain d'un transistor de type N. La figure 5.11 (Dessin de masque CSA+) a été conçu de la même façon que la porte FA.

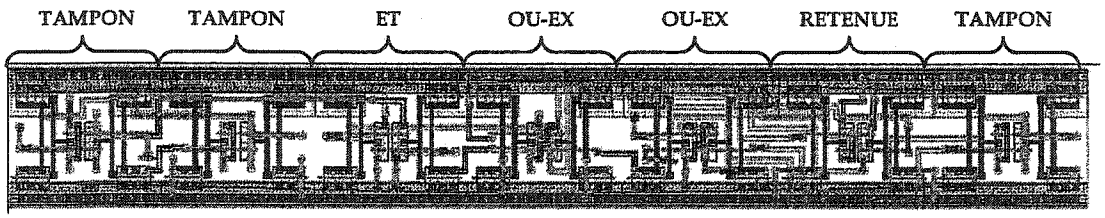


Figure 5.11) Dessin de masque du CSA+ conçu dans Virtuoso®.

La figure 5.12, présente la structure complète de notre multiplieur 8x8 bits de format

complément à 1. En sortie, nous avons les triangles de temporisation permettant de transmettre chaque bit l'un à la suite de l'autre. Ceci est nécessaire car l'additionneur à propagation de la retenue doit recevoir le bit le moins significatif en premier pour générer la retenue transmissible au bit prochain.

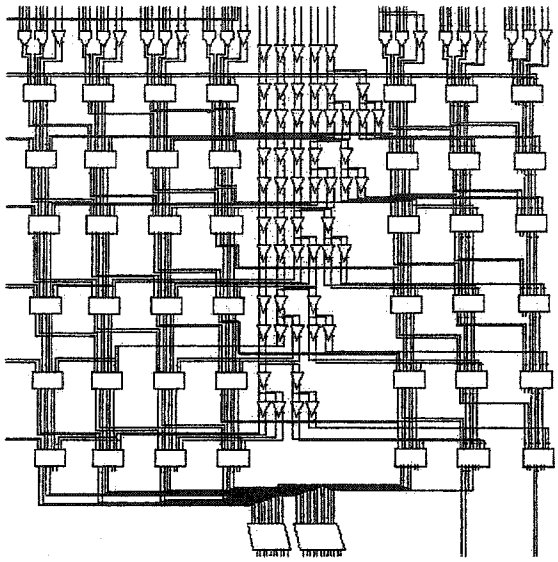


Figure 5.12) L'arbre CSA+ et Triangle temporisateur de notre Mult 8x8.

Un rappel important, notre multiplieur fonctionne sur le format complément à 1, ce qui explique le choix de notre multiplieur en arbre CSA+ qui sera implanté à l'intérieure de nos processeurs élémentaires. Cependant, une étude plus poussée sur le multiplieur vectorielle de format complément à deux de Baugh et Wooley [ZAR95] a été fais intégrant la technique du Pipeline par vagues. Celui-ci étant encore dans un stage très précoce, il a été conçu en format 4x4 bits et il a atteint une fréquence maximum de 950 MHz. Cependant, le même multiplieur fabriqué ensuite sous format 8x8 bits rencontre une limite

fonctionnelle (déjà mentionné auparavant) limitant la charge pouvant être fournie par nos cellules NPCPL avec une capacité limitée en charge de 4 cellules en avale. Le taux en tension est insuffisamment élevé (min. Vdd/2 + α) pour obtenir un seuil logique pouvant être interprété comme logique CMOS. D'après l'architecture Baugh et Wooley, nous observons un trop grand nombre de portes devant être fournies en parallèle lorsque l'architecture dépasse une certaine dimension. Dans notre cas, c'est une dimension 8x8. La prochaine étape de notre étude sera d'utiliser la cellule NPCPL modifiée proposée par [CHO96]. Spécialement conçu pour contrer le problème de surcharge, elle propose simplement l'ajout d'un second inverseur de plus grande dimension, rajouté en sortie de chaque côté. Ceci permettra de contrer les effets néfastes d'une structure à base de demi-porte de transmission (*Pass Transistor Logic*). Malheureusement, elle ne fera pas l'objet de recherche plus approfondie pour ce qui est de ce mémoire.

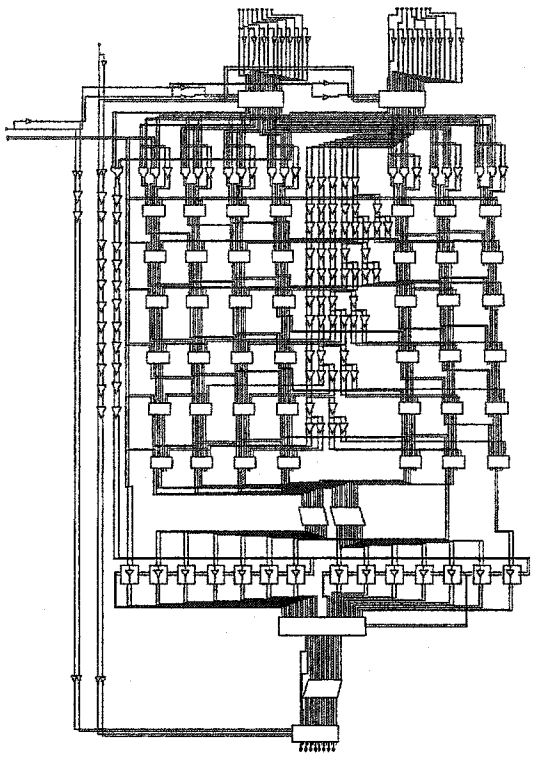


Figure 5.13) Schématique du multiplieur (8x8).

5.2.2 Résultats de simulation

Les résultats suivants présentent le comportement de notre multiplieur en fonction de plusieurs procédés. Ils démontrent bien quelles sont les faiblesses et qualités obtenues lors d'une implantation. Plusieurs recherches et articles de journaux présentent sensiblement les mêmes résultats. Par exemple, K. J. Nowka dans [NOW94], [NOW95b] présente les faiblesses de ce type de pipeline. Cependant, il utilise des portes logiques CMOS comme implantation principale. Comme on le sait bien, les portes CMOS on de très grands défauts

par rapport à la dépendance des données. Cependant, ils sont très robustes aux procédés et ils permettent d'obtenir une plus grande charge de sortie. Les mêmes conclusions ont été présentées par [GRA93] lors de procédés de température et de tension.

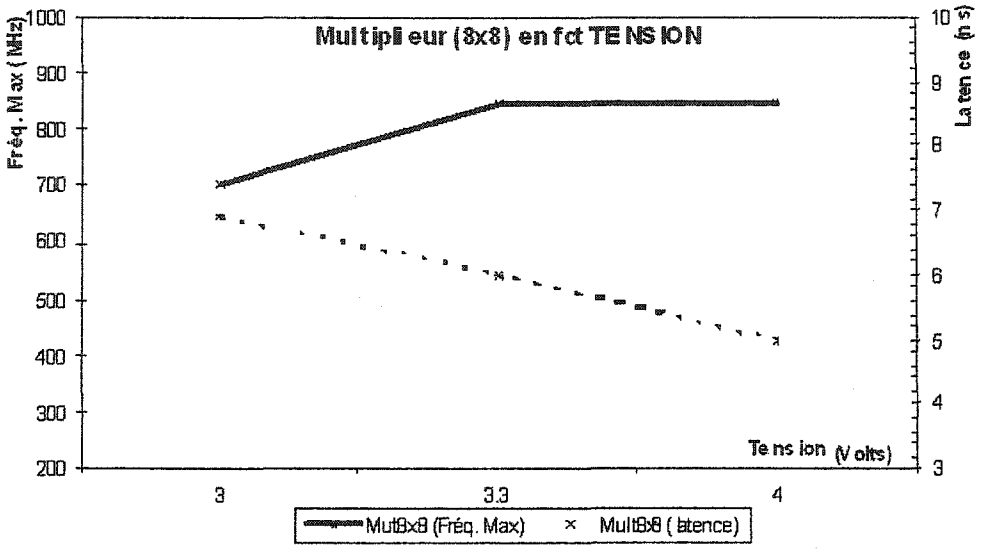


Figure 5.14) Multiplieur 8x8 en fonction de la tension.

Le multiplieur pipeline par vagues fabriqué à partir de cellule NPCPL présente (figure 5.14) des résultats assez impressionnants et plus robustes que prévu. Nous obtenons un plateau en fréquence entre 3.3 à 4.5 volts. Ainsi la latence obtenue est entre 4.5 à 6 ns pour une fréquence maximum de 850 MHz à 25°C.

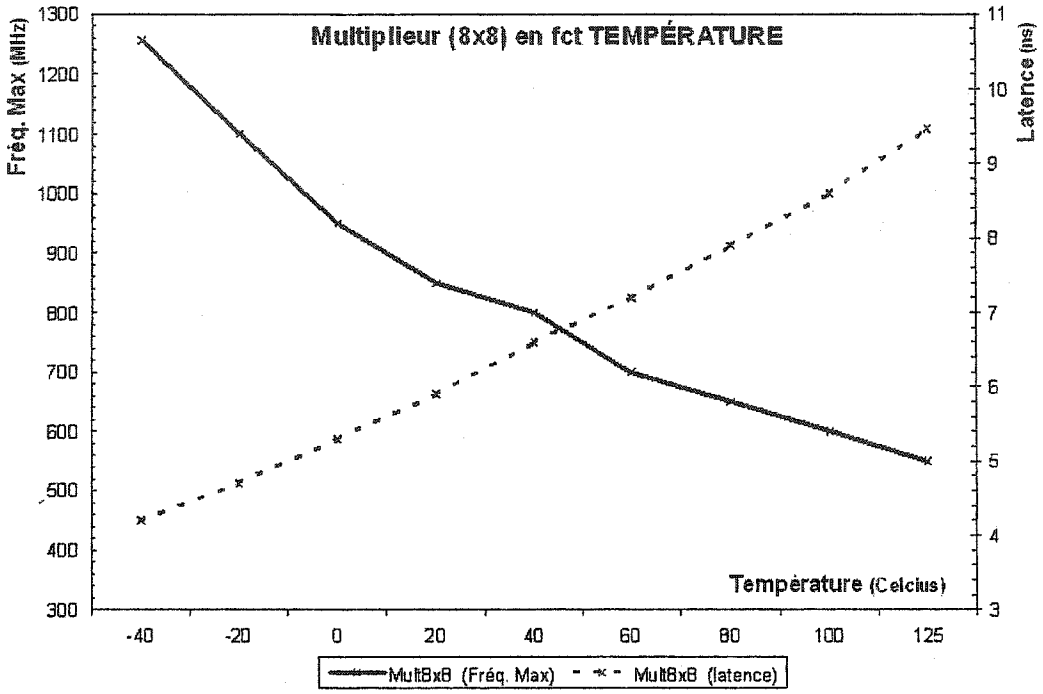


Figure 5.15) Multiplieur 8x8 en fonction de la température.

La figure 5.15, présente comment notre multiplieur réagit aux variations de température. Beaucoup moins robuste à la température, il présente donc des signes de faiblesse. On peut observer que la mobilité des électrons (courant) joue un rôle crucial lorsque la température varie radicalement. Améliorer le courant de charge de la cellule permettrait d'être plus robuste au changement de la température. Ainsi, la cellule NPCPL modifiée présentée dans [CHO96] pourrait servir à améliorer notre circuit sous une variation de la température. Comme seconde courbe, nous avons la latence. Avec sa courbe quasi-linéaire, elle permet d'observer la même conclusion. Une variation énorme

d'environ 5ns a été observée entre le procédé le plus rapide (-40°C) et le procédé le plus lent (125°C) à 3.3 Volts.

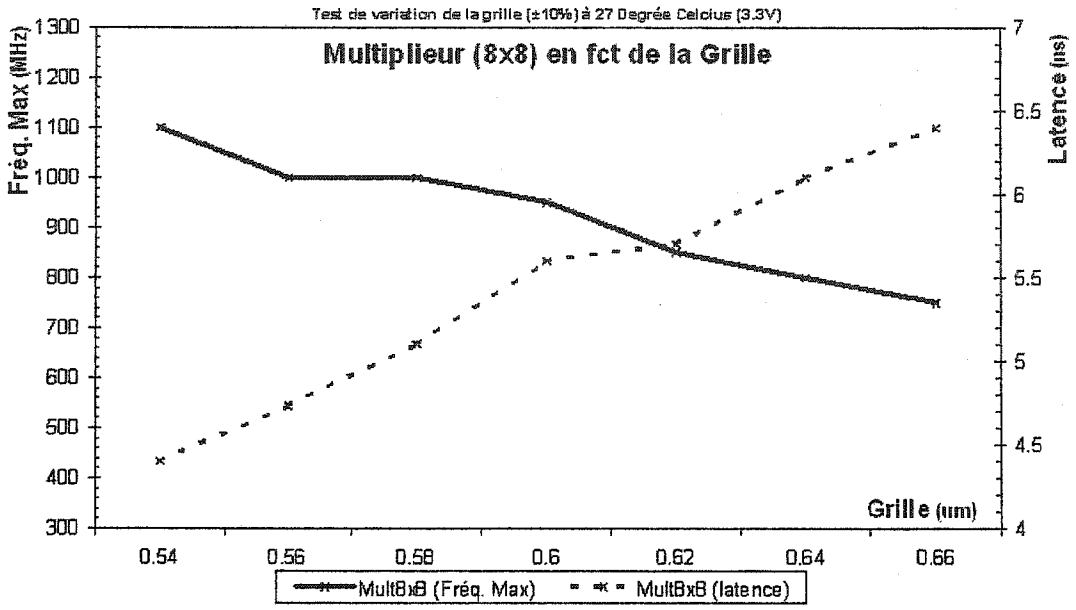


Figure 5.16) Multiplieur 8x8 en fonction de la grille.

Comme dernier test, nous avons décidé de varier d'environs 10% la dimension de notre grille. Cependant, même si notre procédé a été implanté en utilisant une technologie 0.5µm, la dimension de la grille suggérée par le fabricant est de 0.6µm. La fréquence maximum aurait facilement atteint les GigaHertz si nous avions conçu les grilles à 0.5µm. En bref, les résultats observés ont le même comportement lorsque nous varions la grille de nos transistors vis-à-vis la variation en température.

La figure 5.17 présentée ci-dessous démontre l'exactitude des résultats obtenus par

notre multiplicateur. L'architecture fonctionne sous la forme module et signe. Donc, le bit le plus significatif représente le signe de la multiplication. La figure ci-dessous présente trois sections. La section a est la multiplication de 3x3=9, la section b représente 11x4=44 et la section c représente une multiplication 29x1=29.

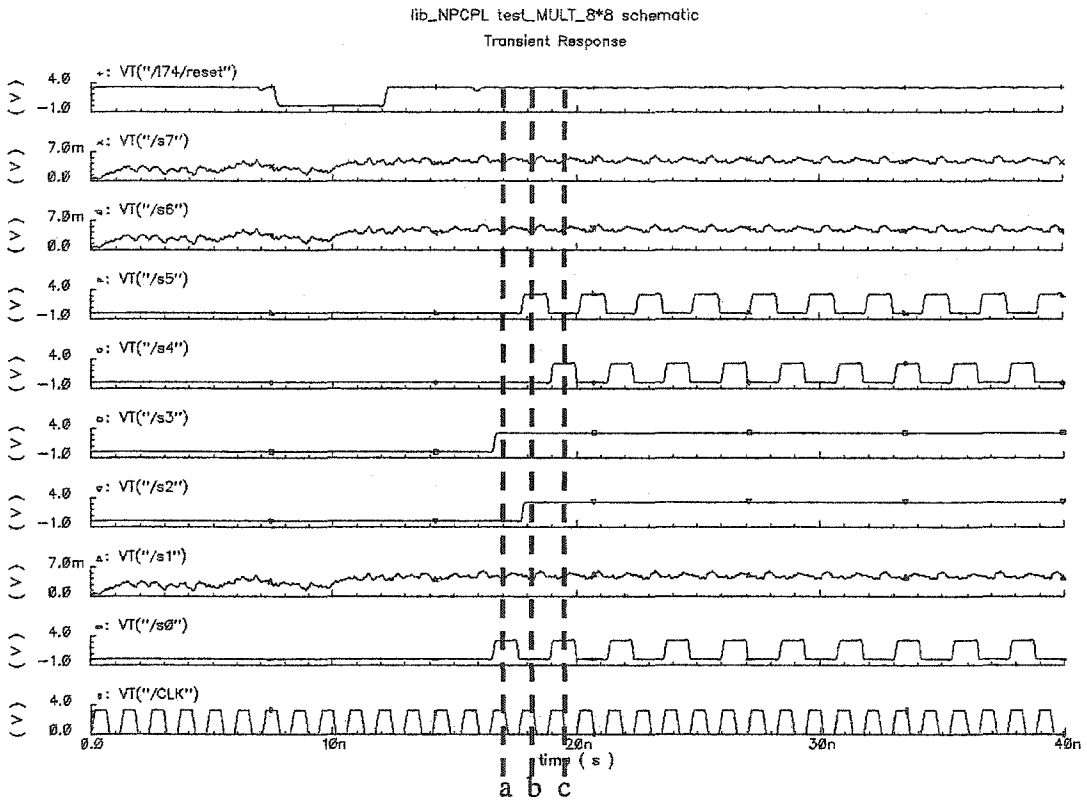
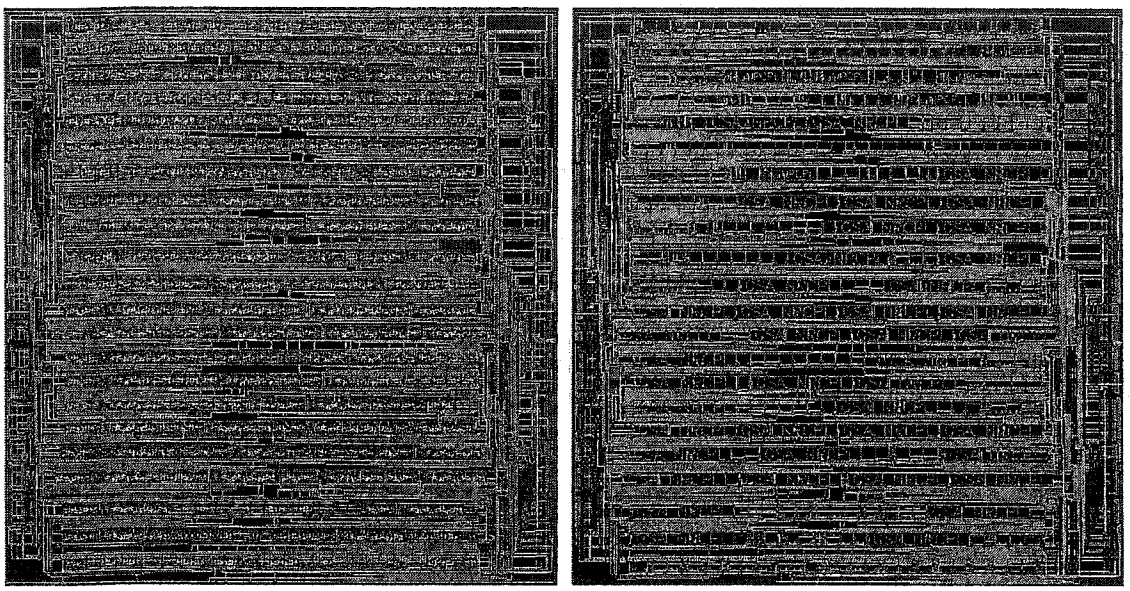


Figure 5.17) Valeur en sortie du Multiplicateur (8x8).

Le multiplieur a été conçu par la CMC (Société Canadienne de Microélectronique), utilisant une technologie 0.5µm avec trois couches de métaux. Nous avons la représentation de ce circuit ci-dessous. La partie a) est le multiplieur 8x8 sans les entrées et

sorties. Nous présentons aussi la disposition de chaque cellule en partie b). Après avoir disposé de nos cellules et optimisé l'horloge interne, nous avons exécuté un DRU (*Design rules check*). Puis, nous avons re-simulé notre circuit avec capacité parasite (Post-layout).

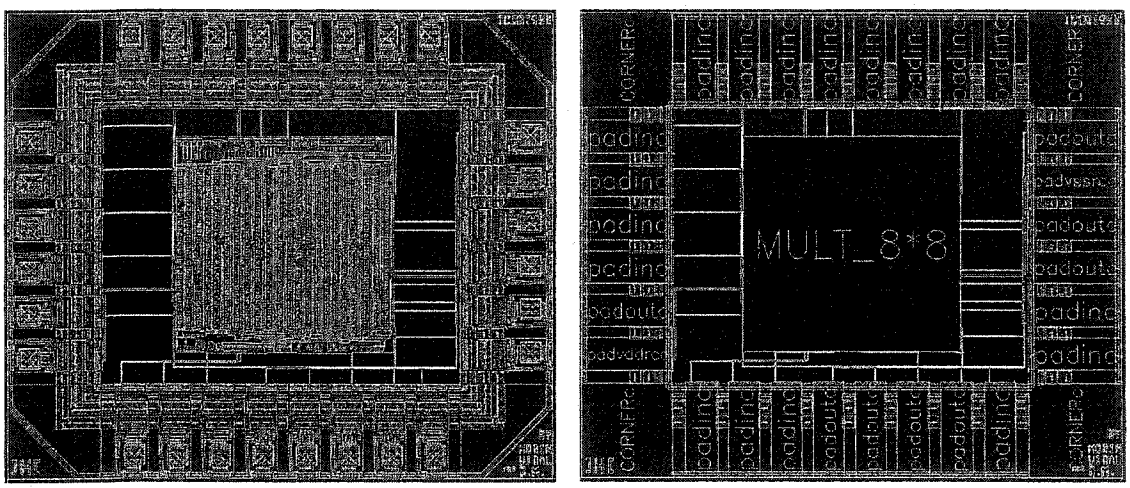


a) b)

Figure 5.18) Multiplieur (8x8)

a) Dessin de masque b) Positionnement des cellules.

La dernière étape avant la fabrication est le positionnement des broches d'entrée et de sortie. La figure 5.19, montre notre circuit entouré d'un anneau de garde où chaque broche est reliée à un dispositif contre les décharges électrostatiques (*ESD*).



a) b)

Figure 5.19) Multiplieur (8x8) avec broche entrée-sortie

a) Puce fabriquée b) Disposition des IO sur la puce.

Le multiplicateur a été fabriqué indépendamment du MAC pour 2 raisons. D'abord, puisque le multiplieur sera utilisé plusieurs fois à l'intérieur de nos processeurs élémentaires dédiés à l'égalisation de canaux, on ce devait de vérifier son bon fonctionnement comme élément indépendant. En second, puisque notre architecture MAC fonctionne avec une boucle de rétro-propagation des données, les variations reliées au procédé doivent être mesurées. La synchronisation est très différente comparativement à une architecture linéaire simple. Il est donc important de vérifier si notre structure utilisant le pipeline par vagues fonctionne et ceci avant d'implanter une structure beaucoup plus complexe, comme notre architecture.

5.3 Pipeline par vagues du multiplieur-accumulateur

Le multiplieur-accumulateur a été conçu sur les mêmes principes que notre multiplieur CSA. Puisque les données accumulées utilisent la boucle de rétro propagation, ils emmènent une complexité au niveau de l'ajustement des données ou des vagues. Les simulations ont donc été effectuées d'abord au niveau schématique simple (aucune capacité parasite et délai d'interconnexion) ce qui diminue le temps de simulation de plusieurs heures à quelques minutes. Par la suite, nous avons conçu le dessin de masque et extrait les informations nécessaires durant les simulations post-layout, permettant des résultats qui s'approchent des résultats physiques. Les performances au niveau fréquentiel et de la propagation sont présentées plus loin. Faisant attention aux règles de dessin et aux limites réelles du dessin de masque, nous avons fourni en format GDSII (Gerber Data Stream) notre dessin de masque.

5.3.1 Description

Pour la conception du MAC, nous avons réunie notre additionneur à propagation de la retenue et le multiplieur 8x8 de type CSA. Cependant, l'ajout de l'additionneur et du multiplicateur n'est pas suffisant. Le MAC nécessite plusieurs structures additionnelles fabriquées pour contrer les effets néfastes d'une structure décalée dans le temps. Puisque nous avons choisi un additionneur de type RCA on se devait de propager chaque bit des

octets (8bits) d'entrée devant être additionnée en décalant du bit de poids à valeur faible (2^0) au bit de poids à valeur maximum ($2^{7)}$. Chaque addition bit par bit transmet donc la retenue du bit précédent. On remarque sur la figure suivante (figure 5.20) les triangles de temporisation en sortie du CSA permettent de décaler dans le temps chaque bit de la gauche vers la droite. Le même système a été conçu en sortie, mais ceux-ci (Triangle de dé-temporisation) remettent nos données de façon parallèle dans le temps. Ainsi, chaque donnée accumulée doit être décalée dans le temps et être additionnée à la valeur précédente ou être mémorisée par les registres de sortie.

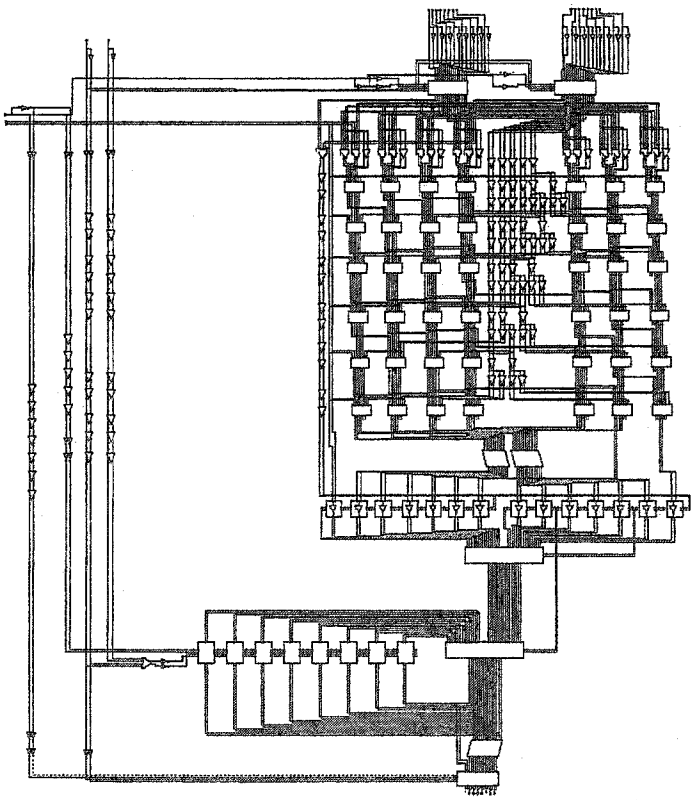


Figure 5.20) Schématique du MAC (8x8).

5.3.2 Résultats de simulation

Les résultats suivants présentent le comportement de notre multiplieur-accumulateur en fonction de plusieurs procédés. Ils démontrent bien les faiblesses et qualités obtenues lors d'une implantation utilisant le pipeline par vagues. Durant cette recherche, une implantation d'un MAC utilisant le pipeline par vagues a été présentée par W. P. Burleson [BUR94]. Cependant, l'utilisation de registres entre le multiplieur et l'addition simplifiait grandement la conception du MAC. Pour notre cas, nous n'utilisons aucun registre entre chaque multiplication additionnée. C'est-à-dire, chaque donnée multipliée est directement additionnée et le seul élément mémoire (registre) inclus à notre structure est implanté à notre boucle de rétro-propagation. Une excellente représentation d'un MAC a été présentée par [JOU97], utilisant la méthode du pipeline comme optimisation du débit. Il fait appel à des triangles de temporisation ou décalage pour effectuer son accumulation. D'après nous, notre MAC constitue dès lors le premier MAC implantant le Pipeline par Vagues comme méthode d'optimisation du débit.

La conception du Multiplieur-Accumulateur peut devenir assez complexe si les données accumulées doivent nécessiter une dépendance dans le temps, c'est-à-dire que chaque donnée doit être accumulée à la donnée précédente. Plus spécifiquement, le tout se complique en générale lorsque la propagation entre une nouvelle multiplication arrive plus rapidement ou plus lentement que les données accumulée précédemment. Ceci est plus particulier à l'implantation du pipeline par vagues. Heureusement, notre architecture à base

de neurones ne nécessite aucune accumulation chronologique puisque c'est la moyenne des gains qui ajuste les poids de chaque neurone. Mais pour cette recherche, il importe d'obtenir des accumulations régulières dans le temps. Voilà pourquoi la boucle de retour inclus une synchronisation par registre avec l'horloge. Le temps de propagation des données d'entrée ajoute alors un délai qui doit être compensé par l'horloge reliée aux registres. Remarqué à la figure 5.20, que notre réseau emprunté par l'horloge inclus des temporisateurs. L'idée derrière n'est pas de rajouter un délai semblable à la donnée traversant l'arithmétique mais bien d'exercer un déphasage inférieur à 180° de notre horloge d'entrée. Ce qui aura pour but de rallonger le temps de stabilisation (*Hold time*) des données avant quelles ne soient capturées par nos registres à l'intérieur de la boucle.

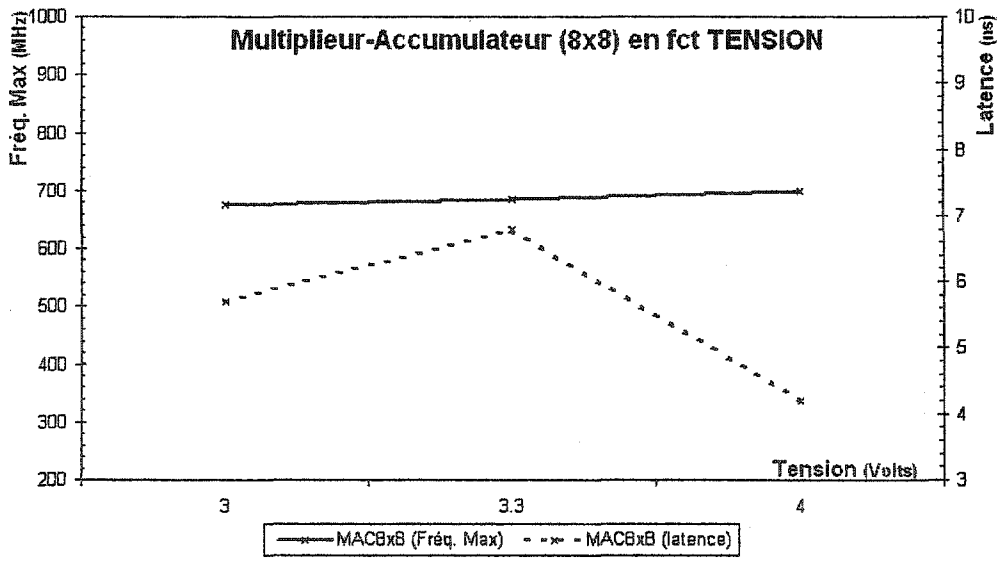


Figure 5.21) MAC 8x8 en fonction de la tension.

Voilà pourquoi nos résultats présentent une certaine caractéristique en dehors des fréquences synchroniser par notre multiplieur lorsque notre circuit est testé avec un procédé et une température normale (3.3Volts, 27° Celsius et 0.6µm). En plus, on remarque une légère diminution de la fréquence maximale dans la région de 3.3 Volts et aussi une latence plus élevée. Ceci représente l'endroit où la phase de notre horloge arrive très tard à notre registre ce qui permet d'être robuste aux paramètres externes. Nos données seront stables avant d'être capturées par l'horloge.

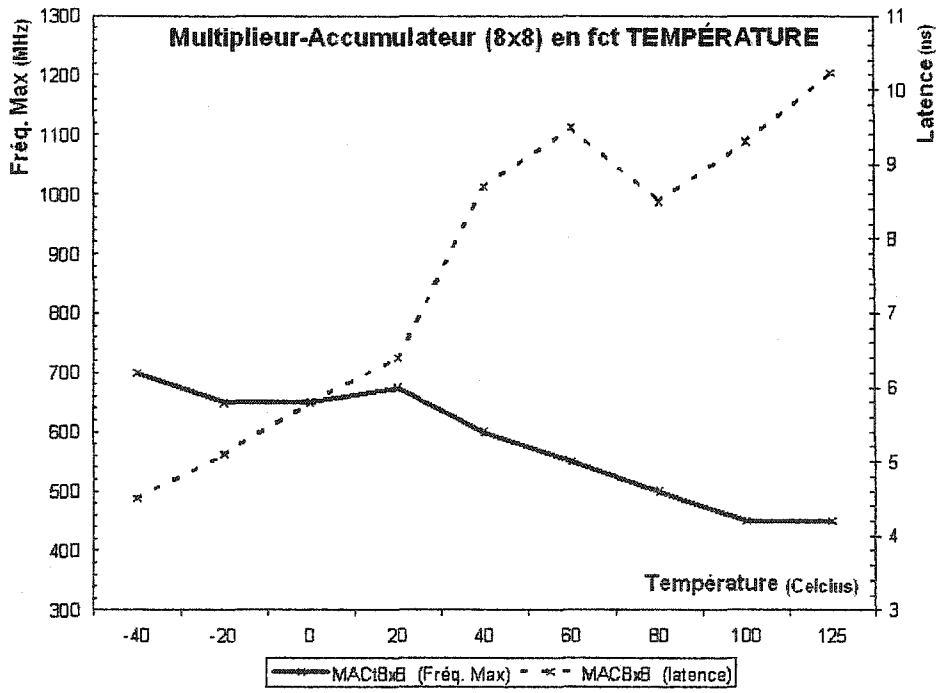


Figure 5.22) MAC 8x8 en fonction de la température.

La figure 5.22, présente substantiellement les mêmes résultats que la figure précédente. On remarque encore une légère différence lors de procédés normaux. Nos résultats

présentés ci-haut montre la piètre robustesse au changement de température lors d'une implantation de type pipeline par vagues. Les vitesses atteintes par notre MAC sans dépendance implantée dans notre architecture pourraient très bien atteindre des vitesses qui se rapprochent des vitesses de notre multiplieur.

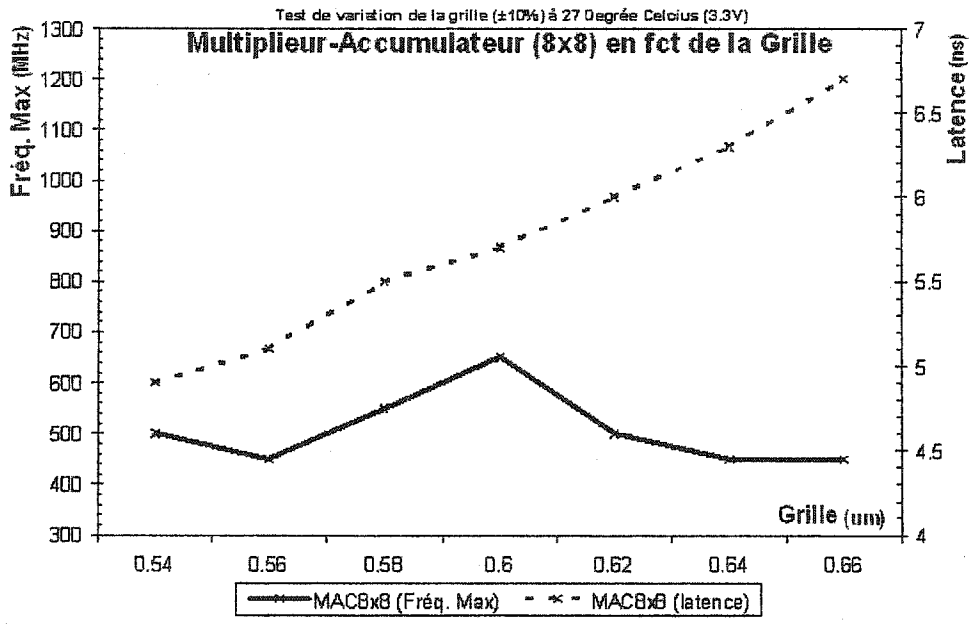


Figure 5.23) MAC 8x8 en fonction de la grille.

Comme dernier test, nous varions la dimension de notre grille d'environs 10%. La dimension de la grille suggérée par le fabricant est de 0.6µm, ce qui devrait nous limiter en fréquence. Pourtant ce n'est pas le cas, notre architecture dépend plus du déphasage entre la donnée accumulée et la dernière multiplication.

La figure 5.24 présentée ci-dessous démontre la véridicité des résultas obtenues par

notre multiplicateur-accumulateur. Pour chaque accumulation nous avons la valeur multipliée et additionnée avec la prochaine multiplication future. Le signale "coco" représente la remise à zéro logique inverse. Le signale "CLK" représente l'horloge d'entrée, le signal "CCLK" représente l'horloge de sortie non-restauré. Ainsi, la figure ci-dessous présente trois sections. La section **a**, est la multiplication de 29x1 =29, la section **b** représente 11x4+**a**= et la section **c** est 29x1+**b**=102. Ainsi, nous alternons entre deux multiplications de valeur décimale 29 et 44.

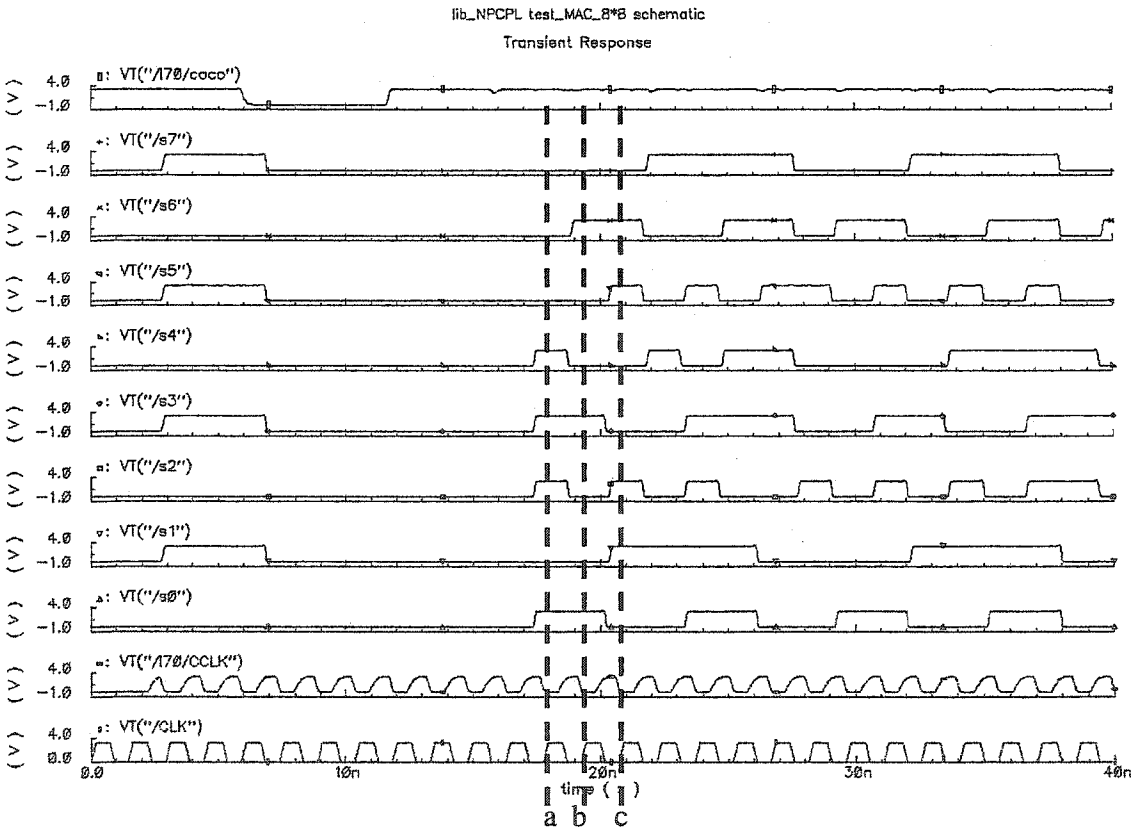
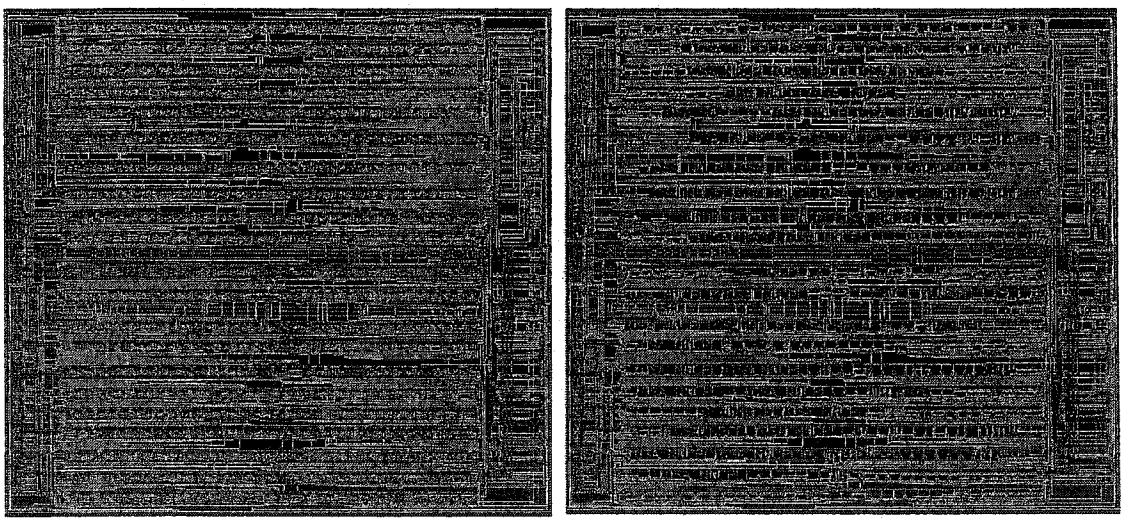


Figure 5.24) Valeur en sortie du MAC(8x8).

Le MAC complément à 1 a été conçu et fabriqué grâce à la générosité de la CMC, utilisant une technologie 0.5µm et trois couches de métaux propriété intellectuelle de Hewlett-Packard®. Il est important de spécifier que la dimension des grilles utilisées est de 0.6µm recommandé par HP®. La représentation du dessin de masque est présentée ci-dessous. La partie a) représente le MAC 8x8 sans les entrées et sortie. En partie b) nous présentons la disposition des cellules. Après avoir disposé de nos cellules et optimisé l'horloge interne, nous avons exécuté un DRU (*Design rules check*). Puis, nous avons re-simulé notre circuit avec capacité parasite (Post-Layout).



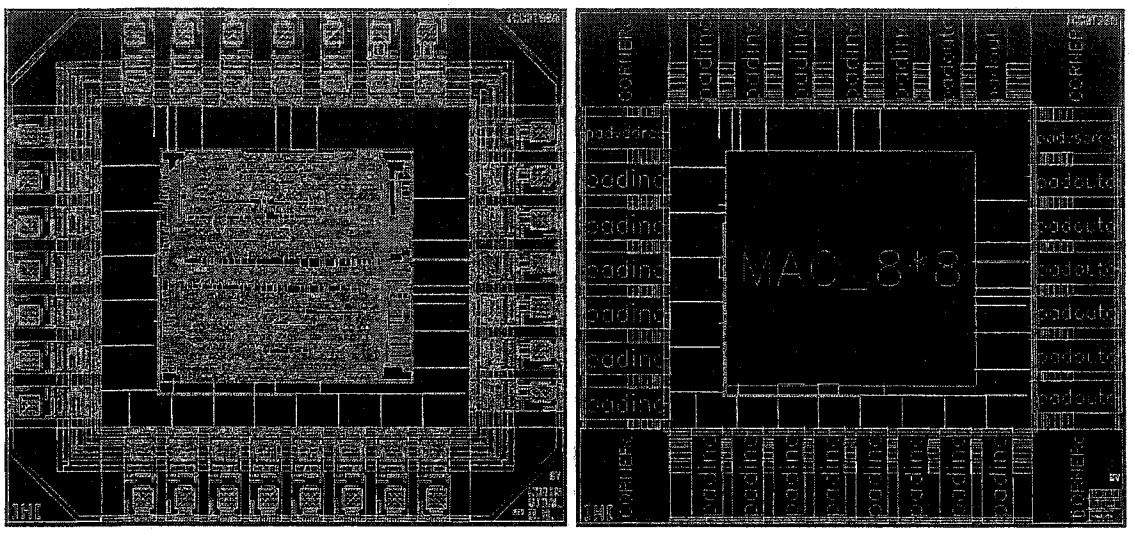
a) b)

Figure 5.25) MAC (8x8)

a) Dessin de masque b) Positionnement des cellules

La dernière étape avant la fabrication est le positionnement des broches d'entrées et de

sorties. La figure 5.19, montre notre circuit entouré d'un anneau de garde où chaque broche est reliée à un dispositif contre les décharges électrostatiques (*ESD*).



a) b)

Figure 5.26) MAC (8x8) avec broche entrée-sortie

a) Puce fabriquée b) Disposition des entrées-sorties.

5.4 Évaluation des performances de l'architecture arithmétique proposée

Les architectures proposées ont été sélectionnées en vu d'une implantation optimale utilisant le pipeline par vagues et pouvant respecter néanmoins les caractéristiques nécessaires pour obtenir un grand débit de transmission lors de leurs utilisation à notre architecture PL-MNN avec apprentissage hors ligne [VID99d]. La décision fût prise d'utiliser des architectures avec une grande latence et augmenter ainsi la quantité de vagues

pouvant être introduites. Voilà pourquoi même s'il existe des architectures arithmétiques parallèles beaucoup plus rapide au niveau du débit présenté par J. L. Hennessy [HEN92], M. R. Zargham [ZAR95], ils n'ont pas les paramètres requis pour une implantation pipeline par vagues utilisant la cellule NPCPL plus précisément. Les architectures hautement parallèles conviennent mieux à une implantation conventionnelle ou pipeline.

À partir des résultats de l'étude de quantification dans [VID99d], un modèle analogique utilisant Hspice® par Cadence® et VHDL utilisant Synopsys® comme outils de synthèse a été créé pour valider les architectures arithmétiques et finalement testé sur le modèle architectural PL-MNN avec apprentissage hors ligne.

Pour la création individuelle de chaque arithmétique sur puce de silicium nous avons limité la longueur des mots binaires à 8-bits. Par contre, une étude de quantification présentée par [VID99d] démontre l'utilisation d'un convertisseur de 8-bits. Une longueur de mots binaires de 16 bits interne offre un bon compromis précision/surface à notre architecture PL-MNN hors ligne. Le tableau ci-dessous présente le dimensionnement finale des nos arithmétiques.

	Multiplieur (8x8)	MAC (8x8)
Surface avec broche :	2762.3 um x 2306.3 um (6.37 um²)	2793 um x 2541 um (7.10um²)
Surface sans broche :	1119.6 um x 1070 um (1.198 um²)	1170.70 um x 1281 um (1.5 um²)
Nombre de transistor :	3628	4294

Table 5.2) Dimensionnements et Puissances des U.A. fabriquée

5.4.1 Débits et latence de l'architecture arithmétique proposée

Les débits obtenus sont présentés à titre indicatif dans le but de comparer les différences entre les divers types d'implantations. Ainsi, les comparaisons se font à température ambiante (25°C) avec une source de tension normale (3.3 Volt). Utilisant d'une part Synopsys® comme élément d'estimation statique et Cadence® pour nos modèles analogiques *post-layout*. Nous obtenons en bref :

	Débits	# Transistors	Latence
Conventionnel	100 MHz	2000	11 ns
Pipeline (22 étages)	500 MHz	5168	44 ns
Pipeline par vagues (4 vagues)	650 MHz	4300	8 ns

Table 5.3) Résultats et Tableau comparatif du MAC 8x8

Il est à noter que nos architectures ne pourront obtenir ces fréquences présentées ci-haut sur silicium puisqu'il n'existe aucun port d'entrée-sortie pouvant atteindre plus de 50MHz pour la technologie 0.5µm SPTM. Pour atteindre les fréquences obtenues par simulation synthétique, il faudra comme tout architecture à haut débit, implanter un multiplieur fréquentiel utilisant soit un PLL ou DLL (*Phase Lock Loop* ou *Delay Lock Loop*) à l'intérieur de notre architecture [MEH94], [PAR99]. Notre décision de fabriquer ces deux architectures sur silicium était d'abord de vérifier l'intégrité de nos architectures utilisant une toute nouvelle méthode (Pipeline par vagues). Et d'autre part, d'étudier la première

phase avant l'implantation d'une architecture neuronale complète impliquant une architecture de type mixte (Analogique-Digital) dans un même procédé.

5.5 Conclusion

Les performances obtenues par nos deux architectures démontrent bien les capacités et les avantages d'implanter des unités utilisant le pipeline par vagues. Cependant, beaucoup de travail reste à faire avant de pouvoir utiliser une unité purement pipeline par vagues, puisque les procédés des variation emmènent un écart en fréquence non négligeable. Pour l'industrie, toute fabrication ITGE doit fonctionner normalement à l'intérieur des paramètres de standardisation, c'est-à-dire entre -40°C à 125°C et 3.0 à 3.6 Volts. Néanmoins, nous démontrons théoriquement qu'il est possible d'atteindre des fréquences maximales de 4 à 5 fois supérieures aux méthodes conventionnelles et ayant une latence de 4 à 5 fois moins grande que le pipeline conventionnel. En plus, il est primordial de mentionner la consommation de puissance dynamique et de surface réduit comparativement aux architectures pipelines conventionnelles.

De futures recherches sur une architecture arithmétique utilisant l'algorithme Baugh-Wooley [ZAR95] présentée ultérieurement au chapitre 3 ont été faites. L'architecture permet une multiplication complément à 2 directs et simplifie les conversions intermédiaires. La fabrication et simulation pré-layout d'une multiplication 4x4 bits a été effectué et présente une architecture plus compacte, obtenant de meilleures performances.

Il n'est donc plus nécessaire d'ajouter des temporisateurs pour décaler nos données avant de passer à travers nos RCA. D'autres recherches devront être faites utilisant dans le futur la cellule NPCPL modifiée, si nous voulons des arithmétiques avec un nombre de bit supérieur à 4-bits.

6 Application à l'égalisation de canaux

L'intégration d'unités arithmétiques utilisant la méthode du pipeline par vagues est donc utilisée pour l'intégration à très grande échelle d'une architecture dédiée au traitement à temps réel. Nous proposons au présent chapitre une architecture ITGE à laquelle sera implanté un algorithme dédié aux traitements des signaux numériques. L'algorithme propose l'approche systolique comme structure qui permet des communications locales entre processeurs élémentaires et apporte une plus grande flexibilité au cas ou une modification de l'algorithme dédié plus spécifiquement aux réseaux neuronaux serait nécessaire dans le futur. Un réseau de neurones non-récursifs avec apprentissage hors-ligne permet l'implantation de structure arithmétique pipeline par vagues.

La conception de notre architecture sera exécutée en deux étapes. D'abord nous utiliserons le langage VHDL (*VHSIC Hardware Description Language*) comme élément CAO pour la conception de notre architecture permettant ainsi d'utiliser des cellules normalisées CMOS offertes par la technologie. Les arithmétiques conçues précédemment seront uniquement introduites à la toute dernière étape puisqu'elles sont considérées comme élément analogique. Nous devrons remplacer pour nos simulations nos éléments arithmétiques par des boîtes noires effectuant les mêmes opérations arithmétiques et avec la

même propagation. Les outils de synthèses automatiques et de simulations remplaceront les simulations P-Spice.

Dans ce chapitre, il sera question à la section 6.1 des contraintes et des critères architecturaux en ITGE. Discutant des avantages d'une implantation numérique face aux contraintes d'une architecture totalement analogique. Nous discuterons aussi d'une technique d'architecture parallèle simplifiant l'intégration d'algorithmes neuronaux. Ensuite, à la section 6.2, une architecture sera proposée pour l'algorithme PL-MNN avec fonction d'activation canonique par morceaux [VID99] permettant l'intégration d'arithmétique pipeline par vagues. À la section 4.3, nous présentons une estimation de la surface d'intégration de notre architecture et des performances qui pourront être atteintes en utilisant une technologie de 0.5 µm.

6.1 Contraintes et critères architecturaux en technologie ITGE

Le développement d'une architecture principalement numérique sera choisi puisqu'il permet alors d'utiliser des outils de conception assistés par ordinateur (CAO) principalement orienté au circuit numérique utilisant des librairies pré-conçues et testées par le fabriquant ce qui restreint les coûts reliés au développent de produit ASIC (*Application Specific Integrated Circuit*) et au temps de développement de circuit de fonction de plus en plus complexe où le nombre de transistors continu d'augmenter exponentiellement tel qu'indiqué par la loi de Moore. Ainsi, la conception numérique avec des librairies pré-conçues rend ce type de circuit plus simple à réaliser. De nos jours, il est

même question de circuits pouvant être intégrés par différent fabricant appelé système sur silicium (SOC). Les circuits analogiques et « custom », comme pour ce qui est de nos cellules NPCPL, ont été pour longtemps l'unique méthode à la réalisation de projet ITGE. Cependant, très coûteuse en développement et en vérification elle est principalement utilisée maintenant pour le développement de propriétés intellectuelles (IP) axées sur un créneau très spécifique de composant.

Les circuits analogiques apportent certains inconvénients que les circuits numériques n'ont pas. Un circuit analogique requière beaucoup plus de temps de conception et nécessite plusieurs types de protection pour permettre une stabilité du circuit. Car les circuits analogiques sont très sensibles aux sources externes (Champ électrostatique et magnétique, bruits thermiques et quantiques) et à la stabilité des sources d'alimentation. Les procédés de fabrication peuvent aussi déterminer si notre circuit fonctionne adéquatement, car il est très difficile d'obtenir des circuits analogiques imitant les caractéristiques physiques au quelles ils ont été conçu comme par exemple la variation de gain des transistors. Les capacités parasites restent aussi un important facteur qu'il ne faut surtout pas négliger ainsi que la durée de vie des circuits analogiques qui est de beaucoup moins élevée. En plus, l'architecture analogique peut obtenir des résultats différents de fabrication en fabrication.

Cependant, les circuits analogiques restent toujours très alléchant puisqu'ils permettent de fonctionner en courant. Contrairement aux circuits numériques qui fonctionnent en saturation de tension. Ainsi les performances en fréquences élevées son plus facilement

atteignables. La consommation en puissance dynamique et en courant de fuite est aussi très faible.

Les circuits numériques, quant à eux, fonctionnent en saturation. La variation des gains des transistors n'ont aucune influence sur le fonctionnement de l'architecture et ils ont une durée de vie très longue. De plus, les circuits numériques sont très peu affectés par les procédées de fabrication. Les circuits numériques permettent d'utiliser la représentation binaire comme système de calculs et ainsi d'exécuter des opérations arithmétiques plus facilement ce qui simplifie son intégration. Cependant, la consommation de puissance dynamique et les courants de fuite ne cessent d'être un problème grandissant. Les circuits numériques consomment beaucoup en énergie dynamique lors des transitions. Alors que la diminution des transistors augmente la quantité de courrant de fuite. Pour notre cas, notre choix s'arrête bien évidemment sur les intégrations numériques, car d'une part l'environnement auquel devra faire face notre architecture est dédiée plus précisément au canal non-linéaire et de type non-variant. Voilà pourquoi l'égalisation de canaux a d'abord été choisie. L'égalisation de canaux est souvent utilisée à l'intérieur d'un environnement bruité, alors que sa consommation de puissance n'est pas un facteur prédominant. D'autre part, l'élément de vérification sur notre circuit pousse sur le numérique puisque étant déjà testé par le fabriquant de la technologie, il permet de concevoir une architecture complète rapidement sans les contraintes que notre circuit analogique aurait pu apporter. Les circuits numériques sont choisis pour implanter en technologie ITGE l'algorithme PL-MNN.

Une autre contrainte qu'il ne faut pas négliger est la quantité phénoménale de calculs

devant être effectués pour arriver à corriger les ISI sans introduire une latence trop élevée. Nous proposons donc une architecture utilisant le parallélisme pour arriver à gagner en fréquences ce qui normalement prend plusieurs étapes de calculs. Diviser pour régner consiste donc à diviser les tâches de calculs et de contrôles en plus petits procédés. Le principe général du parallélisme est d'obtenir des traitements plus rapides en effectuant des calculs simples et rapides simultanément. Les réseaux de neurones ce prête bien à ce type d'architecture puisque chaque cellule est en soi un processeur élémentaire (PE). Celle plus connu sous le nom d'architecture systolique a été initialement proposée par [KUN82].

Les architectures systoliques peuvent être appliquées pour plusieurs types d'applications. [MOR98], [ZAR95], [ELO98], [ELO98b], [MAS98], [VID99b], [VID99d], et [ZAK99] proposent tous des algorithmes pouvant être appliqués sur une architecture systolique. Ces architectures sont constituées de processeurs élémentaires (PE) dédiés qui communiquent localement entre eux. Une classe d'applications spécifiques est donc appliquée à différents types de topologies. La topologie constitue alors un système complet. L'avantage derrière l'application d'architectures systoliques est avant tout pour augmenter le débit de notre architecture tout en obtenant une architecture régulière facilement implantable et pouvant même être modifié facilement. Finalement, les méthodes pipelines sont très faciles à implanter sur des topologies systoliques.

6.2 Choix et description de l'architecture

Au chapitre 2, l'algorithme PL-MNN a été choisi comme méthode optimale à l'égalisation d'un canal non-linéaire. Nous savons qu'un réseau neuronal s'avère à être idéal pour une implantation systolique parallèle. Chaque neurone sera considérée comme processeur élémentaire (PE) car comme présenté au chapitre 2, les équations 2.11, 2.12, et 2.13 présentent des équations sous forme récurrente et régulière. Les communications sont donc locales entre chaque neurone ce qui explique la préférence au choix d'une architecture dédiée aux topologies systoliques en vue d'une intégration ITGE. Notre architecture n'aura aucune récurrence, car l'utilisation de technique pipeline par vagues permet difficilement l'utilisation d'équations récurrentes alors que les poids de chaque neurone seront calculés hors-ligne. L'algorithme PL-MNN et les équations présentées au chapitre 2 seront dédiés aux communications utilisant la modulation PAM.

L'architecture représentée à la figure 6.1.a a comme entrée un vecteur $\tilde{y}(n-m)$ avec un nombre limite de délais M+1. Le signal bruité $\tilde{y}$ traité sera bouclé par une pile circulaire à l'entrée jusqu'à ce que le premier PE_1 traite le signal $\tilde{y}(n)$. Lorsque toutes les valeurs du vecteur d'entrée ont été livrées à l'architecture, l'élément est expulsé de la pile et un nouveau élément est introduit et ceci jusqu'au dernier PE_K, un par un. La dernière étape est la somme partielle de chaque PE_k avant de passer à travers la fonction d'activation.

Chacun des PE_k possède les mêmes caractéristiques que son voisin. Pour assurer une convergence des performances de notre égalisateur, nous avons un nombre limité de

neurones dans une couché cachée. Notre réseau de neurones non-récursifs a donc un neurone par PE où l'indice K représente le nombre de neurones sur la couche cachée. La figure 6.1.b présente en détail la structure de chaque processeur élémentaire PE_k. À l'intérieur de chaque PE, puisque nous fonctionnons hors-ligne une pile circulaire d'élément mémoire contenant les poids w_{km} sera utilisée. Les poids seront ensuite multipliés et accumulés avec la sortie de notre canal $\tilde{y}(n)$, où la cellule octogonale représente le MAC utilisé. Par la suite, le signal s_k à la sortie du MAC traversera notre fonction d'activation canonique linéaire par morceau et générera le signal en sortie z_k. z_k sera ensuite multiplié par le poids de la sortie de la neurone q_k. Finalement, le résultat de cette multiplication sera partiellement additionné avec la sortie du processeur élémentaire précédent PE_{k-1}.

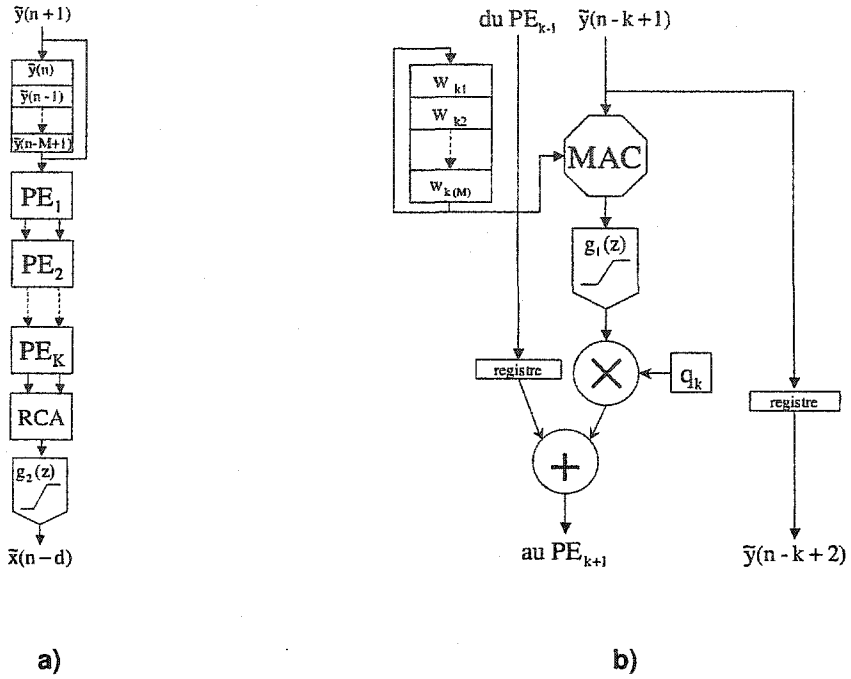


Figure 6.1) L'architecture PL-MNN (a), Processeur élémentaire (b).

Une étude de quantification sur la convergence de l'algorithme, car celle-ci est la plus affectée par cette quantification pour des signaux variant dans l'intervalle [-1, 1], démontre que l'utilisation d'un convertisseur analogue-digital de 8-bits pour la conversion de l'entrée du signal $\tilde{y}(n)$ et une longueur de mots binaires de 16-bits pour les constantes (w_{kn}, q_k) et pour les variables (s_k, z_k, $\hat{x}_k$), offre un bon compromis précision-surface [VID99d].

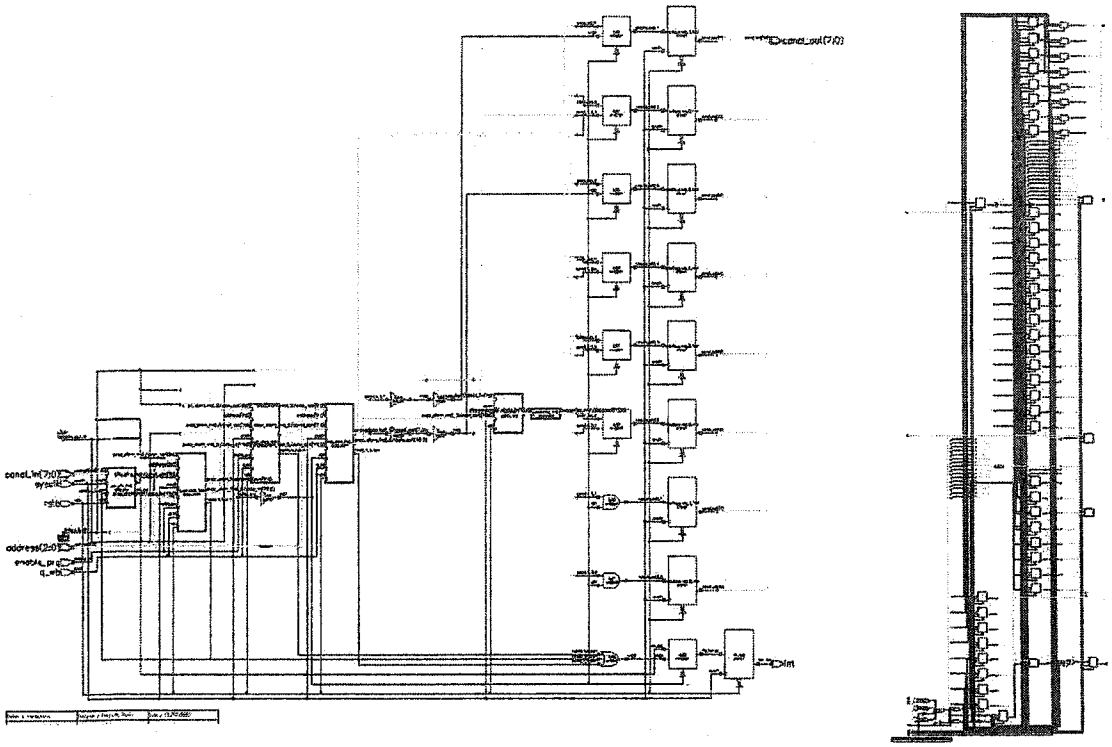
6.3 Estimation de la surface de l'architecture

Notre architecture à été synthétisée à l'aide de l'outil de Synopsys® utilisant la technologie CMOS 0.5µm de Hewlett-Packard®. Fonctionnant à 3.3 Volts, utilisant une couche de poly et cinq couches de métaux. Après avoir conçu notre architecture utilisant le langage VHDL présenté en annexe B après avoir utilisé Cadence comme compilateur de code, nous avons utilisé l'outil de synthèse automatique pour évaluer l'estimation de surface qui serait obtenue par note architecture si le nombre de neurones sur la couche cachée était de K=3 et un délai en entrée de M=2. Puisque le rapport de surface obtenue par Synopsys® est une estimation du nombre de portes logiques. Le Concepteur de logiciel Synopsys® compare une porte logique à une porte Non-Et, c'est à dire environs quatre transistors. Le rapport de surface obtenue par Synopsys® ne tient pas compte du MAC et du Multiplieur pipeline par vague conçu précédemment sous forme de macros. Le tableau ci-dessous présente le nombre de transistors estimés de notre architecture PL-MNN.

Processeur Élémentaire	#Transistors	Architecture PL-MNN	#Transistors
additionneur (8x16)	1086	additionneur RCA (16x16)	1136
Pile circulaire (8 bits)	296	Pile circulaire (8 bits)	296
Canonique lin. (16 bits)	161	Canonique lin. (16 bits)	161
Mult pipeline par vagues (8 x 16)	3628	Processeur Élémentaire (3x)	34374
MAC pipeline par vagues (8 x 8)	4294		
Logique Contrôle	1993	Logique Contrôle	1123
Totale :	11458	Totale :	37090

Table 6.1) Estimation du # transistors utilisés par l'architecture PL-MNN.

En annexe B, nous présentons l'architecture en langage VHDL utilisée pour notre estimation utilisant trois processeurs élémentaires. L'annexe C, présente les fichiers de commande et les contraintes appliquées à notre architecture. Les figures ci-dessous présente l'architecture synthétisée sous Synopsys® Design Vision™. La figure 6.2.a représente notre architecture sous vision hiérarchique, et peut être comparée avec la figure 6.1.a. Alors que la figure 6.2.b représente un processeur élémentaire PEk. Notre comparaison peut aussi être faite avec la figure 6.1.b.



a) b)

Figure 6.2) L'architecture PL-MNN (a), Processeur élémentaire (b) après synthèse automatique.

La latence estimée sera pour M=2 et K=3, de (M+K+5/*fréq.Max*) et de (M+K+35/*fréq.Max*) pour les versions non-pipelinés et pipelinés. Alors que pour la version pipeline par vagues elle est de (M/*fréq.Max* + (K+16)τ_{temp} + 24ns). Le tableau 5.3 présente une estimation des fréquences maximale pouvant être atteintes par l'architecture.

Pour terminer nous avons synthétisé et égalisé tout les niveaux de l'architecture pour obtenir une architecture avec porte logiques. Les MAC et les Multiplieurs pipeline par

vagues restent des boites noires puisqu'ils seront intégré sous forme de macro lors de la création de dessin de masque de l'architecture complet. La figure suivante présente les entrées/sorties de l'architecture.

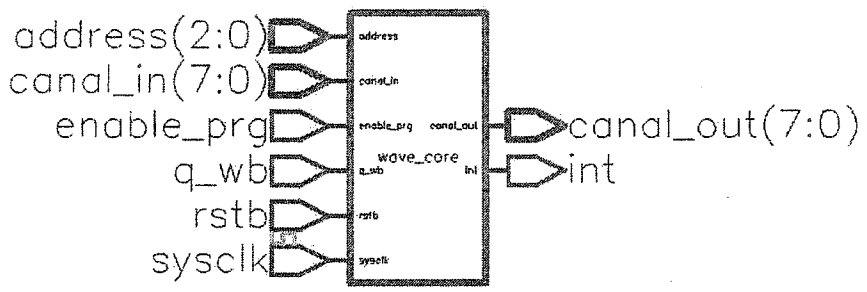


Figure 6.3) Entrées/Sorties de l'architecture PL-MNN avec programmation hors-ligne.

6.4 Conclusion

Nous avons présenté une architecture basée sur un type de réseau de neurones dite PL-MNN. L'architecture [VID99] a été utilisée et modifiée en fonction d'une implantation pipeline par vagues. L'architecture présentée utilise donc des équations sans récurrences, et un apprentissage hors-ligne. L'architecture permet des performances supérieures à l'utilisation de méthodes d'optimisations pipeline. Cependant, elle doit obtenir de façon externe les poids des neurones avant de pourvoir effectuer une égalisation du canal à traité. Plusieurs articles ([VID99], [VID99b], [VID99c], [VID99d] mentionnent l'utilisation de l'algorithme PL-MNN en vu d'une implantation ITGE. Trois points importants poussent au

choix de cet algorithme. D'abord, elle permet l'implantation d'une structure parallèle systolique. D'autre part, la fonction d'activation canonique linéaire par morceau est simplement un décalage binaire. Et en troisième lieu, l'architecture permet l'implantation ITGE sous forme numérique.

Ce chapitre présente les principes qui nous ont amenés au choix de l'architecture. Nous discutons d'abord des avantages et des inconvénients entre une architecture analogique et numérique. Puis nous présentons un type d'architecture parallèle qui préconise l'intégration de réseau de neurone sous forme systolique. Le tout est discuté à la section 6.1 (Contrainte et critères architectureaux en technologie ITGE). Ensuite à la section 6.2 (Choix et description de l'architecture) nous présentons en détail l'implantation ITGE de notre architecture PL-MNN. La dernière section présente une estimation de surface de l'architecture utilisant d'abord un outil de synthèse automatique pour la logique numérique et d'autres part la dimension du nombre de transistors utilisés par nos unités arithmétiques par l'entremise de l'outil de dessin de masque Virtuoso™ de Cadence®.

7 Conclusion

L'objectif de cette recherche était donc l'application du pipeline par vague à l'intérieur d'une architecture systolique dédiée à l'égalisation de canaux. Ainsi, l'utilisation fréquente d'algorithmes mathématiques de plus en plus performant dans le domaine des communications tel que SONET/SDH, DWDM et Cellulaire sont la preuve que les techniques de compression, de codage, et de filtre numérique linéaire seront nécessaires pour satisfaire la demande toujours croissante d'information nécessitant un débit très élevé et une qualité de service remarquable à coût toujours plus faible. Au niveau physique, cette forte croissance limite la largeur des bandes en fréquences et crée des interférences entre symboles (ISI) qui peuvent erroner l'information reçue. C'est-à-dire, que les ISI peuvent créer une inversion de polarité lors de transmission où la modulation est aussi simple qu'une transmission de type binaire (PAM). Et même un changement de phase lors de communications de signaux modulés complexes transmettant plusieurs constellations polaires (QAMn) pourrait être causé. Une architecture axée sur l'égalisation de canaux ITGE a donc été proposée. Elle utilise une méthode d'optimisation des calculs arithmétiques (pipeline par vagues), permettant d'augmenter nos performances en correction d'information.

Alors, nous avons donc proposé l'implantation d'une architecture utilisant l'égalisation de canaux comme algorithme adaptatif dans le but d'estimer la séquence transmise à partir du signal disponible à la réception. La difficulté reste dans la recherche d'une inversion parfaite d'un canal non-linéaire pouvant obtenir une reproduction fidèle dans un environnement difficilement estimable. Le côté algorithmique présente plusieurs méthodes basées sur l'égalisation (Chapitre 2) et un choix utilisant les réseaux de neurones a été choisi. Nous proposons d'utiliser une modification de l'algorithme PL-MNN à l'intérieur d'une architecture systolique non-récursive permettant le développement d'unités arithmétiques utilisant le pipeline par vagues comme méthode d'optimisation des performances de l'architecture.

Étant donnée que les unités arithmétiques constituent le goulot d'étranglement d'une architecture ITGE, le chapitre 3 présente une investigation des unités arithmétiques, plus particulièrement l'étude portée sur les additionneurs et les multiplieurs. Le choix c'est arrêté sur deux unités afin de répondre aux contraintes d'intégration impliquant une structure pipeline par vagues. Ainsi, l'intégration d'une architecture non-récursive implique l'application de calculs utilisant des multiplieurs-accumulateurs. Faisant très peu l'objet de recherche scientifique, nous proposons une toute nouvelle arithmétique pipeline par vagues utilisant le complément à un comme méthode binaire. De futures recherches sur l'application de multiplieurs-accumulateurs utilisant le complément à deux sont aussi proposées.

Le travail présenté au chapitre 4 propose une solution à la problématique d'une

implantation ITGE utilisant les méthodes d'augmentation de débit pipeline. Une recherche sur les principales techniques pipelines est effectuée. La nécessité de réaliser les calculs en temps réel au cours d'une transmission numérique justifie l'intégration impliquant de nouvelle technique pipeline. Notre choix s'arrête au pipeline par vagues. Elle permet une réduction de puissance et de surface tout en obtenant une augmentation du débit substantiellement plus élevé. Elle permet aussi d'être innovateur puisqu'elle est rarement un sujet scientifique et amène le développement sur silicium d'une cellule ayant une dépendance logique idéale (NPCPL) qui est plus particulièrement adaptée au pipeline avec logique purement combinatoire. La cellule NPCPL a été développée sur silicium 0.5µm. Elle permet l'utilisation de plusieurs types de logique booléenne (ET, Non-ET, OU, Non-OU, OUEX, Non-OUEX, Inverseur, et Temporisateur) sans la complexité de chacune des intégrations. Cette recherche présente même une toute nouvelle utilisation de notre cellule non-présentée par les articles [GHO93-94]. L'implantation d'un registre ayant les similitudes d'une bascule de type D sans la surface et la latence encourue. Un délai de 0.3ns est atteint par la cellule NPCPL non dépendante de l'opération, incluant la fonction de registre.

L'implantation et la comparaison d'unités utilisant le pipeline par vagues est le sujet de notre chapitre 5. Les résultats présentés démontrent la véridicité des performances pouvant être atteintes et leurs faible robustesse au procédé et température. Les simulations de deux unités ont fait l'objet d'intenses recherches. La validation et l'évaluation des performances se sont faites grâce à un modèle PSPICE (pré-layout et post-layout) dans l'environnement

Cadence® utilisant l'outil Analog Artist®. Son intégration sur silicium a été possible grâce à l'outil Virtuoso® et à la technologie CMOS de 0.5µm (CMOSIS5). Pour être finalement fabriqué avec générosité par la société Canadienne de MicroÉlectronique (CMC).

Nous retrouvons dans la littérature, plusieurs propositions d'algorithmes d'égalisation de canaux dont certains sont pour des canaux linéaires et d'autres pour des canaux non-linéaires. Cependant, nous retrouvons peu d'architecture VLSI dédiée à l'égalisation de canaux non linéaires. L'architecture par M. Vidal [VID99b] est constituée de processeurs élémentaires utilisant les équations de récurrences d'un réseau de neurones. Ainsi, nous retrouvons par ces équations une forme régulière, récursive et qui emmène une communication locale des données. L'architecture proposée au chapitre 6 est une modification de la structure du *Piecewise linear recurrent network* (PL-MNN) qui a été proposée par M. Vidal [VID99]. Cette modification permet de diminuer une certaine dépendance des données lors de l'implantation en ITGE effectuée par l'utilisation du pipeline par vagues comme méthode d'optimisation du débit. N'ayant aucune récurrence, nous limitons les facteurs qui pourraient introduire une trop grande complexité lors de l'introduction de pipeline par vagues. L'architecture ainsi développée pourra atteindre un débit d'environ 1.7Gbps lorsque deux retards (M = 3) sur la sortie du canal de transmission sont considérés en utilisant une modulation PAM et une architecture parallèle de 8-bits dans une technologie 0.5µm.

Ce projet apporte une solution intéressante à l'intégration d'une architecture ITGE en vu d'une utilisation axée sur l'égalisation de canaux numériques à haut débit. L'application

utilisant le pipeline par vagues dans les architectures nécessitant des traitements à haut débit assure une transmission de données plus rapide avec un niveau de correction plus proche des communications à temps réel et ce, à un coût beaucoup plus faible que les méthodes conventionnelles. De plus, nous proposons pour les recherches futures une nouvelle unitée arithmétique pipeline par vagues (complément-à-deux) qui pourrait faire l'objet d'implantations architecturales plus complexe.

Ce projet a donc permis une participation à deux conférences [MOR99a] (voir annexe A), [MOR99b] et un symposium [MOR99c] pour publier la majeure partie du travail présenté. Ces articles présentent la possibilité d'appliquer la technique du pipeline par vagues sur l'architecture impliquant l'algorithme d'égalisation PL-MNN et le multiplieur-accumulateur (complément-à-un) à des processeurs élémentaires. De plus, la présentation [MOR99c] s'est vu décerner le prix "*PMC-Sierra High-Speed Networking & Communications Award*" au symposium Canadien sur la Recherche et Développement axée sur l'utilisation de la microélectronique présenté le 14 juin 1999 à Ottawa pour sa contribution au niveau des réseaux et des communications à haute vitesse.

Bibliographie

[BAR97] H. Barringer, D. Fellows, G. Gough, A. Williams, C.D. Kloos, and E. Cerny, "Abstract modeling of asynchronous micropipeline systems using Rainbow", International Conference on Computer Hardware Description Languages and their Applications. (Conference Papers), Chapman & Hall, London, UK. 1997.

[BER98] K. Bernstein, K. M. Carrig, C. M. Durham, and P. Hensen, "High Speed CMOS Design Styles", Kluwer Academic Publication, July 1998.

[BOE96] E.I. Boemo, S. Lopez-Buedo, and J.M. Menesses, "Wave pipelines via look up tables", 1996 IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World, ISCAS 96 (Conference Paper), 4 vol. (xlvii+692+801+612+845 pp.) 1996.

[BOE98] E.I. Boemo, S. Lopez-Buedo, and J.M. Meneses, "Some experiments about wave pipelining on FPGA's", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, (Conference Paper in Journal), IEEE, June 1998.

[BUR94] W. P. Burleson, C.Y. Lee, and E. J. Tan, "A 150 MHz Wave-Pipelined Adaptive Digital Filter in 2μm CMOS", VLSI Signal Processing Workshop, 1994, pp. 296-305.

[BUR98] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-Pipelining: A Tutorial and Research Survey", IEEE Transactions on very large scale integration (VLSI) systems, VOL. 6, NO.3, September 1998, pp. 464-474.

[CAM98] Colin K. Campbell. "Surface Acoustic Wave Devices for Mobile and Wireless Communications", Application of Modern Acoustics, Academic Press, inc., 1998.

[CHA95] C-H. Chang, E. S. Davidson, and K. A. Sakallah, "Maximum Rate Single-Phase Clocking of a Closed Pipeline including Wave Pipelining, Stoppability, and Startability", IEEE Transactions on computer-aided design of integrated circuits and systems, VOL. 14, NO. 12, December 1995, pp. 1526-1545.

[CHO96] H. Choi, and S.H. Hwang, "Design of Wave-Pipelined 900MHz 16b ripple-carry adder using modified NPCPL", 1996 IEEE International Symposium on Circuits and Systems, Circuits and Systems Connecting the world, ISCAS 96, 1996 4 vol.(xlvii+692+801+612+845 pp.)

[COT69] L.W. Cotten, "Maximum-rate pipeline systems", Proceeding AFIPS Spring Joint Computer Conference, 1969, pp. 581-586.

[DAE98] S. Daeyun, K. Wonchan, M.A. Soderstand, and S. Micheal, "The Design of 16*16 Wave Pipelined Multiplier using Fan-in equalization Technique", Proceedings of Midwest Symposium on circuits and Systems Dedicated to the Memory of Professor Mac Van Valkenburg, 1998, 2 vol. lii_1480 pp.

[DAN92] I. Dancea et P. Marchand, "Architecture des ordinateurs", Édition Gaëtan Morin, QA76.9A73D36, 1992.

[DAY95] P. Day, and J.V. Woods, "Investigation into Micropipeline Latch Design Styles", IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 2, June 1995.

[DE_96] V.K. De, and J.D. Meindl, "A Dynamic Energy Recycling Logic Family for Ultra Low Power Gigascale integration (GSI)", 1996 International Symposium on Low Power Electronics and Design Digest of Technical Papers (Conference Paper), 1996 390 pp.

[ELO98] B.M. Elouafay and D. Massicotte, "A Pipeline Systolic Architecture For Kalman-Filter-Based Signal Reconstruction Algorithm", CCECE'98, 24-28 May 1998.

[ELO98b] M. B. Elouafay, "Étude de l'application du pipeline par vague sur une architecture systolique dédiée au filtre de Kalman", Mémoire présenté à l'UQTR, Décembre 1998.

[GHO93] D. Ghosh, S.K. Nandy, K. Parthasarathy, and V. Visvanathan, "NPCPL: Normal Process Complementary Pass Transistor Logic for Low Latency, High Throughput Designs", 6th International conference on VLSI Design-January 1993, pp. 341-346.

[GHO94] D. Ghosh, S. Sural, and S. K. Nandy, "A 600 MHz Half-Bit Pipelined Multiplier Macrocell", 7th International Conference on VLSI Design-January 1994, pp. 95-100.

[GHO95] D. Ghosh, and S. K. Nandy, "Design and realization of high-performance Wave-Pipelined 8 x 8 b Multiplier in CMOS Technology", IEEE Transactions on very large scale integration (VLSI) systems, VOL. 3, NO. 1, March 1995, pp.36-48.

[GRA92] C.T. Gray, T. Hughes, S. Arora, W. Liu, R.K. Cavin III, "Theoretical and practical issues in CMOS wave pipelining", IFIP Transaction A (Computer Science and Technology) (Conference Paper in Journal), 1992.

[GRA93] C. T. Gray, W. Liu, R. K. Cavin III, "Wave Pipelining: Theory and CMOS Implementation", The Kluwer international series in engineering and computer science. VLSI, computer architecture, and digital signal processing, TK7871.99.M44G73, Kluwer Academic Publishers, 1993.

[HAU98] O. Hauck, and S. A. Huss, " Asynchronous Wave Pipelines For High Throughput Datapaths", 5th IEEE International Conference on Electronics, Circuits and Systems, ICECS September 1998, pp. 283-286.

[HAY96] S. Haykin, Adaptative filter theory, Prentice Hall, 1996.

[HEN92] J. L. Hennessy et D. A. Patterson, "Architecture des ordinateurs (une approche quantitative)", Édition Française Ediscience 1994.

[HSI95] H-Y. Hsieh, W. Liu, R. K. Cavin III, and C. T. Gray, "Concurrent timing optimization of latch based digital systems", Proceedings, International Conference on Computer Design: VLSI in Computers and Processors (Cat. No. 95CB35861), 2-4 Oct. 1995, xvii+711 pp.

[HUG91] T. Hughes, T Gray, W. Farlow, W. Liu, and R. Cavin, "Advantages of MCM technology for CMOS Wave Pipelined systems", 1991 Multichip Module Workshop (Conference Paper), IEEE Comput. Soc., 173 pp., 1991.

[JOH98] D. Johnson, V. Akella, and B. Stott, "Micropipelined Asynchronous Discrete Cosine Transform (DCT/IDCT) Processor", IEEE Transactions on very large scale integration (VLSI) systems, Vol.6, No.4, December 1998.

[JOU97] S-J. Jou, C-Y Chen, E-C. Yang, and C-C. Su, "A Pipelined Multiplier-Accumulator Using a High-Speed, Low-Power Static and Dynamic Full Adder Design", IEEE Journal of solid-state Circuits, vol. 32, No.1, January 1997, pp. 114-118.

[JOY93] D.A. Joy, and M.J. Ciesielki, "Clock period minimization with Wave Pipelining", IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems (Journal Paper), April 1993.

[KEA99] M. Keating, and P. Bricaud, "Reuse Methodology Manual (For System-on-a-Chip Designs)", Les Éditions Kluwer Academic 1999.

[KEC91] G. Kechriotis, and E.S. Manolakos, "A VLSI Architecture for the On-line Training of recurrent Neural Networks", Proc. IEEE Asilomar conference on signals, systems and computers, November 1991, pp. 506-510.

[KOS97] B. Kosko, "Fuzzy Engineering", University of southern california, Edition Prentice Hall, TJ217.5.K67 1997.

[KUN82] H. T. Kung, "Why Systolic Architectures?", IEEE Computer, Jan. 1982, pp. 300-309

[LEE94] E.A. Lee, D. G. Messerschmitt, "Digital Communication" 2ed. Kluwer Academic Publishers, TK5103.7.L44 1994, USA.

[LIE92] W. Lien, and W. Burleson, "Wave-domino logic: timing analysis and applications", 1992 IEEE International Symposium on Circuits and Systems (Conference Paper), 6 vol. 3028 pp. 1992.

[LIU98] X. Liu, T. Adah, and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", IACCASP 98, Seattle, 1998, pp. 1221-1224.

[MAR97] P. Markovic, and D.V. Simic, "A combination of Wave-Pipelining Timing Methodology and DPTL circuit design technique", 21st International Conference on Microelectronics Proceedings MIEL'97, 1997 2vol. xvi+852 pp.

[MAS98] D. Massicotte and B. M. Elouafay, "A Systolic Architecture For Kalman-Filter-Based Signal Reconstruction using the Wave Pipeline Method", 5th IEEE International conference on Electronics, Circuits and Systems (ICECS 98), Lispone, Portugal, 7-10 sept., 1998.

[MAT97] S. Mathew and R. Sridhar, "Efficient Clocking of a Wave-Domino Pipeline", ISCAS 97 4vol. IXVI+2832 pp. (9-12 june 97).

[MEH94] S. Mehrotra, P. Franzon, and W. Liu, "Skew and Delay Minimization of High Speed CMOS Circuits using Stochastic Optimization", IEEE 1994 Custom Integrated Circuits Conference, pp. 11.41-11.4.4, 1994

[MEH96] K Mehrotra, C. K. Mohan, S. Ranka, "Elements of artificial neural networks", Complex adaptive systems/ Bradford Book, MIT Press, QA76.87.M45 1996, Cambrige, Massachusetts.

[MIC92] F. Michaut, "Méthodes adaptatives pour le signal outils mathématiques et mise en œuvre des algorithmes" Edition HERMES, Traité des Nouvelles Technologies, série Traitement du signal

[MOR98] F. Morin, M. Vidal, "Modal: Un processeur de convolution", Projet pour le Groupe de recherche en Électronique Industrielle (GRÉI), Mars 1998, Université du Québec à Trois-Rivières.

[MOR99a] F. Morin, M. Vidal and D. Massicotte, "A High Throughput Architecture for nonlinear channel equalization based on neural network using wave pipeline method", IEEE Canadian Conference on Electrical and Engineering (CCECE'99), Edmonton, Alberta, Canada, 9-12 May 1999.

[MOR99b] F. Morin, M. Vidal et D. Massicotte, "Intégration d'un MAC en utilisant la technique du pipeline par vagues", Congrès de l'ACFAS, Université d'Ottawa, mai 1999.

[MOR99c] F. Morin, M. Vidal et D. Massicotte, "A Wave pipeline Architecture For High Speed Nonlinear Channel Equalization", TEXPO'99, Ottawa, 1999.

[NOO97] S. Nooshabadi, J.A. Montiel-Nelson, G.S. Visweswaran, and D. Nagchoudhurhi, "Micropipeline architecture for multiplier-less FIR filters", Proceedings Tenth International Conference on VLSI Design (Conference Paper) IEEE Comput. Soc. Press, ps Alamitos, CA, USA 1997.

[NOW94] K.J. Nowka, and M.J. Flynn, "Environmental limits on the performance of CMOS Wave-Pipelined circuits", Technical Report CSL-TR-94-600, January 1994.

[NOW94b] K.J. Nowka, and M.J. Flynn, "Wave Pipelining of High Performance CMOS Static RAM", Technical Report CSL-TR-94-615, January 1994.

[NOW95] K. J. Nowka, and M. J. Flynn, "System design using wave-pipelining: a CMOS VLSI vector unit", 1995 IEEE Symposium on Circuits and Systems, NY. USA 1995, 3 vol. 1+2346 pp.

[NOW95b] K. J. Nowka, "high-performance CMOS System Design Using Wave Pipelining", Technical Report CSL-TR-95-XXX, August 1995.

[PAR96] K.K. Parthi, F. Catthoor, R.K. Cavin III, and W. Liu, Design of High Performance DSP Systems", Emerging Technologies Designing Low Power Digital Systems (Conference Paper), 1996 x+516 pp.

[PAR97] R.S. Parthasarathy, and R. Sridhar, "Double Pass Transistor logic for High performance Wave Pipeline Circuits", Proceedings Eleventh International Conference on VLSI Design, 1997 xxxvii+583 pp.

[PAR99] K. K. Parhi, "VLSI digital signal processing systems: design and implementation", A Wiley-Interscience publication, TK7874.75.P37 1999.

[PRO94] J. G. Proakis, M. Salehi, "Communication systems engineering" Prentice hall, Englewood cliff, New Jersey, TK5101.P75 1994.

[PRO95] J.G. Proakis, "Digital communication", McGraw Hill series in Electrical and computer engineering. Communication and signal processing, TK5103.7.P76 1995.

[SAV88] Y. Savaria, "Conception et Vérification des circuits VLSI", Les éditions de l'école Polytechnique de Montréal, 1988.

[SRI96] R. Sridhar, A. Martinez-Smith, and B. McGee, "Speed and Power comparison of CMOS wave pipelined systems and low power WTGL", IEEE International Symposium on Circuits and Systems Circuits and Systems Connecting the world, ISCAS 96, 4 vol.(xlviii+692+801+612+845 pp.), 1998.

[SUT89] I. E. Sutherland, "Micropipelines", Commun. ACM, vol.32, pp. 720-738, June 1989.

[TAK98] K. Takano, T. Sasaki, N. Oba, H. Kobayashi, and T. Nakamura, "Automated design of wave pipelined multiport register files", Proceeding of the ASP DAC'98 Asian and South pacific Design Automation Conference 1998, IEEE, NY., xxxviii+606 pp.

[TAY98] G. S. Taylor, and G. M. Blair, "Reduce Complexity Two-Phase Micropipeline Latch Controller", IEEE Journal of Solid-State Circuits, vol.33, No.10, October 1998.

[UYE88] J. P. Uyemura, "Fundamentals of MOS. digital integrated circuits.", Addison-Wesley Publishing, TK7874.U94 1988.

[VID99] M. Vidal and D. Massicotte, "A VLSI Parallel Architecture of a Piecewise Linear Neural Network for Nonlinear Channel Equalization", Conference IMTC 99, Italie, Mai 1999.

[VID99b] M. Vidal et D. Massicotte, "Algorithme et architectures systoliques pour l'égalisation de canaux", Congrès de l'ACFAS, Université d'Ottawa, mai 1999.

[VID99c] M. Vidal et D. Massicotte, "QPSK Equalizer Based on Piecewise Neural Network", International Conference on Wireless Communications (Wireless'99), Calgary, Juillet 1999.

[VID99d] M. Vidal, " Architecture systolique pour un algorithme basé sur les réseaux de neurones pour l'égalisation de canaux", Mémoire présenté à l'UQTR, Septembre 1999.

[WAN93] L-X, Wang and J. M. Mendel, "An Fuzzy Adaptive Filter, With Application To Nonlinear Channel Equalization", Second IEEE International Conference on Fuzzy Systems, Avril 1993.

[WEI95] L. Wei-Han, and W.P. Burleson, "Wave-Domino logic theory and applications", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing (Journal Paper), Fev. 1995.

[WON92] D.C. Wong, G. De Micheli, M.J. Flynn, and R.E. Huston, "A bipolar population counter using wave pipelining to achieve 2.5* normal clock frequency", IEEE Journal of solid-state Circuits. (Journal Paper), Mai 1992.

[WON93] D.C. Wong, G. De Micheli, and M.J. Flynn, "Designing high performance digital circuits using wave pipelining algorithms and practical experiences", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Journal Paper), Jan 1993.

[YUK97] P. Yuk-Wah, S. Wing-Yun, C. Cheong-Fat, and C. Wai-Kuen, "An asynchronous cell library for self timed system designs", IEICE Transactions on Information and Systems (Journal Paper), Inst. Electron. Inf. & Commun. Eng. March 1997.

[ZAK99] M. Zakhama and D. Massicotte, "A Systolic Architecture for Channel Equalization Based on Piecewise Linear Fuzzy Logic Algorithm", IEEE Canadian Conference on Electrical and Engineering (CCECE'99), Edmonton, Alberta, Canada, 9-12 May 1999.

[ZAR95] M. R. Zargham, "Computer architecture: single and parallel systems", Edition Prentice-Hall, QA76.9.A73Z36, 1995.

Annexe A
(Articles)

[MOR99a] F. Morin, M. Vidal and D. Massicotte, "A High Throughput Architecture for nonlinear channel equalization based on neural network using wave pipeline method", IEEE Canadian Conference on Electrical and Engineering (CCECE'99), Edmonton, Alberta, Canada, 9-12 May 1999.

IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'99)
Edmonton, Alberta, Canada, 9-12 may, 1999

A High Throughput Architecture for Channel Equalization Based on a Neural Network Using a Wave Pipeline Method

Frédéric Morin, Martin Vidal and Daniel Massicotte

Electrical Engineering Department, Université du Québec à Trois-Rivières
Research Group on Industrial Electronics
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
Tel.: 1-(819)-376-5071, Fax : 1-(819)-376-5219
Email : {Frederic_Morin, Martin_Vidal, Daniel_Massicotte}@uqtr.uquebec.ca

ABSTRACT

The use of a wave pipelining method for the design of a systolic architecture dedicated to channel equalization is proposed. A description is given of the piecewise linear multilayer neural network (PL-MNN) algorithm and the architecture. To improve the throughput of the architecture we propose a wave-pipelined version of the multiplier-accumulator (MAC) the presents the bottleneck of the architecture. A 16x8-bit MAC is performed using a Normal Process Complementary Pass Transistor (NPCPL) as a universal cell for the creation of conventional logic gates and is used to optimize the wave pipelined MAC. The throughput and the latency of the MAC have been evaluated at 650 MHz and 8 ns respectively. The performance has been evaluated in a 0.5 µm CMOS technology in comparison with the systolic architecture with and without a conventional pipeline and the proposed wave pipeline structure.

1. INTRODUCTION

The field of communications (e.g. satellites, modems, and cellular phones) often uses methods of compression to increase transmission speed. Nevertheless, these techniques do not satisfy increasing demands for the high throughput communication of data. Thus, channel equalization is interesting because it supplements these methods of digital communication by having a direct impact on the data flow.

On the whole, the forward model of the nonlinear channel can be applied as follows

$$\{\tilde{y}(n)\} = \Im[\{x(n)\}, \Theta] + \eta(n) \qquad (1)$$

where the channel's output $\tilde{y}(n)$ is a scalar corrupted with additive noise $\eta(n)$ and Θ represents the vector of the parameters of the operator $\Im$. The problem of nonlinear channel equalization is to solve the reconstruction problem, which consists of a regularized inversion of the operator $\Im$, i.e., an operator of reconstruction $\Re$

$$\{\hat{x}(n)\} = \Re[\{\tilde{y}(n)\}, \Theta] \qquad (2)$$

The measurement of $\tilde{y}(n)$ leads us to deduce the value of the estimated entry $x(n)$ through a corrective processor called a channel equalizer that makes it possible to obtain $\hat{x}(n)$.

Since data communication is in constant evolution, the need for fast systolic architecture is always in demands and becomes a necessity. To improve the throughput of VLSI architectures [1], [6], the wave pipeline method is proposed to bring higher transmission rate in a nonlinear channel. This method is applied on the VLSI architecture proposed in [6], using the NPCPL cell as basic logic gates [2]. The multiplier-accumulator (MAC) represents the principal combinatory logic block (CLB) unlikely it acts as a bottleneck.

In Section 2, we discuss the adaptive PL-MNN algorithm for channel equalization, and in Section 3, we describe the systolic VLSI architecture dedicated to this algorithm. In Section 4, we present the structure of the MAC and the wave-pipeline technique using the NPCPL cell for universal logic. A comparison study of the architecture both with and without a conventional pipeline and with a wave pipeline is performed in Section 5. Our conclusions are presented in the final section.

2. MULTILAYER NEURAL NETWORK ALGORITHM

There are many algorithm propositions of channel equalization in the literature. Some are for linear channel [3], and others are for a nonlinear channel [4], [7]. However, some VLSI architectures are dedicated to

This work was supported by the Natural Science and Engineering Research Council of Canada and by Le Fonds FCAR Québec.

0-7803-5579-2/99$10.00© 1999 IEEE

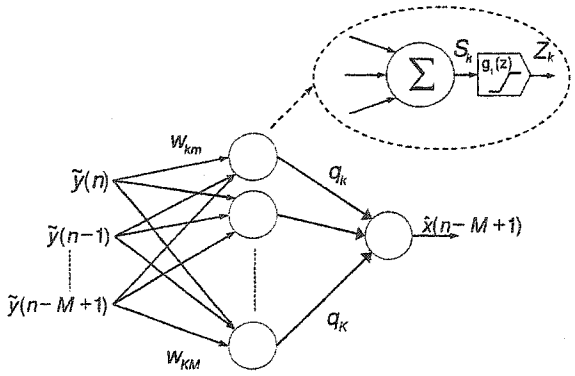


Figure 1: The structure of the neural network (PL-MNN).

nonlinear channel equalization [6]. In the paper [6], a systolic array architecture that uses the recurrent equations of the neural network is proposed. Therefore, these equations were found to have a regular and recursive form produces a local communication of data. Fig. 1 depicts the piecewise linear multilayer neural network (PL-MNN) representation of the algorithm of correction. In the case of an invariant and stationary channel, we need only consider the propagation of the data $\tilde{y}$ through the neural network to get the result of the correction. The following relation represents the algorithm of correction:

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{K} q_k g(s_k)\right) \tag{3}$$

and

$$s_k = \sum_{m=1}^{M} w_{km}\tilde{y}(n-m+1) \text{ for } k = 1,2,...,K \tag{4}$$

where $g(s)$ defines the piecewise linear function:

$$g(s) = \frac{|s+4|-|s-4|}{8} \tag{5}$$

The value of M depends on the frequency characteristics of the channel, and the numbers of hidden neurons K are tuned for optimized algorithm performance.

3. VLSI IMPLEMENTATION OF THE PL-MNN

To obtain a VLSI architecture for channel equalization at high throughput, we used the wave pipeline method on this architecture. The on-line learning will not be included, because of the difficulty of incorporating a recurrence equation in the architecture when using wave pipelining.

Fig. 2 shows the systolic architecture, and the elementary processor (PE_k) [6]. The architecture has an array of $\tilde{y}(n-m)$ for an input with a maximum stage of

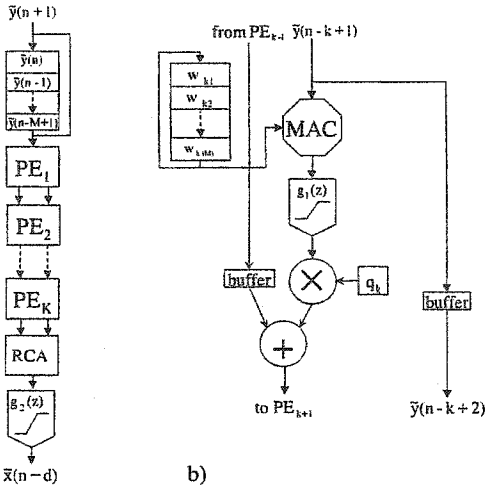


Figure 2: The systolic architecture of PL-MNN algorithm (a) and the processing element PE_k (b).

M+1 delay. The signal $\tilde{y}$ introduced inside the array is loop until the signal $\tilde{y}(n)$ has crossed the PE_1. The last step is to add up the partial products from each elementary processor, before they pass through the activation function. The combinatory logic blocks use a full adder (FA) cell for the main structure.

All PE blocks are identical, and a number of hidden neurons K is fixed to assure the convergence of the equalizer performance. Inside the PE, we must use an array containing the weight w_{km} in the hidden layer since we are using only off-line learning. These will then multiply and accumulate with the output of the channel $\tilde{y}(n)$, where the octagonal shape represents the MAC. Thereafter, s_k will pass through the activation function and give z_k, which is multiplied with the weight of the neural output q_k. Finally, the output will be partially added to the output value of the PE_{k-1}. A study of the data quantization in the scale [-1,1] demonstrates that an 8-bit analog-to-digital converter quantify the input signal $\tilde{y}(n)$ and an 8 bits to 16 bits wordlength for the constants (w_{km}, q_k) and variables $(s_k, z_k, \hat{x}_k)$ data respectively present a good tradeoff for the VLSI implementation [6].

4. THE INTEGRATION OF A WAVE PIPELINED MULTIPLIER-ACCUMULATOR

The wave pipeline, like the conventional pipeline, is an optimization method of the flow inside the CLB. The pipeline technique introduces a number N registers inside a CLB to allow the division of the structure into several small parts, which establishes the order of depth

0-7803-5579-2/99$10.00© 1999 IEEE

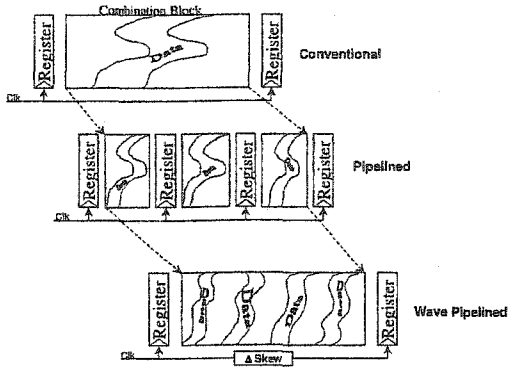


Figure 3: Concept of wave pipeline structure

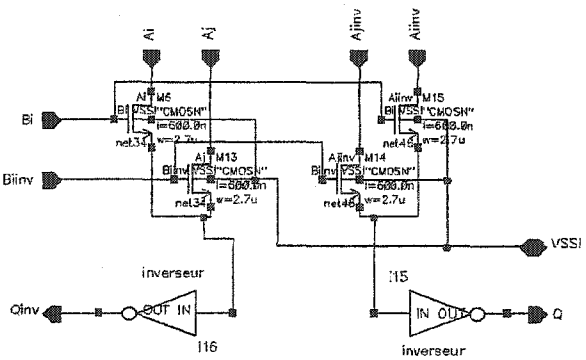


Figure 4: Structure of the NPCPL

of the pipeline. The principle of the wave pipeline introduces inside the CLB a number of waves of data superior to the propagation delay through the combinatory logic. This allows one to reduce the difference between the maximum and minimum propagation delays of the CLB without using internal registers. As result, a concrete reduction of integration surfaces is obtained if we suppose an equivalent performance and an increase in the data flows relative to the number of introduced waves [5]. However, this method of implementation remains complex and requires the use of advanced technology with respect to the delay of transistors (fine-tuning) and connections (coarse tuning). Unlike the conventional pipeline, which aims to minimize the length of path between registers, the wave pipeline aims to equalize it [1]. In other words, the shortest path must be almost equal to the longest. Fig. 3 depicts both types of pipeline methods.

One of the solutions adopted to deal with the problem generated by the transmission of logic is the use of a cell called the NPCPL (Normal Process Complementary Pass Transistor Logic), shown in Fig. 4. Indeed the cell NPCPL makes it possible to balance the delays granted the logical gates as explained in [2]. To apply the wave pipeline method, each logic gate (NAND, AND, OR, NOR, XOR, DELAY, etc) consists of one NPCPL cell. Moreover, the optimization generated by the use of the wave pipeline, unlike the conventional pipeline, makes possible a certain decrease in power consumption, because the conventional pipeline must use a synchronous combinatory logic, which implies a higher number of transitions. Of course, there is the problem of clock skew during its propagation inside the CLB. In a wave pipeline, therefore, we use an intentional clock skew, or a control skews (CS) to reduce the effect of non-desirable path delays. Nevertheless, the addition of a buffer to equalize the paths may increase the surface integration compared with a conventional pipeline.

0-7803-5579-2/99$10.00© 1999 IEEE

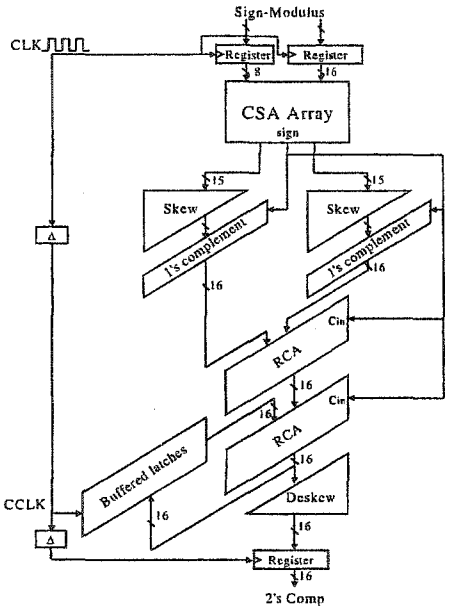


Figure 5: Structure of the MAC

For the suggested architecture, the use of a multiplier-accumulator shown in Fig. 5 makes it possible to increase the data flow associated with the processing of the equalizer. The structure of the MAC permits a visualization of the flow and waves propagation inside the MAC arithmetic. To reduce both the surface area and latency we only use the 16 first significant bits (MSB) during the accumulation feedback. The arithmetic begins in sign-modulus and end in 2's complement. We used a temporal control skew and deskew before and after the arithmetic as proposed in [9]. A modified carry save adder (CSA+) structure used for this implementation is proposed in [2], where Fig. 6 represents the cell used during the multiplier array or CSA array. The regularity and low dependency of data justify the use of CSA array went applying the wave pipeline method. This CSA+

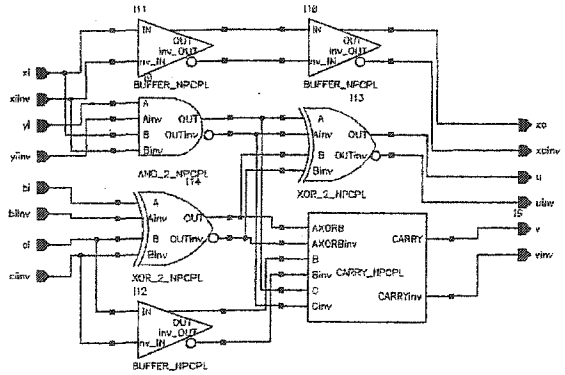


Figure 6: CSA+ structure

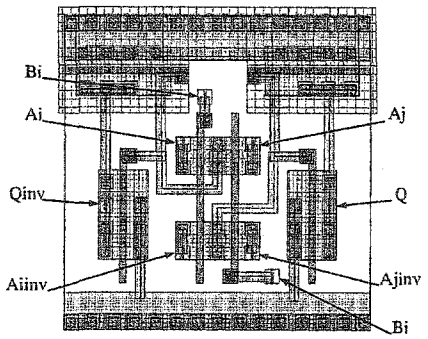


Figure 7: Layout of the NPCPL cell

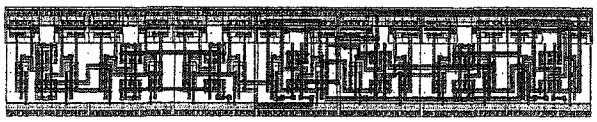


Figure 8: Layout of the CSA+

cell produces an equal path throughout the multiplier structure.

The addition at the end of the partial sum (u, v) is generally called the ripple carry adder (RCA), where the CSA inside are modified so the delay of the sign propagation is equalize. To use the ripple carry adders, we must first pass through a triangular skew made of a singular buffer to synchronize data. Since we are in sign-modulus, we use the sign (MSB) to see if it is necessary to transform the value by a 2's complement conversion if the value is supposed to be negative. The sign value linked to the carry of the RCA will then complete the necessary conversion. The output is then is deskewed by a triangular topology of buffer [9].

The MAC used to resolve Eq. (4), represents the bottleneck of this architecture; therefore, the use of a high throughput method is needed. What constitutes the advantages of this MAC is the use of the carry save adder (CSA) cell instead of the CLB. The cell is defined by the following equation [10]:

$$u = a \oplus b \oplus c \tag{6}$$

$$v = ab + bc + ac \tag{7}$$

where a,b and c are the added bits.

The modified CSA cell use mentioned earlier in [2], used the equations (6) and (7) and added a logical AND for the half adder.

The accumulation is realized one bit at a time since the bits have been delayed by the triangular skewed. Indeed, this technique helps to reduce the data dependency, and permits finer tuning when equalizing the paths for the wave pipeline. The registers used to synchronize the data in the feedback make it possible to use the wave pipelined MAC for different values of clock frequency. Since the CLB requires a certain data dependency, which means that the accumulation of data i and i+1... is necessary, the possibility of a structure without a latch (register) in the feedback is almost impossible. There is one exception where the control of data and clock must be restricted to a special frequency, but it is unacceptable to obtain a low sensitivity to fabrication process. Without data dependency, we believe that our architecture gives a good regularity and the shortest and longest paths are almost equal. So, the shorter path from wave j+1 does not meet the longer path from wave j.

5. Comparison Study and Performance Evaluation

The performance of the proposed architecture was done using a 3.3 V, single poly, triple metal 0.5 µm CMOS technology from the Canadian Microelectronics Corporation (CMC) with Cadence® tools. Simulation of the multiplier-accumulator was performed using a SpectreS™ simulator. To obtain a more accurate level of simulation, the layout of the NPCPL and CSA cells was made, and we have extracted all the parasitic and inter-layer coupling capacitance. Those layouts are represented in Figs. 7 and 8. The surface used by the NPCPL is 745 µm^2 and that for the CSA 5209 µm^2. The NPCPL cell has advantages, but is restrictive. Disadvantages are first, the necessity to give the inverse of all data being treated, and therefore to upset the path delay by one inverter (i.e. $\bar{a}+\bar{b} \Rightarrow (ab)$) from the beginning. Another problem is the limited fanout capability, where each gate cannot drive more then two other gates if we want a substantial fast slope. Those factors are less significant than the advantages of the NPCPL and may be avoided, even if more buffers are needed.

A test was made on a 4×4-bit MAC to lighted-up simulation and tuning time. The output of the MAC,

0-7803-5579-2/99$10.00© 1999 IEEE

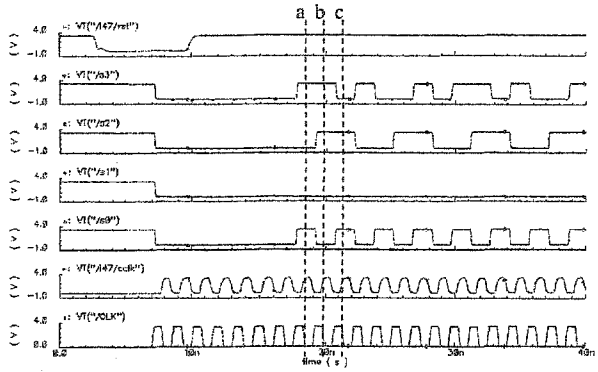


Figure 9: Results of the 4×4-bit MAC with a clock frequency of 650 MHz, where a=1101_b×1100_b+0, b=0101_b×1010_b+a, and c=1101_b×1100_b+b, etc...

s3:s0, is shown in Fig. 9. The master clock (CLK), the reset signal (RESET) and the intentional clock skew (cclk) of the MAC are indicated. We observed from Fig. 9 a good quality of synchronization and shape of the output waves. The speed of the 16×8-bit MAC was simulated at 650 MHz. A comparison study was made with the non-pipeline and conventional pipeline MAC structures. Table I represent the maximum clock frequency, number of transistors needed for each method and latency.

The performance of the systolic architecture (Fig. 2) is evaluated in terms of throughput and latency. The throughput is evaluated to f/M with and without a conventional pipeline and with a wave-pipeline. The latency is evaluated to $(M+K+35)/f$ and $(M+K+5)/f$ with and without a pipelined MAC respectively, and to $M/f+(K+16)\tau_{buf}+24$ ns with the wave-pipelined version, where τ_{buf} is the delay of one NPCPL buffer (τ_{buf}=0.3 ns).

6. Conclusion

In this paper we have presented a wave-pipelined multiplier-accumulator and a systolic architecture of a piecewise linear multilayer neural network for nonlinear channel equalization using 0.5 µm CMOS technology. The properties of recursiveness, regularity, and localized communication of the systolic implementation in an off-line adaptation of the equalizer have made it possible to apply a faster method of optimization of the throughput using the wave-pipelining method. The validation of the proposed 16×8-bit multiplier-accumulator using a NPCPL cell is made using Cadence® tools. The NPCPL cell makes it possible to equalize the paths of the combination logic blocks. The maximum clock rate obtained for a 16×8 bits is 650 MHz and a latency of 8 ns. The number of waves is evaluated at 5 in the MAC structure. The proposed multiplier-accumulator and systolic architectures can be applied in high throughput architectures and communication systems.

Table I: Performance comparison of the 16×8-bits MAC.

	Max. Clock	# Transistors	Latency
Conventional	100 MHz	3500	10 ns
Pipelined*	500 MHz	15000	52 ns
Wave-pipelined	650 MHz	12000	8 ns

* Results obtained with Synopsys® tools without considering the delay of registers and the clock skew, it corresponds to an overestimate.

References

[1] W.P. Burleson, M. Ciesielski, F. Klass, W. Lui, "Wave-Pipelining : A Tutorial and Research Survey", *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol. 6, No. 3, September 1998, pp.464-474.

[2] D. Ghosh, S. K. Nandy, "Design and Realisation of High-Performance Wave-Pipeline 8 × 8 b Multiplier in CMOS Technology", *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 1, March 1995, pp.36-48.

[3] S. Haykin, *Adaptive filter theory*, Prentice Hall, 1991, pp. 342-347;492-497.

[4] G. Kechtiotis, E. Zervas, E. S. Manolakos, "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization", *IEEE Transaction on Neural Networks*, Vol. 5, No. 2, Mars 1994, pp. 267-278.

[5] K. J. Nowka, "High-Performance CMOS System Design Using Wave Pipelining", *Technical Report CSL-TR-95-XXX*, Stanford University, August 1995.

[6] M. Vidal and D. Massicotte, "A Parallel Architecture of a Piecewise Linear Neural Network for Channel Equalization", Accepted for publication in *Instrumentation and Measurement Technology Conference (IMTC'99)*, Venice, May 1999.

[7] X. Liu, T. Adah and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", *International Conference on Acoustics, Speech and Signal Processing (ICASSP'98)*, Seattle, 1998, pp.1221-1224.

[8] D. Massicotte and My B. Elouafay, "A Systolic Architecture for Kalman-Filter-Based Signal Reconstruction Using the Wave Pipeline Method", *5th IEEE International Conference on Electronics, Circuits and Systems (ICECS'98)*, Portugal, Lisbon, 7-10 Sept 1998, pp.199-202.

[9] S-J. Jou, C-H. Chen, E-C. Yang, and C-C. Su, "A Pipelined Multiplier-Accumulator Using a High-Speed, Low-Power Static and Dynamic Full Adder Design", *IEEE Journal of solid-state circuits*, Vol. 32, No. 1, January 1997.

[10] V.K. Madisetti, "VLSI Digital Signal Processors: An introduction to Rapid Prototyping and Design Synthesis", *IEEE Press*, 1995.

Annexe B
(Modèle d'architecture en VHDL)

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                      --
--                  Exigence Partielle a la Memoire de Maitrise         --
--                                                                      --
--                                                                      --
--------------------------------------------------------------------------


--------------------------------------------------------------------------
--
--   File:     adder_rtl.vhd
--
--   Purpose:
--             Additioneur
--
--   Author: Frederic Morin
--
--   Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;

-- CONNECTIVITY DEFINITION
ENTITY adder IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;

    data_in           : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    data_in2          : IN  std_logic_vector(bit_int-1 DOWNTO 0);

   -- OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END adder;



-- IMPLEMENTATION
ARCHITECTURE rtl OF adder IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss          : std_logic;
  SIGNAL vdd          : std_logic;
  SIGNAL vss_vector   : std_logic_vector(15 DOWNTO 0);
```

```
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);

BEGIN

--------------------------------------------------------------------

 -- Static signal
  vss               <= '0';
  vdd               <= '1';
  vss_vector        <= (OTHERS => '0');
  vdd_vector        <= (OTHERS => '1');

--------------------------------------------------------------------



  ------------------------------------------------------------------------
  --  Process:  Effectue une addition partiel
  --  Purpose:
  ------------------------------------------------------------------------
  adder_proc : PROCESS(rstb, clk)
    VARIABLE data_var : signed(bit_int-1 DOWNTO 0);


  BEGIN

    IF (rstb /= '1') THEN
      data_var      := (OTHERS => '0');
      data_out      <= (OTHERS => '0');

    ELSIF (clk'EVENT AND clk = '1') THEN
      data_var      := (OTHERS => '0');


      data_var      := (SIGNED(data_in2) + SIGNED(data_in));

      data_out      <= STD_LOGIC_VECTOR(data_var);
    END IF;

  END PROCESS;

END rtl;


--
-- END OF CODE
--
```

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                      --
--                  Exigence Partielle a la Memoire de Maitrise         --
--                                                                      --
--                                                                      --
--------------------------------------------------------------------------


--------------------------------------------------------------------------
--
--    File:      adder_rca_rtl.vhd
--
--    Purpose:
--              Additioneur RCA (Ripple Carry Adder)
--
--    Author: Frederic Morin
--
--    Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
--

--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY adder_rca IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;

    data_in           : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    canal_in          : IN  std_logic_vector(bit_io-1 DOWNTO 0);

   -- OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END adder_rca;


-- IMPLEMENTATION
ARCHITECTURE rtl OF adder_rca IS
```

```
 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss           : std_logic;
  SIGNAL vdd           : std_logic;
  SIGNAL vss_vector    : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);


BEGIN

-------------------------------------------------------------------------

 -- Static signal
  vss                 <= '0';
  vdd                 <= '1';
  vss_vector          <= (OTHERS => '0');
  vdd_vector          <= (OTHERS => '1');


-------------------------------------------------------------------------
  ---------------------------------------------------------------------------
  --  Process:  Effectue une addition partiel
  --  Purpose:
  ---------------------------------------------------------------------------
  adder_proc : PROCESS(rstb, clk)
    VARIABLE data_var : signed(bit_int-1 DOWNTO 0);


  BEGIN

    IF (rstb /= '1') THEN
      data_var       := (OTHERS => '0');
      data_out       <= (OTHERS => '0');

    ELSIF (clk'EVENT AND clk = '1') THEN
      data_var       := (OTHERS => '0');


      data_var       := (SIGNED(canal_in) + SIGNED(data_in));

      data_out       <= STD_LOGIC_VECTOR(data_var);
    END IF;

  END PROCESS;

END rtl;


--
-- END OF CODE
--
```

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                      --
--                Exigence Partielle a la Memoire de Maitrise           --
--                                                                      --
--                                                                      --
--------------------------------------------------------------------------


--------------------------------------------------------------------------
--
--   File:      mac_blackbox.vhd
--
--   Purpose:
--      Boite noire du MAC pipeline par vagues, sera remplace par
--      la macro durant layout. Effectue les operations basic et ajoute
--      la latence.
--
--
--
--
--   Author: Frederic Morin
--
--   Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
--
--
--


--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY mac_wavepipeline IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;
    canal_in          : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    poids_in          : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    neg_in            : IN  std_logic;

   -- OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
```

```
END mac_wavepipeline;


-- IMPLEMENTATION
ARCHITECTURE blackbox OF mac_wavepipeline IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss           : std_logic;
  SIGNAL vdd           : std_logic;
  SIGNAL vss_vector    : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);

  SIGNAL poids_mem     : std_logic_vector(bit_int-1 DOWNTO 0);
  SIGNAL data_mem      : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_s    : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_ss   : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_sss  : signed(bit_int-1 DOWNTO 0);


BEGIN

-------------------------------------------------------------------------

 -- Static signal
  vss               <= '0';
  vdd               <= '1';
  vss_vector        <= (OTHERS => '0');
  vdd_vector        <= (OTHERS => '1');

 -- Signal Intermedaire
  poids_mem         <= (vss_vector(7 DOWNTO 0) & poids_in);

-------------------------------------------------------------------------



  ---------------------------------------------------------------------
  --  Process:  Effectue une multiplication et accumule les retards
  --  Purpose:
  ---------------------------------------------------------------------
  mac_proc : PROCESS(rstb, clk)


  BEGIN

    -- synopsys synthesis_off
    IF (rstb /= '1') THEN
      data_mem      <= (OTHERS => '0');
      data_mem_s    <= (OTHERS => '0');
      data_mem_ss   <= (OTHERS => '0');
      data_mem_sss  <= (OTHERS => '0');
      data_out      <= (OTHERS => '0');

    ELSIF (clk'EVENT AND clk = '1') THEN
```

```
        data_mem       <= (SIGNED(canal_in) * SIGNED(poids_mem)) + data_mem;

        data_mem_s     <= data_mem;
        data_mem_ss    <= data_mem_s;
        data_mem_sss   <= data_mem_ss;
        data_out       <= STD_LOGIC_VECTOR(data_mem_sss);

    END IF;

    -- synopsys synthesis_on

  END PROCESS;

END blackbox;


--
-- END OF CODE
--
```

```
------------------------------------------------------------------------
--                    UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                    --
--                                                                    --
--                                                                    --
--                  Exigence Partielle a la Memoire de Maitrise       --
--                                                                    --
------------------------------------------------------------------------


------------------------------------------------------------------------
--
--   File:     mult_blackbox.vhd
--
--   Purpose:
--          Boite noire du MULT pipeline par vagues, sera remplace par
--         la macro durant layout. Effectue les operations elementaires
--         et ajoute la latence du pipeline.
--
--
--
--
--   Author: Frederic Morin
--
--   Version: 1.0
--
------------------------------------------------------------------------

-- NOTES :
--
--
--


--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY mult_wavepipeline IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;
    canal_in          : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    gains_in          : IN  std_logic_vector(bit_io-1  DOWNTO 0);

   -- OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
```

```
END mult_wavepipeline;


-- IMPLEMENTATION
ARCHITECTURE blackbox OF mult_wavepipeline IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss           : std_logic;
  SIGNAL vdd           : std_logic;
  SIGNAL vss_vector    : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);

  SIGNAL gains_mem     : std_logic_vector(bit_int-1 DOWNTO 0);
  SIGNAL data_mem      : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_s    : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_ss   : signed(bit_int-1 DOWNTO 0);
  SIGNAL data_mem_sss  : signed(bit_int-1 DOWNTO 0);


BEGIN

-------------------------------------------------------------------------

 -- Static signal
  vss               <= '0';
  vdd               <= '1';
  vss_vector        <= (OTHERS => '0');
  vdd_vector        <= (OTHERS => '1');

 -- Signal Intermedaire
  gains_mem         <= (vss_vector(7 DOWNTO 0) & gains_in);

-------------------------------------------------------------------------



  ---------------------------------------------------------------------
  --  Process: Effectue une multiplication du gain et du signal corrige
  --  Purpose:
  ---------------------------------------------------------------------
  mult_proc : PROCESS(rstb, clk)


  BEGIN

    -- synopsys synthesis_off
    IF (rstb /= '1') THEN
      data_mem      <= (OTHERS => '0');
      data_mem_s    <= (OTHERS => '0');
      data_mem_ss   <= (OTHERS => '0');
      data_mem_sss  <= (OTHERS => '0');

      data_out      <= (OTHERS => '0');
```

```
    ELSIF (clk'EVENT AND clk = '1') THEN

      data_mem      <= (SIGNED(canal_in) * SIGNED(gains_mem));

      data_mem_s    <= data_mem;
      data_mem_ss   <= data_mem_s;
      data_mem_sss  <= data_mem_ss;

      data_out      <= STD_LOGIC_VECTOR(data_mem_sss);

    END IF;


    -- synopsys synthesis_on

  END PROCESS;

END blackbox;


--
-- END OF CODE
--
```

```
-------------------------------------------------------------------------
--                    UNIVERSITE DU QUEBEC A TROIS-RIVIERES            --
--                  Exigence Partielle a la Memoire de Maitrise        --
-------------------------------------------------------------------------


-------------------------------------------------------------------------
--
--   File:     piecewise_rtl.vhd
--
--   Purpose:
--         Structure de la fonction de decision utilisant la canonique
--         lineaire par morceaux (piecewise lineaire)
--
--
--   Author: Frederic Morin
--
--   Version: 1.0
--
-------------------------------------------------------------------------

-- NOTES :
--
--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY piecewise IS
  PORT(
   --INPUTs
    data_in           : IN  std_logic_vector(bit_int-1 DOWNTO 0);

   --OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END piecewise;



-- IMPLEMENTATION
ARCHITECTURE rtl OF piecewise IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss          : std_logic;
  SIGNAL vdd          : std_logic;
  SIGNAL vss_vector   : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector   : std_logic_vector(15 DOWNTO 0);
```

```
BEGIN

-------------------------------------------------------------------------

 -- Static signal
  vss                   <= '0';
  vdd                   <= '1';
  vss_vector            <= (OTHERS => '0');
  vdd_vector            <= (OTHERS => '1');


-------------------------------------------------------------------------


  ---------------------------------------------------------------------
  --  Process:  effectue la fonction de decision par morceaux.
  --  Purpose:  Logique Combinatoire
  ---------------------------------------------------------------------
  pl_proc : PROCESS(data_in)
    VARIABLE   data_var       : signed(bit_int-1 DOWNTO 0);

  BEGIN
    IF   (data_in(bit_int-1 DOWNTO bit_int-12) = ("111111111111"))
      OR (data_in(bit_int-1 DOWNTO bit_int-12) = ("000000000000")) THEN

      data_var       := (OTHERS => '0');

    ELSE

      data_var       := SHIFT_LEFT(signed(data_in),2);  --Divise par 4

    END IF;

      data_out <= STD_LOGIC_VECTOR(data_var);

  END PROCESS;

END rtl;


--
-- END OF CODE
--
```

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                      --
--                    Exigence Partielle a la Memoire de Maitrise       --
--                                                                      --
--------------------------------------------------------------------------


--------------------------------------------------------------------------
--
--   File:     pile_circ_rtl.vhd
--
--   Purpose:
--          Structure d'une pile circulaire memorisant les retards de
--         l'entree ou des poids de chaque cellule.
--
--
--
--
--   Author: Frederic Morin
--
--   Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
--
--
--


--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY pile_circ IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;
    canal_in          : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    ready_on          : IN  std_logic;

   -- OUTPUTs
    int               : OUT std_logic;
    canal_out         : OUT std_logic_vector(bit_io-1 DOWNTO 0)
    );
END pile_circ;
```

```
-- IMPLEMENTATION
ARCHITECTURE rtl OF pile_circ IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss           : std_logic;
  SIGNAL vdd           : std_logic;
  SIGNAL vss_vector    : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);

  SIGNAL data_feed     : std_logic_vector(bit_io-1 DOWNTO 0);
  SIGNAL data_mem      : array_nbr_retardxbit_io;
  SIGNAL data_out_o    : std_logic_vector(bit_io-1 DOWNTO 0);


BEGIN

-------------------------------------------------------------------------

 -- Static signal
  vss                  <= '0';
  vdd                  <= '1';
  vss_vector           <= (OTHERS => '0');
  vdd_vector           <= (OTHERS => '1');

  canal_out            <= data_out_o;
-------------------------------------------------------------------------


  -------------------------------------------------------------------------
  --  Process:  Control programmation des piles (poids et retard)
  --  Purpose:
  -------------------------------------------------------------------------
  pile_proc : PROCESS(rstb, clk)
  BEGIN
    IF (rstb /= '1') THEN

      data_feed  <= (OTHERS => '0');
      data_out_o <= (OTHERS => '0');
      data_mem   <= (OTHERS =>(OTHERS => '0'));
      int        <= '0';


    ELSIF (clk'EVENT AND clk = '1') THEN

      IF (ready_on = '1') THEN
        data_feed <= canal_in;
        int       <= '1';

      ELSE
        data_feed <= data_out_o;
        int       <= '0';
```

```
      END IF;

      FOR i IN 1 TO nbr_retard LOOP
        IF (i = 1) THEN
          data_mem(i) <= data_feed;
        ELSIF (i < nbr_retard) THEN
          data_mem(i) <= data_mem(i-1);
        ELSIF (i = nbr_retard) THEN
          data_out_o  <= data_mem(i);
        ELSE
          data_mem    <= data_mem;
        END IF;
        data_mem  <= data_mem;
      END LOOP;
    END IF;
  END PROCESS;
END rtl;

--
-- END OF CODE
--
```

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                    Exigence Partielle a la Memoire de Maitrise       --
--------------------------------------------------------------------------


--------------------------------------------------------------------------
--
--    File:      proc_elem_rtl.vhd
--
--    Purpose:
--               Architecture Processeur Elementaire PL-MNN
--
--
--    Author: Frederic Morin
--
--    Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
--
--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY proc_elem IS
  PORT(
   -- INPUTs
    rstb              : IN  std_logic;
    clk               : IN  std_logic;
    data_in           : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    canal_in          : IN  std_logic_vector(bit_io-1  DOWNTO 0);
    addr_set          : IN  unsigned(2 DOWNTO 0);
    address           : IN  std_logic_vector(2 DOWNTO 0);
    enable_prg        : IN  std_logic;
    q_wb              : IN  std_logic;

   -- OUTPUTs
    data_out          : OUT std_logic_vector(bit_int-1 DOWNTO 0);
    canal_out         : OUT std_logic_vector(bit_io-1  DOWNTO 0);
    ready_on          : OUT std_logic
    );
END proc_elem;


-- IMPLEMENTATION
```

```
ARCHITECTURE rtl OF proc_elem IS


 -- LOCAL SIGNAL DECLARATION
  SIGNAL vss           : std_logic;
  SIGNAL vdd           : std_logic;
  SIGNAL vss_vector    : std_logic_vector(15 DOWNTO 0);
  SIGNAL vdd_vector    : std_logic_vector(15 DOWNTO 0);

 -- Pile signals
  SIGNAL pile_rdy      : std_logic;
  SIGNAL pile_int      : std_logic;
  SIGNAL pile_out      : std_logic_vector(bit_io-1 DOWNTO 0);
  SIGNAL poids_in      : std_logic_vector(bit_io-1 DOWNTO 0);


 -- MAC signals
  SIGNAL mac_out       : std_logic_vector(bit_int-1 DOWNTO 0);


 -- MULT signals
  SIGNAL gains_in      : std_logic_vector(bit_io-1  DOWNTO 0);
  SIGNAL mult_out      : std_logic_vector(bit_int-1 DOWNTO 0);

 -- PIECEWISE signals
  SIGNAL piece_out     : std_logic_vector(bit_int-1 DOWNTO 0);


 -- SAMPLE OUTPUT
  SIGNAL canal_in_s    : std_logic_vector(bit_io-1  DOWNTO 0);
  SIGNAL data_in_s     : std_logic_vector(bit_int-1 DOWNTO 0);


-- COMPONENTS DECLARATION
COMPONENT pile_circ
  PORT(
   -- INPUTs
    rstb               : IN  std_logic;
    clk                : IN  std_logic;
    canal_in           : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    ready_on           : IN  std_logic;

   -- OUTPUTs
    int                : OUT std_logic;
    canal_out          : OUT std_logic_vector(bit_io-1 DOWNTO 0)
    );
END COMPONENT;


COMPONENT mac_wavepipeline
  PORT(
   -- INPUTs
    rstb               : IN  std_logic;
    clk                : IN  std_logic;
    canal_in           : IN  std_logic_vector(bit_io-1 DOWNTO 0);
```

```
    poids_in            : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    neg_in              : IN  std_logic;

   -- OUTPUTs
    data_out            : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END COMPONENT;



COMPONENT piecewise
  PORT(
   -- INPUTs
    data_in             : IN  std_logic_vector(bit_int-1 DOWNTO 0);

   -- OUTPUTs
    data_out            : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END COMPONENT;



COMPONENT mult_wavepipeline
  PORT(
   -- INPUTs
    rstb                : IN  std_logic;
    clk                 : IN  std_logic;
    canal_in            : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    gains_in            : IN  std_logic_vector(bit_io-1  DOWNTO 0);

   -- OUTPUTs
    data_out            : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END COMPONENT;



COMPONENT adder
  PORT(
   -- INPUTs
    rstb                : IN  std_logic;
    clk                 : IN  std_logic;

    data_in             : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    data_in2            : IN  std_logic_vector(bit_int-1 DOWNTO 0);

   -- OUTPUTs
    data_out            : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END COMPONENT;
```

```
BEGIN

--------------------------------------------------------------------

 -- Static signal
  vss                   <= '0';
  vdd                   <= '1';
  vss_vector            <= (OTHERS => '0');
  vdd_vector            <= (OTHERS => '1');


--------------------------------------------------------------------
-- Lors de la programmation des poids et gains pass data direct au
-- prochaine PEk
--
--------------------------------------------------------------------
  canal_out  <= canal_in WHEN (enable_prg = '1') ELSE canal_in_s;



--------------------------------------------------------------------

  pile_circ_inst: pile_circ
    PORT MAP(
     -- INPUTs
      rstb              =>  rstb,
      clk               =>  clk,
      canal_in          =>  poids_in,

      ready_on          =>  pile_rdy,

     -- OUTPUTs
      int               =>  pile_int,
      canal_out         =>  pile_out
    );


  mac_inst: mac_wavepipeline
    PORT MAP(
     -- INPUTs
      rstb              =>  rstb,
      clk               =>  clk,
      canal_in          =>  canal_in,
      poids_in          =>  pile_out,
      neg_in            =>  vss,

     -- OUTPUTs
      data_out          =>  mac_out
    );


  piecewise_inst: piecewise
    PORT MAP(
     -- INPUTs
      data_in           => mac_out,
```

```
    -- OUTPUTs
     data_out            => piece_out
     );


mult_inst: mult_wavepipeline
  PORT MAP(
    -- INPUTs
     rstb                => rstb,
     clk                 => clk,
     canal_in            => piece_out,
     gains_in            => gains_in,

    -- OUTPUTs
     data_out            => mult_out
     );


adder_inst: adder
  PORT MAP(
    -- INPUTs
     rstb                => rstb,
     clk                 => clk,
     data_in             => mult_out,
     data_in2            => data_in_s,

    -- OUTPUTs
     data_out            => data_out
     );


-------------------------------------------------------------------------
--  Process:  Control programmation des piles (poids et gains)
--  Purpose:
-------------------------------------------------------------------------
pile_proc : PROCESS(rstb, clk)
BEGIN
  IF (rstb /= '1') THEN
    gains_in  <= (OTHERS => '0');
    poids_in  <= (OTHERS => '0');
    ready_on  <= '0';
    pile_rdy  <= '0';


  ELSIF (clk'EVENT AND clk = '1') THEN

    IF (enable_prg = '1') THEN

      IF (unsigned(address) = addr_set) THEN

        IF (q_wb = '1') THEN
          gains_in  <= canal_in;
          ready_on  <= '1';
```

```
          ELSE
            poids_in  <= canal_in;
            pile_rdy  <= '1';
            ready_on  <= pile_int;

          END IF;

        ELSE
          pile_rdy <= '0';
          ready_on <= '0';

        END IF;

      ELSE
        pile_rdy <= '0';
        ready_on <= '0';

      END IF;

    END IF;

  END PROCESS;


  ------------------------------------------------------------------------
  --  Process:  sample signals
  --  Purpose:
  ------------------------------------------------------------------------
  sample_proc : PROCESS(rstb, clk)
  BEGIN
    IF (rstb /= '1') THEN
      data_in_s  <= (OTHERS => '0');
      canal_in_s <= (OTHERS => '0');


    ELSIF (clk'EVENT AND clk = '1') THEN
      data_in_s  <= data_in;
      canal_in_s <= canal_in;

    END IF;

  END PROCESS;

END rtl;


--
-- END OF CODE
--
```

```
--------------------------------------------------------------------------
--                      UNIVERSITE DU QUEBEC A TROIS-RIVIERES           --
--                                                                      --
--                    Exigence Partielle a la Memoire de Maitrise       --
--                                                                      --
--------------------------------------------------------------------------

--------------------------------------------------------------------------
--
--    File:      wave_core_arch.vhd
--
--    Purpose:
--               Architecture top_level du reseau PL-MNN
--
--
--
--
--    Author: Frederic Morin
--
--    Version: 1.0
--
--------------------------------------------------------------------------

-- NOTES :
--
--
--
-- LIBRARIES
--
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.numeric_std.ALL;

LIBRARY wave_lib;
  USE wave_lib.wave_pkg.ALL;


-- CONNECTIVITY DEFINITION
ENTITY wave_core IS
  PORT(
    rstb                : IN std_logic;
    sysclk              : IN std_logic;

    canal_in            : IN std_logic_vector(bit_io-1 DOWNTO 0);
    enable_prg          : IN std_logic;
    address             : IN std_logic_vector(2 DOWNTO 0);
    q_wb                : IN std_logic;

    canal_out           : OUT std_logic_vector(bit_io-1 DOWNTO 0);
    int                 : OUT std_logic
    );
END wave_core;



-- IMPLEMENTATION
ARCHITECTURE rtl OF wave_core IS
```

```
 -- LOCAL SIGNAL DECLARATION
  SIGNAL data_x        : array_nbr_cellxbit_int;  -- Define in package
  SIGNAL canal_x       : array_nbr_cellxbit_io;   -- Define in package
  SIGNAL ready_x       : std_logic_vector(1 TO nbr_cell);

  SIGNAL int_out       : std_logic;
  SIGNAL data_out      : std_logic_vector(bit_io-1 DOWNTO 0);

  SIGNAL rca_out       : std_logic_vector(bit_int-1 DOWNTO 0);
  SIGNAL piece_out     : std_logic_vector(bit_int-1 DOWNTO 0);

  SIGNAL results_o     : std_logic_vector(bit_io-1 DOWNTO 0);

  SIGNAL num_pe        : array_nbr_cellxunsigned;


-- COMPONENTS DECLARATION
COMPONENT proc_elem
  PORT(
   -- INPUTs
    rstb               : IN  std_logic;
    clk                : IN  std_logic;
    data_in            : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    canal_in           : IN  std_logic_vector(bit_io-1  DOWNTO 0);
    addr_set           : IN  unsigned(2 DOWNTO 0);
    address            : IN  std_logic_vector(2 DOWNTO 0);
    enable_prg         : IN  std_logic;
    q_wb               : IN  std_logic;

   -- OUTPUTs
    data_out           : OUT std_logic_vector(bit_int-1 DOWNTO 0);
    canal_out          : OUT std_logic_vector(bit_io-1  DOWNTO 0);
    ready_on           : OUT std_logic
    );
END COMPONENT;


COMPONENT adder_rca
  PORT(
   --INPUTs
    rstb               : IN  std_logic;
    clk                : IN  std_logic;
    data_in            : IN  std_logic_vector(bit_int-1 DOWNTO 0);
    canal_in           : IN  std_logic_vector(bit_io-1  DOWNTO 0);

   --OUTPUTs
    data_out           : OUT std_logic_vector(15 DOWNTO 0)
    );
END COMPONENT;


COMPONENT piecewise
  PORT(
   --INPUTs
```

```
    data_in             : IN  std_logic_vector(bit_int-1 DOWNTO 0);

   --OUTPUTs
    data_out            : OUT std_logic_vector(bit_int-1 DOWNTO 0)
    );
END COMPONENT;


COMPONENT pile_circ
  PORT(
   --INPUTs
    rstb                : IN  std_logic;
    clk                 : IN  std_logic;
    canal_in            : IN  std_logic_vector(bit_io-1 DOWNTO 0);
    ready_on            : IN  std_logic;

   --OUTPUTs
    int                 : OUT std_logic;
    canal_out           : OUT std_logic_vector(bit_io-1 DOWNTO 0)
    );
END COMPONENT;


BEGIN

-------------------------------------------------------------------------
 -- Static signal

  data_x(0)             <= all_zero_bus(bit_int);
-------------------------------------------------------------------------

  pile_circ_inst: pile_circ
    PORT MAP(
   --INPUTs
    rstb                =>  rstb,
    clk                 =>  sysclk,
    canal_in            =>  canal_in,

    ready_on            =>  ready_x(1),

   --OUTPUTs
    int                 =>  int_out,
    canal_out           =>  canal_x(0)
    );


gen_tsbs : FOR i IN 1 TO 3 GENERATE

      num_pe(i)         <=  to_unsigned(i,3);

      proc_elem_inst: proc_elem
      PORT MAP(
       -- INPUTs
        rstb            =>  rstb,
        clk             =>  sysclk,
        data_in         =>  data_x(i-1),
        canal_in        =>  canal_x(i-1),
```

```
        addr_set        =>  num_pe(i),
        address         =>  address,
        enable_prg      =>  enable_prg,
        q_wb            =>  q_wb,

       -- OUTPUTs
        data_out        =>  data_x(i),
        canal_out       =>  canal_x(i),
        ready_on        =>  ready_x(i)
        );

END GENERATE;


  rca_inst: adder_rca
    PORT MAP(
     --INPUTs
      rstb              =>  rstb,
      clk               =>  sysclk,

      data_in           =>  data_x(nbr_cell),
      canal_in          =>  canal_x(nbr_cell),

     --OUTPUTs
      data_out          =>  rca_out
      );


  piecewise_inst: piecewise
    PORT MAP(
     --INPUTs
      data_in           =>  rca_out,

     --OUTPUTs
      data_out          =>  piece_out
      );


  ------------------------------------------------------------------------
  --  Process:  sample output signals
  --  Purpose:
  ------------------------------------------------------------------------
  sample_proc : PROCESS (rstb, sysclk)

    VARIABLE int_var : std_logic;

  BEGIN

    IF (rstb /= '1') THEN
      int_var       := '0';
      results_o     <= (OTHERS => '0');
      int           <= '0';

    ELSIF (sysclk'EVENT AND sysclk = '1') THEN
      int_var       := '0';
```

```
      IF (enable_prg = '1') THEN

        FOR i in 1 TO nbr_cell LOOP
          int_var  := (int_var OR ready_x(i));
        END LOOP;

        results_o <= canal_x(nbr_cell);

      ELSE

        int_var   := int_out;
        results_o <= piece_out(7 DOWNTO 0);

      END IF;

      int  <= int_var;

    END IF;

  END PROCESS;


  canal_out <= results_o(bit_io-1 DOWNTO 0);

END rtl;


--
-- END OF CODE
--
```

Annexe C
(Fichiers de commande et contrainte)

```
#! /usr/local/bin/tcsh -f
##############################################################################
#
# FILE NAME: do_compile_rtl.cmd
# AUTHOR: Frederic Morin
# DATE:
#
# $Author: morinfre $
# $Date: Sun Oct 20 11:56:29 2002 $
# $Locker: morinfre $
# $Revision: 1.1 $
#
# INVOCATION SYNTAX:
# APP. RUN TIME:
# FUNCTION:
# INPUT FILES:
# OUTPUT FILES:
#
##############################################################################
### Variables
##-----------------------------------
set LOG=./${0:r}.log
set SOURCE="../rtl"
set LIBS="wave_lib"
set COMP_OPTS=" -errormax 8 -messages -work $LIBS -linedebug"
##-----------------------------------

### Clear Log file
##-----------------------------------
if ( -e $LOG ) then
     echo " removing old $LOG file"
     rm -f $LOG
endif
##-----------------------------------


### Compile rtl list
##-----------------------------------
set FILES= (                    \
             wave_pkg.vhd       \
             adder_rtl.vhd      \
             adder_rca_rtl.vhd  \
             mac_blackbox.vhd   \
             mult_blackbox.vhd  \
             piecewise_rtl.vhd  \
             pile_circ_rtl.vhd  \
             proc_elem_rtl.vhd  \
             wave_core_arch.vhd \
           )
##-----------------------------------

echo "Compiling RTL files.............started on `date`" | tee $LOG


foreach FILE ($FILES)
```

```
   if (-e ${SOURCE}/${FILE} ) then
      echo "\n ncvhdl => ${SOURCE}/${FILE}"
      ncvhdl $COMP_OPTS ${SOURCE}/${FILE}      >> $LOG
      echo "          "
   else
      echo "No ${FILE} found in ${SOURCE}/..."
   endif
end


#clear
echo "                  "
echo " _______________ "
echo " TOTAL of ERRORS "
echo " _______________ "
echo "                  "
egrep -i "error|.vhd"  $LOG

#echo " $0 is over on the `date` on $HOST..." | tee -a $LOG | mailx -s
"$0" -m `whoami`
#####################################################


#--EOF
```

```
#! /usr/local/bin/tcsh -f

##########################################################################
# FILE NAME: do_synth_core_tcl.cmd
# AUTHOR: Frederic Morin
##########################################################################
### Variables
##---------------------------------
set LOG="./${0:r}.log"
set SYN_DIR="../syn"
set LIBS="wave_lib"
set LOG_DIR="./log"
set SOURCE="../rtl"
##---------------------------------

### Clear Log file
##---------------------------------
if ( -e $LOG ) then
     echo " removing old $LOG file"
     rm -f $LOG
endif
##---------------------------------


### Synth list
##---------------------------------
set FILES= ( \
             adder_rtl       \
             adder_rca_rtl   \
             mult_blackbox   \
             mac_blackbox    \
             piecewise_rtl   \
             pile_circ_rtl   \
             proc_elem_rtl   \
             wave_core_arch  \
           )
##---------------------------------

echo "Synthesizing RTL files using DC-CL.started on `date`" | tee $LOG
echo "Log files are located in the $LOG_DIR dir......." | tee -a $LOG


foreach FILE ($FILES)
   set DC_SCRIPT="${SYN_DIR}/${FILE}.dctcl"
   set LOGFILE="${LOG_DIR}/${FILE}.log"
   set VHDL_FILE="${SOURCE}/${FILE}.vhd"

   if ( -e $DC_SCRIPT ) then
      if ( -e $VHDL_FILE ) then
        echo "Running $DC_SCRIPT..."                | tee -a $LOG
        dc_shell-t >$LOGFILE  <<cat_end

        #
        # Get a VHDL-Compiler license and hold it for
        # the duration of the synthesis.
        # This is OK, since the license is required for
        # analyze/elaborate/compile commands
```

```
        # (i.e. it is required for every step of the
        # synthesis process).
        #
        set license_list [list VHDL-Compiler]
        foreach lic license_list {
           while { {true} } {
              set license_availability [ get_license [list lic] ]
              if {license_availability=="1"} {
                 break
              } else {
                 echo [format "Waiting for %s license" lic]
                 sh sleep 30
              }
           }
        }
        source $DC_SCRIPT

cat_end

      else
        echo "No $VHDL_FILE"                    | tee -a $LOG
      endif
   else
      echo "Can't find $DC_SCRIPT"              | tee -a $LOG
   endif
end

echo "                    "                     | tee -a $LOG
echo " ____________________________ "           | tee -a $LOG
echo " TOTAL of ERRORS in LOG file "            | tee -a $LOG
echo " ____________________________ "           | tee -a $LOG
echo "                    "                     | tee -a $LOG
egrep -i 'error'  $LOG                          | tee -a $LOG


echo "                    "                     | tee -a $LOG
echo " ____________________________ "           | tee -a $LOG
echo " TOTAL of ERRORS all syn LOG file "       | tee -a $LOG
echo " ____________________________ "           | tee -a $LOG
echo "                    "                     | tee -a $LOG
foreach FILE ($FILES)

   egrep -i 'error'  ${LOG_DIR}/$FILE.log       | tee -a $LOG

end

egrep -i 'error'  ../syn/tsb_ecbi/*.log         | tee -a $LOG


#echo " $0 is over on the `date` on $HOST..." | tee -a $LOG | mailx -s
"$0" -m `whoami`
################################################
##EOF
```

```
##########################################################################
##
##   File: adder_rtl.dctcl
##
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

# Load VHDL
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/adder_rtl.vhd}]
elaborate adder -arch rtl -lib wave_lib

reset_design
source ../syn/generic.dctcl

create_clock -period 1.5 -name clk [get_ports clk]
set_dont_touch_network [get_ports clk]
set_drive 0 [get_ports clk]
set_resistance 0 [get_nets clk]
set_load 0 -subtract_pin_load [get_nets clk]

set_drive 0 [get_ports rstb]
set_dont_touch_network [get_ports rstb]
set_ideal_net [get_nets rstb]
set_resistance 0 [get_nets rstb]
set_load 0 -subtract_pin_load [get_nets rstb]

set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

set_case_analysis 1 [get_ports rstb]

compile -map_effort low

redirect         ../syn/adder_gates.rpt { report_area }
redirect -append ../syn/adder_gates.rpt { report_timing -path end }

ungroup -all -flatten

compile -map_effort high  -verify_effort high -area_effort high
#
# Generate reports
#
redirect -append ../syn/adder_gates.rpt { report_area }
redirect -append ../syn/adder_gates.rpt { report_reference }
redirect -append ../syn/adder_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/adder_gates.rpt { report_timing -path end }
redirect -append ../syn/adder_gates.rpt { check_design }

change_names -rules pmc_vhdl -hierarchy

write -hierarchy -format db -output ../syn/db/adder_gates.db

::CAD::setup::remove_license {VHDL-Compiler}
quit
```

```
##########################################################################
##   File: adder_rca_rtl.dctcl
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

# Load VHDL
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/adder_rca_rtl.vhd}]
elaborate adder_rca -arch rtl -lib wave_lib

reset_design
source ../syn/generic.dctcl

create_clock -period 1.5 -name clk [get_ports clk]
set_dont_touch_network [get_ports clk]
set_drive 0 [get_ports clk]
set_resistance 0 [get_nets clk]
set_load 0 -subtract_pin_load [get_nets clk]

set_drive 0 [get_ports rstb]
set_dont_touch_network [get_ports rstb]
set_ideal_net [get_nets rstb]
set_resistance 0 [get_nets rstb]
set_load 0 -subtract_pin_load [get_nets rstb]

set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

set_case_analysis 1 [get_ports rstb]
compile -map_effort low
redirect         ../syn/adder_rca_gates.rpt { report_area }
redirect -append ../syn/adder_rca_gates.rpt { report_timing -path end }
ungroup -all -flatten
compile -map_effort high  -verify_effort high -area_effort high
# Generate reports
redirect -append ../syn/adder_rca_gates.rpt { report_area }
redirect -append ../syn/adder_rca_gates.rpt { report_reference }
redirect -append ../syn/adder_rca_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/adder_rca_gates.rpt { report_timing -path end }
redirect -append ../syn/adder_rca_gates.rpt { check_design }

change_names -rules pmc_vhdl -hierarchy

write -hierarchy -format db -output ../syn/db/adder_rca_gates.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/adder_rca_gates.vhd

# Release the VHDL-Compiler license
#
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##
##    File: mac_blackbox.dctcl
##
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

#
# Load VHDL
#
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/mac_blackbox.vhd}]
elaborate mac_wavepipeline -arch blackbox -lib wave_lib

reset_design
source ../syn/generic.dctcl

set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

# set_case_analysis 1 [get_ports rstb]

compile -map_effort low

redirect          ../syn/mac_blackbox.rpt { report_area }
redirect -append ../syn/mac_blackbox.rpt { report_timing -path end }

# ungroup -all -flatten

compile -map_effort high -verify_effort high -area_effort high
#
# Generate reports
#
redirect -append ../syn/mac_blackbox.rpt { report_area }
redirect -append ../syn/mac_blackbox.rpt { report_reference }
redirect -append ../syn/mac_blackbox.rpt { report_constraint -
all_violators }
redirect -append ../syn/mac_blackbox.rpt { report_timing -path end }
redirect -append ../syn/mac_blackbox.rpt { check_design }

# ungroup -all -flatten

change_names -rules pmc_vhdl -hierarchy


write -hierarchy -format db -output ../syn/db/mac_blackbox.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/mac_blackbox.vhd

# Release the VHDL-Compiler license
#
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##
##   File: mult_blackbox.dctcl
##
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}
# Load VHDL
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/mult_blackbox.vhd}]
elaborate mult_wavepipeline -arch blackbox -lib wave_lib

reset_design
source ../syn/generic.dctcl

# create_clock -period 1.5 -name clk [get_ports clk]
# set_clock_uncertainty $standard_clock_skew_plus_uncertainty

#
set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

# set_case_analysis 1 [get_ports rstb]

compile -map_effort low

redirect         ../syn/mult_blackbox.rpt { report_area }
redirect -append ../syn/mult_blackbox.rpt { report_timing -path end }

# ungroup -all -flatten

compile -map_effort high  -verify_effort high -area_effort high

# Generate reports
redirect -append ../syn/mult_blackbox.rpt { report_area }
redirect -append ../syn/mult_blackbox.rpt { report_reference }
redirect -append ../syn/mult_blackbox.rpt { report_constraint -
all_violators }
redirect -append ../syn/mult_blackbox.rpt { report_timing -path end }
redirect -append ../syn/mult_blackbox.rpt { check_design }

change_names -rules pmc_vhdl -hierarchy

write -hierarchy -format db -output ../syn/db/mult_blackbox.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/mult_blackbox.vhd

# Release the VHDL-Compiler license
#
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##   File: piecewise_rtl.dctcl
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

# Load VHDL
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/piecewise_rtl.vhd}]
elaborate piecewise -arch rtl -lib wave_lib


reset_design
source ../syn/generic.dctcl


set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

# set_case_analysis 1 [get_ports rstb]

compile -map_effort low


redirect         ../syn/piecewise_gates.rpt { report_area }
redirect -append ../syn/piecewise_gates.rpt { report_timing -path end }



# ungroup -all -flatten

compile -map_effort high  -verify_effort high -area_effort high

#
# Generate reports
#
redirect -append ../syn/piecewise_gates.rpt { report_area }
redirect -append ../syn/piecewise_gates.rpt { report_reference }
redirect -append ../syn/piecewise_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/piecewise_gates.rpt { report_timing -path end }
redirect -append ../syn/piecewise_gates.rpt { check_design }

change_names -rules pmc_vhdl -hierarchy


write -hierarchy -format db -output ../syn/db/piecewise_gates.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/piecewise_gates.vhd

# Release the VHDL-Compiler license
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##   File: pile_circ_rtl.dctcl
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

# Load VHDL
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/pile_circ_rtl.vhd}]
elaborate pile_circ -arch rtl -lib wave_lib

reset_design
source ../syn/generic.dctcl

create_clock -period 1.5 -name clk [get_ports clk]
set_dont_touch_network [get_ports clk]
set_drive 0 [get_ports clk]
set_resistance 0 [get_nets clk]
set_load 0 -subtract_pin_load [get_nets clk]

set_drive 0 [get_ports rstb]
set_dont_touch_network [get_ports rstb]
set_ideal_net [get_nets rstb]
set_resistance 0 [get_nets rstb]
set_load 0 -subtract_pin_load [get_nets rstb]

set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl
set_case_analysis 1 [get_ports rstb]
compile -map_effort low

redirect         ../syn/pile_circ_gates.rpt { report_area }
redirect -append ../syn/pile_circ_gates.rpt { report_timing -path end }

# ungroup -all -flatten
compile -map_effort high  -verify_effort high -area_effort high
# Generate reports
redirect -append ../syn/pile_circ_gates.rpt { report_area }
redirect -append ../syn/pile_circ_gates.rpt { report_reference }
redirect -append ../syn/pile_circ_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/pile_circ_gates.rpt { report_timing -path end }
redirect -append ../syn/pile_circ_gates.rpt { check_design }

change_names -rules pmc_vhdl -hierarchy

write -hierarchy -format db -output ../syn/db/pile_circ_gates.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/pile_circ_gates.vhd

# Release the VHDL-Compiler license
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##   File: proc_elem_rtl.dctcl
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}
# Read in Leaf Cells
#
read_file -format db ../syn/db/mac_blackbox.db
read_file -format db ../syn/db/mult_blackbox.db
read_file -format db ../syn/db/adder_gates.db
read_file -format db ../syn/db/piecewise_gates.db
read_file -format db ../syn/db/pile_circ_gates.db

#
# Load VHDL
#
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/proc_elem_rtl.vhd}]
elaborate proc_elem -arch rtl -lib wave_lib


reset_design
source ../syn/generic.dctcl


create_clock -period 1.5 -name clk [get_ports clk]
# set_clock_uncertainty $standard_clock_skew_plus_uncertainty [get_clocks
clk]
set_dont_touch_network [get_ports clk]
set_drive 0 [get_ports clk]
set_resistance 0 [get_nets clk]
set_load 0 -subtract_pin_load [get_nets clk]


set_drive 0 [get_ports rstb]
set_dont_touch_network [get_ports rstb]
set_ideal_net [get_nets rstb]
set_resistance 0 [get_nets rstb]
set_load 0 -subtract_pin_load [get_nets rstb]


set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

set_case_analysis 1 [get_ports rstb]


set_dont_touch [get_designs mac_wavepipeline]
set_dont_touch [get_designs mult_wavepipeline]
set_dont_touch [get_designs adder]
set_dont_touch [get_designs pile_circ]
set_dont_touch [get_designs piecewise]

compile -map_effort low


redirect         ../syn/proc_elem_gates.rpt { report_area }
redirect -append ../syn/proc_elem_gates.rpt { report_timing -path end }
```

```
# ungroup -all -flatten

compile -map_effort high  -verify_effort high -area_effort high

#
# Generate reports
#
redirect -append ../syn/proc_elem_gates.rpt { report_area }
redirect -append ../syn/proc_elem_gates.rpt { report_reference }
redirect -append ../syn/proc_elem_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/proc_elem_gates.rpt { report_timing -path end }
redirect -append ../syn/proc_elem_gates.rpt { check_design }
#
change_names -rules pmc_vhdl -hierarchy


write -hierarchy -format db -output ../syn/db/proc_elem_gates.db

# set vhdlout_write_entity true
# set vhdlout_write_architecture true
# write -hierarchy -format vhdl -output ../gates/proc_elem_gates.vhd


# Release the VHDL-Compiler license
#
::CAD::setup::remove_license {VHDL-Compiler}

quit
```

```
##########################################################################
##    File: wave_core_arch.dctcl
##########################################################################
::CAD::setup::get_license {VHDL-Compiler}

#
# Read in Leaf Cells
#
read_file -format db ../syn/db/adder_rca_gates.db
read_file -format db ../syn/db/piecewise_gates.db
read_file -format db ../syn/db/pile_circ_gates.db
read_file -format db ../syn/db/proc_elem_gates.db


#
# Load VHDL
#
analyze -format vhdl -lib wave_lib [list {../rtl/wave_pkg.vhd}]
analyze -format vhdl -lib wave_lib [list {../rtl/wave_core_arch.vhd}]
elaborate wave_core -arch rtl -lib wave_lib


reset_design
source ../syn/generic.dctcl


create_clock -period 1.5 -name clk [get_ports sysclk]
# set_clock_uncertainty $standard_clock_skew_plus_uncertainty [get_clocks
sysclk]
set_dont_touch_network [get_ports sysclk]
set_drive 0 [get_ports sysclk]
set_resistance 0 [get_nets sysclk]
set_load 0 -subtract_pin_load [get_nets sysclk]


set_drive 0 [get_ports rstb]
set_dont_touch_network [get_ports rstb]
set_ideal_net [get_nets rstb]
set_resistance 0 [get_nets rstb]
set_load 0 -subtract_pin_load [get_nets rstb]


set_dont_touch [get_designs proc_elem]
set_dont_touch [get_designs pile_circ]
set_dont_touch [get_designs adder_rca]
set_dont_touch [get_designs piecewise]


set_max_area 0
set_wire_load_model -library wlm_op_cond -name medium_wl

set_case_analysis 1 [get_ports rstb]


compile -map_effort low
```

```
redirect         ../syn/wave_core_gates.rpt { report_area }
redirect -append ../syn/wave_core_gates.rpt { report_timing -path end }


# ungroup -all -flatten

compile -map_effort high  -verify_effort high -area_effort high

#
# Generate reports
#
redirect -append ../syn/wave_core_gates.rpt { report_area }
redirect -append ../syn/wave_core_gates.rpt { report_reference }
redirect -append ../syn/wave_core_gates.rpt { report_constraint -
all_violators }
redirect -append ../syn/wave_core_gates.rpt { report_timing -path end }
redirect -append ../syn/wave_core_gates.rpt { check_design }


change_names -rules pmc_vhdl -hierarchy

write -hierarchy -format db -output ../syn/db/wave_core_gates.db

set vhdlout_write_entity true
set vhdlout_write_architecture true
write -hierarchy -format vhdl -output ../gates/wave_core_gates.vhd



# Release the VHDL-Compiler license
#
::CAD::setup::remove_license {VHDL-Compiler}

quit
```