ÉCOLE DE TECHNOLOGIE SUPÉIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE
LA MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. ING.

PAR
HASSAN CHERRAJ

# CONCEPTION ET RÉALISATION D'UN DISPOSITIF DE PRÉTRAITEMENT DE SIGNAUX RADAR EN UTILISANT LE VHDL

MONTRÉAL, 29 MARS 2001

# CONCEPTION ET RÉALISATION D'UN DISPOSITIF DE PRÉTRAITEMENT DE SIGNAUX RADAR EN UTILISANT LE VHDL

Hassan Cherraj

## Sommaire

Au cours des dernières décennies, les méthodes de conception des fonctions numériques ont subi une évolution importante. Dans le passé, la majorité des applications était élaborée autour des circuits intégrés standards. Plus récemment, les circuits programmables ont beaucoup attiré l'attention par leurs propriétés multiples. Les outils d'aide à la conception se sont unifiés contribuant ainsi à l'essor de telles technologies.

Notre projet se situe dans le cadre de l'exploitation de cette évolution, en collaboration avec le centre de recherche de la Défense à Ottawa et la compagnie Lockeed Martin Canada, pour la réalisation d'une unité de prétraitement des signaux radar.

Notre travail, dans ce projet, consiste à concevoir un circuit numérique pour la détection des impulsions radar et l'estimation de leur temps d'arrivée, en utilisant une description en langage VHDL pour cibler une technologie de type FPGA.

Au cours de ce travail, nous proposons trois structures pour la réalisation de l'estimateur. Elles se distinguent par le type d'interpolation et la méthode de la résolution de l'équation qui en découle. Les trois solutions ont été simulées par le logiciel VSS de SYNOPSYS, synthétisées en vue de cibler deux familles de circuits FPGA, XC4000 et VIRTEX de la firme XILINX. Les trois solutions proposées satisfont les spécifications. Les résultats de réalisation nous permettent d'analyser la complexité et les performances des trois stratégies.

CE MÉMOIRE A ÉTÉ ÉVALUÉ PAR UN JURY COMPOSÉ DE :

- M. Claude Thibeault, directeur de mémoire
  Département de génie électrique, École de technologie supérieure

- M. Jean Belzile, codirecteur
  Département de génie électrique, École de technologie supérieure

- M. François Gagnon, professeur
  Département de génie électrique, École de technologie supérieure

- M. Naïm Batani, Professeur
  Département de génie électrique, École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT CE JURY ET UN PUBLIC

LE 9 MARS 2001

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## REMERCIEMENTS

Je dédie ce mémoire aux membres de ma famille : Mon épouse Bahija, mes chers enfants Fadwa, Mohaned-Amine et Réda Ismail pour leur soutien et leurs encouragements qui m'ont permis de mener à terme mes études de Maîtrise.

Je témoigne toute ma reconnaissance envers mon professeur et directeur Claude Thibeault et mon codirecteur Jean Belzile pour la qualité de leur encadrement leur disponibilité.

Enfin, j'exprime toute ma reconnaissance envers mon DIEU de m'avoir donné la force nécessaire pour la réalisation de mon travail.

## LISTE DES TABLEAUX

Page

## LISTE DES FIGURES

Page

## LISTE DES ABRÉVIATIONS

| | |
|---|---|
| AGC | Automatic Gain Control |
| ASIC | Application-Specific Integrated Circuit |
| CLB | Configurable Logic Block |
| CMOS | Complementary Metal Oxide Semiconductor |
| DLL | Digital Looked Loop |
| DREO | Defense Research Establishment of Ottawa |
| DRC | design Rule Check |
| EPROM | Erasable-Programmable Read Only Memory |
| FPGA | Field Programmable Gate Array |
| GCK | Global Clock |
| GRM | Global Routing Matrix |
| IOB | Input Output Block |
| LACIME | Laboratoire de Communications et Intégration de la Micro-Électronique |
| LUT | Look Up Table |
| LC | Logic Cell |
| MH | Nombre d'échantillons nécessaires pour valider le début d'une impulsion |
| ML | Nombre d'échantillons nécessaires pour valider la fin d'une impulsion |
| NGD | Native generic Database |
| PLD | Programmable Logic Device |
| PAL | Programmable Array Logic |
| PLA | Programmable Logic Array |
| PTOA | Pulse Time of Arrival |
| RADAR | Radio Detection And Ranging |
| RAM | Random Access Memory |
| RTL | Register Transfer Logic |

| | |
|---|---|
| ROM | Read Only Memory |
| SRAM | Static Random Access Memory |
| TH1 | Seuil de détection du front montant |
| TH2 | Seuil de détection du front descendant |
| TTL | Transistor-Transistor Logic |
| VSS | VHDL System Simulation |
| VHSIC | Very High Speed Integrated circuit |
| VHDL | VHSIC High Description Language |
| VLSI | Very Large Scale Integration |

## TABLES DES MATIERES

ANNEXES :

# INTRODUCTION

## Historique

Les premières applications de la radioélectricité furent les télécommunications, puis la radionavigation. Le développement des moyens de transports aériens et maritimes, tant civils que militaires a fait rapidement apparaître le besoin de détecter la direction et la position d'un objet. Le radar a été développé pour détecter la présence d'objets. Le mot R.A.D.A.R désigne une abréviation de l'expression anglaise : 'RAdio Detection And Ranging'.

Depuis les années quarante, le progrès dans ce domaine n'a pas cessé de croître à cause du deuxième conflit mondial au départ, et de la conquête de l'espace par la suite. Actuellement, le radar nous côtoie dans la vie de tous les jours : prévisions météorologiques, contrôle de vitesse, surveillance, localisation, etc...

En général, le radar est composé d'une antenne d'émission rayonnant une onde électromagnétique générée par l'émetteur, et d'une antenne de réception alimentant un récepteur. Une fraction de l'onde émise par le radar est interceptée par l'objet à détecter, qui à son tour, la rayonne dans toutes les directions. Une fraction de l'onde réfléchie est captée par l'antenne réceptrice, qui alimente le récepteur où s'effectue le traitement du signal, en vue d'obtenir certaines informations sur la cible : direction, distance, vitesse...

Les possibilités offertes par le radar ont conduit à de nombreuses applications et réalisations dans le domaine civil et militaire. Plusieurs recherches ont été menées dans ce domaine. Le projet qui nous concerne s'inscrit dans cette optique.

## Objectifs

Ce projet s'inscrit dans le cadre d'une collaboration avec le Centre de Recherche pour la Défense Nationale à Ottawa, et la société Lockheed Martin Canada. Il consiste en la conception et la réalisation d'un dispositif de prétraitement des impulsions radar. L'architecture globale du système est représentée par la figure suivante :

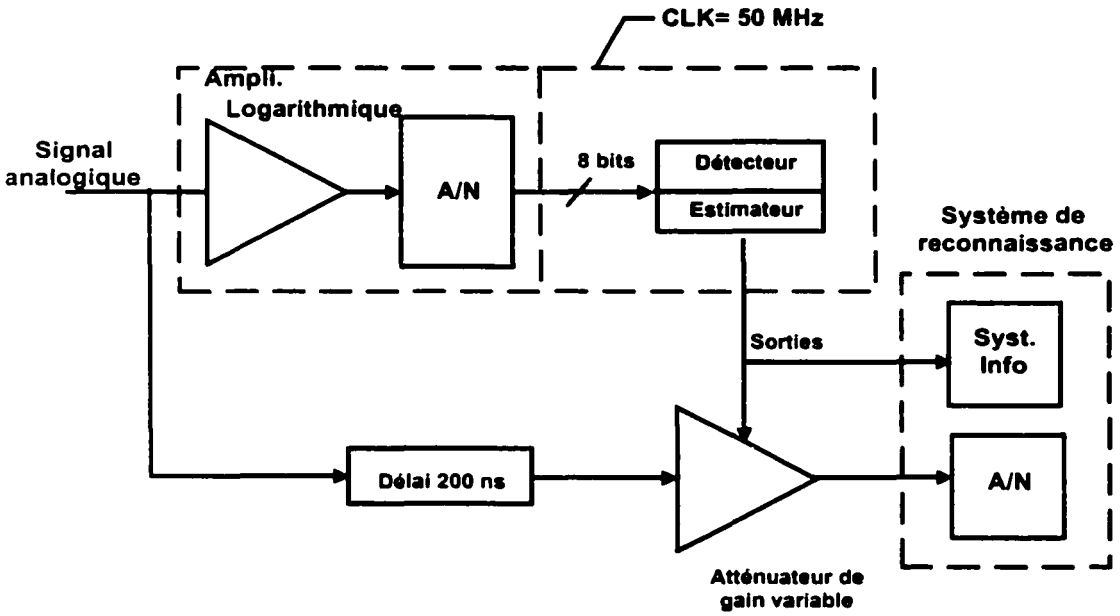


Figure 1 Principe de l'unité de prétraitement

Le signal analogique à l'entrée du système est amplifié par un amplificateur vidéo. L'amplificateur sert à réduire la plage dynamique du signal. Le nombre de bits nécessaires pour représenter les différents niveaux du signal quantifié à l'entrée du convertisseur analogique à numérique, sera ainsi réduit. Par conséquent, la complexité du circuit numérique à réaliser sera diminuée. Le circuit détecte les impulsions, les caractérise et estime leur temps d'arrivée. Il génère par la suite un signal de commande pour l'amplificateur à gain variable, qui agira sur le signal analogique retardé par une

ligne à délai de 200 ns. Les caractéristiques de l'impulsion sont aussi envoyées vers le système numérique de reconnaissance, ce qui augmentera la fiabilité de cette opération.

Mon travail dans ce projet est de traduire en VHDL et de réaliser un circuit numérique pour la détection des impulsions et l'estimation de leur temps d'arrivée. Ce travail devrait se baser sur celui déjà accompli en utilisant le logiciel Viewlogic, tout en explorant différentes stratégies pour la réalisation de l'estimateur. La description du circuit est effectuée en vue de cibler une technologie FPGA.

Le circuit traitera des mots de 8 bits en format non signé. Sa fréquence d'horloge sera de 50 Mhz. La commande de l'amplificateur à gain variable devrait avoir une latence maximale de 200 ns, l'estimation du temps d'arrivée des impulsions devrait avoir une précision de 1 ns en absence de bruit.

**Contributions**

À l'issue de ce projet, j'aurai contribué à mettre en application deux nouvelles stratégies pour la réalisation matérielle du circuit d'estimation. La première stratégie consiste en une interpolation linéaire de l'impulsion autour d'un seuil adaptatif. La résolution nécessite le calcul d'une division de deux nombres. Pour réduire la complexité du circuit, le calcul de la division a été remplacé par le calcul de l'inverse du dénominateur suivi d'un produit du résultat par le numérateur. La seconde consiste en une interpolation quadratique de l'impulsion. Le temps d'arrivée des impulsions est estimé en comparant la valeur de la quadratique, en N points de l'intervalle entre deux échantillons, au seuil adaptatif.

J'aurai également développé, une expertise au niveau de la réalisation de circuits numériques complexes en utilisant le langage VHDL pour cibler une technologie de type

FPGA, et en explorant les différents problèmes inhérents aux divers outils de développements mis à ma disposition.

**Contenu du rapport**

Ce rapport est composé de quatre chapitres : introduction, principes de détection et d'estimation, description VHDL du circuit, technologie de réalisation, résultats d'implémentation ainsi qu'une conclusion.

L'introduction débute par un historique sur les systèmes radar, et définit ensuite les objectifs fixés et la contribution du projet.

Le premier chapitre, principes de détection et d'estimation, consiste à expliquer les principes de base de fonctionnement du détecteur et de l'estimateur; il résume ensuite le travail déjà effectué, ainsi que les raisons qui ont mené au choix des méthodes d'interpolation et les stratégies de réalisation.

Le deuxième chapitre, description VHDL du circuit, introduit le langage VHDL, il explique ensuite la hiérarchie de l'implantation des fonctions, ainsi que leurs interactions.

Le troisième chapitre, technologie de réalisation, introduit les différents types de solutions disponibles pour la réalisation d'un circuit numérique. L'accent sera mis sur les architectures des circuits programmables qui ont été retenus pour la réalisation du circuit.

Le quatrième chapitre, résultats d'implémentation, présente les résultats obtenus pour les différentes stratégies, ainsi que la comparaison des résultats obtenus par Viewlogic et ceux obtenus par VHDL.

Le rapport se termine par une conclusion, et par des recommandations pour les travaux futurs.

# CHAPITRE I

# PRINCIPES DE DÉTECTION ET D'ESTIMATION

## 1.1 Introduction

Ce chapitre traite de la description fonctionnelle du circuit numérique à réaliser. Au début, on fera une mise en situation du circuit dans son environnement extérieur. Elle sera suivie par une description détaillée des deux modules principaux : le détecteur et l'estimateur, ainsi que les différents signaux et paramètres mis en jeux. Enfin, on abordera les différentes méthodes d'interpolation ainsi que les différentes stratégies de résolution de l'équation de l'estimateur.

## 1.2 Environnement du circuit

La figure 1-1 donne une vue d'ensemble du circuit numérique à concevoir ainsi que des différents paramètres et des signaux d'échange avec les modules externes composant le reste de l'unité de prétraitement.

Les données de 8 bits à la sortie du convertisseur analogique à numérique sont fournis au détecteur et à l'estimateur. Le détecteur détermine la validité, l'amplitude, le début et la fin de l'impulsion. La valeur du maximum servira pour le calcul du gain à l'aide d'une table de correspondance LUT ('Look Up Table') qui fournira la valeur servant à contrôler l'atténuateur à gain variable. Les paragraphes suivants détailleront les différents éléments et leur fonctionnement.

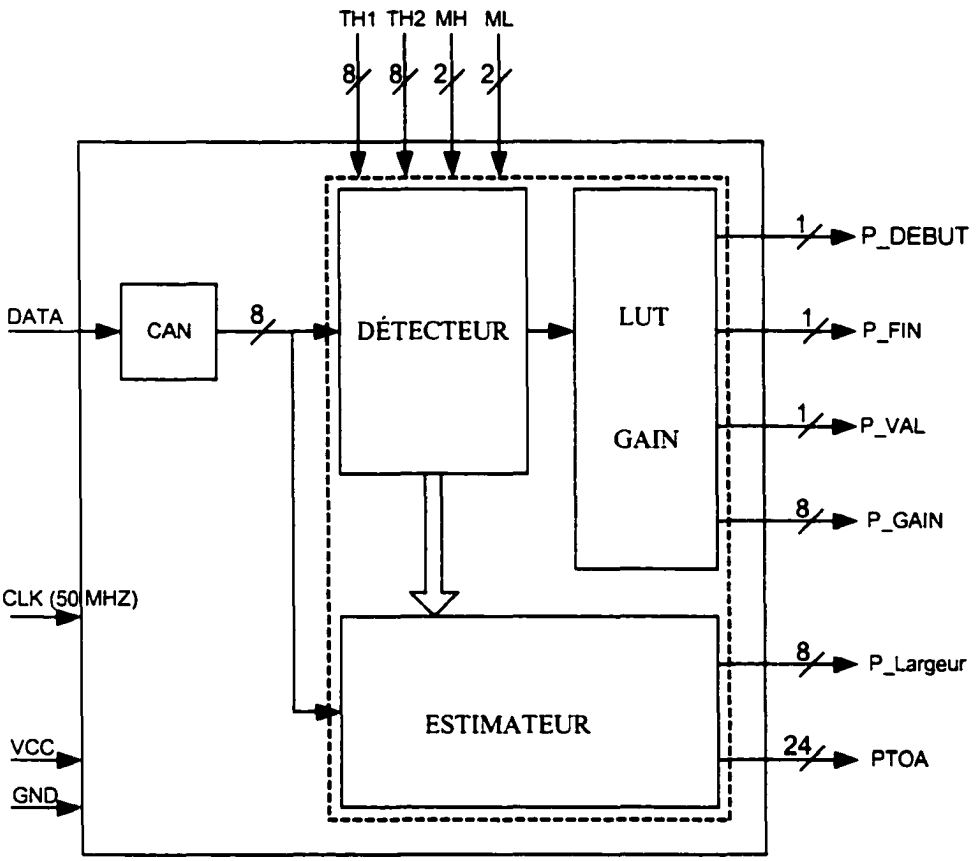


Figure 1-1 Environnement du circuit

## 1.3 Détecteur

Son rôle est de différentier les véritables impulsions du bruit, et de caractériser l'impulsion en détectant son début, sa fin ainsi que son amplitude maximale. L'amplitude maximale servira à déterminer la commande de l'atténuateur à gain variable, via la table de correspondance, participant ainsi à la réalisation du contrôle automatique du gain sur le signal analogique retardé.

La détection des impulsions est basée sur l'utilisation de deux seuils de comparaison TH1 et TH2. Les échantillons formant l'impulsion sont comparés au premier seuil TH1 pour détecter le front montant. La même opération avec TH2 permet la détection du front descendant. Pour diminuer les effets du bruit et réduire le nombre des fausses alarmes, l'impulsion n'est considérée valide que si un nombre déterminé d'échantillons successifs est supérieur à TH1. Suite à la validation de l'impulsion, la fin de l'impulsion est effective après avoir détecter un nombre défini d'échantillons en dessous du seuil TH2. Le processus utilisé est un processus probabiliste, puisque la décision est prise sur un signal bruité. Il existe une certaine probabilité sur l'exactitude des décisions. On est en présence de trois cas de figures :

- Les fausses alarmes : en l'absence du signal utile, une rafale de bruit entraîne une fausse décision de détection d'impulsion.
- Les détections manquées : l'amplitude du signal est suffisamment plus petite que TH1, l'impulsion sera considérée comme du bruit, entraînant ainsi une fausse décision.
- Décision correcte en présence du signal

Le paragraphe suivant donne la définition des différents paramètres et signaux utilisés dans le détecteur.

### 1.3.1 Paramètres de détection

La figure 1-2 met en évidence les différents paramètres et signaux utilisés pour la détection des impulsions

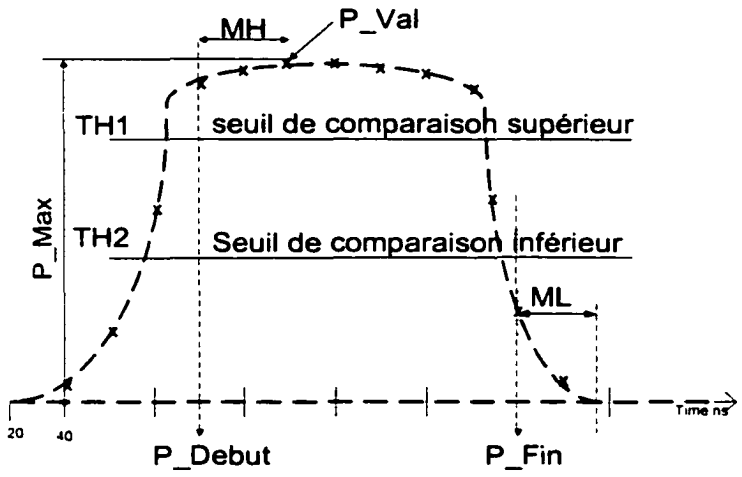


Figure 1-2 Paramètres du détecteur

***- Seuil de comparaison supérieur : TH1 (8 bits)***

Ce paramètre, réglable par l'utilisateur, est un mot de 8 bits en format non signé. Il permet la détection du début de l'impulsion.

***- Seuil de comparaison inférieur : TH2 (8bits)***

Ce paramètre, réglable par l'utilisateur, est un mot de 8 bits en format non signé. Il permet la détection de la fin de l'impulsion.

***- Début de l'impulsion : P_DEBUT (1 bit)***

C'est un signal de sortie sur 1 bit. Son passage à '1' durant un cycle de l'horloge permet de signaler le début de l'impulsion.

***- Fin de l'impulsion : P_FIN (1 bit)***

C'est un signal de sortie sur 1 bit. Son passage à '1' durant un cycle de l'horloge permet de signaler la fin de l'impulsion.

***- Nombre d'échantillons de validation : MH (2 bits)***

Ce paramètre, réglable par l'utilisateur, est un mot de 2 bits. Il fixe le nombre d'échantillons, successifs supérieurs à TH1, nécessaires pour déclarer l'impulsion valide. Il permettra de diminuer le nombre des fausses alarmes dues au bruit.

***- Nombre d'échantillons du front descendant : ML (2 bits)***

Ce paramètre, réglable par l'utilisateur, est un mot de 2 bits. Il fixe le nombre d'échantillons successifs inférieurs à TH2, nécessaires pour déclarer l'impulsion valide.

***- Amplitude maximale : P_MAX (8 bits)***

Ce signal de sortie sur 8 bits en format non signé, correspond à la valeur maximale des amplitudes des MH+1 échantillons consécutifs supérieurs au seuil maximum TH2. Il servira au calcul du seuil de comparaison de l'estimateur et au contrôle automatique du gain du signal analogique retardé.

***- Validation : P_VAL (1 bit)***

Ce signal peut prendre la valeur "1" une fois par cycle de détection. À l'aide de ce signal, le détecteur confirme l'existence de MH+1 échantillons consécutifs supérieurs au seuil TH2. Il servira de référence pour toutes les opérations du système.

### 1.3.2 Contrôle automatique du gain

La figure 1-3 illustre le délai de réaction du système pour effectuer le contrôle automatique du gain. Le circuit de détection placera sa commande de gain 200 ns après le passage du signal P_VAL à "1". La commande du gain est maintenue pendant une durée égale à la largeur estimée par le détecteur. La valeur de commande, appelée GAIN

est reliée au maximum détecté par une relation définie par l'utilisateur et effectuée à l'aide d'une table de correspondance LUT.

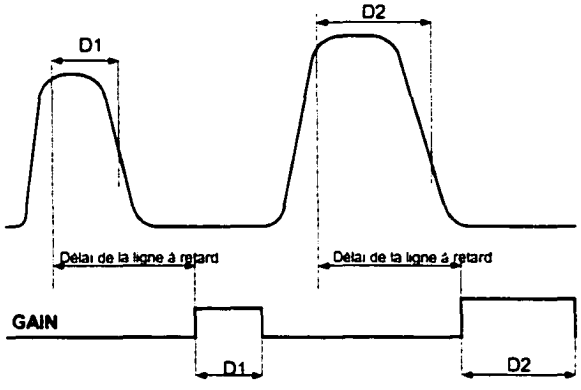


Figure 1-3 Réaction du gain et contrôle automatique du gain

## 1.4 Estimateur

Ce module détermine la durée de l'impulsion et son temps d'arrivée PTOA. La durée de l'impulsion est estimée par le temps séparant la détection du début d'une impulsion valide et la détection de sa fin. Le PTOA, acronyme anglais ('Pulse Time Of Arrival'), est le temps du passage de l'impulsion par la valeur à mi-hauteur de son amplitude maximale comme le montre la figure 1-4. Le caractère discret des échantillons fait que cet estimé ne correspondra pas forcément à un échantillon, ou à un multiple de la période d'échantillonnage. Pour affiner le résultat et diminuer l'erreur sur la valeur du PTOA, on procède par interpolation de l'impulsion entre deux échantillons consécutifs.

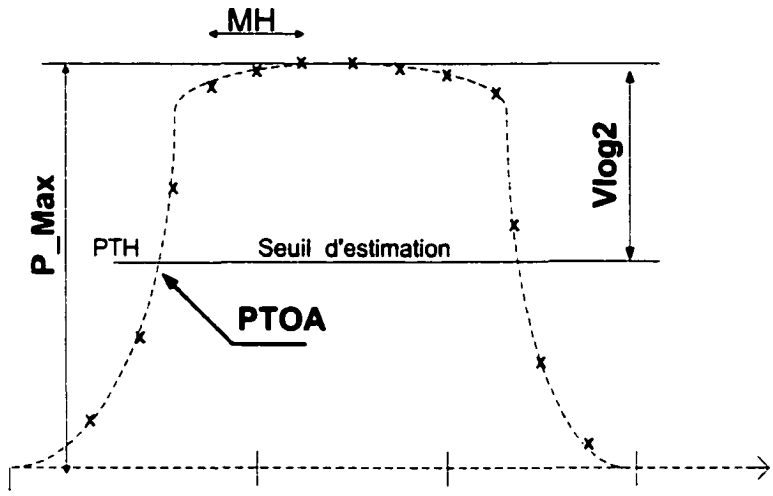


Figure 1-4 Paramètres de l'estimateur

### 1.4.1 Seuil d'estimation

Le seuil d'estimation est le niveau correspondant à la moitié de l'amplitude maximale d'une impulsion valide. Pour définir ce seuil, nous disposons de deux alternatives : Un seuil fixe pour toutes les impulsions ou un seuil adaptatif, selon le niveau de chaque impulsion. La figure 1-5 illustre ces deux types de seuil.

La première alternative, bien que simple sur le plan de la réalisation, reste peu précise puisqu'elle est très sensible aux variations de l'amplitude de chaque impulsion. Or, nous sommes en face du cas où l'amplitude des impulsions varie beaucoup. Et comme le montre la figure 1-5, si deux impulsions ont le même point de montée et deux amplitudes différentes, on aura tendance à trouver comme estimé de leur PTOA deux valeurs très distinctes, c'est à dire qu'on va conclure que l'une a précédé l'autre.

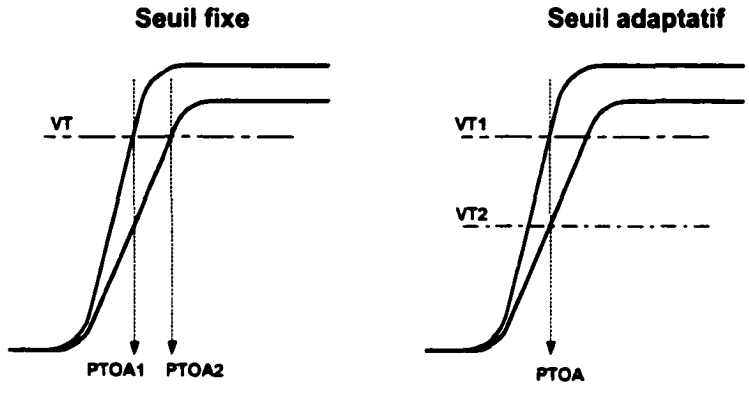


Figure 1-5 Types du seuil d'estimation

La deuxième alternative est plus précise puisqu'elle prends en compte les variations d'amplitude subies par les différentes impulsions. Dans ce cas, si deux impulsions ont le même point de montée et deux amplitudes différentes, on aura tendance à trouver comme estimé de leur PTOA deux valeurs très proches. Dans notre réalisation, nous avons opté pour cette solution. Le calcul du seuil est donnée dans le paragraphe suivant.

### 1.4.2 Calcul du seuil adaptatif

Le signal de sortie de l'amplificateur logarithmique est une fonction logarithmique de l'amplitude de l'échantillon à son entrée. Pour l'échantillon numéro n d'amplitude $V_e(n)$ la sortie est de la forme :

$$V_s(n) = V\log(V_e(n)/V_0) = V\log(V_e(n)) - V\log(V_0)$$

$V_0$ et V sont des paramètres de l'amplificateur logarithmique utilisé.

En appliquant cette formule à l'échantillon d'amplitude maximale $V_{max}$ et à $V_{max}/2$ on obtient :

$$P\_Max = V\log(V_{max}) - V\log(V_0)$$

$$SEUIL = Vlog(V_{max}) - Vlog2 - VlogV_0$$

des deux équations on tire :

$$SEUIL = P\_Max - Vlog(2).$$

Donc, le calcul du seuil se réduit à soustraire une valeur constante Vlog(2) du maximum détecté pour chaque impulsion valide.

### 1.4.3 Méthodes d'interpolation

L'interpolation est une opération mathématique qui consiste en l'approximation d'une suite de valeurs discrètes par une fonction continue sur un domaine déterminé. Les valeurs de la fonction coïncident avec ceux des données de départ aux différents points d'échantillonnage dans le domaine d'approximation considéré. Pour le problème qui nous concerne, le choix de la méthode sera dicté par un compromis entre deux éléments antagonistes : la complexité d'une part, et la précision d'une autre part. Une étude théorique a été effectuée dans le cadre d'un projet de maîtrise portant sur la même question. Je retiens de cet étude les méthodes les plus pratiques suivantes :

- **La méthode linéaire** : après la localisation des deux échantillons M0 et M1 entourant le seuil. L'impulsion est approximée par un polynôme du premier degré, ce qui correspond à la droite passant par ces deux points et qui coupe la droite horizontale du seuil au point d'estimation du PTOA. Cette méthode est très intéressante par sa précision. En effet, les résultats de simulation ont révélé qu'on pourrait répondre aux spécifications du projet en utilisant cette méthode.

- **La méthode quadratique** : Pour cette méthode, on localise les trois échantillons au voisinage du seuil M0, le point juste en dessous du seuil, M1 et M2 les deux échantillons successifs juste après le seuil. Le polynôme d'approximation est d'ordre deux. C'est une parabole passant par les trois points et coupant la droite horizontale du seuil. Cette méthode est plus précise que la linéaire au prix d'une implantation plus complexe.

- **La méthode de la Spleen cubique** : Pour cette méthode, on a besoin de localiser quatre points au voisinage du seuil. On choisira les deux points successifs juste avant le seuil ainsi que les deux juste au-dessus. On est en face de la résolution d'une équation du troisième ordre. Le nombre de solutions est soit un ou trois. On aura aussi à localiser la racine adéquate.

### 1.4.4 Stratégies de résolution

Selon le type d'interpolation retenu, nous aurons à résoudre une équation du premier degré pour une interpolation linéaire, ou une équation du second degré dans le cas d'une interpolation quadratique. Pour ces deux méthodes nous exposerons les différentes stratégies retenues pour la réalisation matérielle du projet.

#### 1.4.4.1 Interpolation Linéaire

L'équation à résoudre est de la forme :

$$AX + B = SEUIL \qquad (1\text{-}1)$$

Les coefficients A et B sont définis en fonction des coordonnées des points M0(0,Y0) et M1(1,Y1). Dans ce cas :

$$A = Y1 - Y0 \text{ et } B = Y0.$$

L'équation à résoudre est donc (Y1 – Y0) X + Y0 = SEUIL. Pour résoudre cette équation nous avons retenu les deux stratégies suivantes :

### *A - Subdivision de l'intervalle [M0,M1]*

Cette méthode a été implantée dans le cadre de l'étude précédente sous Viewlogic. Elle consiste en la façon de résoudre l'équation du premier degré en question. La résolution est basée sur la subdivision de l'intervalle M0, M1 en 16 sous intervalles et le calcul de la valeur de la droite d'interpolation pour ces 16 points. Le PTOA correspond au point qui a l'ordonnée la plus proche de la valeur du seuil.

L'équation 1-1 devient (Y1- Y0) X = ( SEUIL – Y0 ). Sa résolution se ramène à une comparaison de la valeur de la droite en 16 points équidistants de l'intervalle [Y0,Y1] avec (SEUIL – Y0).

### *B - Inversion de la pente*

Cette stratégie consiste en l'implantation de la solution de l'équation 1-1 sous la forme suivante :

$$PTOA = \frac{1}{(Y1 - Y0)} * (SEUIL - Y0).$$

La résolution est faite en deux temps. Dans un premier temps, on calcule 1/(Y1-Y0) à l'aide d'une table de correspondance, ensuite on effectue le produit de la valeur obtenue par (SEUIL – Y0).

### 1.4.4.2 Interpolation Quadratique

L'équation à résoudre est de la forme : $AX^2 + BX + C = SEUIL$. Les coefficients A, B, et C sont définis en fonction des coordonnées des trois points M0, M1 et M2. Le choix de l'origine facilitera les calculs et l'adaptation du résultat. On a deux possibilités :

***Premier choix d'origine*** : M0 ( -1 , Y0) ; M1(0,Y1) ; M2(1,Y2)

Pour ce choix d'origine

$$A = .5 ( Y0 + Y2 ) - Y1$$
$$B = .5 ( Y2 - Y0)$$
$$C = Y1.$$

***Deuxième choix d'origine***: M0( 0 , Y0) ; M1 (1, Y1) ; M2(2,Y2)

Pour ce choix d'origine

$$A = 0.5\,Y2 - Y1 + .5\,Y0$$
$$B = 2\,Y1 - 0.5\,Y2 - 1.5\,Y0$$
$$C = Y0.$$

Nous avons retenu le deuxième choix ; Il nécessite plus de calcul pour trouver les coefficients. Par contre le résultat obtenu sera directement utilisé. En effet on tombera sur un nombre qu'on ajoute à la valeur de l'instant de Y0. Le calcul des coefficients

pour la première méthode bien qu'il soit moins compliqué, la racine sera de signe négatif et on doit manipuler des nombres signés. En plus on doit faire un calcul pour retrouver la valeur du PTOA.

Par définition des points M0, M1, M2 le problème possède deux solutions réelles ; celle qui nous intéresse se trouve entre M0 et M1. Donc le choix de la racine sera effectué par une comparaison des deux racines avec 1.

L'équation à résoudre est donc :

$$(.5Y2 - Y1 + .5Y0)\,X^2 + (-.5Y2 + 2Y1 - 1.5Y0)X + Y0 = SEUIL. \qquad (1\text{-}2)$$

Pour résoudre cette équation nous avons retenu les deux stratégies suivantes :

***A - Subdivision de l'intervalle [M0,M1]***

Comme dans le cas de l'équation du premier degré, la résolution est basée sur la subdivision de l'intervalle M0, M1 en 16 sous intervalles et le calcul de la valeur de la parabole d'approximation pour les 16 points. Le PTOA correspond au point qui a l'ordonnée la plus proche de la valeur du seuil. L'équation 1-2 devient

$$(.5Y2 - Y1 + .5Y0)\,X^2 + (-.5Y2 + 2Y1 - 1.5Y0)X = SEUIL - Y0.$$

La résolution se ramène à une simple comparaison de la valeur de la droite en 16 points équidistants de l'intervalle [Y0,Y1] avec (SEUIL – Y0).

### *B - Résolution algébrique*

À partir des coefficients A, B et C on calcule $\Delta = B^2 - 4A(C\text{-}SEUIL)$. Ensuite, on détermine la racine carrée de delta à l'aide d'une table de correspondance (LUT). Ayant la racine de delta les deux racines de l'équation sont données par :

$$R1 = \frac{-B + \sqrt{\Delta}}{2A} = \frac{1}{2A} * (-B + \sqrt{\Delta})$$

$$R2 = \frac{-B - \sqrt{\Delta}}{2A} = \frac{1}{2A} * (-B - \sqrt{\Delta})$$

Les opérations à implémenter dans ce cas sont : deux tables de correspondance, la première pour le calcul de l'inverse de A, la deuxième pour le calcul de la racine carrée de delta, ainsi que le produit de ces deux résultats suivi d'un multiplexeur et d'un comparateur des deux racines.

## 1.5 Conclusion

Dans ce chapitre nous avons décrit les fonctionnalités du circuit numérique à réaliser. Les deux blocs principaux étant le détecteur et l'estimateur. Les différentes possibilités et stratégies de réalisation ont été aussi abordées avec un rappel sur les travaux effectués dans le cadre d'une étude de maîtrise sur le même sujet [1], et dont les résultats seront aussi revus plus tard pour effectuer une comparaison des performances des deux architectures. Dans le chapitre suivant nous aborderons en détail les fichiers de description de l'architecture du circuit en VHDL.

# CHAPITRE II

# DESCRIPTION VHDL DU CIRCUIT

## 2.1 Introduction

Au chapitre précédent, nous avons exposé les spécifications de l'unité de prétraitement à réaliser. Dans le présent chapitre, nous abordons les deux premières étapes de la conception de circuits numériques de haut niveau, qui sont la description en VHDL du circuit et la simulation du code obtenu. Ces deux étapes sont les plus longues et les plus ardues du processus de conception. Il s'agit d'un processus itératif, au cours duquel ces deux étapes sont répétées jusqu'à ce que les résultats soient satisfaisants et correspondent aux spécifications. Le chapitre commence par une introduction succincte sur le langage VHDL, ensuite il décrit les différents modules et leurs interactions. Des exemples de résultats de simulations sont également présentés.

## 2.2 VHDL

Le VHDL est un langage permettant la description d'un circuit ou un système, sous forme syntaxique [9]. Cette description, qui sera évaluée par un compilateur du langage, varie du niveau d'abstraction le plus élevé, description comportementale, au niveau le plus bas, proche du matériel. Entre ces deux niveaux on trouve la description d'équations booléennes. L'aspect comportemental de ce langage permet d'obtenir un code source indépendant de la technologie ciblée et donc satisfaire aux critères de portabilité. À l'origine, ce langage a été créé pour la modélisation et la simulation. Il

s'est imposé aujourd'hui comme standard de description destinée à la synthèse. Tous les constructeurs de composants programmables possèdent des outils intégrant les ressources VHDL. Pour ces raisons il est considéré par la plupart des concepteurs comme un des maillons indispensables dans la chaîne de développement des composants programmables.

Une description en VHDL peut être écrite à différents niveaux d'abstraction : Comportemental, structurel, RTL transfert de registre, porte logique. Ces différents niveaux de description peuvent être utilisés simultanément dans un même design. La description RTL, décrit le comportement du circuit coup d'horloge par coup d'horloge. Pour la description structurelle le 'netlist' représente un bon exemple. Il décrit les interconnexions entre les éléments de base qui sont les portes logiques, en faisant appel aux éléments spécifiques décrits dans des librairies. Le 'netlist' peut être utilisé dans les différentes phases de développement comme la simulation, le placement et routage. Pour la simulation, le 'netlist' sera utilisé conjointement avec une librairie de modèles de simulation. Pour le placement et routage, la librairie associée à ce fichier contiendra les modèles d'implantation physique de chaque composant.

Les designs en VHDL sont composés de plusieurs unités compilées et sauvegardées dans des librairies. Les différentes unités compilables sont :

- Les entités (Entity) : Elles décrivent l'interface du module avec l'extérieur.
- Les architectures (Architecture) : Elles décrivent le comportement du module. Une entité peut avoir plusieurs architectures, chacune peut avoir son propre type : comportemental ou structurel.
- Les configurations (Configuration) : Elles permettent la sélection des architectures pour construire une variante de description du design à partir d'une librairie. C'est l'équivalent d'une liste qui défini le comportement à utiliser pour chaque entité du design.

- Les packages (Package) : Ils permettent de sauvegarder ensemble des spécifications fréquemment utilisées telles que : types de données, constantes et sous programmes. Ils les rendent ensuite visibles à toutes les autres unités. Les packages peuvent être compilés dans des librairies et utilisés au besoin à l'aide de l'expression 'Use'.

En utilisant la démarche descendante ( top down design), on représente le système de la façon la plus abstraite possible et on ajoute les détails par la suite. Le code obtenu peut être ensuite simulé à l'aide d'un simulateur pour vérifier le comportement du modèle. La simulation nécessite la génération des stimulus pour faire réagir le modèle.

## 2.3 Présentation du code

Rappelons que nous avons retenu trois stratégies différentes pour la conception de l'estimateur : la première basée sur une interpolation linéaire et inversion de la pente, la deuxième est basée sur une interpolation linéaire et subdivision de l'intervalle [M0,M1], la troisième et dernière est basée sur une interpolation quadratique et subdivision de l'intervalle [M0,M1]. Par conséquent notre code VHDL est composé d'un noyau commun aux trois stratégies, plus des modules spécifiques. La figure 2-1 illustre la structure des fichiers pour la première stratégie.

Un module global, appelé ***Agc.vhd***, comporte trois blocs structurels : ***Detecteur.vhd***, ***Ptoa.vhd***, et ***Gain_sortie.vhd***. Ces derniers modules, à leur tour, comportent d'autres modules plus détaillés. Les différents blocs feront l'objet des paragraphes suivants. Le code VHDL des différentes entités est présenté en annexes A, B, C, D et E.

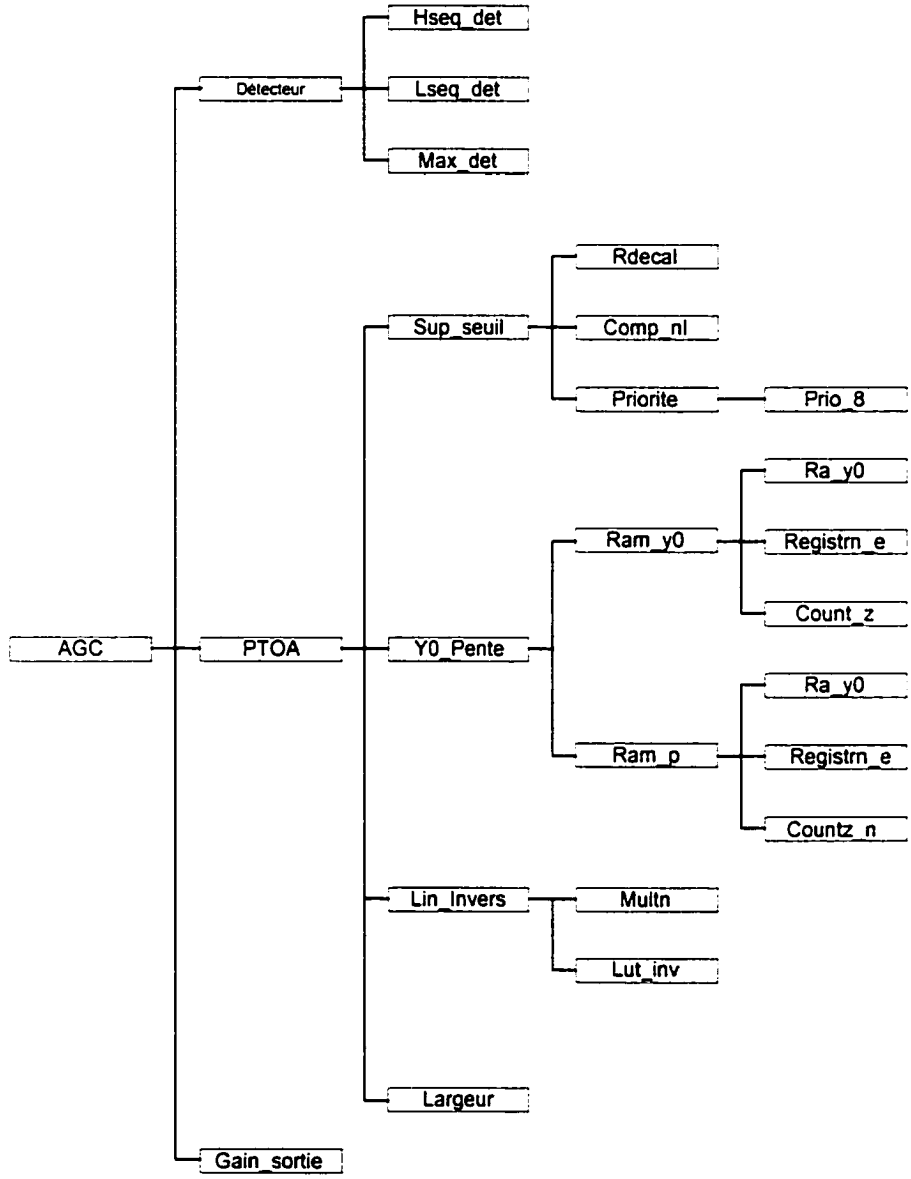


Figure 2-1 Arborescence des fichiers VHDL, première stratégie

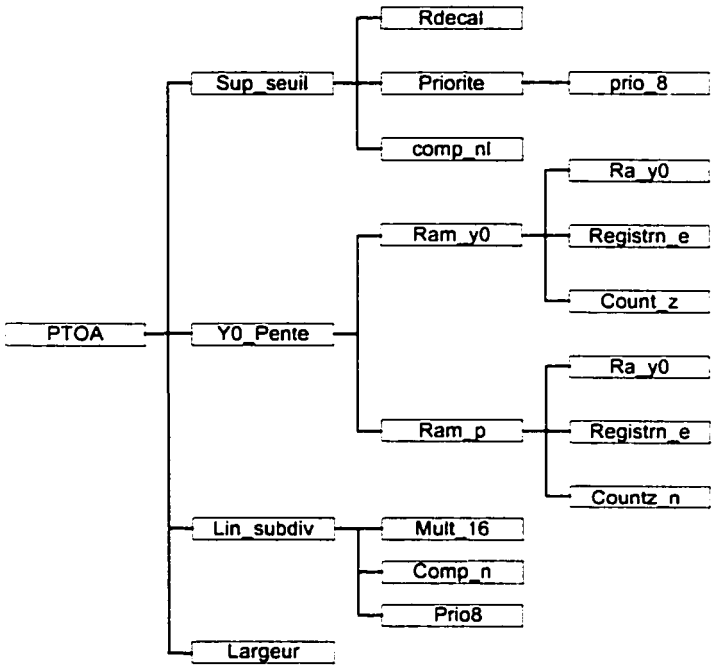


Figure 2-1a Structure du module PTOA, (linéaire et subdivision)

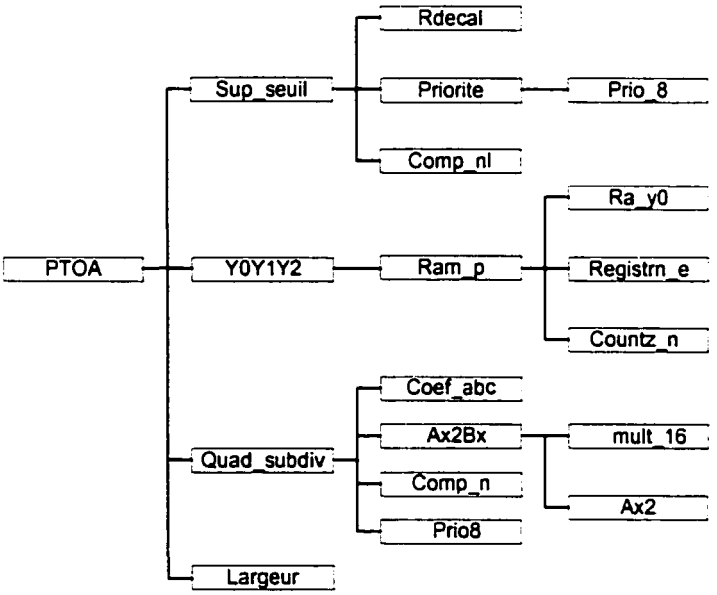


Figure 2-1b Structure du module PTOA, (quadratique et subdivision)

### 2.3.1 Détection : *Detecteur.vhd*

Le détecteur a pour mission de détecter les impulsions et de déterminer leurs caractéristiques. Il comporte trois modules : ***Hseq_det.vhd***, ***Lseq_det.vhd***, ***Max_det.vhd***. Ces modules seront détaillés dans les paragraphes suivants.

#### 2.3.1.1 Début de l'impulsion : *hseq_det.vhd*

Ce module génère le signal **START_DET** pour signaler le début d'une impulsion valide. L'échantillon à l'entrée est comparé au seuil **TH2**. Le résultat de la comparaison génère le signal **HIGH_DET**. Les résultats de comparaison des trois échantillons précédents sont mémorisés par les signaux : **HIGH_DET1, HIGH_DET2 et HIGH_DET3**. On dispose du signal PEE, qui indique le sens du front en cours de traitement. Le signal **START_DET** passe à '1' pour indiquer le début d'une impulsion valide. Le paramètre MH permet la sélection entre quatre fonctions combinatoires formant **START_DET**.

Si MH = 00 START_DET = HIGH_DET et (non PEE)
Si MH = 01 START_DET = HIGH_DET et HIGH_DET1 et ( non PEE)
Si MH = 10 START_DET = HIGH_DET et HIGH_DET1 et HIGH_DET2 et ( non PEE)
Si MH = 11 START_DET = HIGH_DET et HIGH_DET1 et HIGH_DET2 et HIGH_DET3 et ( non PEE)

Le signal ainsi obtenu sera retardé d'un coup d'horloge et sera utilisé comme référence pour synchroniser et effectuer toutes les opérations. Il est nommé **P_VALIDE**.

#### 2.3.1.2 Fin de l'impulsion : *lseq_det.vhd*

Ce module génère le signal **END_DET** pour signaler la fin de l'impulsion. Sa structure est similaire à celle du module précédent. L'échantillon à l'entrée est comparé au seuil **TH1**. Le résultat de la comparaison génère le signal **LOW_DET**. Les résultats de comparaison des trois échantillons précédents sont mémorisés par les signaux : **LOW_DET1, LOW_DET2 et LOW_DET3**. On dispose du signal PEE, qui indique le sens du front en cours de traitement. Le signal END_DET passe à '1' pour indiquer la fin de l'impulsion. Le paramètre LH permet la sélection entre quatre fonctions combinatoires formant **END_DET**.

Si ML = 00 END_DET = LOW_DET et PEE
Si ML = 01 END_DET = LOW_DET et LOW_DET1 et PEE
Si ML = 10 END_DET = LOW_DET et LOW_DET1 et LOW_DET2 et PEE
Si ML = 11 END_DET = LOW_DET et LOW_DET1 et LOW_DET2 et LOW_DET3 et PEE

#### 2.3.1.3 Détection du maximum : *max_det.vhd*

Ce module est basé sur un système de comparaison bouclé. Chaque échantillon à l'entrée du circuit est comparé au maximum déjà mémorisé. Si l'échantillon actuel est supérieur au maximum provisoire, il remplacera l'ancien. Sinon la valeur précédente est conservée. La valeur finale du maximum est prise en compte lorsque l'impulsion est déclarée valide.

Pour accélérer la cadence du traitement du circuit, le seuil d'estimation est calculé simultanément. À chaque échantillon on retranche une donnée constante appelée

**ECART**. Le seuil provisoire change à chaque modification du maximum. La valeur retenue coïncide aussi avec le signal de validité de l'impulsion.

Les valeurs du maximum et du seuil sont initialisées à zéro à l'aide du signal **MAX_INIT,** au début de chaque impulsion ou à chaque fois que le niveau de l'échantillon passe en dessous de TH2 avant la validation de l'impulsion.

Le module est composé de trois processus : le premier effectue la comparaison et génère le signal **MAX_DET** qui valide la mémorisation du nouveau maximum. Le seuil est calculé par une expression concurrentielle. Le deuxième et le troisième processus valident et remettent à zéro le maximum et le seuil en utilisant les signaux **MAX_INIT** et **MAX_DET**.

#### 2.3.1.4 Validation de la fin de l'impulsion

Le signal **PEE** a pour but de mémoriser l'état de détection. Il permet de différencier la détection des deux fronts. Il passe à 0 pour indiquer que la séquence présente correspond à un front montant. Il prend la valeur 1 pour indiquer que la séquence présente correspond à un front descendant. Ce signal est donc utilisé par les deux processus précédents.

### 2.3.2 Générateur de gain : *Gain_s.vhd*

Le module ***Gain_s.vhd*** permet le calcul et la synchronisation du signal de commande de l'atténuateur à gain variable pour réguler l'amplitude du signal analogique retardé. La valeur de commande est fonction du maximum détecté. La relation entre les deux est imposée par l'utilisateur sous forme de paramètres mémorisés dans une table de conversion, définie par le module ***Lut_p.vhd***. Cette relation n'étant pas définie, nous

avons fait nos tests en utilisant la fonction identité. C'est une ROM de 625 mots de 8bits. Le signal de commande obtenu correspond donc au maximum détecté représenté par 8 bits.

La commande doit avoir lieu dix coups d'horloge après la détection de début de l'impulsion. Cela correspond au délai de 200 ns introduit par la ligne à retard. Pour rendre cette commande indépendante de la valeur du paramètre MH, on introduit un retard réglable sur les signaux **START_DET**, **END_DET** et **GAIN_IN**. Un multiplexeur est utilisé pour aiguiller la bonne valeur vers la sortie. Cette opération demande 6 coups d'horloge, on rajoute un autre délai de 4 coups d'horloge pour la synchronisation.

### 2.3.3 Estimation : Interpolation linéaire avec inversion de la pente

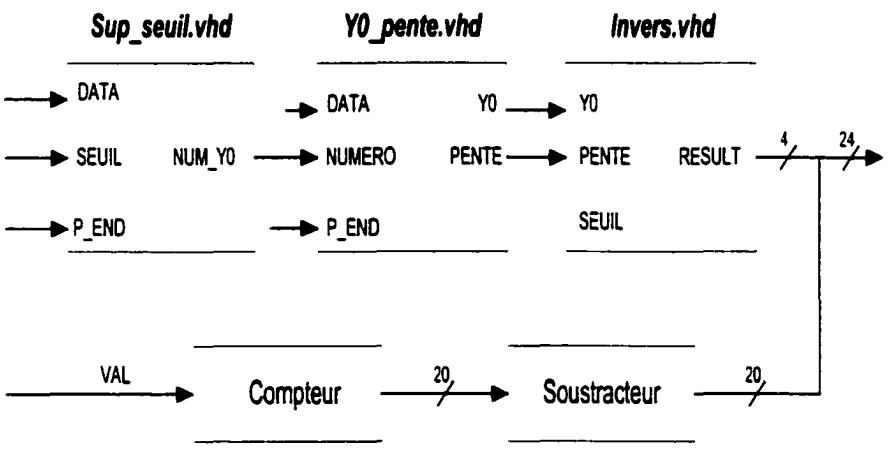


Figure 2-2 Architecture de L'estimateur, stratégie numéro 1

L'estimation du temps d'arrivée de l'impulsion est réalisée à l'aide du module hiérarchique ***ptoa.vhd***. Ce module hiérarchique instancie les trois modules suivants : ***Sup_seuil.vhd, Y0_Pente.vhd*** et ***Lin_invers.vhd***. Le premier détermine le nombre des échantillons supérieurs au seuil d'estimation, le deuxième localise l'échantillon Y0 ainsi que la pente P entre M0 et M1, les deux échantillons qui entourent le seuil d'estimation, le troisième et dernier module permet le calcul de l'inverse de la pente ainsi que le produit final pour le calcul de la partie décimale du PTOA. Dans ce qui suit nous détaillerons ces trois modules.

#### 2.3.3.1 Échantillons supérieurs au seuil : *Sup_seuil.vhd*

Ce module utilise un registre à décalage pour mémoriser les valeurs des différents échantillons de l'impulsion jusqu'à ce qu'elle soit déclarée valide. À chaque moment, nous disposons des valeurs des 32 derniers échantillons. Ils sont simultanément comparés au seuil adaptatif ***SEUIL***. Le résultat de comparaison est un mot de 32 bits débutant par une série de 0 suivie d'une série de 1. La série des 1 correspond à la suite d'échantillons consécutifs supérieurs au seuil, alors que la série des 0 correspond à la suite d'échantillons consécutifs inférieurs au seuil. Le passage de la série des 1 à la série des 0 correspond au premier échantillon juste au-dessus du seuil. À noter, que la série des 1 peut contenir des 0, dans le cas ou, après validation, certains échantillons se trouvent en dessous de TH2. Néanmoins, le principe de décodage reste valable, car on est intéressé uniquement par la première transition de 0 à 1.

Pour localiser le rang du premier bit à 1 de cette séquence, on utilise un décodeur de priorité. Cette opération est réalisée par le module ***priorite.vhd***. Le mot de 32 bits est traité par paquets de 8 bits de façon parallèle. Au résultat du premier décodage on rajoute le poids du groupe de 8 bits le plus élevé non nul. Ce poids est égal à 0 pour les bits d'ordre 0 à 7, égal à 8 pour les bits d'ordre 8 à 15, égal à 16 pour les bits de 16 à 24 et il est égal à 24 pour les bits de 24 à 32. On obtient ainsi le numéro d'ordre du premier

échantillon juste au-dessus du seuil parmi les 32 échantillons du début de l'impulsion valide. Cette valeur est portée par le signal **NUM_Y0**. Elle servira comme entrée pour le module ***Y0_pente.vhd*** qui sera détaillé dans le paragraphe suivant.

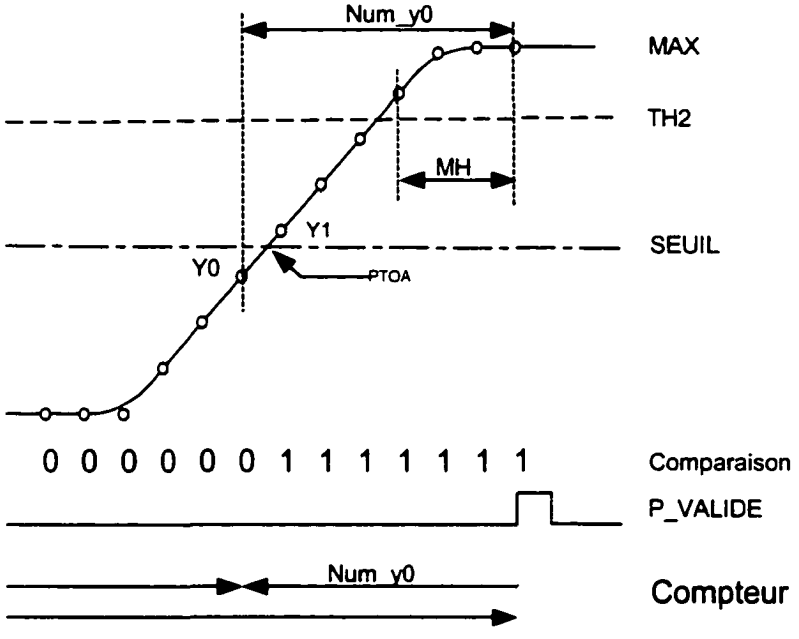


Figure 2-3 Détermination du rang de M0

### 2.3.3.2 Localisation du point M0 et calcul de la pente

En parallèle, on dispose de deux RAM de 32 mots de 8 bits chacune. Les échantillons sont mémorisés dans la première appelée **RAM_Y0**, tandis que la pente est mémorisée dans la seconde appelée **RAM_P**. La pente est évaluée en calculant à chaque coup d'horloge la différence entre l'échantillon actuel et celui qui l'a précédé.

Le compteur générant les adresses de mémorisation est remis à zéro au début de chaque impulsion. Sa valeur au moment du passage du signal VAL de 0 à 1, validation de l'impulsion, est mémorisée dans un registre, en attendant la détermination du nombre des échantillons supérieurs au seuil.

Pour obtenir l'adresse de l'échantillon correspondant à Y0, on retranche la valeur NUM_Y0 du contenu du registre précèdent. Au cinquième coup d'horloge après la validation, cette adresse sera appliquée aux deux RAM pour effectuer la lecture de Y0 et de la pente des deux échantillons entourant le seuil d'estimation.

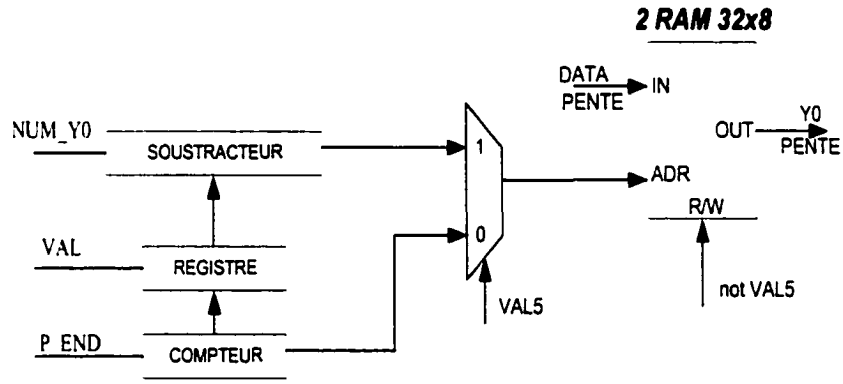


Figure 2-4 Détermination de Y0 et de la pente

### 2.3.3.3 Inversion de la pente et calcul du produit

Pour le calcul du PTOA nous disposons de Y0 et de la PENTE. On calcule l'inverse de la pente, en utilisant une table de valeur sous forme de ROM de 256 valeurs sur 12 bits. La valeur de la pente est fournie comme adresse en sortie on a la valeur de 1/PENTE. Le nombre de bits pour la quantification est obtenu en tenant compte de l'erreur relative permise de 20%. On effectue ensuite la multiplication de la sortie de la ROM par la différence entre le SEUIL et Y0. Le résultat obtenu nous fournit les quatre

bits de la partie décimale du PTOA. La partie entière étant donnée par le compteur des échantillons

### 2.3.4 Estimation : linéaire avec subdivision de [M0,M1]

La structure de ce module est similaire à celle du paragraphe précédent. Nous utilisons le module ***sub_seuil.vhd***, pour calculer le nombre des échantillons supérieurs au seuil d'estimation, et le module ***Y0_pente.vhd*** pour déterminer Y0 et la pente. Ainsi, nous aurons définit les coefficients de l'équation du premier degré à résoudre :

PENTE * X = SEUIL – Y0 avec X = I/16 et I variant de 1 à 15.

Cette équation est équivalente à :

PENTE * I = 16 * ( SEUIL – Y0) avec I variant de 1 à 15.

La résolution, est effectuée à l'aide du module, ***subdiv_lin.vhd***, dont le principe est représenté par la figure 2-5.

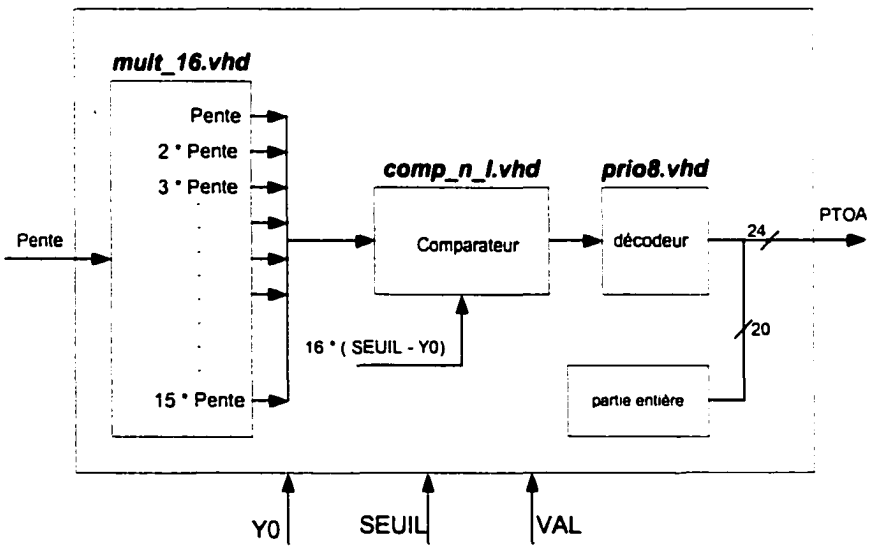


Figure 2-5 Architecture du module *lin_subdiv.vhd*

Le bloc ***subdiv_comp.vhd*** est composé du module ***mult_16.vhd,*** qui calcule les quinze produits (PENTE * I), d'un deuxième module qui effectue la comparaison des résultats obtenus avec le terme 16 * (SEUIL – Y0). Ce dernier terme est obtenu en effectuant une soustraction suivie par un décalage de quatre bits. Le résultat de la comparaison est ensuite décodé à l'aide d'un décodeur de priorité qui fournit à la sortie la partie décimale du PTOA.

### 2.3.5 Estimation : quadratique avec subdivision de [M0,M1]

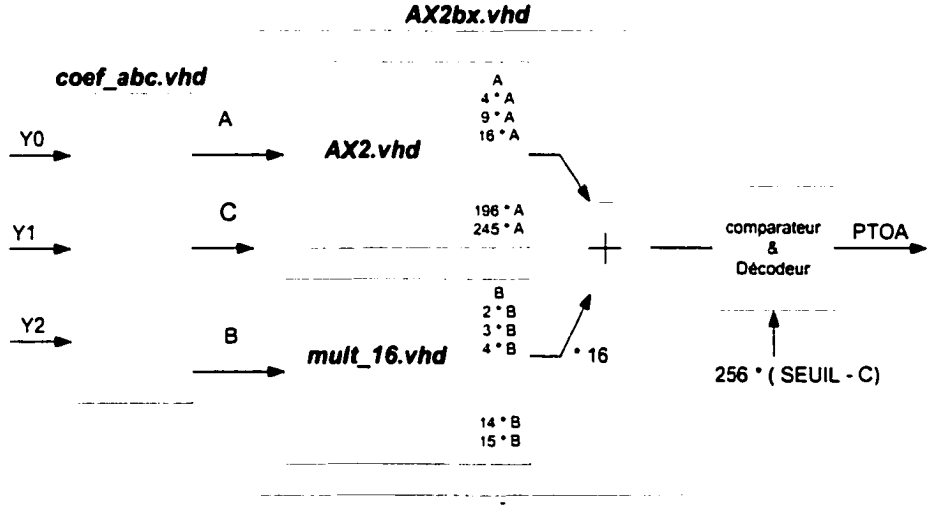


Figure 2-6 Schéma bloc du module *quad_subdiv.vhd*

Ce bloc hiérarchique instancie les quatre modules suivants : ***Sup_seuil.vhd, Y0y1y2.vhd, Coef_abc.vhd*** et ***Quad_subdiv.vhd***. Le premier module a été décrit précédemment. Le module ***Y0y1y2.vhd*** localise les trois points entourant le suil d'estimation. Il contient trois RAM de 32 mots de 8 bits. À chaque coup d'horloge il mémorise la valeur de l'échantillon en cours ainsi que les deux valeurs précédentes. Le

module C***oef_abc.vhd*** effectue le calcul des coefficients de l'équation du second degré à résoudre :

$$A X^2 + B X + C = SEUIL \quad \text{avec } X = I/16 \text{ et I variant de 1 à 15}$$

Cette équation est équivalente à :

$$A I^2 + 16 B I = 256 * ( SEUIL - C) \quad \text{avec I variant de 1 à 15}$$

Le module ***Ax2bx.vhd*** (voir figure 2-6) calcule les quinze opérations $A I^2 + 16 B I$. Les sorties de ce module sont comparés ensuite au terme 256 * (SEUIL – C). Le résultat de la comparaison est décodé par un décodeur de priorité pour déduire la partie décimale du PTOA.

### 2.3.6 Synchronisation des résultats

Les échantillons à l'entrée du circuit sont comptés à l'aide d'un compteur de vingt bits. Au moment de la validation de l'impulsion, le signal **P_VAL** entraîne le chargement du contenu du compteur dans le registre **COMPT**. Après le calcul du nombre des échantillons supérieurs au seuil, trois coups d'horloge après le passage de **P_VAL** à '1', on retranche **NUM_Y0** du contenu de **COMPT**. On aura ainsi le rang de Y0. C'est la partie entière du PTOA. Les quatre bits formant la partie décimale seront déduits du résultat de la multiplication de l'inverse de la pente par la différence entre le SEUIL et Y0. La partie entière sera retardée de quatre coups d'horloge pour la synchronisation avec la partie décimale. Le résultat final du PTOA sur vingt et quatre bits est obtenu par concaténation de la partie entière avec la partie décimale.

On a besoin de neuf coups d'horloge pour calculer le PTOA pour les deux premières stratégies, dans ce cas la latence du circuit varie entre dix et treize selon la valeur de MH. Pour l'interpolation quadratique Le PTOA est calculé en onze coups d'horloge, la latence varie entre douze et quinze pour la même raison.

### 2.3.7 Calcul de la largeur : *Largeur.vhd*

Ce module permet le calcul de la durée d'une impulsion valide. Il s'agit d'un compteur de huit bits commandé par les deux signaux **P_START et P_END**. Le premier initialise la valeur du compteur à une valeur égale à MH plus un, le second arrête le comptage. La largeur de l'impulsion correspond au contenu du compteur moins la valeur du paramètre ML. La valeur ainsi obtenue est retardée d'un coup d'horloge pour la synchroniser avec le signal **P_Fin**.

## 2.4 Simulation

La simulation RTL représente la seconde étape après l'écriture du code. Elle est utilisée pour vérifier l'exactitude de la description RTL, qui décrit le comportement du circuit coup d'horloge par coup d'horloge. La simulation applique des stimulus qui représentent l'environnement du circuit pour contrôler le circuit et de s'assurer que les résultats sont corrects. Dans notre cas nous avons utilisé le simulateur VSS de SYNOPSYS pour vérifier le code VHDL. Le simulateur lit le code, le compile dans un format interne et ensuite exécute la forme compilée en utilisant les vecteurs de test. Les formes d'onde sont ensuite vérifiées visuellement pour déterminer si le circuit fonctionne correctement. Les stimulus sont générés à l'aide du fichier *testbench*. Le test fonctionnel est effectué pour chaque fichier VHDL en commençant du bas de la

hiérarchie vers le haut. Dans ce qui suit, on donne un exemple des résultats obtenu par le *testbench* ***Tb_agc.vhd*** qui simule la totalité du circuit.

Le *testbench* ***Tb_agc.vhd*** instancie le module ***Agc.vhd*** à tester ainsi que le module ***gene_par_sim.vhd*** qui génère les échantillons et les paramètres de détection TH1, TH2, MH, ML et ECART. Les échantillons sont représentés par une ROM dont le contenu est lu de façon périodique à l'aide des adresses générées par un compteur. Le vecteur de test retenu comme exemple est représenté par le tableau 2-1. Il est composé de trois impulsions de dix échantillons chacune. La figure 2-7 donne une représentation graphique de la série des impulsions ainsi que les paramètres TH1, TH2 et ECART.

Tableau 2-1

Vecteur de test, valeurs des échantillons en décimal

| | ***VALEURS DES ÉCHANTILLONS*** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **IMPULSION** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **I** | 1 | 5 | 16 | 26 | 33 | 50 | 36 | 6 | 5 | 1 |
| **II** | 10 | 27 | 28 | 29 | 35 | 27 | 5 | 3 | 2 | 0 |
| **III** | 8 | 40 | 60 | 80 | 80 | 90 | 28 | 15 | 8 | 3 |

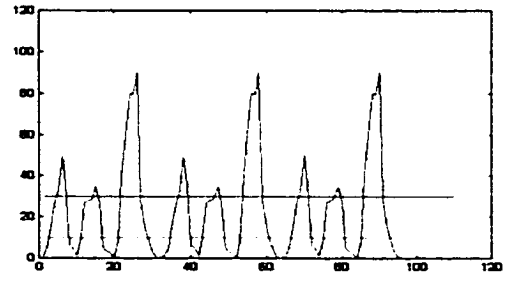

Figure 2-7 : Vecteur de test 1, TH1 = 10, TH2 = 10 et ECART = 7

Tableau 2-2

Résultats de simulation par Matlab

| | MH =2, ML = 2 | | | MH = 3, ML= 2 | | | MH = 0, ML = 2 | | |
|---|---|---|---|---|---|---|---|---|---|
| IMPULSION | I | II | III | I | II | III | I | II | III |
| MAX | 50 | - | 80 | - | - | 80 | 33 | 35 | 40 |
| DEBUT | 33 | - | 40 | - | - | 40 | 33 | 35 | 40 |
| FIN | 36 | - | 16 | - | - | 16 | 36 | 27 | 16 |
| LARGEUR | 3 | - | 7 | - | - | 7 | 3 | 2 | 8 |
| SEUIL | 43 | - | 73 | - | - | 73 | 26 | 28 | 33 |
| Y0 | 33 | - | 60 | - | - | 60 | 15 | 28 | 8 |
| Y1 | 50 | - | 80 | - | - | 80 | 26 | 29 | 40 |
| PTOA stratégie 1 | 9 | - | 10 | - | - | 10 | 13 | 0 | 12 |
| PTOA stratégie 2 | 10 | - | 11 | - | - | 11 | 1 | 1 | 13 |
| PTOA stratégie 3 | 06 | - | 09 | - | - | 09 | 1 | 1 | 12 |

Les figures 2-8a à 2-10b montrent des exemples de simulations pour les trois stratégies. Pour faciliter la lecture des résultats de simulations, on a affiché le signal **DATA6**. Il représente les échantillons retardés de six coups d'horloge. Ainsi, la commande du gain coïncide avec les échantillons de l'impulsion en question. La sortie du compteur générale des impulsions, **COUT**, est aussi affichée pour permettre la vérification de la valeur du PTOA. Cette dernière est localisée par le signal **P_VAL 9** ou **P_VAL11**, selon la stratégie considérée.

On peut vérifier que les résultats de simulation sont identiques à ceux obtenus sous MATLAB, ceci confirme que le code VHDL fonctionne correctement et qu'il est conforme aux spécifications.

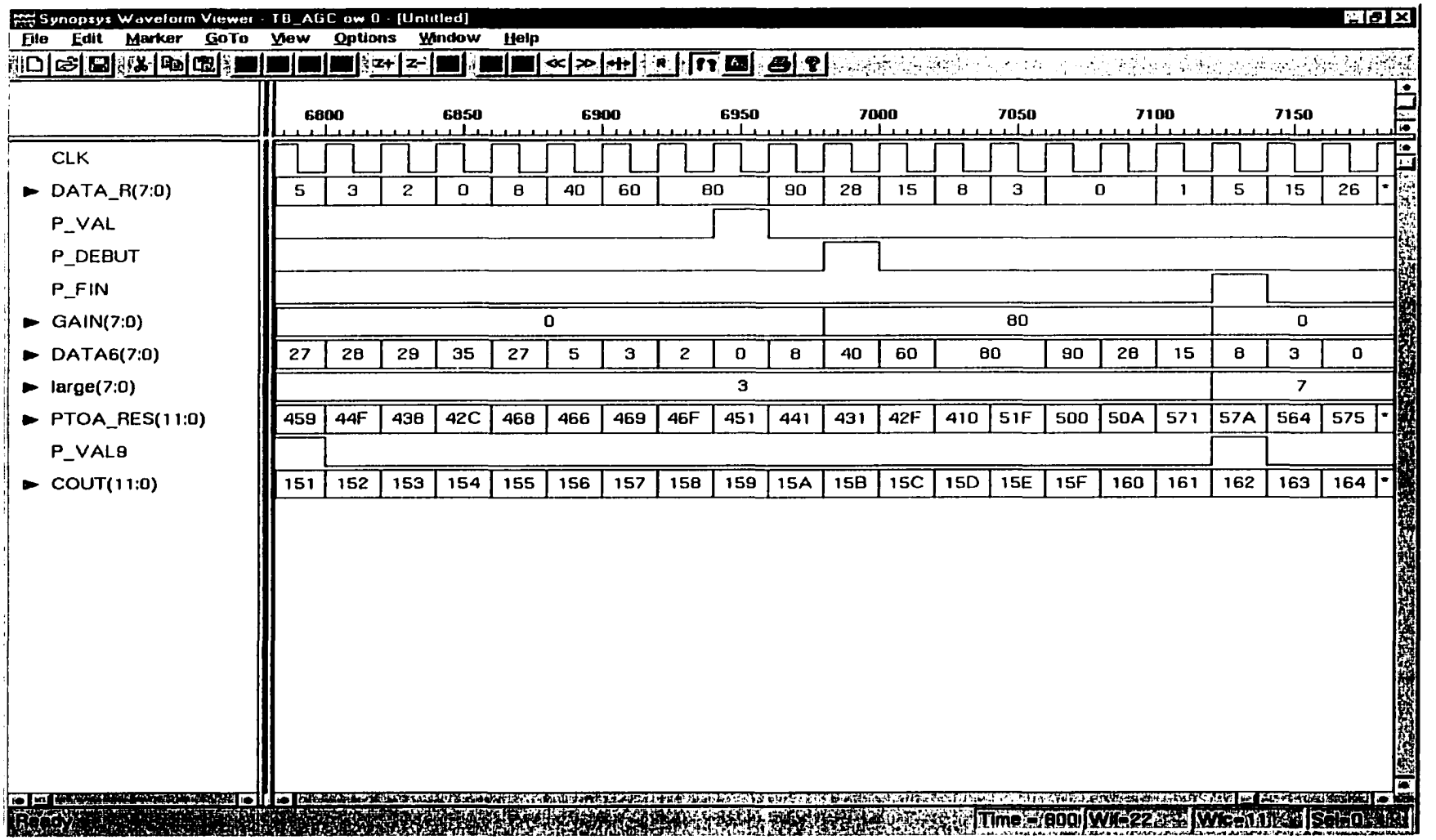


Figure 2-8a Simulation VSS, Stratégie 1, MH = 2 et ML = 2

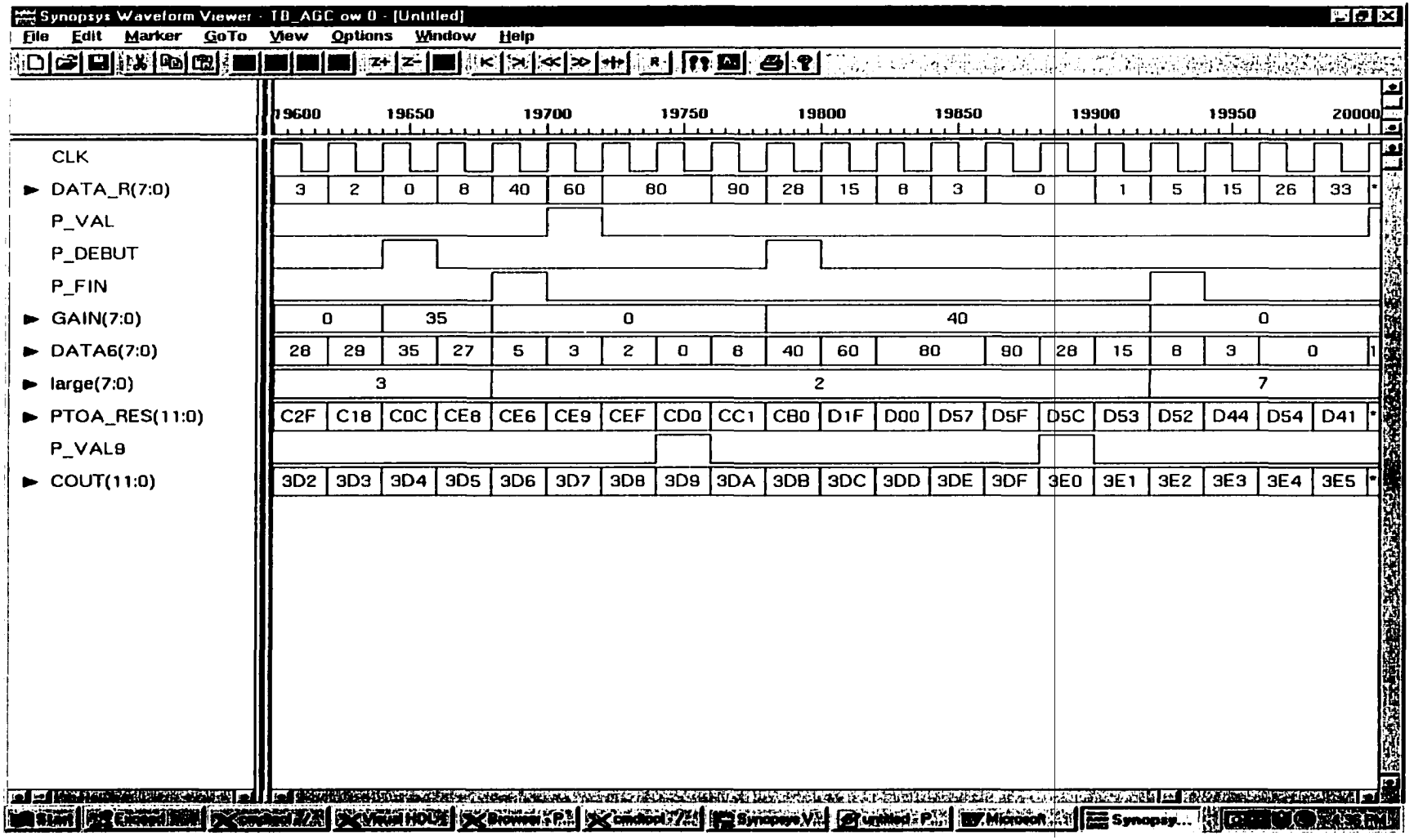


Figure 2-8b Simulation VSS, Stratégie 1, MH = 0 et ML = 2

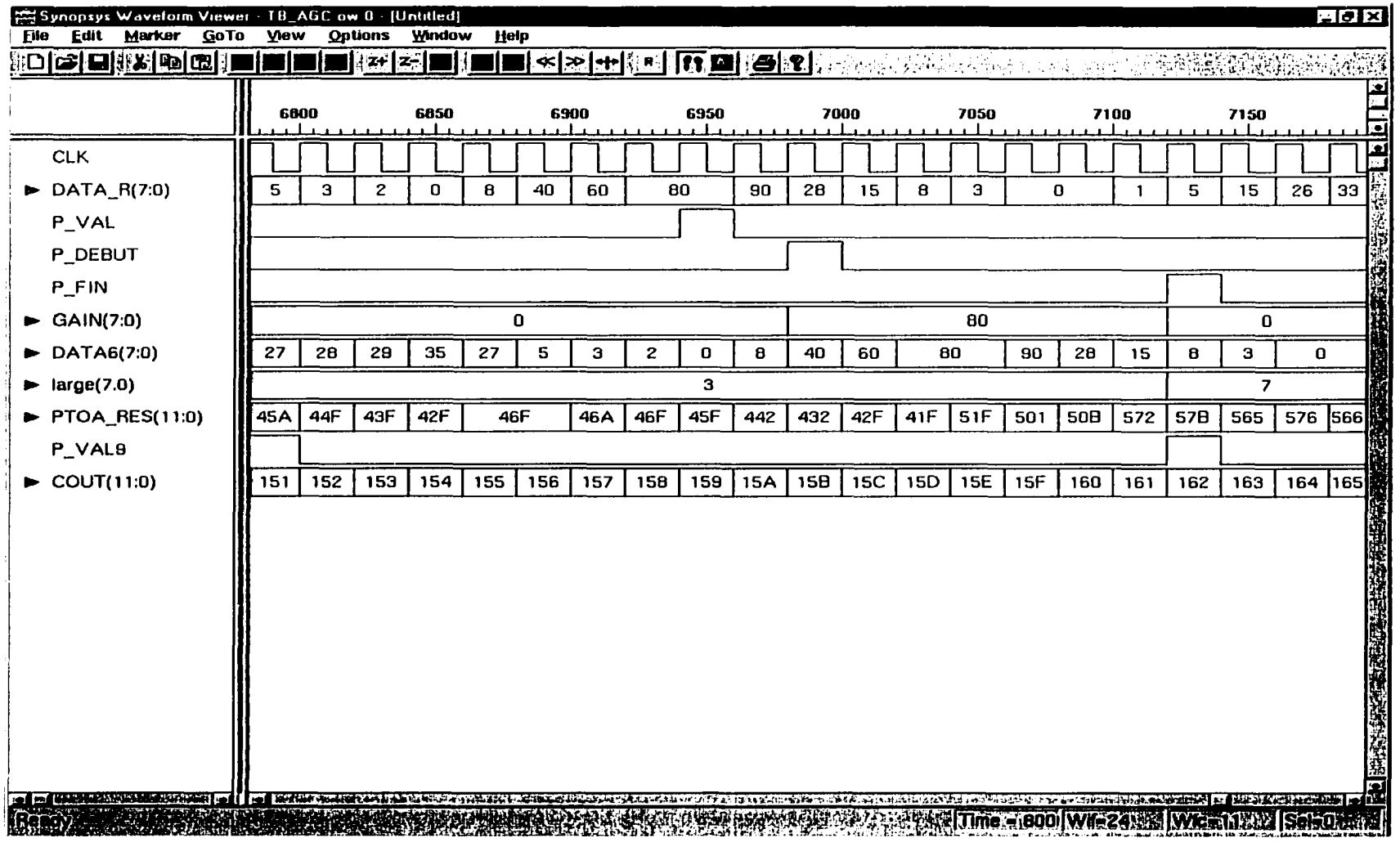


Figure 2-9a Simulation VSS, Stratégie 2, MH = 2 et ML = 2

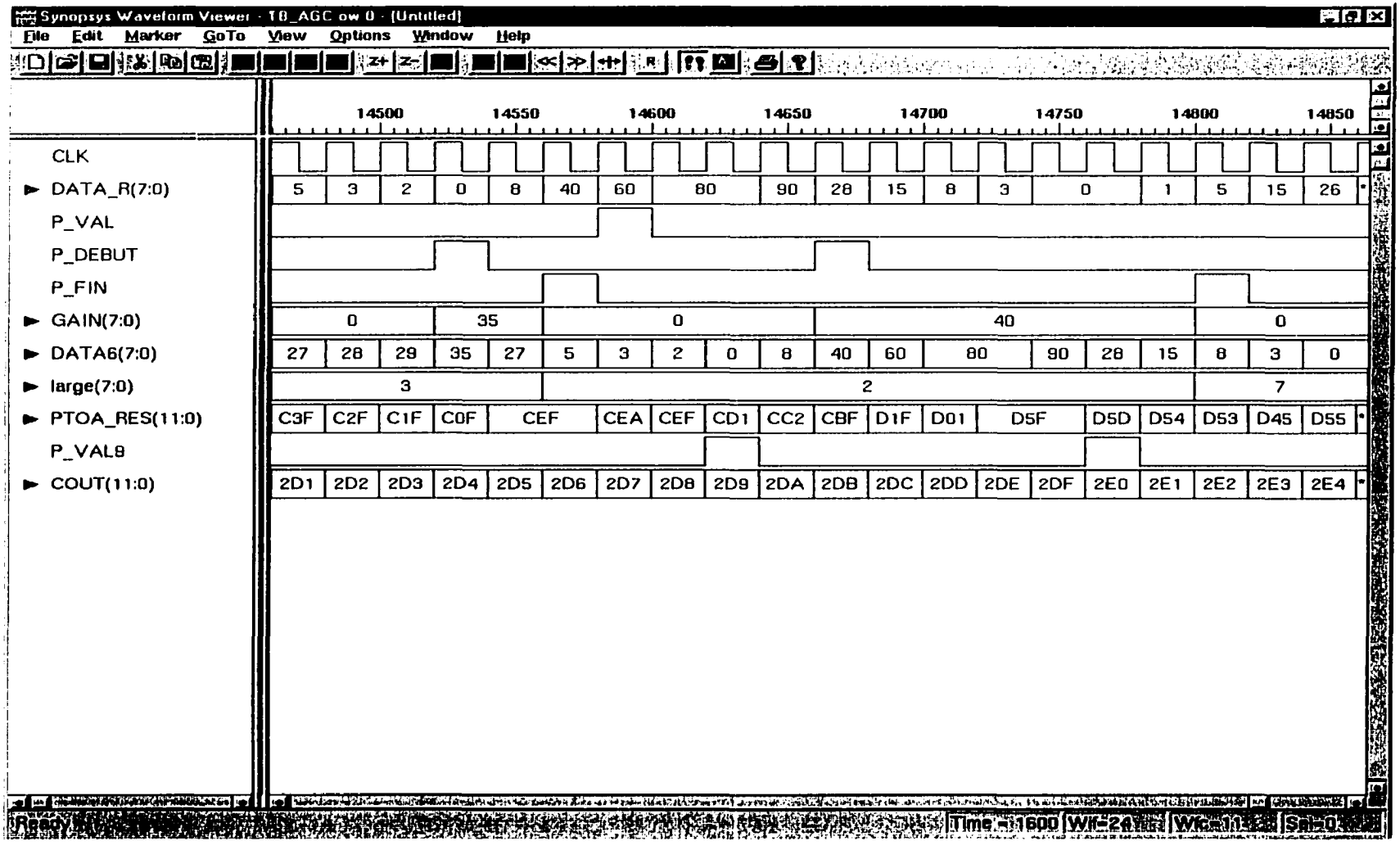


Figure 2-9b Simulation VSS, Strarégie 2, MH = 0 et ML = 2

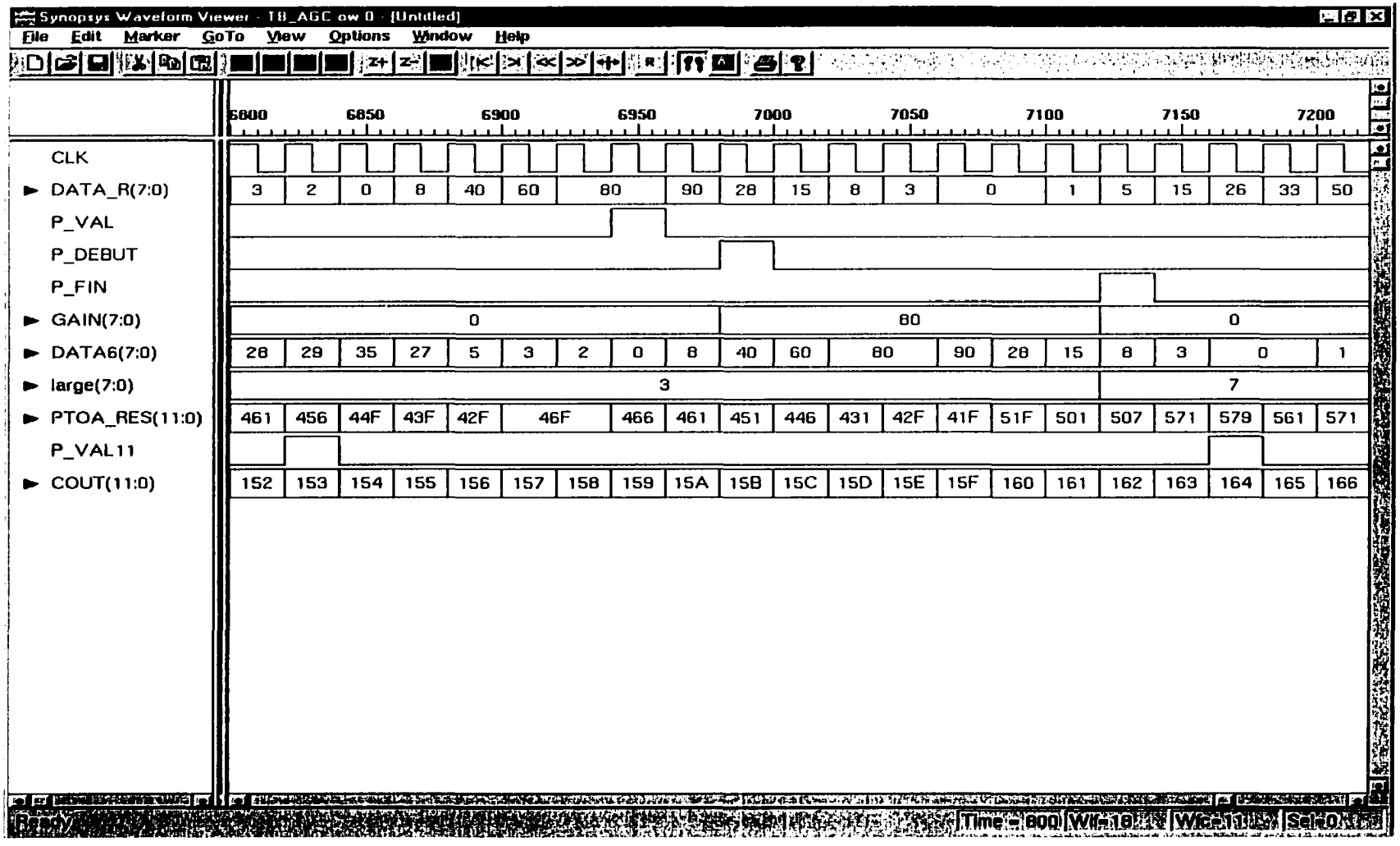


Figure 2-10a Simulation VSS, Stratégie 3, MH = 2 et ML = 2

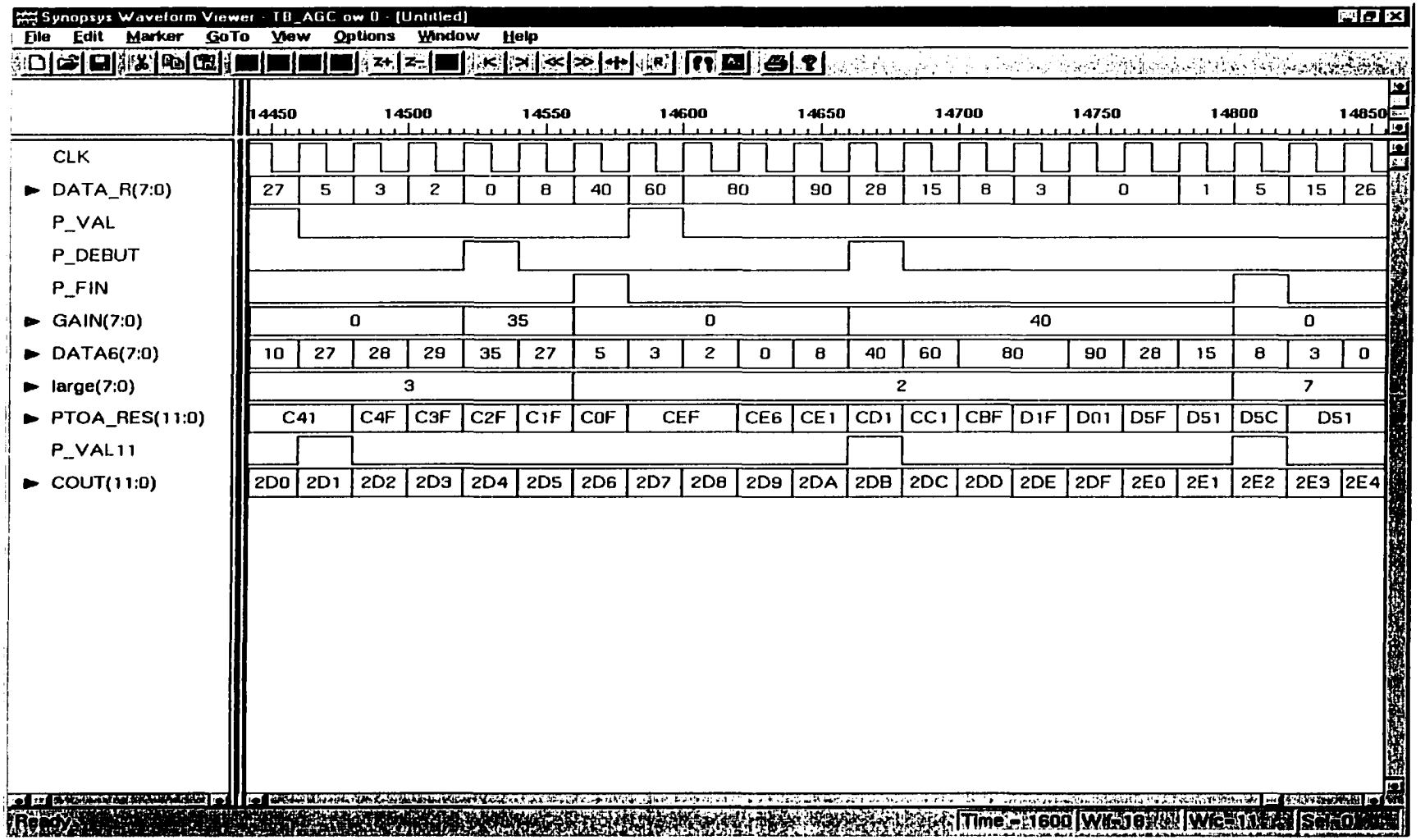


Figure 2-10b Simulation VSS, Stratégie 3, MH = 0 et ML = 2

## 2.5 Conclusion

Ce chapitre a fait l'objet d'une introduction au langage VHDL, suivie par la description RTL détaillée de la structure du circuit sous forme de code VHDL fonctionnel. Le code comporte la description des trois stratégies retenues pour la réalisation du circuit d'estimation. Elles différent par leur type d'interpolation linéaire ou quadratique, et par la méthode de résolution de l'équation premier ou deuxième degré qui en découle. Des exemples des résultats de simulation de l'architecture globale ont été fournis. Ils seront comparés aux résultats de la simulation au niveau porte logique après synthèse qui fera l'objet du chapitre cinq.

# CHAPITRE III

# TECHNOLOGIE DE RÉALISATION

## 3.1 Introduction

Suite à la description des différents modules composant notre circuit, on introduit les circuits numériques programmables de type FPGAs ('Field Programmable Gate Array' ) comme une technologie cible pour la réalisation du prototype de notre projet. La connaissance et la maîtrise des ressources et de l'architecture de tels composants sont autant de conditions indispensables pour mener à bien la réalisation d'un circuit répondant aux spécifications du projet. En effet, le respect des contraintes de vitesse et de surface nécessite une utilisation judicieuse des ressources et une très bonne exploitation des outils de conception.

Ce chapitre permet un survol des caractéristiques des circuits FPGAs en les comparant à celles des circuits dédiés. Ensuite, il présente les architectures des circuits FPGA utilisés pour la réalisation du prototype ainsi que les raisons qui ont dicté ce choix.

## 3.2 Circuits numériques programmables

Les FPGAs représentent un développement récent dans le domaine des circuits VLSI. Ils permettent de réaliser un grand nombre de portes logiques dans des structures matricielles. L'architecture d'un FPGA est similaire à celle d'un circuit prédiffusé. Elle

consiste en une matrice de cellules logiques pouvant être interconnectées par programmation pour réaliser différents designs. La différence entre ces deux types de circuits réside dans la façon de réaliser les interconnexions. Les transistors ou les portes logiques du circuit prédiffusé sont interconnectés en effectuant les différentes couches de métallisation en fonderie, alors que les FPGAs sont programmés par l'utilisateur final, en programmant électriquement des interrupteurs programmables similaires à ceux des circuits PLD ( 'Programmable Logic Device' ) , tels que PLA, PAL et GAL.

Le coût unitaire d'un FPGA est plus élevé que celui d'un circuit dédié de la même densité. Cette différence de coût est largement compensée, pour un volume de production restreint, par la réduction de frais de conception pour réaliser le composant. En plus d'une grande flexibilité qui résulte en un style de conception à faible risque où les conséquences des erreurs logiques sont faibles à la fois en coût et en délais de projet. Les FPGAs sont donc, pratiques pour le développement rapide de produits et de prototypes. Ils permettent des cycles de conception très courts. En plus, ils sont totalement testés après la production, donc les designs ne demandent pas la génération de programmes de test du composant, ni la génération automatique des vecteurs de test ou le design pour la testabilité, mais seulement des tests de type fonctionnels.

Au niveau de la vitesse, les FPGAs offrent des unités qui opèrent à des vitesses approchant les 250 MHz dans plusieurs applications. Évidemment, les vitesses sont plus élevées que celles dans les systèmes réalisés en utilisant des circuits discrets, mais elles restent plus faibles que celles obtenues par les circuits dédiés. La raison principale provient de la programmabilité. Les interconnections programmables rajoutent des résistances aux chemins internes. La vitesse des FPGAs est adéquate pour la plupart des applications. Dans les cas critiques, les applications peuvent être accélérées par simple utilisation d'unités plus rapides, souvent sans modifier la conception du circuit. Avec les circuits dédiés, la situation est totalement différente. Les nouveaux procédés de

fabrication nécessitent la réalisation de masques et augmentent le coût global et le délai de la production.

La programmabilité des FPGAs introduit, sur le composant, de la circuiterie supplémentaire. Elle occupe une surface qui ne sera pas utilisée par le design. Par conséquent, pour la même quantité de logique, le FPGA sera toujours plus gros qu'un circuit dédié et reviendra plus cher.

Le développement des FPGAs s'accompagne par une évolution constante des outils de conception. Ces outils de haut niveau restent abordables même par des petites entreprises de conception. Le temps de développement est composé uniquement du temps de réalisation du prototype et de simulation, tandis que les temps des autres phases nécessaires pour les circuits dédiés : génération des vecteurs de test, production des masques, fabrication des gaufres, mise en boîtier et le test final en manufacture, sont évités. Ce qui mène à un temps de développement pour les FPGAs mesuré en journées ou semaines. Alors que pour les circuits dédiés les durées sont calculées en mois.

## 3.3 Caractéristiques architecturales des FPGAs

L'architecture typique d'un FPGA est donnée par la figure 3-1. Un FPGA consiste en un nombre de cellules logiques utilisées pour réaliser des foncions logiques. Les cellules logiques sont arrangées en matrice. Les ressources d'interconnexion permettent des liaisons programmables entre les entrées sorties des cellules logiques et les blocs d'entrées/sorties. Ces derniers sont utilisés pour connecter le FPGA avec le monde extérieur.

Les cellules logiques sont typiquement capables de réaliser de la logique combinatoire ou séquentielle de différentes complexités. Les FPGAs emploient des cellules logiques basées sur une ou plusieurs techniques tels que les multiplexeurs ou les tables de conversion 'look up tables'.

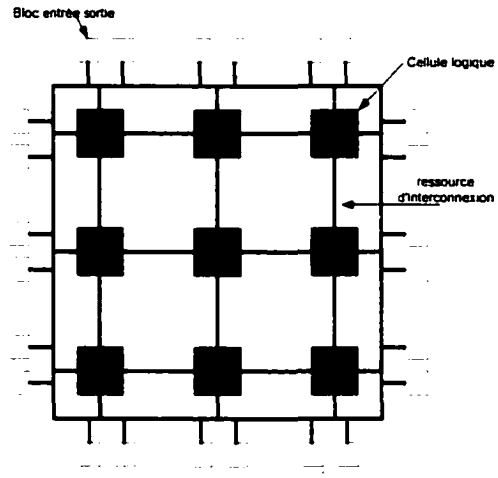


Figure 3-1 Structure d'un circuit FPGA

Pour la programmation des FPGAs, trois technologies sont utilisées. Elles diffèrent par leur densité, coût, performances et maintien de la configuration. Ils peuvent être aussi programmables une seule fois, en utilisant des fusibles ou des antifusibles, ou programmables plusieurs fois, en utilisant des EPROMs ou des SRAMs.

Les cellules à antifusibles sont des interrupteurs ouverts par défaut. Une fois programmés, ils forment des chemins de faible résistance, non réversibles, programmés de façon définitive. Ces cellules ont l'avantage d'être plus rapides et plus denses que les cellules à base des EPROMs et des SRAMs. Les cellules SRAM sont à base de mémoires RAM statiques. Elles ont l'avantage d'être indéfiniment reprogrammables. Par contre, il faut les reconfigurer à chaque mise sous tension.

## 3.4 Choix de technologie

Sur le marché des composants numériques programmables, on trouve plusieurs fournisseurs, tels que : ALTERA, ACTEL, et XILINX. La réalisation de notre circuit est faite sur des composants de XILINX. Ce choix est dicté par l'expertise développée au sein du laboratoire LACIME ainsi que la disponibilité des outils et du matériel conçu par cette technologie. La réalisation a été entreprise en deux étapes, ce qui explique l'utilisation des deux circuits : le XC4028EX et le Virtex XCV600. Le premier représente une solution transitoire. En effet, on a profité de la disponibilité de la carte 'mezzanine'. Cette carte de développement comporte quatre circuits XC4028. Elle nous a permis de réaliser et de tester le fonctionnement des premiers modules en attendant la réalisation du prototype final à base du Virtex XCV600. Ce dernier était choisi à cause de sa capacité au niveau CLBs, ses multiples ressources de routage et de calcul arithmétique ainsi que sa mémoire dédiée, cette dernière étant à la base de la première implémentation par Viewlogic. Ces caractéristiques répondent largement aux dimensions de notre circuit, et permettent aussi de disposer de la surface pour le développement futur de l'unité de prétraitement.

Les deux paragraphes suivants donnent plus de détails sur les caractéristiques de ces deux composants.

### 3.4.1 FPGA XC4028

C'est un FPGA de la famille XC4000. Il comporte 1024 CLBs et 256 blocs d'entrées/sorties. Ses multiples ressources d'interconnexion permettent d'atteindre des vitesses système de l'ordre de 80 MHz. Chaque CLB est doté d'une ressource dédiée pour les calculs arithmétiques 'Fast Carry Logic'. Un survol de l'architecture est donné dans les paragraphes suivants.

### 3.4.1.1 Les blocs logiques (CLB)

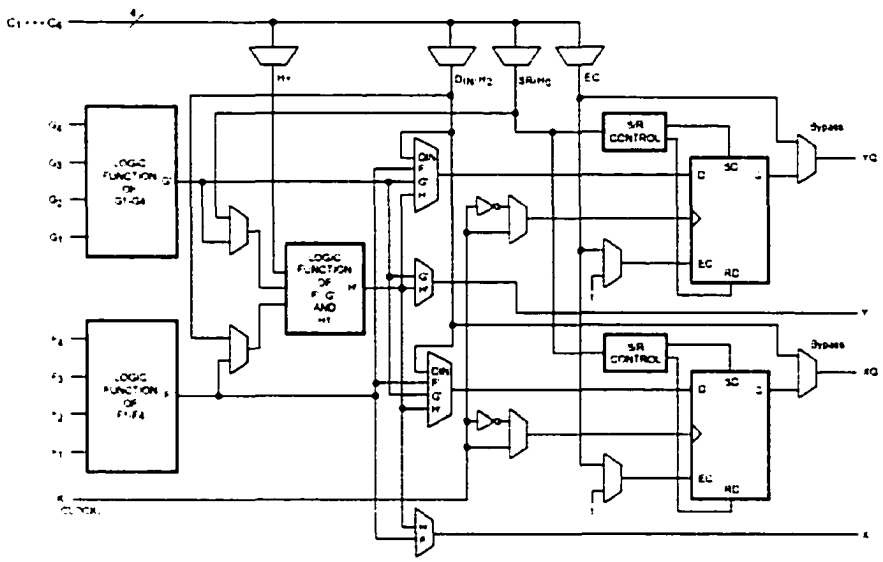

Figure 3 -2 Schéma simplifié d'un bloc logique (CLB)

Les blocs logiques de base, appelés CLB (Configurable Logic Block), réalisent l'essentiel des fonctions logiques du composant. On trouve à l'entrée, deux générateurs de fonction ( F et G) à quatre variables. Un troisième générateur de fonctions (H) permet la combinaison du résultat des deux premiers avec une variable extérieure. Les générateurs de fonctions constituent des LUTs (Look-Up Tables). Ils permettent la réalisation de la table de vérité de la fonction à réaliser. Chaque CLB contient deux bascules, les sorties des générateurs peuvent être dirigées directement vers l'extérieur du CLB (sortie X,Y) ou passer par les bascules internes (XQ, YQ).

Les CLBs peuvent être configurés en mémoires ROM, RAM à simple ou à double port. Le fonctionnement est à déclenchement sur niveau ou sur front. Selon le mode

opératoire, chaque CLB peut être transformé respectivement en réseau de 16x2, 32x1 ou 16x1 bits.

### 3.4.1.2 Les blocs Entrée/Sortie

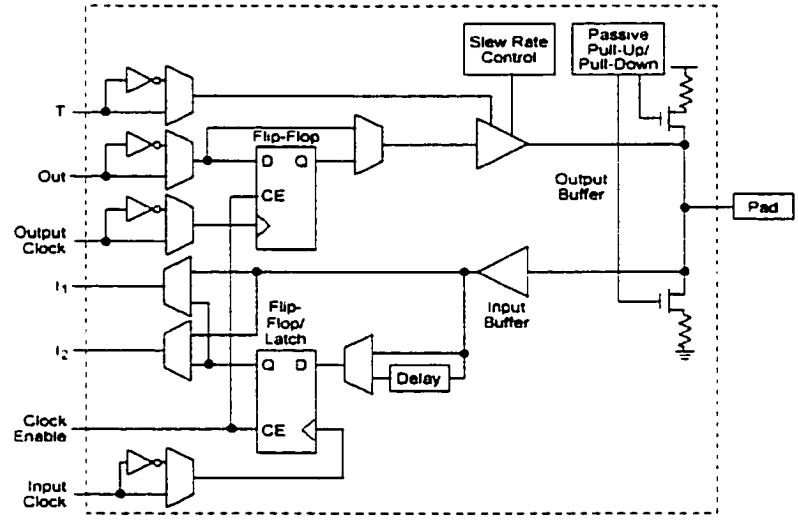


Figure 3-3 Schéma simplifié d'un bloc d'entrée/sortie

Les blocs d'entrées/sorties (IOB) ont une configuration programmable. Ils réalisent l'interface entre les broches d'entrées/sorties et la logique interne. Les niveaux peuvent être programmés pour des niveaux compatibles CMOS ou TTL. La figure 3-3 illustre le schéma de principe. On y retrouve une bascule D associée au signal d'entrée et de sortie, un amplificateur pouvant positionner la sortie en état de haute impédance. Les modules de retard permettent de compenser le retard engendré par la distribution interne du signal d'horloge.

#### 3.4.1.3 Le système d'interconnexion

Chaque CLB est associé à plusieurs ressources de routage. Cette structure se répète d'un CLB à l'autre de manière à obtenir une structure globale régulière bien adaptée aux algorithmes des outils de routage.

La ressource principale d'interconnexion est constituée de lignes simples et doubles. Les lignes simples sont des segments métalliques dont la longueur correspond à une zone CLB. Chaque segment peut être connecté à son homologue par l'intermédiaire d'une matrice programmable.

Des lignes, dites longues, forment une grille de segments métalliques traversant la matrice des CLBs de bout en bout. Ces lignes seront particulièrement utilisées pour router les signaux critiques ou à forte charge entre des CLBs éloignés les uns des autres. Parmi les six lignes horizontales, deux sont parfaitement continues et peuvent être pilotées par les deux amplificateurs trois états associés à chaque CLB. Les autres sont séparées en quatre segments pouvant être reliés entre eux par des connexions actives.

#### 3.4.1.4 Le réseau d'horloges

Huit broches d'entrée sont spécifiquement dédiées à des signaux d'horloge. Le signal de chacune de ces broches est dirigé vers deux amplificateurs de nature différente. Les huit amplificateurs appelés BUFGLS sont capables de supporter une très forte charge et les horloges qu'ils génèrent sont accessibles par tous les CLBs et les IOBs du composant. Les huit amplificateurs appelés BUFGE supportent une charge moins élevée mais leur temps de transfert est plus faible que celui des BUFGLS. Les amplificateurs BUFGE sont groupés par deux dans chaque coin du FPGA et les signaux qu'ils génèrent n'accèdent qu'à un quart du composant. Pour chacun des deux types d'amplificateurs,

l'horloge a une très faible variation de phase, ce qui est important pour la conception des systèmes synchrones. Il est ainsi possible de synchroniser l'ensemble du composant à partir d'une horloge fournie sur une des broches GCK1 à GCK8.

### 3.4.2 Virtex XCV600

Le XCV600 fait partie de la famille Virtex de Xilinx. Il comporte 2056 CLBs et 512 blocs d'entrée/sortie programmables compatibles avec plusieurs standards. En plus de ses nombreuses ressources de routages, ses multiples possibilités de distribution du signal d'horloge, il est doté de 98304 bits de mémoire RAM dédiée. Les paragraphes suivants présentent un survol de certains aspects de son architecture.

#### 3.4.2.1 Les blocs logiques CLB

L'architecture du Virtex possède une structure plus fine que celle de la famille du XC4000. Elle est basée sur des cellules logiques (LC). Chaque LC comporte un générateur de fonction à quatre entrées, un contrôleur logique pour traiter et propager la retenue, et une bascule D. La sortie du générateur peut être directe ou issue de la bascule D. Chaque CLB de cette famille est constitué de quatre LCs, organisées par paire, formant deux unités appelées 'Slice', comme le montre la figure 3-4. En plus de ces deux slices, le CLB contient de la logique qui permet de combiner les sorties des quatre générateurs donnant ainsi la possibilité de générer des fonctions pouvant avoir jusqu'à dix-neuf entrées dans certains cas.

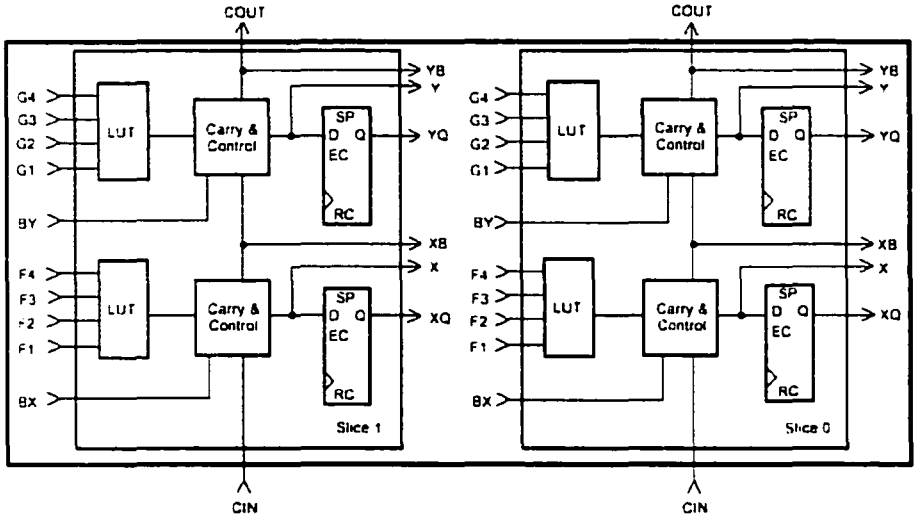


Figure 3-4 Schéma simplifié d'un CLB du Virtex.

Tout comme la famille XC4000, les CLBs peuvent être configurés pour former de la mémoire RAM synchrone de 16 bits. En plus, les deux LUTs d'un Slice peuvent être combinés pour former une mémoire RAM de 16x2 bits ou 32x1 bits ou une mémoire RAM à double port de 16x1 bits.

### 3.4.2.2 Les blocs d'entrée/sortie

Les blocs d'entrées/sorties (IOB) sont programmables. Leurs niveaux peuvent être programmés pour être compatibles avec treize standards différents. Tous les broches d'entrée/sortie sont protégées contre les décharges électrostatiques et les surtensions transitoires. Comme le montre la figure 3-5, chacun des signaux d'entrées et de sorties est associé à une bascule D, un amplificateur pouvant positionner la sortie en état de haute impédance est munie d'un signal de contrôle direct ou pouvant passer par une bascule.

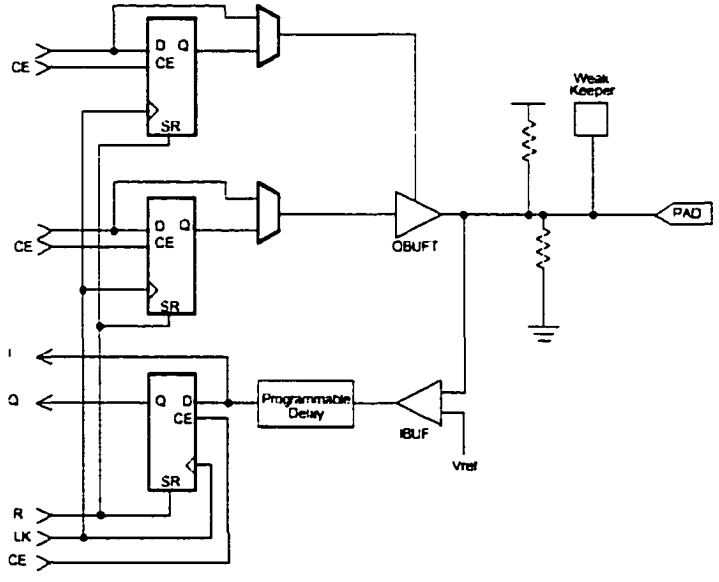


Figure 3-5 Schéma simplifié d'un bloc d'entrée/sortie du Virtex

### 3.4.2.3 Système d'interconnexion

Le routage est basé sur une architecture hiérarchique très élaborée. Cette structure comporte plusieurs niveaux. On retient les quatre principaux :

- Routage local : Il permet l'interconnexion entre les LUTs d'un même CLB, entre ces LUTs et les matrices de routage global appelées GRM ('Global Routing Matrix'), et entre les CLBs horizontalement adjacents.
- Routage à usage général : Représenté par les canaux de routage horizontaux et verticaux qui relient les lignes et les colonnes de CLBs. La majorité des interconnexions est située à ce niveau de la hiérarchie.
- Routage dédié : Chaque rangée de CLBs dispose de quatre longues lignes horizontales pouvant servir à réaliser les bus à trois états. Chaque CLB contient deux lignes verticales dédiées pour la propagation de la retenue.

- Routage global : Permet la distribution des signaux d'horloge ainsi que les signaux alimentant des grosses charges ( fanout).

#### 3.4.2.4 Distribution du signal d'horloge

Bien que les ressources de routage global permettent la distribution du signal d'horloge avec un faible biais de synchronisation, le Virtex est doté de quatre amplificateurs globaux dédiés pour les signaux d'horloge. Chaque amplificateur est associé à une boucle à verrouillage de phase appelée DLL ('Delay Locked Loop'). Cette dernière sert à générer des horloges multiples ou sous multiples de l'horloge principale, éliminer le biais de synchronisation, et contrôler les différents domaines d'horloge.

## 3.5 Conclusion

Dans ce chapitre, nous avons introduit les circuits FPGAs comme une technologie cible pour la réalisation des circuits numériques. L'exploitation adéquate de ces circuits programmables passe par la maîtrise des ressources offertes et une connaissance de l'architecture de tels composants. Cette technologie répond aux exigences et à la nature de notre projet.

En effet, au niveau du coût, elle représente une solution idéale pour la réalisation d'un prototype. Grâce à leur reprogrammabilité, il était possible de réaliser et comparer différentes stratégies. La combinaison de la technologie FPGA avec une description en VHDL a permis de réduire considérablement les temps de conception.

Dans le chapitre suivant, nous exposerons et comparerons les résultats d'implémentation obtenus pour les différentes stratégies définies dans le chapitre 2.

# CHAPITRE IV

# RÉSULTATS DE RÉALISATION

## 4.1 Introduction

Dans ce chapitre, les différents résultats de réalisation seront exposés. La première section présentera le processus de conception à base de circuits FPGA. Ensuite, nous présenterons les résultats détaillés de la réalisation pour chacune des trois stratégies de conception de l'estimateur, ainsi qu'une comparaison entre ces différentes solutions.

## 4.2 Processus de conception à base de circuits FPGA

La conception de circuits numériques complexes à base de FPGA peut se résumer à cinq étapes : la description RTL, la simulation fonctionnelle du code, la synthèse, le placement et routage et la simulation au niveau portes logiques. Les deux premières étapes ont fait l'objet du chapitre deux. Les trois autres seront introduites dans les sections suivantes.

### 4.2.1 Synthèse VHDL

Après la description du circuit, et sa simulation fonctionnelle, l'étape suivante est la synthèse et l'optimisation du circuit. Cette étape consiste à traduire la description VHDL, du circuit numérique à réaliser, en portes logiques. Dans notre cas, nous avons utilisé le logiciel **FPGA COMPILER II**. Le code VHDL en entrée doit être

synthétisable. Les modules de mémoires utilisés dans la phase de simulation n'étant pas synthétisables, nous avons eu recours au logiciel **CORGEN**, pour la synthèse de ces modules. **CORGEN** génère des modules synthétisables et optimisés pour le type de FPGA utilisé. À partir du reste du code, le logiciel **FPGA COMPILER II** procède en deux phases. Dans la première, il compile les différents fichiers. Dans la seconde, il les optimise selon le contenu du fichier de contraintes et le type de circuit programmable ciblé. Le résultat est un fichier de type '.edf', des rapports de synchronisation et de surface. Les fichiers '.edn' générés par **CORGEN** et les fichiers '.edf' serviront d'entrée pour l'outil de placement et routage. Cette phase fera l'objet des paragraphes suivants.

### 4.2.2 Placement et routage

Cette opération est effectuée à l'aide du logiciel **Alliance M3.1** de XILINX. Il accepte comme entrée, les fichiers '.edf' et '.edn' générés précédemment, ainsi qu'un fichier de contraintes de type '.ucf'. Ce dernier est composé essentiellement de la localisation de broches d'entrée/sortie, ainsi que la contrainte sur la période de l'horloge. Il permet aussi de contraindre un chemin particulier ayant comme départ une broche d'entrée ou la sortie d'une bascule, et se terminant par l'entrée d'une bascule ou une broche de sortie. Un modèle de fichier de contraintes est donné en annexe. Comme le montre la figure 4-1, la génération du fichier '.mcs', qui servira pour la programmation du FPGA, est obtenue en passant par différentes étapes : Traduction, partionnement, placement et routage, synchronisation et configuration.

Durant l'étape de traduction 'Translate', le fichier '.edf' est converti en format NGD (Native Generic Database). Un rapport sera généré. Il contient les messages d'erreurs, les avertissements et la liste des blocs manquants ou non traduits.

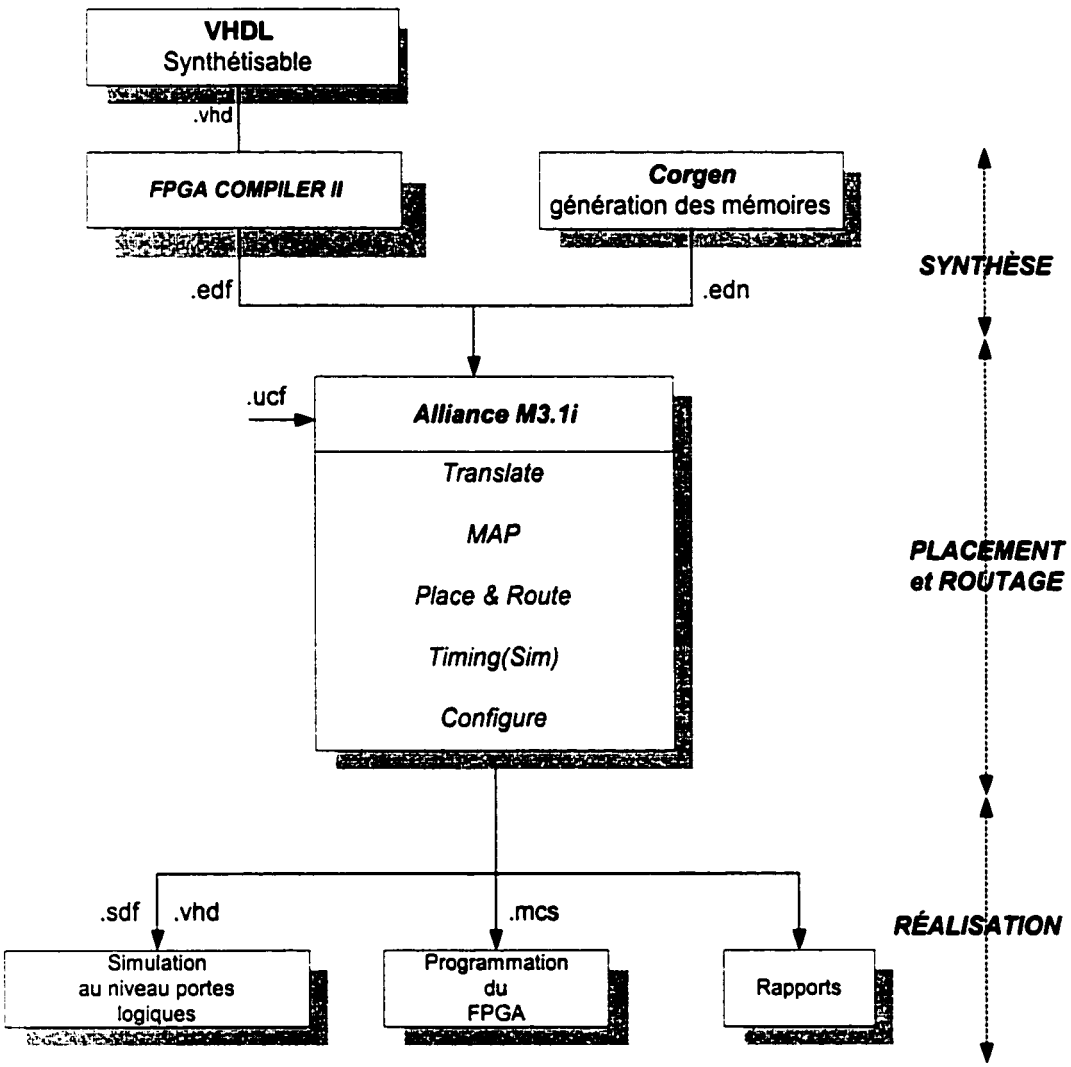


Figure 4-1 Processus de réalisation à base de FPGA

Durant l'étape de répartition 'MAP', le logiciel effectue une vérification des règles de design (DRC), sur le contenu du fichier '.ngd'. Ensuite il répartit la logique du design dans les différentes ressources du FPGA. Le rapport généré précise les parties de la logique enlevées, à cause de l'optimisation, ou rajoutées , pour augmenter la vitesse du circuit .

Durant l'étape 'Place & Route', le logiciel répartit les différentes fonctions combinatoires ainsi que les registres dans les CLBs. Ensuite, il procède au routage des différents signaux entre les CLBs. Le rapport résultant de cette étape récapitule le nombre de ressources utilisées, le nombre de signaux non routés et la localisation des broches. Un deuxième rapport,'Post Layout Timing Report', fournit une analyse du de la synchronisation pour déterminer la fréquence maximale.

L'étape 'Timing(Sim)' produit un fichier VHDL, pour la simulation au niveau portes logiques en tenant compte des délais introduits par le routage.

L'étape 'configuration' génère un fichier '.bit' qui sera utilisé pour la programmation finale du FPGA.

### 4.2.3 Simulation après routage

Pour vérifier le résultat de la synthèse et du placement et routage, on effectue une simulation au niveau porte logique. Elle combine le 'netlist' qui a suivi le placement et routage avec le fichier de type '.sdf' généré suite à l'étape 'Timing(sim)'. Cette simulation tient donc compte des délais des portes logiques et de ceux introduits par le routage. Ainsi, elle permet de vérifier simultanément la fonctionnalité et la synchronisation.

## 4.3 Résultats de réalisation

Rappelons que nous avons effectué la description de trois stratégies différentes pour la réalisation de l'estimateur :

- Interpolation linéaire avec inversion de la pente
- Interpolation linéaire avec subdivision de l'intervalle [M0,M1]

- Interpolation quadratique avec subdivision de l'intervalle [M0,M1]

Dans ce qui suit nous exposerons les résultats de réalisation obtenus après placement et routage pour ces différentes stratégies.

### 4.3.1 Interpolation linéaire avec inversion de la pente

Le tableau 4-1 représente les résultats obtenus lors du placement et routage de la première stratégie, d'abord avec un FPGA de la famille XC4000 et ensuite avec un de la famille VIRTEX.

Tableau 4-1

Résultats de réalisation pour la linéaire et inversion

| | XC4028-4 | XCV600 |
|---|---|---|
| Fréquence | **48.525 MHz** | **62.635 MHz** |
| Portes logiques | **34695** | **48924** |
| *Slices* ou *CLBs* | **748** | **599** |
| *Flip-flop* | **790** | **719** |
| LUT | **1231** | **752** |
| *Select RAM* | **24** | **16** |
| *Bloc RAM* | **1** | **2** |
| Latence pour PTOA | **14** | **13** |

On constate que les deux alternatives utilisent sensiblement le même nombre de ressources. Au niveau de la fréquence maximale du circuit, la réalisation avec le XC4028 résulte en une fréquence 25% plus faible que celle obtenue avec le VIRTEX. Pour une latence de 13 cette fréquence chute à d'environ 41MHz. Même en augmentant la latence et en dupliquant la logique sur les chemins critiques, on n'est pas arrivé à la

fréquence exigée de 50 MHz. Dans ce cas, la taille du FPGA devient la source principale de limitation, ce qui justifie l'utilisation d'un FPGA plus performant. Le choix d'un FPGA de la famille XC4000 avec un grade de vitesse supérieur répondra aux exigences. Nous avons opté pour la famille VIRTEX dans un soucis de suivre l'évolution technologique, et dans la perspective de disposer de plus de ressources pour le développement futur du circuit.

### 4.3.2 Interpolation linéaire avec subdivision de [M0,M1]

Le tableau 4-2 représente les résultats obtenus lors du placement et routage de la deuxième stratégie. Il comporte aussi les résultats d'une réalisation basée sur une entrée schématique à l'aide du logiciel Viewlogic.

Tableau 4-2

Résultats de réalisation pour la linéaire avec subdivision

| | XCV600<br>Entrée en VHDL | XCV600<br>Entrée schématique |
|---|---|---|
| Fréquence | **56.370 MHz** | **56.912 MHz** |
| Portes logiques | **34128** | **29457** |
| ***Slices*** | **733** | **730** |
| *Flip-flop* | **779** | **610** |
| *LUT* | **864** | **-** |
| *Select RAM* | **16** | **24** |
| *Bloc RAM* | **1** | **16** |
| Latence max. pour le PTOA | **13** | **14** |

On constate que les deux alternatives utilisent le même nombre de 'slices'. Ils atteignent pratiquement la même fréquence maximale environ 56 MHz. La solution VHDL utilise beaucoup moins de mémoires. La différence est de 8 modules 'Select RAM' et 15 modules 'Bloc RAM'. L'équivalent en portes logiques de cette quantité de mémoire est très important. Cette différence est due au choix de la réalisation des 15 multiplicateurs. Dans la solution en entrée schématique, ces multiplicateurs ont été réalisés uniquement par la mémoire dédiée. Dans le cas de l'entrée VHDL, les multiplicateurs ont été réduits à des simples additions et décalages, l'outil d'optimisation ayant complété le reste.

### 4.3.3 Interpolation quadratique avec subdivision de [M0,M1]

Le tableau 4-3 représente les résultats obtenus lors du placement et routage de la troisième stratégie.

Tableau 4-3
Résultats de réalisation pour la quadratique avec subdivision

| | XCV600 |
|---|---|
| Fréquence | **55.679 MHz** |
| Portes logiques | **44912** |
| *Slices ou CLBs* | **1061** |
| *Flip-flop* | **1211** |
| *LUT* | **1325** |
| *Select RAM* | **24** |
| *Bloc RAM* | **1** |
| Latence max. pour le PTOA | **15** |

## 4.4 Comparaison des résultats des différentes stratégies

Le tableau 4-4 récapitule les résultats de réalisation des trois stratégies retenues pour la conception de l'estimateur.

Tableau 4-4
Résultats de réalisation des trois stratégies

| | Linéaire inversion | Linéaire subdivision | Quadratique subdivision |
|---|---|---|---|
| Fréquence | **62.635 MHz** | **56.370 MHz** | **55.679 MHz** |
| *Slices ou CLBs* | **599** | **733** | **1061** |
| *Flip-flop* | **719** | **779** | **1211** |
| *LUT* | **752** | **864** | **1325** |
| *Select RAM* | **16** | **16** | **24** |
| *Bloc RAM* | **2** | **1** | **1** |
| Latence pour le PTOA | **13** | **13** | **15** |
| Précision relative moyenne | **3.9 %** | **3.2 %** | **3.2 %** |

On constate que les trois stratégies répondent aux spécifications du projet. Les résultats de la première stratégie sont plus intéressants au niveau du nombre de ressources utilisées, ainsi qu'au niveau de la fréquence maximale. En tenant compte du nombre des 'Slices' occupés par le détecteur, qui est de l'ordre de 50 'Slices', on peut déduire que la complexité de la quadratique est pratiquement le double de celle de la première stratégie. Au niveau de la précision, la stratégie avec inversion de la pente devrait donner un meilleur résultat par rapport aux stratégies avec subdivision. La première est basée sur un calcul alors que les deux autres sont basées sur une approximation. Or, le résultat obtenu sur un ensemble d'impulsions fournit par le DREO prouve le contraire. Cela s'explique par l'erreur de quantification commise sur les

valeurs mémorisées dans la table de correspondance servant à calculer l'inverse de la pente. Cette erreur augmente avec la valeur de la pente. L'amélioration de la précision de la première stratégie passe donc par l'augmentation du nombre des bits utilisés pour le calcul de l'inverse de la pente. Toutefois la précision relative de 3.9% répond aux spécifications du projet.

## 4.5 Conclusion

Ce chapitre nous a permis de présenter les résultats de réalisation des trois stratégies retenues. Ces résultats montrent que la logique utilisée répond aux exigences du projet, et que le choix du VIRTEX est bien justifié. Ils nous ont permis de confirmer que l'interpolation linéaire autour d'un seuil adaptatif donne de bons résultats. La stratégie avec inversion de la pente donne des résultats meilleurs que ceux obtenus par la méthode de subdivision de l'intervalle [M0,M1], au niveau de la fréquence maximale, de la latence et de la quantité des ressources utilisées. L'interpolation quadratique n'a pas permis d'améliorer la précision. Malgré l'utilisation de 50 % de plus des ressources que dans la première stratégie, sa précision reste identique à celle de la deuxième.

Les résultats obtenus montrent que l'entrée en VHDL n'a pas apporté de gain significatif par rapport à une entrée schématique au niveau de la quantité des ressources utilisées. Les avantages du VHDL résident dans sa flexibilité, sa concision, ses différents styles de description, et son indépendance de la technologie. Les corrections et les modifications du code sont plus simples et plus rapides que pour des schémas. La standardisation du VHDL permet d'utiliser différentes plates-formes, différents simulateurs et surtout différents outils de synthèse. La performance de ces derniers est d'une importance capitale. En effet, l'optimisation a une influence capitale sur les performances du circuit synthétisé. L'équivalence de ces résultats peut être attribuable à

notre méthode de conception orientée vers la synthèse avec une perspective de porte logique.

# CONCLUSION

Nos objectifs pour ce projet étaient de décrire en VHDL et de réaliser un circuit numérique pour la détection des impulsions radar et l'estimation de leur temps d'arrivée. La description du circuit est effectuée en vue de cibler une technologie du type FPGA. Le circuit devrait traiter des mots de 8 bits en format non signé à une fréquence d'horloge de 50 MHz. La latence maximale de détection devrait être de 200 ns, tandis que l'estimation du temps d'arrivée des impulsions devrait être de 1 ns en absence du bruit. Ce qui correspond à une erreur relative de 5%.

À partir de ces spécifications, et du travail effectué lors d'une étude précédente sur Viewlogic, nous avons retenu trois stratégies différentes pour réaliser l'estimateur. Ces trois stratégies répondent aux exigences. Elles se distinguent par le type d'interpolation de l'impulsion autour d'un seuil adaptatif, ainsi que par la méthode de résolution de l'équation qui en découle.

Dans la première stratégie, nous avons utilisé une interpolation linéaire de l'impulsion autour d'un seuil adaptatif. La résolution de l'équation du premier degré nécessite le calcul d'une division de deux nombres. Pour réduire la complexité du circuit, le calcul de cette division a été remplacé par la détermination de l'inverse du dénominateur suivi d'un produit du résultat par le numérateur. Le résultat de réalisation indique une fréquence maximale de 62 MHz, une latence de détection de 6, une latence d'estimation de 13 et une erreur relative moyenne de 3.9 %.

Dans la deuxième stratégie, nous avons utilisé aussi une interpolation linéaire. La résolution de l'équation est obtenue en subdivisant l'intervalle des deux échantillons, entourant le seuil adaptatif, en 16 sous-intervalles et en calculant la valeur de la droite

d'interpolation pour ces 15 points. L'estimation du PTOA correspond au point ayant l'ordonnée la plus proche du seuil. Les résultats de la réalisation indiquent une fréquence maximale de 56 MHz, des valeurs de latence maximale similaires à celles obtenues par la première stratégie et une erreur relative moyenne de 3.2 %.

Dans la troisième stratégie, nous avons utilisé une interpolation quadratique. La résolution de l'équation du second ordre est similaire à celle utilisée pour la deuxième stratégie. Les résultats de réalisation indiquent une fréquence maximale de 56 MHz, une latence maximale d'estimation de 15 et une erreur relative moyenne égale à 3.2 %.

À la lumière des résultats de réalisation pour les trois stratégies, on constate que la première stratégie offre les meilleurs résultats au niveau de la vitesse, ainsi qu'au niveau de la complexité : elle occupe environ le même nombre de CLB que la deuxième stratégie et presque la moitié du nombre utilisé par la quadratique. Par contre, au niveau de la précision, les deux autres stratégies présentent une erreur relative moyenne plus faible, sur un ensemble d'impulsions de test fournit par le DREO. L'amélioration de la précision passe par la diminution de l'erreur de quantification commise sur le calcul de l'inverse de la pente.

La comparaison des résultats de réalisation de la deuxième stratégie avec ceux obtenus suite à une entrée schématique par Viewlogic, montre que la description VHDL n'a permis de réaliser aucun gain au niveau de la complexité, de la précision et de la fréquence maximale. Cela confirme que le VHDL reste un outil qui ne pourra pas se substituer au savoir-faire. La maîtrise de la conception au niveau porte logique est une condition nécessaire pour réussir une bonne description VHDL en vue de la synthèse. Le VHDL apporte la flexibilité du langage par rapport à la rigidité du schéma, permet d'explorer facilement différentes solutions, et d'essayer différentes technologies. Le passage d'une technologie à une autre est très simple, alors que dans le cas d'une entrée

schématique, le passage d'une famille de FPGA à une autre demande beaucoup plus de temps et d'efforts.

La combinaison du VHDL et du FPGA, nous a permis d'essayer plusieurs alternatives de conception au niveau simulation et ensuite au niveau du FPGA lui même. La conception a été conduite de façon progressive. À chaque étape, le code est simulé, synthétisé et testé au niveau du FPGA, ce qui a facilité et accéléré la mise au point du circuit.

Dans le cadre des travaux futurs, nous recommandons une étude approfondie de l'erreur relative ainsi que l'effet de la quantification sur la précision des différentes stratégies de réalisation de l'estimateur. Il serait intéressant de tirer profit d'une technologie du type ASIC puisque le code VHDL le permet. Le passage nécessite des modifications au niveau des mémoires. Ces dernières étant générées spécifiquement pour la famille du FPGA considéré. Pour la solution ASIC, nous disposons de deux alternatives : Utiliser des cellules optimisées par le fournisseur de la technologie cible, ou bien les synthétiser par des registres puisque la quantité de mémoire n'est pas très importante. Dans ce cas le code comportemental de simulation des mémoires sera fourni directement à l'outil de synthèse. Ensuite les autres modifications seront effectuées automatiquement par l'outil de placement et routage.

## BIBLIOGRAPHIE


[1] Bongo, I., (2000). Détection des signaux radar et estimation de leurs temps d'arrivée. mémoire de maîtrise. École de technologie supérieure. Montréal.

[2] Chang, K. C. , (1997). Digital design and modeling with VHDL and synthesis. Los Alamitos, Californie. : IEEE Computer Society Press.

[3] Kurup, P., Abbasi T., (1995). Logic synthesis using synopsys (2$^{e}$ éd.). Norwell, Massachussetts : Kluwer Academic Publishers.

[4] Merrill, I., Shonik(1980). Introduction to Radar Systems. Mc Graw-Hill Book Company.

[5] Perry, D., (1998). VHDL (3$^{e}$ éd.). New York : McGraw-Hill.

[6] Savaria, Y., (1988). Conception et vérification des circuits VLSI. Montréal Quebec éditions de l'École Polytechnique de Montréal

[7] Merrill, I., Shonik(1980). Introduction to radar systems. Mc Graw-Hill Book Company.

[8] The Programmable Logic Data Book 2000, Xilinx.

[9] Zoran, S., (1998). VHDL and FPLDs in digital systems design, prototyping and customization. Massachusetts : Kluwer Academic Publishers.

# ANNEXE A

# STRATEGIE AVEC INVERSION DE LA PENTE

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

--------------------------------------------------------------------
--
-- Fichier : agc.vhd
--
-- detection des impulsions
-- estimation de leur temps d'arrivee
--
--------------------------------------------------------------------

entity agc is
        port(CLK,RESET      : in  std_logic;
             DATA_PULSE     : in  std_logic_vector(7 downto 0);
             TH1, TH2       : in  std_logic_vector(7 downto 0);
             MH, ML         : in  std_logic_vector(1 downto 0);
             ECART          : in  std_logic_vector(7 downto 0);
             GAIN           : out std_logic_vector(7 downto 0);
             P_VAL          : out std_logic;
             P_DEBUT        : out std_logic;
             P_FIN          : out std_logic;
             LARGE          : out std_logic_vector(7 downto 0);
             PTOA_RES       : out std_logic_vector(23 downto 0));
end agc;


architecture comport of agc is


  component detecteur
  port( CLK, RESET   : in  std_logic;
        IPAD_IN      : in  std_logic_vector(7 downto 0);
        TH1, TH2     : in  std_logic_vector(7 downto 0);
        MH, ML       : in  std_logic_vector(1 downto 0);
        DCO          : in  std_logic_vector(7 downto 0);
        MAX          : out std_logic_vector(7 downto 0);
        SEUIL        : out std_logic_vector(31 downto 0);
        START_DET    : out std_logic;
        END_DET      : out std_logic);
  end component;

  component gain_s
  port( CLK            : in  std_logic;
        RESET          : in  std_logic;
        MH             : in  std_logic_vector(1 downto 0);
        ML             : in  std_logic_vector(1 downto 0);
        MAX_IN         : in  std_logic_vector(7 downto 0);
        START_DET      : in  std_logic;
        END_DET        : in  std_logic;
        GAIN_OUT       : out std_logic_vector(7 downto 0);
        PULSE_START_F  : out std_logic;
```

```
        PULSE_END_F    : out std_logic);
  end component;

  component ptoa
  port( CLK,RESET   : in std_logic;
        ECART       : in std_logic_vector(7 downto 0);
        DATA        : in std_logic_vector(7 downto 0);
        SEUIL       : in std_logic_vector(31 downto 0);
        P_START     : in std_logic;
        P_END       : in std_logic;
        LARGE       : out std_logic_vector(7 downto 0);
        RESULT      : out std_logic_vector(23 downto 0));
  end component;


 signal P_START         : std_logic;
 signal P_END           : std_logic;
 signal MAX             : std_logic_vector(7 downto 0);
 signal VAL_PTOA        : std_logic;
 signal SEUIL            : std_logic_vector(31 downto 0);
 signal RESULT          : std_logic_vector(23 downto 0);

begin

 P_VAL         <= P_START;
 PTOA_RES      <= RESULT;


 detect : detecteur port map(CLK,RESET,DATA_PULSE,TH1,TH2,MH,ML,ECART,
                             MAX,SEUIL,P_START,P_END);


ptoa1   : ptoa port map(CLK,RESET,ECART,DATA_PULSE,SEUIL,P_START,P_END,
                        LARGE,RESULT);

 gain1   : gain_s port map(CLK,RESET,MH,ML,MAX,P_START,P_END,GAIN,
                           P_DEBUT,P_FIN);
end ;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

------------------------------------------------------------------
-- Fichier : detecteur.vhd
--
-- detecte les impulsions valides
-- determine  le debut et la fin
-- determine le maximum et calcule le seuil pour l'estimatiom
--
------------------------------------------------------------------

entity detecteur is
port( CLK, RESET : in std_logic;
      IPAD_IN    : in std_logic_vector(7 downto 0);
      TH1, TH2   : in std_logic_vector(7 downto 0);
      MH, ML     : in std_logic_vector(1 downto 0);
      DCO        : in std_logic_vector(7 downto 0);
      MAX        : out std_logic_vector(7 downto 0);
      SEUIL      : out std_logic_vector(31 downto 0);
      START_DET  : out std_logic;
      END_DET    : out std_logic);
end detecteur;

-- IPAD_IN     : arrivee des echantillons
-- TH1         : seuil inferieur
-- TH2         : seuil siperieur
-- DATA_D      : echantillons synchronises
-- MAX         : serie des maximums
-- PULSE_START : debut d'impulsion
-- PULSE_END   : fin d'impulsion


architecture comport of detecteur is

component hseq_det
port( CLK,RESET       : in  std_logic ;
      ENTREE          : in std_logic_vector( 7 downto 0);
      TH2             : in std_logic_vector( 7 downto 0);
      PEE             : in  std_logic ;
      MH              : in  std_logic_vector(1 downto 0);
      HIGH_DET        : out std_logic;
      START_DET       : out std_logic);
end component;

component lseq_det
  port( CLK,RESET       : in  std_logic ;
        ENTREE          : in std_logic_vector( 7 downto 0);
        TH1             : in std_logic_vector( 7 downto 0);
        PEE             : in  std_logic ;
        ML              : in  std_logic_vector(1 downto 0);
```

```
        END_DET          : out std_logic);
end component;


component max_det
port( CLK,RESET     : in  std_logic;
      DATA          : in  std_logic_vector(7 downto 0);
      MAX_INIT      : in  std_logic;
      ECART         : in  std_logic_vector(7 downto 0);
      SEUIL         : out std_logic_vector(31 downto 0);
      MAX           : out std_logic_vector(7 downto 0));
end component;


  signal DATA_R           : std_logic_vector( 7 downto 0);
  signal HIGH_DET         : std_logic;
  signal START_DETI       : std_logic;
  signal END_DETI         : std_logic;
  signal PEE              : std_logic;

  -- DATA_R    : echantillons synchronisees a l'entree
  -- HIGH_DET  : detection d'un echantillon superieur au seuil TH2


begin

  START_DET <= START_DETI;
  END_DET   <= END_DETI;

  hd1  : hseq_det port map(CLK, RESET, DATA_R, TH2, PEE, MH, HIGH_DET,
                           START_DETI);

  ld1  : lseq_det port map(CLK, RESET, DATA_R, TH1, PEE, ML, END_DETI);

  max1 : max_det  port map(CLK, RESET, DATA_R, HIGH_DET, DCO ,SEUIL,
MAX);



-- **********************************
-- determination du PEE
-- **********************************
   pp1 : process(CLK)
   begin
      if( CLK' Event and CLK = '1' ) then
         if ( (RESET or END_DETI) = '1') then
            PEE <=  '0';
         elsif START_DETI = '1' then
            PEE <= '1';
         else
            PEE <= PEE;
         end if;
      end if;
```

```
    end process;

--*************************************************************************
--  Registre de synchronisation des impulsions a l'entree
--*************************************************************************
   pp2 : process(CLK,RESET)
  begin
    if (RESET = '1') then
       DATA_R <= ( others => '0');
     elsif (CLK'event and CLK = '1') then
       DATA_R <= IPAD_IN;
     end if ;
  end process;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

------------------------------------------------------------------
--
-- Fichier : hseq_det.vhd
--
-- detecte le debut de l'impulsion
-- detection de la sequence haute selon TH2 et MH
--
------------------------------------------------------------------

entity hseq_det is
port( CLK,RESET  : in  std_logic ;
      ENTREE     : in std_logic_vector( 7 downto 0);
      TH2        : in std_logic_vector( 7 downto 0);
      PEE        : in  std_logic ;
      MH         : in  std_logic_vector(1 downto 0);
      HIGH_DET   : out std_logic;
      START_DET  : out std_logic);
end hseq_det;


architecture comport of hseq_det is
signal HIGH_DET1        : std_logic;
signal HIGH_DET2        : std_logic;
signal HIGH_DET3        : std_logic;
signal HIGH_DETI        : std_logic;
signal START_DETI       : std_logic;
signal START_DETI1      : std_logic;

begin

HIGH_DET  <= HIGH_DETI;

--*****************************************
-- determination de debut de pulse
--*****************************************
 un :  process(ENTREE,TH2)
 begin
   if (ENTREE > TH2) then
      HIGH_DETI <= '1';
   else
      HIGH_DETI <= '0';
   end if;
 end process;


--*****************************************
-- detection sequence haute
--*****************************************
```

```
process(MH,PEE,HIGH_DETI,HIGH_DET1,HIGH_DET2,HIGH_DET3)
 begin
  case MH is
    when "00" => START_DETI <= (not PEE) and HIGH_DETI;
    when "01" => START_DETI <= (not PEE) and HIGH_DETI and HIGH_DET1;
    when "10" => START_DETI <= (not PEE) and HIGH_DETI and HIGH_DET1
                                         and HIGH_DET2;
    when others => START_DETI <= (not PEE) and HIGH_DETI and HIGH_DET1
                                  and HIGH_DET2 and HIGH_DET3;
   end case;
 end process;



deux : process(CLK,RESET)
   begin
      if (RESET = '1') then
         HIGH_DET1 <= '0' ; HIGH_DET2 <= '0';
         HIGH_DET3 <= '0';
      elsif (CLK' event and CLK = '1') then
         if ( START_DETI = '1') then
            HIGH_DET1 <= '0' ; HIGH_DET2 <= '0';HIGH_DET3 <= '0';
         else
            HIGH_DET1 <= HIGH_DETI ; HIGH_DET2 <= HIGH_DET1 ;
            HIGH_DET3 <= HIGH_DET2;
         end if;
      end if;
 end process;

-- **********************
--   signal sortie
-- **********************
 trois : process(CLK, RESET)

 begin
   if (RESET =  '1' ) then
     START_DET <= '0';
   elsif( CLK' EVENT and CLK = '1') then
     START_DET <= START_DETI;
   end if;
 end process;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

------------------------------------------------------------
--
-- Fichier : lseq_det.vhd
--
-- detecte le debut de l'impulsion
-- detection de la sequence haute selon TH1 et ML
--
------------------------------------------------------------


entity lseq_det is
  port( CLK,RESET       : in  std_logic ;
        ENTREE          : in std_logic_vector( 7 downto 0);
        TH1             : in std_logic_vector( 7 downto 0);
        PEE             : in  std_logic ;
        ML              : in  std_logic_vector(1 downto 0);
        END_DET         : out std_logic);
end lseq_det;



architecture comport of lseq_det is

signal LOW_DET1         : std_logic;
signal LOW_DET2         : std_logic;
signal LOW_DET3         : std_logic;
signal LOW_DET          : std_logic;
signal END_DETI         : std_logic;
signal END_DETI1        : std_logic;

begin

-- ************************************************
-- determination de la fin du pulse
-- ************************************************
un :  process(ENTREE,TH1)
 begin
   if (TH1 > ENTREE) then
      LOW_DET <= '1';
   else
      LOW_DET <= '0';
   end if;
 end process;


-- ************************************************
-- detection sequence basse
-- ************************************************
```

```
process(ML,PEE,LOW_DET,LOW_DET1,LOW_DET2,LOW_DET3)
 begin
    case ML is
       when "00" => END_DETI    <=  PEE and LOW_DET;
       when "01" => END_DETI    <=  PEE and LOW_DET and LOW_DET1;
       when "10" => END_DETI    <=  PEE and LOW_DET and LOW_DET1
                                          and LOW_DET2;
       when others => END_DETI <=  PEE and LOW_DET and LOW_DET1
                                          and LOW_DET2 and LOW_DET3;
    end case;
 end process;



deux : process(CLK, RESET)
   begin
      if (RESET = '1') then
         LOW_DET1 <= '0' ; LOW_DET2 <= '0';
LOW_DET3 <= '0';
      elsif (CLK' event and CLK = '1') then
         if (  END_DETI = '1') then
            LOW_DET1 <= '0' ; LOW_DET2 <= '0';
            LOW_DET3 <= '0';
         else
            LOW_DET1 <= LOW_DET ; LOW_DET2 <= LOW_DET1;
            LOW_DET3 <= LOW_DET2;
         end if;
      end if;
 end process;


-- **********************
--   signal sortie
-- **********************
 trois : process(CLK,RESET)

 begin
   if (RESET =  '1' ) then
     END_DET  <= '0';
     END_DETI1 <= '0';
   elsif( CLK' EVENT and CLK = '1') then
     END_DETI1 <= END_DETI;
     END_DET <= END_DETI;
   end if;
 end process;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

--------------------------------------------------------------------
--
-- Fichier : max_de.vhd
--
-- detection du maximum apres Nv echantillons depassant le seuil
-- superieur TH2
-- la valeur du maximum est utilisee pour controler le gain de l'ampli
--
--------------------------------------------------------------------

entity max_det is
port( CLK,RESET     : in  std_logic;
      DATA          : in  std_logic_vector(7 downto 0);
      MAX_INIT      : in  std_logic;
      ECART         : in  std_logic_vector(7 downto 0);
      SEUIL         : out std_logic_vector(31 downto 0);
      MAX           : out std_logic_vector(7 downto 0));
end max_det;


-- DATA     : les echantillons a l'entree
-- MAX_INIT : initialisation du mximum apres detection d'un echantillon
--            superieur au seuil superieur TH2
-- MH       : nombre d'echantillons au dessus de TH2
-- MAX      : les maximums detectes


architecture comport of max_det is

  signal MAX_I      : std_logic_vector(7 downto 0);
  signal SEUILX     : std_logic_vector(7 downto 0);
  signal SEUILX1    : std_logic_vector(7 downto 0);

begin

  SEUIL  <= SEUILX & SEUILX & SEUILX &SEUILX;
  MAX    <= MAX_I;

--*****************************************************************
-- determination du maximum et du seuil
--*****************************************************************
maximum : process(CLK, RESET)
   begin
      if ( RESET = '1') then
         MAX_I  <= (others => '0');
         SEUILX1  <= (others => '0');
      elsif( CLK' event and CLK = '1') then
         if (MAX_INIT = '0') then
```

```
            MAX_I <= (others => '0');
            SEUILX1 <= (others => '0');
        elsif (DATA > MAX_I) then
            MAX_I   <= DATA;
            SEUILX  <= DATA - ECART;
        else
            MAX_I   <= MAX_I;
            SEUILX  <= SEUILX;
        end if;
     end if;
 end process;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------
--
-- Fichier : ptoa.vhd
--
-- Calcul du PTOA : pulse time of arrival
-- on utilise la methode lineaire entre 2 impulsion entourant
-- le SEUIL
-- le seuil est obtenu a mi hauteur par rapport au maximum detecte
--
--------------------------------------------------------------------

entity ptoa is
   port( CLK,RESET : in std_logic;
         ECART     : in std_logic_vector(7 downto 0);
         DATA      : in std_logic_vector(7 downto 0);
         SEUIL     : in std_logic_vector(31 downto 0);
         P_START   : in std_logic;
         P_END     : in std_logic;
         LARGE     : out std_logic_vector(7 downto 0);
         RESULT    : out std_logic_vector(23 downto 0));
end ptoa;

-- ************************************************************
-- DATA     : echantillons de l'impulsion
-- MAXI     : valeur maximale detectee apres Nv echantillons
-- P_START  : debut de l'impulsion, impulsion valide : VAL
-- P_END    : fin de l'impulsion
-- COMPT    : compteur general des impulsions
-- RESULT   : PTOA sur 24 bits
-- ************************************************************

architecture comport of ptoa is

  component nu_y0
  port    ( CLK,RESET  : in std_logic;
            DIN        : in std_logic_vector(7 downto 0);
            SEUIL_IN   : in std_logic_vector(31 downto 0);
            P_END      : in std_logic;
            SEUIL_OUT  : out std_logic_vector(7 downto 0);
            NUM_Y0     : out std_logic_vector(4 downto 0));
  end component;

  component ram_y0y1
  port ( CLK,RESET : in std_logic;
       VAL         : in std_logic;
       P_END       : in std_logic;
       DATA        : in std_logic_vector(7 downto 0);
       NUMERO      : in std_logic_vector(4 downto 0);
```

```
      Y0         : out std_logic_vector(7 downto 0);
      PENTE      : out std_logic_vector(7 downto 0)
      );
  end component;


  component lin_invers
  port ( CLK,RESET  : in std_logic;
         Y0         : in std_logic_vector(7 downto 0);
         PENTE      : in std_logic_vector(7 downto 0);
         SEUIL      : in std_logic_vector(7 downto 0);
         RESULT     : out std_logic_vector(3 downto 0));
  end component;


  component largeur
  port (CLK, RESET : in std_logic;
        START      : in std_logic;
        STOP       : in std_logic;
        LARG       :   out std_logic_vector(7 downto 0));
  end component;


  signal VAL              : std_logic;
  signal NUMERO_Y0        : std_logic_vector( 4 downto 0);
  signal NUMERO1          : std_logic_vector( 4 downto 0);
  signal NUMERO1_Q        : std_logic_vector( 4 downto 0);
  signal NUMERO1_R        : std_logic_vector( 4 downto 0);

  signal Y0,PENTE         : std_logic_vector( 7 downto 0);
  signal RESULT1          : std_logic_vector( 3 downto 0);
  signal RANG             : std_logic_vector(19 downto 0);
  signal SEUIL_0,SEUIL_01 : std_logic_vector(7 downto 0);
  signal COUT,COMPT       : std_logic_vector(19 downto 0);
  signal DATA_N,DATA_R    : std_logic_vector(7 downto 0);
  signal RANG1,RANG2      : std_logic_vector(19 downto 0);
  signal RANG3,RANG4      : std_logic_vector(19 downto 0);

begin

 VAL <= P_START;

 numy0   : nu_y0    port map
(CLK,RESET,DATA_N,SEUIL,P_END,SEUIL_0,NUMERO_Y0);

 ramy0y1 : ram_y0y1 port map
(CLK,RESET,VAL,P_END,DATA_R,NUMERO_Y0,Y0,PENTE);

 -- selon la strategie : lin_invers , lin_subdiv , quad_subdiv

 lin_inv   : lin_invers  port map(CLK,RESET,Y0,PENTE,
                                  SEUIL_01,RESULT1);
```

```
 larg1   : largeur port map(CLK,RESET,P_START,P_END,LARGE);


-- ***************************************************************
-- compteur general des echantillons sur 20 bits
-- le contenu du compteur est latche quand VAl passe a 1
-- ***************************************************************

  compt_reg_20 : process(RESET, CLK)
     variable TEMP : std_logic_vector(19 downto 0);
  begin
     if (RESET = '1') then
       COUT <= ( others => '0');
     elsif (CLK'event and CLK = '1') then
       TEMP := unsigned(COUT) + 1;
       COUT <= TEMP;
       if ( VAL = '1') then
         COMPT <= COUT;
       end if;
     end if ;
  end process;


  NUMERO1_Q  <= NUMERO1 + 2;
  RANG       <= COMPT - NUMERO1_R;
  RESULT     <= RANG4 & RESULT1;

--*********************************************************************
--  Registre de synchronisation des impulsions a l'entree
--  retarder la valeur du rang de Y0 de 4 coups d'horloge
--*********************************************************************
  process(RESET, CLK)
  begin
    if (RESET = '1') then
       DATA_R <= ( others => '0');
       DATA_N <= ( others => '0');
       RANG1  <= ( others => '0');
       RANG2  <= ( others => '0');
       RANG3  <= ( others => '0');
       RANG4  <= ( others => '0');
       SEUIL_01 <= ( others => '0');
       NUMERO1 <= (others => '0');
       NUMERO1_R <= (others => '0');
     elsif (CLK'event and CLK = '1') then
       DATA_R <= DATA;
       DATA_N <= DATA;
       RANG1  <= RANG;
       RANG2  <= RANG1;
       RANG3  <= RANG2;
       RANG4  <= RANG3;
       NUMERO1 <= NUMERO_Y0;
       NUMERO1_R <= NUMERO1_Q;
```

```
        SEUIL_O1 <= SEUIL_O;
      end if ;
  end process;
end ;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------------
---
--
-- Fichier : nu_y0.vhd
--
-- determine le nombre d'echantillons superieurs au seuil
--
--
--------------------------------------------------------------------------

entity nu_y0 is
port    ( CLK , RESET : in std_logic;
          DIN         : in std_logic_vector(7 downto 0);
          SEUIL_IN    : in std_logic_vector(31 downto 0);
          P_END       : in std_logic;
          SEUIL_OUT   : out std_logic_vector(7 downto 0);
          NUM_Y0      : out std_logic_vector(4 downto 0)
        );
end nu_y0;


architecture comport of nu_y0 is


component rdecal
generic ( Nbit , longueur : integer := 32);
port ( CLK, RESET : std_logic;
       DIN        : in std_logic_vector(Nbit-1 downto 0);
       INITIAL    : in std_logic;
       QOUT       : out std_logic_vector(longueur*Nbit-1  downto 0));
end component;


component comp_n_1
generic ( Nbit : integer := 8 ; longueur : integer := 32);
port ( DIN1  : in  std_logic_vector(longueur*Nbit-1  downto 0);
       TRES1 : in  std_logic_vector( Nbit-1 downto 0);
       TRES2 : in  std_logic_vector( Nbit-1 downto 0);
       TRES3 : in  std_logic_vector( Nbit-1 downto 0);
       TRES4 : in  std_logic_vector( Nbit-1 downto 0);
       QOUT  : out std_logic_vector( longueur-1  downto 0));
end component;

component prio
port    ( CLK,RESET           : in std_logic;
          VEC_X1              : in std_logic_vector(31 downto 0);
          NUMERO_Y0           : out std_logic_vector(4 downto 0));
end component;
```

```
signal X1         : std_logic_vector(255 downto 0);
signal X2         : std_logic_vector( 31 downto 0);
signal X3         : std_logic_vector( 31 downto 0);

signal SEUIL                    : std_logic_vector(7 downto 0);
signal SEUIL1,SEUIL2,SEUIL3     : std_logic_vector(7 downto 0);
signal SEUI1,SEUI2,SEUI3,SEUI4  : std_logic_vector(7 downto 0);

-- X1       : correspond aux impulsions memorisees
-- X2       : resultat de la comparaison des impulsions avec le seuil
-- X3       : detection du premier passage de 0 a 1 dans X2


begin

  SEUIL <=  SEUIL_IN(7 downto 0);

-- Seuil en plus pour diminuer le fanout
  SEUI1  <=  SEUIL_IN(31 downto 24);
  SEUI2  <=  SEUIL_IN(23 downto 16);
  SEUI3  <=  SEUIL_IN(15 downto 8 );
  SEUI4  <=  SEUIL_IN( 7 downto 0 );


 rg_dec : rdecal generic map (8,32)
                port map (CLK,RESET,DIN,P_END,X1);

 cp_n   : comp_n_1 generic map (8,32)
                port map (X1,SEUI1,SEUI2,SEUI3,SEUI4,X2);

 pro2   : prio port map (CLK,RESET,X3,NUM_Y0);


-- registre de resynchronisation
  pp1 : process(CLK, RESET)
    begin
      if ( RESET = '1') then
         X3        <=  (others => '0');
         SEUIL1    <=  ( others => '0');
         SEUIL2    <=  ( others => '0');
         SEUIL3    <=  ( others => '0');
         SEUIL_OUT <=  ( others => '0');
      elsif( CLK' Event and CLK = '1') then
         X3         <=  X2;
         SEUIL1     <=  SEUIL ;
         SEUIL2     <=  SEUIL1;
         SEUIL3     <=  SEUIL2;
         SEUIL_OUT  <=  SEUIL3;
      end if;
    end process;

 end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

--------------------------------------------------------------------------
--
-- Fichier : rdecal.vhd
--
-- registre a decalage de longueur l de mots de N bits
-- avec remise a zero synchrone
--
--------------------------------------------------------------------------

entity rdecal is
generic ( Nbit , longueur : integer := 32);
port ( CLK,RESET : in std_logic;
       DIN       : in std_logic_vector(Nbit-1 downto 0);
       INITIAL   : in std_logic;
       QOUT      : out std_logic_vector(longueur*Nbit-1  downto 0));
end rdecal;


 architecture comport of rdecal is

 signal X1 : std_logic_vector(longueur*Nbit-1 downto 0);

 begin

 process(CLK,RESET,DIN,INITIAL)

 begin
    if (RESET = '1') then
      X1 <= ( others => '0' );
    elsif ( CLK' event and CLK = '1') then
      if (INITIAL = '1') then
          X1 <= ( others => '0' );
      else
         X1( Nbit - 1 downto 0) <= DIN;
         X1( Nbit * longueur -1 downto Nbit) <=
                                  X1( (longueur - 1) * Nbit -1 downto 0);
      end if;
    end if;
 end process;

 QOUT <= X1;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

--------------------------------------------------------------------------
---
--
-- Fichier : comp_n_l.vhd
--
-- comparateur de L echantillons de N bits
-- avec un seuil de N bits
-- le resultat est mot de N bits de la forme 00000111111
--
--------------------------------------------------------------------------
---

entity comp_n_l is
generic ( Nbit :integer := 8 ; longueur : integer := 32);
port ( DIN1  : in  std_logic_vector(longueur*Nbit-1  downto 0);
       TRES1 : in  std_logic_vector( Nbit-1 downto 0);
       TRES2 : in  std_logic_vector( Nbit-1 downto 0);
       TRES3 : in  std_logic_vector( Nbit-1 downto 0);
       TRES4 : in  std_logic_vector( Nbit-1 downto 0);
       QOUT  : out std_logic_vector( longueur-1  downto 0));
end comp_n_l;


architecture comport of comp_n_l is

begin
 process(DIN1,TRES1,TRES2,TRES3,TRES4)
 begin
    for i in 0 to 7  loop
           if (DIN1((i+1)*Nbit-1 downto (Nbit*i)) >TRES1 ) then
                QOUT(i) <= '1';
           else
                QOUT(i) <= '0';
           end if;
    end loop;


    for i in 8 to 15  loop
           if (DIN1((i+1)*Nbit-1 downto (Nbit*i)) >TRES2 ) then
                QOUT(i) <= '1';
           else
                QOUT(i) <= '0';
           end if;
    end loop;

    for i in 16 to 23  loop
           if (DIN1((i+1)*Nbit-1 downto (Nbit*i)) >TRES3 ) then
                QOUT(i) <= '1';
           else
                QOUT(i) <= '0';
```

```
            end if;
        end loop;

        for i in 23 to 31  loop
            if (DIN1((i+1)*Nbit-1 downto (Nbit*i)) >TRES4 ) then
                QOUT(i) <= '1';
            else
                QOUT(i) <= '0';
            end if;
        end loop;
    end process;
end comport;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
-----------------------------------------------------------------------
-- Fichier : priorite.vhd
--
-- determine le rang du premier bit a 1
-- a partir du MSB
-----------------------------------------------------------------------

entity prio is
port    ( CLK,RESET           : in std_logic;
          VEC_X1              : in std_logic_vector(31 downto 0);
          NUMERO_Y0           : out std_logic_vector(4 downto 0));
end prio;

architecture comport of prio is

  component prio8
  port    ( CLK,RESET  : in std_logic;
            DIN        : in std_logic_vector(7 downto 0);
            NUMERO     : out std_logic_vector(2 downto 0);
            V          : out std_logic);
  end component;

  signal SELE                : std_logic_vector(3 downto 0);
  signal NUM_Y0              : std_logic_vector(4 downto 0);
  signal MEM1,MEM2,MEM3,MEM0 : std_logic_vector(2 downto 0);

begin

  -- decodeurs de priorite sur 8 bits
  pp1 : prio8 port map (CLK,RESET,VEC_X1(31 downto 24),MEM3,SELE(3));
  PP2 : prio8 port map (CLK,RESET,VEC_X1(23 downto 16),MEM2,SELE(2));
  pp3 : prio8 port map (CLK,RESET,VEC_X1(15 downto  8),MEM1,SELE(1));
  PP4 : prio8 port map (CLK,RESET,VEC_X1( 7 downto  0),MEM0,SELE(0));

  pp5 : process(MEM0,MEM1,MEM2,MEM3,SELE)
  begin
        if    SELE(3) = '1'  then  NUM_Y0 <= ("11" & MEM3);
        elsif SELE(2) = '1'  then  NUM_Y0 <= ("10" & MEM2);
        elsif SELE(1) = '1'  then  NUM_Y0 <= ("01" & MEM1);
        else                       NUM_Y0 <= ("00" & MEM0);
      end if;
  end process;

  -- registre de synchronisation
  registre0 : process(CLK, RESET)
       begin
         if (RESET = '1') then
            NUMERO_Y0   <= (others => '0');
        elsif ( CLK'  event and CLK = '1') then
            NUMERO_Y0   <= NUM_Y0 ;
```

```
      end if;
 end process;
end  comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
-----------------------------------------------------------------
--
-- Fichier : prio8.vhd
--
-- decodeur de priorite
-- Donne le rang du bit a 1 ayant le poids maximum
-- Le signal V est a zero pour indiquer que les 8 bits sont a 0
--
-----------------------------------------------------------------

entity prio8 is
port    ( CLK,RESET : std_logic;
          DIN       : in  std_logic_vector(7 downto 0);
          NUMERO    : out std_logic_vector(2 downto 0);
          V         : out std_logic);
end prio8;

architecture comport of prio8 is

  signal NUMEROQ : std_logic_vector(2 downto 0);
  signal VQ      : std_logic;

begin

  pro2 : process(DIN)
  begin
    if    DIN(7)  = '1' then NUMEROQ <= "111";
    elsif DIN(6)  = '1' then NUMEROQ <= "110";
    elsif DIN(5)  = '1' then NUMEROQ <= "101";
    elsif DIN(4)  = '1' then NUMEROQ <= "100";
    elsif DIN(3)  = '1' then NUMEROQ <= "011";
    elsif DIN(2)  = '1' then NUMEROQ <= "010";
    elsif DIN(1)  = '1' then NUMEROQ <= "001";
    else  NUMEROQ <= ( others => '0');
    end if;
  end process;

  VQ <= (DIN(7) or DIN(6) or DIN(5) or DIN(4)) or
        (DIN(3) or DIN(2) or DIN(1) or DIN(0) );

  -- registre de resynchronisation

  ppl : process(CLK, RESET)
    begin
      if ( RESET = '1') then
         NUMERO <=  ( others => '0');
         V  <=   '0';
     elsif( CLK' Event and CLK = '1') then
         NUMERO <=  NUMEROQ;
         V  <=   VQ;
```

```
      end if;
   end process;
end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------------
--
-- Fichier : ram_y0y1.vhd
--
-- Localise les deux points M0 et M1
--
--------------------------------------------------------------------------


entity ram_y0y1 is
  port ( CLK,RESET : in std_logic;
       VAL         : in std_logic;
       P_END      : in std_logic;
       DATA       : in std_logic_vector(7 downto 0);
       NUMERO     : in std_logic_vector(4 downto 0);
       Y0         : out std_logic_vector(7 downto 0);
       PENTE      : out std_logic_vector(7 downto 0)
       );
end ram_y0y1;

architecture comport of ram_y0y1 is

component ram_y0
  port ( CLK,RESET : in std_logic;
       VAL         : in std_logic;
       P_END      : in std_logic;
       DATA       : in  std_logic_vector(7 downto 0);
       NUMERO     : in  std_logic_vector(4 downto 0);
       Y0         : out std_logic_vector(7 downto 0)
       );
end component;

component ram_p
  port ( CLK,RESET : in std_logic;
       VAL         : in std_logic;
       P_END      : in std_logic;
       DATA       : in std_logic_vector(7 downto 0);
       NUMERO     : in std_logic_vector(4 downto 0);
       PENTE      : out std_logic_vector(7 downto 0)
       );
end component;

begin

Y_RAM : ram_y0  port map(CLK,RESET,VAL,P_END,DATA,NUMERO,Y0);
P_RAM : ram_p   port map(CLK,RESET,VAL,P_END,DATA,NUMERO,PENTE);

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------------
--
-- Fichier : ram_Y0.vhd
--
-- determine le rang de l'echantillon juste en dessous du seuil
--
--
--------------------------------------------------------------------------

entity ram_y0 is
  port ( CLK,RESET : in  std_logic;
       VAL       : in  std_logic;
       P_END     : in  std_logic;
       DATA      : in  std_logic_vector(7 downto 0);
       NUMERO    : in  std_logic_vector(4 downto 0);
       Y0        : out std_logic_vector(7 downto 0)
       );
end ram_y0;

architecture comport of ram_y0 is

  component ra_y0
  generic (Nbit : integer := 8);
  port (  CLK             : in  std_logic;
        DATA_IN         : in  std_logic_vector(Nbit-1 downto 0);
        ADRESS_1        : in  std_logic_vector(4 downto 0);
        ADRESS_2        : in  std_logic_vector(4 downto 0);
      EN              : in  std_logic;

        DATA_OUT        : out std_logic_vector(Nbit-1 downto 0));
  end component;

  component count_z
  generic ( N : integer := 8);
  port ( CLK,RESET : in std_logic;
         RZ        : in std_logic;
         COUT      : out std_logic_vector(N-1 downto 0));
  end component;


  component registrn_e
  generic (N : integer := 8);
  port ( CLK, RESET : in std_logic;
         DIN        : in std_logic_vector(N-1 downto 0);
         ENABLE     : in std_logic;
         QOUT       : out std_logic_vector(N-1 downto 0));
  end component;
```

```
  signal ADR, ADR1       : std_logic_vector(4 downto 0);
  signal Y0I             : std_logic_vector(7 downto 0);

  signal CPT, CPT1       : std_logic_vector(4 downto 0);

  signal CPTR            : std_logic_vector(4 downto 0);
  signal CPT_NUM, CPT_NUM1 : std_logic_vector(4 downto 0);
  signal DATA1, DATA2    : std_logic_vector(7 downto 0);
  signal VAL1,VAL2,VAL3 : std_logic;
  signal VAL4,VAL5       : std_logic;
  signal NOT_VAL         : std_logic;

begin

  NOT_VAL <= not VAL4;

  -- Compteur
  CNT_0 : count_z  generic map (5) port map(CLK,RESET,P_END,CPT);


  -- Registre
  reg_01 : registrn_e generic map(5) port map (CLK,RESET,CPT,VAL,CPTR);

  -- Soustraction
  CPT_NUM  <=  CPTR - (NUMERO + 1);


  -- Multiplexeur
  ADR1 <= CPT1       when VAL4 = '0' else "ZZZZZ";
  ADR1 <= CPT_NUM1   when VAL4 = '1' else (others => 'Z');


  RAMY0 : ra_y0 generic map(8)
         port map(CLK,DATA2,ADR1,ADR1,NOT_VAL,Y0I);


  -- registre de synchronisation
  reg0 : process(CLK,RESET)
       begin
         if (RESET = '1') then
            Y0        <= (others => '0');
            DATA1     <= (others => '0');
            DATA2     <= (others => '0');
            VAL1      <= '0'; VAL2 <= '0'; VAL3 <= '0';
            VAL4      <= '0'; VAL5 <= '0';
            CPT_NUM1  <= (others => '0');
            CPT1      <= (others => '0');
        elsif ( CLk'  event and CLK = '1') then
            VAL1      <= VAL   ; VAL2 <= VAL1 ; VAL3 <= VAL2;
            VAL4      <= VAL3  ; VAL5 <= VAL4 ;
            Y0        <= Y0I;
            CPT_NUM1  <= CPT_NUM;
            CPT1      <= CPT;
```

```
            DATA1    <= DATA;
            DATA2    <= DATA1;
        end if;
    end process;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------------
--
-- Fichier : ram_P.vhd
--
-- determine la pente entre M0 et M1
--
--------------------------------------------------------------------------

entity ram_p is
  port ( CLK,RESET : in std_logic;
       VAL       : in std_logic;
       P_END     : in std_logic;
       DATA      : in std_logic_vector(7 downto 0);
       NUMERO    : in std_logic_vector(4 downto 0);
       PENTE     : out std_logic_vector(7 downto 0)
       );
end ram_p;

architecture comport of ram_p is

  component ra_y0
  generic (Nbit : integer := 8);
  port (  CLK             : in std_logic;
          DATA_IN         : in  std_logic_vector(Nbit-1 downto 0);
          ADRESS_1        : in  std_logic_vector(4 downto 0);
          ADRESS_2        : in  std_logic_vector(4 downto 0);
        EN              : in  std_logic;

          DATA_OUT        : out std_logic_vector(Nbit-1 downto 0));
  end component;

  component count_z
  generic ( N : integer := 8);
  port ( CLK,RESET : in std_logic;
         RZ        : in std_logic;
         COUT      : out std_logic_vector(N-1 downto 0));
  end component;


  component registrn_e
  generic (N : integer := 8);
  port ( CLK, RESET : in std_logic;
         DIN        : in std_logic_vector(N-1 downto 0);
         ENABLE     : in std_logic;
         QOUT       : out std_logic_vector(N-1 downto 0));
  end component;
```

```
  signal ADR, ADR1          : std_logic_vector(4 downto 0);
  signal PENTE_I            : std_logic_vector(7 downto 0);
  signal CPT, CPT1          : std_logic_vector(4 downto 0);
  signal CPTR               : std_logic_vector(4 downto 0);
  signal CPT_NUM, CPT_NUM1  : std_logic_vector(4 downto 0);
  signal DATA1              : std_logic_vector(7 downto 0);
  signal PENTQ, PENTQ1      : std_logic_vector(7 downto 0);
  signal VAL1,VAL2,VAL3     : std_logic;
  signal VAL4,VAL5, NOT_VAL : std_logic;

begin

   NOT_VAL <= not VAL4;

   -- Compteur
   CNT_0  : count_z  generic map (5) port map(CLK,RESET,P_END,CPT);

   -- Registre
   reg_01 : registrn_e generic map(5) port map
(CLK,RESET,CPT,VAL,CPTR);

   -- les soustractions
   CPT_NUM <= CPTR -  ( NUMERO + 1);
   PENTQ   <= DATA - DATA1;

   -- Mux
   ADR1 <= CPT1     when VAL4 = '0' else "ZZZZZ";
   ADR1 <= CPT_NUM1 when VAL4 = '1' else (others => 'Z');


   RAMY1 : ra_y0 generic map(8)
           port map(CLK,PENTQ1,ADR1,ADR1,NOT_VAL,PENTE_I);


   -- registre
   reg0 : process(CLK,RESET)
          begin
           if (RESET = '1') then
             VAL1 <= '0'; VAL2 <= '0'; VAL3 <= '0';
             VAL4 <= '0'; VAL5 <= '0';
             PENTE    <= (others => '0');
             DATA1    <= (others => '0');
             CPT_NUM1 <= (others => '0');
             CPT1     <= (others => '0');
             PENTQ1   <= (others => '0');
           elsif ( CLk'  event and CLK = '1') then
             VAL1     <= VAL   ; VAL2 <= VAL1 ; VAL3 <= VAL2;
             VAL4     <= VAL3 ; VAL5 <= VAL4 ;
             PENTE    <= PENTE_I;
             DATA1    <= DATA;
             CPT_NUM1 <= CPT_NUM;
             CPT1     <= CPT;
             PENTQ1   <= PENTQ;
```

```
            end if;
          end process;

end comport;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity ra_y0 is
generic (Nbit : integer := 8);
port (  CLK             : in std_logic;
        DATA_IN         : in  std_logic_vector(Nbit-1 downto 0);
        ADRESS_1        : in  std_logic_vector(4 downto 0);
        ADRESS_2        : in  std_logic_vector(4 downto 0);
      EN              : in  std_logic;

        DATA_OUT        : out std_logic_vector(Nbit-1 downto 0));
end ra_y0;


architecture comport of ra_y0 is

      subtype mots is std_logic_vector(Nbit-1 downto 0);
      type memoire is array (0 to 31) of mots;

      signal RAM_LUT    : memoire;

begin


        DATA_OUT <= RAM_LUT(conv_integer(ADRESS_1(4 downto 0)));


      process(CLK)
      begin
      if (CLK = '1' and CLK' event) then
            -- mode d'ecriture
              if (EN = '1')  then
               RAM_LUT(CONV_INTEGER(ADRESS_1)) <= DATA_IN;
              end if;
      end if;
      end process;


end comport;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.std_logic_arith.all;

--------------------------------------------------------------------------
--
-- Fichier : count_z.vhd
--
-- compteur sur N bits
-- avec remise a zero synchrone
--
--------------------------------------------------------------------------

entity count_z is
generic ( N : integer := 8);
port (CLK, RESET : in std_logic;
      RZ  : std_logic;
      COUT  :   out std_logic_vector(N-1 downto 0));
end count_z;

architecture BEHAVIORAL of count_z is

  signal QOUT : std_logic_vector( N-1 downto 0);

begin

  process(RESET, CLK)
    variable TEMP : std_logic_vector(N-1 downto 0);
  begin
    if (RESET = '1') then
       QOUT <= ( others => '0');
    elsif (CLK'event and CLK = '1') then
       if (RZ = '1') then
         QOUT <= ( others => '0');
       else
         TEMP := unsigned(QOUT) + 1;
         QOUT <= TEMP;
        end if;
    end if;
   end process;

   COUT <= QOUT;

end BEHAVIORAL;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

--------------------------------------------------------------------------
--
-- Fichier : registrn_e.vhd
--
-- registre de N bascules D
-- avec une entree ENABLE
--
--------------------------------------------------------------------------

entity registrn_e is
generic (N : integer := 8);
port ( CLK, RESET : in std_logic;
       DIN        : in std_logic_vector(N-1 downto 0);
       ENABLE     : in std_logic;
       QOUT       : out std_logic_vector(N-1 downto 0));
end registrn_e;


 architecture comport of registrn_e is
 signal QOUTI : std_logic_vector(N-1 downto 0);
 begin
   pp1 : process(CLK)
   begin
      if( CLK' Event and CLK = '1') then
         if ( RESET = '1') then
            QOUTI <= ( others => '0');
         elsif (ENABLE = '1' ) then
            QOUTI <= DIN;
         else
            QOUTI <= QOUTI;
         end if;
      end if;
   end process;

 QOUT <= QOUTI;
 end comport;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

------------------------------------------------------------------
--
-- Fichier : lin_invers.vhd
--
-- Calcul RESult = (SEUIL - Y0) * (1/PENTE)
--
------------------------------------------------------------------


entity lin_invers is
  port ( CLK,RESET  : in  std_logic;
         Y0         : in  std_logic_vector( 7 downto 0);
         PENTE      : in  std_logic_vector( 7 downto 0);
         SEUIL      : in  std_logic_vector( 7 downto 0);
         RESULT     : out std_logic_vector( 3 downto 0));
end invers;


architecture Behave of lin_invers is


component multn
generic ( NA : integer := 8);
port    ( CLK,RESET  : in std_logic;
          A    : in std_logic_vector(NA-1 downto 0);
          B    : in std_logic_vector(7 downto 0);
          PROD  : out std_logic_vector(NA+7 downto 0));
end component;


component lut_inv
      port (
      addr: IN std_logic_VECTOR(7 downto 0);
      clk : IN std_logic;
      di  : IN std_logic_VECTOR(11 downto 0);
      we  : IN std_logic;
      en  : IN std_logic;
      rst : IN std_logic;
      do  : OUT std_logic_VECTOR(11 downto 0));
end component;


signal PROD       : std_logic_vector( 19 downto 0);
signal Y_B        : std_logic_vector(  7 downto 0);
signal Y_BQ       : std_logic_vector(  7 downto 0);
signal INV_PENTE  : std_logic_vector( 11 downto 0);
signal INV_PENTEQ : std_logic_vector( 11 downto 0);
signal DI         : std_logic_vector( 11 downto 0);
```

```
signal WE, EN      : std_logic;

begin

  RESULT <= PROD(11 downto 8);
  Y_BQ <=  SEUIL -  Y0;



-- Calcul de l'invers de la pente
   DI <= ( others => '0');
   WE <= '0'; EN <= '1';
   lutinv : lut_inv port map(PENTE, CLK,DI,WE,EN,RESET,INV_PENTEQ);

   m1      : multn generic map(12)
                port map (CLK,RESET,INV_PENTE,Y_B,PROD);

   pp1 : process(CLK, RESET)
      begin
        if ( RESET = '1') then
           INV_PENTE  <= (others => '0');
           Y_B        <= (others => '0');
        elsif( CLK' Event and CLK = '1') then
           INV_PENTE  <=  INV_PENTEQ;
           Y_B        <=  Y_BQ;
        end if;
  end process;
end ;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

------------------------------------------------------------
-- Fichier : multn.vhd
--                                                          --
--   Multiplication d'un nombre de 8 bits                   --
--   par un nombre de NA bits                               --
--                                                          --
------------------------------------------------------------

entity multn is
generic ( NA  : integer := 12);
port (  CLK,RESET  : in std_logic;
        A          : in std_logic_vector(NA-1 downto 0);
        B          : in std_logic_vector(7 downto 0);
        PROD       : out std_logic_vector(NA+7 downto 0));
end multn;


architecture Behave of multn is

signal X3       : std_logic_vector((8*NA)-1 downto 0);
signal S01, P01 : std_logic_vector(NA + 1 downto 0);  -- max = 2FFD
signal S02, P02 : std_logic_vector(NA + 3 downto 0);  -- max = BFF4
signal S03, P03 : std_logic_vector(NA + 5 downto 0);  -- max = 2FFD0
signal S04, P04 : std_logic_vector(NA + 7 downto 0);  -- max = BFF40
signal S11, P11 : std_logic_vector(NA + 3 downto 0);  -- max =  EFF1
signal S12, P12 : std_logic_vector(NA + 7 downto 0);  -- max = EFF10
signal S21      : std_logic_vector(NA + 7 downto 0);  -- max = FEF01

begin

-- Calcul des produits elementaires :
  mux1 : process(A,B)
         begin
              for i in 7 downto 0 loop
                if B(i) = '1' then
                  X3(NA*(i+1)-1 downto NA*i) <= A;
                else
                  X3(NA*(i+1)-1 downto NA*i) <= ( "000000000000");
                end if;
              end loop;
        end process;


-- quatre additionneurs

  S01 <= ('0'& X3(2*NA-1 downto NA  ) & '0') + X3( NA-1 downto 0 );
  S02 <= ('0'& X3(4*NA-1 downto 3*NA)&"000")+(X3(3*NA-1 downto 2*NA)&
"00");
```

```
  S03 <= ( '0' & X3(6*NA-1 downto 5*NA) & "00000")+(X3(5*NA-1 downto
4*NA) & "0000"  );
  S04 <= ( '0' & X3(8*NA-1 downto 7*NA) & "0000000") + ( X3( 7*NA-1
downto 6*NA) & "000000");

-- process inferant des registre de synchronisation

pp1 : process(CLK,RESET)
   begin
      if ( RESET = '1') then
         P01 <= ( others => '0')  ;  P02 <= ( others => '0');
         P03 <= ( others => '0')  ;  P04 <= ( others => '0');
      elsif( CLK' Event and CLK = '1') then
         P01 <= S01 ;  P02 <= S02 ;
         P03 <= S03 ;  P04 <= S04 ;
      end if;
   end process;

-- Deux additionneurs et deux rregistres

   S11 <=  P01 + P02 ;
   S12 <=  P03 + P04 ;

 -- Un additionneur
   S21 <=  P11 + P12 ;

-- process inferant de registre de synchronisation
pp3 : process(CLK, RESET)
   begin
      if ( RESET = '1') then
         PROD <= ( others => '0');
         P11 <= ( others => '0');
         P12  <= ( others => '0');
      elsif( CLK' Event and CLK = '1') then
         PROD <= S21(NA+7 downto 0);
         P11 <=  S11;
         P12 <=  S12;
      end if;
   end process;

end ;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;

--------------------------------------------------------------------------
--
-- Fichier : lut_inv.vhd
--
-- determine la duree de l'impulsion
--
--------------------------------------------------------------------------

ENTITY lut_inv is
      port (
      addr: IN std_logic_VECTOR(7 downto 0);
      clk : IN std_logic;
      di  : IN std_logic_VECTOR(11 downto 0);
      we  : IN std_logic;
      en  : IN std_logic;
      rst : IN std_logic;
      do  : OUT std_logic_VECTOR(11 downto 0));
end lut_inv;



ARCHITECTURE Beh of lut_inv is
begin

PROCESS(ADDR)

   SUBTYPE word is std_logic_vector(11 downto 0);
   TYPE ROM_TBL is array (0 to 255) of word;

   constant ROM_VAL : ROM_TBL := ROM_TBL'("111111111111",
 "111111111111" , "100000000000" , "010101010101" , "010000000000" ,
 "001100110011" , "001010101010" , "001001001001" , "001000000000" ,
 "000111000111" , "000110011001" , "000101110100" , "000101010101" ,
 "000100111011" , "000100100100" , "000100010001" , "000100000000" ,
 "000011110000" , "000011100011" , "000011010111" , "000011001100" ,
 "000011000011" , "000010111010" , "000010110010" , "000010101010" ,
 "000010100011" , "000010011101" , "000010010111" , "000010010010" ,
 "000010001101" , "000010001000" , "000010000100" , "000010000000" ,
 "000001111100" , "000001111000" , "000001110101" , "000001110001" ,
 "000001101110" , "000001101011" , "000001101001" , "000001100110" ,
 "000001100011" , "000001100001" , "000001011111" , "000001011101" ,
 "000001011011" , "000001011001" , "000001010111" , "000001010101" ,
 "000001010011" , "000001010001" , "000001010000" , "000001001110" ,
 "000001001101" , "000001001011" , "000001001010" , "000001001001" ,
 "000001000111" , "000001000110" , "000001000101" , "000001000100" ,
 "000001000011" , "000001000010" , "000001000001" , "000001000000" ,
 "000000111111" , "000000111110" , "000000111101" , "000000111100" ,
 "000000111011" , "000000111010" , "000000111001" , "000000111000" ,
 "000000111000" , "000000110111" , "000000110110" , "000000110101" ,
```

```
"000000110101" , "000000110100" , "000000110011" , "000000110011" ,
"000000110010" , "000000110001" , "000000110001" , "000000110000" ,
"000000110000" , "000000101111" , "000000101111" , "000000101110" ,
"000000101110" , "000000101101" , "000000101101" , "000000101100" ,
"000000101100" , "000000101011" , "000000101011" , "000000101010" ,
"000000101010" , "000000101001" , "000000101001" , "000000101000" ,
"000000101000" , "000000101000" , "000000100111" , "000000100111" ,
"000000100111" , "000000100110" , "000000100110" , "000000100101" ,
"000000100101" , "000000100101" , "000000100100" , "000000100100" ,
"000000100100" , "000000100011" , "000000100011" , "000000100011" ,
"000000100011" , "000000100010" , "000000100010" , "000000100010" ,
"000000100001" , "000000100001" , "000000100001" , "000000100001" ,
"000000100000" , "000000100000" , "000000100000" , "000000100000" ,
"000000011111" , "000000011111" , "000000011111" , "000000011111" ,
"000000011110" , "000000011110" , "000000011110" , "000000011110" ,
"000000011101" , "000000011101" , "000000011101" , "000000011101" ,
"000000011101" , "000000011100" , "000000011100" , "000000011100" ,
"000000011100" , "000000011100" , "000000011011" , "000000011011" ,
"000000011011" , "000000011011" , "000000011011" , "000000011010" ,
"000000011010" , "000000011010" , "000000011010" , "000000011010" ,
"000000011010" , "000000011001" , "000000011001" , "000000011001" ,
"000000011001" , "000000011001" , "000000011001" , "000000011000" ,
"000000011000" , "000000011000" , "000000011000" , "000000011000" ,
"000000011000" , "000000011000" , "000000010111" , "000000010111" ,
"000000010111" , "000000010111" , "000000010111" , "000000010111" ,
"000000010111" , "000000010111" , "000000010110" , "000000010110" ,
"000000010110" , "000000010110" , "000000010110" , "000000010110" ,
"000000010110" , "000000010110" , "000000010101" , "000000010101" ,
"000000010101" , "000000010101" , "000000010101" , "000000010101" ,
"000000010101" , "000000010101" , "000000010101" , "000000010100" ,
"000000010100" , "000000010100" , "000000010100" , "000000010100" ,
"000000010100" , "000000010100" , "000000010100" , "000000010100" ,
"000000010011" , "000000010011" , "000000010011" , "000000010011" ,
"000000010011" , "000000010011" , "000000010011" , "000000010011" ,
"000000010011" , "000000010011" , "000000010011" , "000000010010" ,
"000000010010" , "000000010010" , "000000010010" , "000000010010" ,
"000000010010" , "000000010010" , "000000010010" , "000000010010" ,
"000000010010" , "000000010010" , "000000010010" , "000000010001" ,
"000000010001" , "000000010001" , "000000010001" , "000000010001" ,
"000000010001" , "000000010001" , "000000010001" , "000000010001" ,
"000000010001" , "000000010001" , "000000010001" , "000000010001" ,
"000000010000" , "000000010000" , "000000010000" , "000000010000" ,
"000000010000" , "000000010000" , "000000010000" , "000000010000" ,
"000000010000" , "000000010000" , "000000010000" , "000000010000" ,
"000000010000" , "000000010000" , "000000010000" );

begin


   do <= ROM_VAL(conv_integer(unsigned(addr)));
end process;
end Beh;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.std_logic_arith.all;

--------------------------------------------------------------------------
--
-- Fichier : largeur.vhd
--
-- determine la duree de l'impulsion
--
--------------------------------------------------------------------------

entity largeur is
port (CLK, RESET : in  std_logic;
      START      : in  std_logic;
      STOP       : in  std_logic;
      LARG       : out std_logic_vector(7 downto 0));
end largeur;

architecture BEHAVIORAL of largeur is

 signal QOUT  : std_logic_vector(7 downto 0);

begin

 process(RESET, CLK)
  variable TEMP : std_logic_vector(7 downto 0);
  begin
    if (RESET = '1') then
       QOUT <= ( others => '0');
       LARG <= ( others => '0');
    elsif (CLK'event and CLK = '1') then
       if (START = '1') then
          TEMP :=  "00000001";
         QOUT <= TEMP;
       elsif (STOP = '1') then
         QOUT <= QOUT;
         LARG <= QOUT;
       else
         TEMP := unsigned(QOUT) + 1 ;
         QOUT <= TEMP;
       end if;
    end if;
  end process;

end BEHAVIORAL;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

------------------------------------------------------------
-- Fichier : gain_s.vhd
--
-- generateur de gain
-- synchronise la sortie du commande de l'attenuateur
-- a gain variable, en la rendant independante de MH
--
------------------------------------------------------------


entity gain_s is
 port( CLK           : in  std_logic;
       RESET         : in  std_logic;
       MH            : in  std_logic_vector(1 downto 0);
       ML            : in  std_logic_vector(1 downto 0);
       MAX_IN        : in  std_logic_vector(7 downto 0);
       START_DET     : in  std_logic;
       END_DET       : in  std_logic;
       GAIN_OUT      : out std_logic_vector(7 downto 0);
       PULSE_START_F : out std_logic;
       PULSE_END_F   : out std_logic);
end gain_s;

architecture rtl of gain_s is

--component lut_p
--generic( adr_bit : integer := 8; Nbit : integer := 8);
--port (  CLK,RESET         : in std_logic;
--       DATA_IN           : in std_logic_vector(adr_bit-1 downto 0);
--       DATA_OUT          : out std_logic_vector(Nbit-1 downto 0));
--end component;

component lut_p
      port (
      addr: IN std_logic_VECTOR(7 downto 0);
      clk: IN std_logic;
      di: IN std_logic_VECTOR(7 downto 0);
      we: IN std_logic;
      en: IN std_logic;
      rst: IN std_logic;
      do: OUT std_logic_VECTOR(7 downto 0));
end component;

 signal GAIN_SET,GAIN_SET1              : std_logic;
 signal GAIN_RESET,GAIN_RESET1          : std_logic;
 signal P_START1, P_START2 ,P_START3    : std_logic;
 signal P_END1,P_END2,P_END3            : std_logic;
 signal GAIN_IN, DATA_OUT    : std_logic_vector(7 downto 0);
```

```
 signal GAIN_IN1 , GAIN_IN2  : std_logic_vector(7 downto 0);
 signal GAIN_IN3             : std_logic_vector(7 downto 0);

 signal WE,EN : std_logic;
 signal DI    : std_logic_vector( 7 downto 0);
begin
--*****************************************

PULSE_START_F <= GAIN_SET1;
PULSE_END_F   <= GAIN_RESET1;

lutpr1 : lut_p port map(MAX_IN,CLK,DI,WE,EN,RESET,GAIN_IN);
DI <= ( others => '0'); WE <= '0'; EN <= '1';

un : process(CLK, RESET)
begin
      if (RESET = '1') then
         P_START1    <= '0';
         P_START2    <= '0';
         P_START3    <= '0';
         P_END1      <= '0';
         P_END2      <= '0';
         P_END3      <= '0';
         GAIN_SET1   <= '0';
         GAIN_RESET1 <= '0';
         GAIN_IN1    <= (others => '0');
         GAIN_IN2    <= (others => '0');
         GAIN_IN3    <= (others => '0');
      elsif (CLK' event and CLK = '1') then
         P_START1    <= START_DET  ;
         P_START2    <= P_START1   ;
         P_START3    <= P_START2   ;
         P_END1      <= END_DET    ;
         P_END2      <= P_END1     ;
         P_END3      <= P_END2     ;
         GAIN_SET1   <= GAIN_SET   ;
         GAIN_RESET1 <= GAIN_RESET ;
         GAIN_IN1    <= GAIN_IN    ;
         GAIN_IN2    <= GAIN_IN1;
         GAIN_IN3    <= GAIN_IN2   ;
      end if;
 end process;


--*************************************************
-- registre de sortie pour le gain final
--*************************************************
deux : process(CLK)
   begin
      if( CLK' Event and CLK = '1') then
         if ( (RESET or GAIN_RESET) = '1') then
            GAIN_OUT <= ( others => '0');
         elsif (GAIN_SET = '1' ) then
```

```
            GAIN_OUT <= DATA_OUT;
          end if;
       end if;
    end process;


mux_01 : process(MH, START_DET, P_START1, P_START2, P_START3)
begin
case MH is
   when "11"   => GAIN_SET <= START_DET ;
   when "10"   => GAIN_SET <= P_START1 ;
   when "01"   => GAIN_SET <= P_START2 ;
   when others => GAIN_SET <= P_START3 ;
end case;
end process;

mux_02 : process(ML, END_DET, P_END1, P_END2, P_END3)
begin
case ML is
   when "11"   => GAIN_RESET <= END_DET ;
   when "10"   => GAIN_RESET <= P_END1 ;
   when "01"   => GAIN_RESET <= P_END2 ;
   when others => GAIN_RESET <= P_END3 ;
end case;
end process;

mux_gain : process(MH,GAIN_IN,GAIN_IN1,GAIN_IN2,GAIN_IN3)
begin
case MH is
  when "11" =>   DATA_OUT <= GAIN_IN;
  when "10" =>   DATA_OUT <= GAIN_IN1;
  when "01" =>   DATA_OUT <= GAIN_IN2;
  when others => DATA_OUT <= GAIN_IN3;
end case;
end process;

end rtl;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
USE IEEE.std_logic_arith.all;
------------------------------------------------------------------
--
-- Fichier : lut_p.vhd
--
-- ROM 256 x 8
-- contient les parametres de gain
--
------------------------------------------------------------------

entity lut_p is
      port (
      addr: IN std_logic_VECTOR(7 downto 0);
      clk : IN std_logic;
      di  : IN std_logic_VECTOR(7 downto 0);
      we  : IN std_logic;
      en  : IN std_logic;
      rst : IN std_logic;
      do  : OUT std_logic_VECTOR(7 downto 0));
end lut_p;


architecture comport of lut_p is
signal DDD,DDD1 : std_logic_vector(7 downto 0);

begin

-- genere une ROM et son contenu
process(addr)

subtype WORD is std_logic_vector(7 downto 0);
TYPE ROM_TBL is array (0 to 255) of WORD;

constant ROM_VAL : ROM_TBL := ROM_TBL'(
 "00000000" , "00000001" , "00000010" , "00000011" , "00000100" ,
 "00000101" , "00000110" , "00000111" , "00001000" , "00001001" ,
 "00001010" , "00001011" , "00001100" , "00001101" , "00001110" ,
 "00001111" , "00010000" , "00010001" , "00010010" , "00010011" ,
 "00010100" , "00010101" , "00010110" , "00010111" , "00011000" ,
 "00011001" , "00011010" , "00011011" , "00011100" , "00011101" ,
 "00011110" , "00011111" , "00100000" , "00100001" , "00100010" ,
 "00100011" , "00100100" , "00100101" , "00100110" , "00100111" ,
 "00101000" , "00101001" , "00101010" , "00101011" , "00101100" ,
 "00101101" , "00101110" , "00101111" , "00110000" , "00110001" ,
 "00110010" , "00110011" , "00110100" , "00110101" , "00110110" ,
 "00110111" , "00111000" , "00111001" , "00111010" , "00111011" ,
 "00111100" , "00111101" , "00111110" , "00111111" , "01000000" ,
 "01000001" , "01000010" , "01000011" , "01000100" , "01000101" ,
 "01000110" , "01000111" , "01001000" , "01001001" , "01001010" ,
 "01001011" , "01001100" , "01001101" , "01001110" , "01001111" ,
```

```
 "01010000" , "01010001" , "01010010" , "01010011" , "01010100" ,
 "01010101" , "01010110" , "01010111" , "01011000" , "01011001" ,
 "01011010" , "01011011" , "01011100" , "01011101" , "01011110" ,
 "01011111" , "01100000" , "01100001" , "01100010" , "01100011" ,
 "01100100" , "01100101" , "01100110" , "01100111" , "01101000" ,
 "01101001" , "01101010" , "01101011" , "01101100" , "01101101" ,
 "01101110" , "01101111" , "01110000" , "01110001" , "01110010" ,
 "01110011" , "01110100" , "01110101" , "01110110" , "01110111" ,
 "01111000" , "01111001" , "01111010" , "01111011" , "01111100" ,
 "01111101" , "01111110" , "01111111" , "10000000" , "10000001" ,
 "10000010" , "10000011" , "10000100" , "10000101" , "10000110" ,
 "10000111" , "10001000" , "10001001" , "10001010" , "10001011" ,
 "10001100" , "10001101" , "10001110" , "10001111" , "10010000" ,
 "10010001" , "10010010" , "10010011" , "10010100" , "10010101" ,
 "10010110" , "10010111" , "10011000" , "10011001" , "10011010" ,
 "10011011" , "10011100" , "10011101" , "10011110" , "10011111" ,
 "10100000" , "10100001" , "10100010" , "10100011" , "10100100" ,
 "10100101" , "10100110" , "10100111" , "10101000" , "10101001" ,
 "10101010" , "10101011" , "10101100" , "10101101" , "10101110" ,
 "10101111" , "10110000" , "10110001" , "10110010" , "10110011" ,
 "10110100" , "10110101" , "10110110" , "10110111" , "10111000" ,
 "10111001" , "10111010" , "10111011" , "10111100" , "10111101" ,
 "10111110" , "10111111" , "11000000" , "11000001" , "11000010" ,
 "11000011" , "11000100" , "11000101" , "11000110" , "11000111" ,
 "11001000" , "11001001" , "11001010" , "11001011" , "11001100" ,
 "11001101" , "11001110" , "11001111" , "11010000" , "11010001" ,
 "11010010" , "11010011" , "11010100" , "11010101" , "11010110" ,
 "11010111" , "11011000" , "11011001" , "11011010" , "11011011" ,
 "11011100" , "11011101" , "11011110" , "11011111" , "11100000" ,
 "11100001" , "11100010" , "11100011" , "11100100" , "11100101" ,
 "11100110" , "11100111" , "11101000" , "11101001" , "11101010" ,
 "11101011" , "11101100" , "11101101" , "11101110" , "11101111" ,
 "11110000" , "11110001" , "11110010" , "11110011" , "11110100" ,
 "11110101" , "11110110" , "11110111" , "11111000" , "11111001" ,
 "11111010" , "11111011" , "11111100" , "11111101" , "11111110" ,
 "11111111" );
 begin
 DO <= ROM_VAL(conv_integer(unsigned(addr)));

end process;


end comport;
```

# ANNEXE B

# STRATEGIE : LINÉAIRE ET SUBDIVISION

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


--------------------------------------------------------------------------
--
-- Fichier : lin_subdiv.vhd
--
-- determine la partie decimale du PTOA sur 4 bits
--
--------------------------------------------------------------------------

entity lin_subdiv is
port    ( CLK , RESET : in  std_logic;
          Y0          : in  std_logic_vector(7 downto 0);
          PENTE       : in  std_logic_vector(7 downto 0);
          SEUIL       : in  std_logic_vector(7 downto 0);
          RESULT      : out std_logic_vector(3 downto 0)
        );
end lin_subdiv;


architecture  comport  of  lin_subdiv  is


component mult_16
    port (
         CLK,RESET: in STD_LOGIC;
         PENTE: in STD_LOGIC_VECTOR (7 downto 0);
         P1  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P2  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P3  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P4  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P5  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P6  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P7  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P8  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P9  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P10 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P11 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P12 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P13 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P14 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P15 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
    );
end component;


component comp_n
generic ( N : integer := 8);
```

```
  port( A,B   : in std_logic_vector(N-1 downto 0);
        O       : out std_logic);
end component;


component prio8
port     ( CLK,RESET : std_logic;
           DIN       : in  std_logic_vector(7 downto 0);
           NUMERO    : out std_logic_vector(2 downto 0);
           V         : out std_logic);
end component;

signal NUM1,NUM2 : std_logic_vector(2 downto 0);
signal V1,V2     : std_logic;

signal X1        : std_logic_vector(179 downto 0);
signal X2,X2_1   : std_logic_vector( 16 downto 0);
signal X3,X4     : std_logic_vector( 17 downto 0);

--signal RESULT1       : std_logic_vector(3 downto 0);
signal SEUIL_16, SEUIL_16_1, SEUIL_16_2     : std_logic_vector(11
downto 0);
signal SEUIL_Y0  : std_logic_vector(7 downto 0);

signal PRD1, PRD2, PRD3 , PRD4 : std_logic_vector(11 downto 0);
signal PRD5, PRD6, PRD7 , PRD8 : std_logic_vector(11 downto 0);
signal PRD9, PRD10, PRD11 , PRD12 : std_logic_vector(11 downto 0);
signal PRD13, PRD14, PRD15  : std_logic_vector(11 downto 0);



begin
  SEUIL_Y0 <= SEUIL - Y0;
  SEUIL_16 <=  SEUIL_Y0 & "0000";



 multiplication :
       mult_16 port map (CLK , RESET, PENTE, PRD1 , PRD2 , PRD3 ,PRD4 ,
PRD5 , PRD6,PRD7,
                         PRD8, PRD9 , PRD10, PRD11, PRD12,PRD13, PRD14,
PRD15);

-- comparaison :
    comp_00 : comp_n generic map(12) port map (PRD1,SEUIL_16_2, X2(1));
    comp_01 : comp_n generic map(12) port map (PRD2,SEUIL_16_2, X2(2));
    comp_02 : comp_n generic map(12) port map (PRD3,SEUIL_16_2, X2(3));
    comp_03 : comp_n generic map(12) port map (PRD4,SEUIL_16_2, X2(4));
    comp_04 : comp_n generic map(12) port map (PRD5,SEUIL_16_2, X2(5));
    comp_05 : comp_n generic map(12) port map (PRD6,SEUIL_16_2, X2(6));
    comp_06 : comp_n generic map(12) port map (PRD7,SEUIL_16_2, X2(7));
    comp_07 : comp_n generic map(12) port map (PRD8,SEUIL_16_2, X2(8));
```

```
    comp_08 : comp_n generic map(12) port map (PRD9,SEUIL_16_2, X2(9));
    comp_09 : comp_n generic map(12) port map(PRD10,SEUIL_16_2,X2(10));
    comp_10 : comp_n generic map(12) port map(PRD11,SEUIL_16_2,X2(11));
    comp_11 : comp_n generic map(12) port map(PRD12,SEUIL_16_2,X2(12));
    comp_12 : comp_n generic map(12) port map(PRD13,SEUIL_16_2,2(13));
    comp_13 : comp_n generic map(12) port map(PRD14,SEUIL_16_2,X2(14));
    comp_14 : comp_n generic map(12) port map(PRD15,SEUIL_16_2,X2(15));
      X2(0) <= '0'; X2(16) <= '1'; -- Y0 plus petit que le seuil, Y1
superieur au seuil
-- decodage:

  deco1    : prio8 port map (CLK, RESET, X4( 7 downto 0), NUM1, V1);
  deco2    : prio8 port map (CLK, RESET, X4(15 downto 8), NUM2, V2);

-- selection:
   selec : process(NUM1,NUM2,V1,V2)
           begin
              if V2 = '1' then
                   RESULT <=  ("1" & NUM2);
               else
                   RESULT <= ("0" & NUM1);
              end if;
           end process;

 X4 <= not X3;

-- registre de resynchronisation
  pp1 : process(CLK,RESET)
    begin
      if ( RESET = '1') then
         X3     <=  (others => '0');
         --RESULT <=  (others => '1');
         SEUIL_16_1 <= (others => '1');
         SEUIL_16_2 <= (others => '1');
      elsif( CLK' Event and CLK = '1') then
         X3(17 downto 1)  <=  X2;
         --RESULT <= RESULT1;
         SEUIL_16_1 <= SEUIL_16;
         SEUIL_16_2 <= SEUIL_16_1;
      end if;
    end process;

 end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.std_logic_arith.all;

----------------------------------------------------------------
-- Fichier mult_16
-- Ce module effectue le calcul de pente * i/16
-- i variant de 1 a 15
-- le calcul est effectue par des declage et addition
----------------------------------------------------------------

entity mult_16 is
    port (
         CLK,RESET: in STD_LOGIC;
         PENTE: in STD_LOGIC_VECTOR (7 downto 0);
         P1  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P2  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P3  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P4  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P5  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P6  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P7  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P8  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P9  : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P10 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P11 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P12 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P13 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P14 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
         P15 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)

    );
end mult_16;

architecture mult_16_arch of mult_16 is

 signal PI1 : std_logic_vector(11 downto 0);
 signal PI2 : std_logic_vector(11 downto 0);
 signal PI3 : std_logic_vector(11 downto 0);
 signal PI4 : std_logic_vector(11 downto 0);
 signal PI5 : std_logic_vector(11 downto 0);
 signal PI6 : std_logic_vector(11 downto 0);
 signal PI7 : std_logic_vector(11 downto 0);
 signal PI8 : std_logic_vector(11 downto 0);
 signal PI9 : std_logic_vector(11 downto 0);
 signal PI10 : std_logic_vector(11 downto 0);
 signal PI11 : std_logic_vector(11 downto 0);
 signal PI12 : std_logic_vector(11 downto 0);
 signal PI13 : std_logic_vector(11 downto 0);
 signal PI14 : std_logic_vector(11 downto 0);
 signal PI15 : std_logic_vector(11 downto 0);
```

```
 signal PA1 : std_logic_vector(11 downto 0);
 signal PA2 : std_logic_vector(11 downto 0);
 signal PA3 : std_logic_vector(11 downto 0);
 signal PA4 : std_logic_vector(11 downto 0);
 signal PA5 : std_logic_vector(11 downto 0);
 signal PA6 : std_logic_vector(11 downto 0);
 signal PA7 : std_logic_vector(11 downto 0);
 signal PA8 : std_logic_vector(11 downto 0);
 signal PA9 : std_logic_vector(11 downto 0);
 signal PA10 : std_logic_vector(11 downto 0);
 signal PA11 : std_logic_vector(11 downto 0);
 signal PA12 : std_logic_vector(11 downto 0);
 signal PA13 : std_logic_vector(11 downto 0);
 signal PA14 : std_logic_vector(11 downto 0);
 signal PA15 : std_logic_vector(11 downto 0);

begin

-- Multiplication par 1
     PI1 <= "0000" & PENTE;
-- Multiplication par 2 = 2*P
     PI2 <= "000" & PENTE & '0';
-- Multiplication par 3
     PI3 <= PI1 + PI2 ;
-- Multiplication par 4
     PI4 <=  "00" & PENTE & "00";
-- Multiplication par 5
     PI5 <= PI1 + PI4;
-- Multiplication par 8 = 8P
    PI8 <= '0' & PENTE & "000";
-- Multiplication par 9
     PI9 <= PI8 + PI1;

registres : process(CLK,RESET)
   begin
      if (RESET = '1') then
         PA1 <= ( others => '0');
         PA2 <= ( others => '0');
         PA3 <= ( others => '0');
         PA4 <= ( others => '0');
         PA5 <= ( others => '0');
         PA8 <= ( others => '0');
         PA9 <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         PA1 <= PI1;
         PA2 <= PI2;
         PA3 <= PI3;
         PA4 <= PI4;
         PA5 <= PI5;
         PA8 <= PI8;
         PA9 <= PI9;
      end if;
```

```
 end process;

-- Multiplication par 6 = 2 * P3
    PI6 <= PA3(10 downto 0) & '0' ;
-- Multiplication par 7
    PI7 <= PI6 + PA1;
-- Multiplication par 10 = 2*P5
    PI10 <= PA5(10 downto 0) & '0';
-- Multiplication par 11
    PI11 <= PI10 + PA1;
-- Multiplication par 12
     PI12 <= PI6(10 downto 0) & '0';
-- Multiplication par 13
    PI13 <= PI12 + PA1;
-- Multiplication par 14
   PI14 <= PI7(10 downto 0) & '0';
-- Multiplication par 15
    PI15 <= PA3(10 downto 0) & '0' + PA9;

registr_01 : process(CLK,RESET)
   begin
      if (RESET = '1') then
         P1 <= ( others => '0');
         P2 <= ( others => '0');
         P3 <= ( others => '0');
         P4 <= ( others => '0');
         P5 <= ( others => '0');
         P6 <= ( others => '0');
         P7 <= ( others => '0');
         P8 <= ( others => '0');
         P9 <= ( others => '0');
         P10 <= ( others => '0');
         P11 <= ( others => '0');
         P12 <= ( others => '0');
         P13 <= ( others => '0');
         P14 <= ( others => '0');
         P15 <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         P1  <= PA1;
         P2  <= PA2;
         P3  <= PA3;
         P4  <= PA4;
         P5  <= PA5;
         P8  <= PA8;
         P9  <= PA9;

         P6  <= PI6;
         P7  <= PI7;
         P10 <= PI10;
         P11 <= PI11;
         P12 <= PI12;
         P13 <= PI13;
         P14 <= PI14;
```

```
            P15 <= PI15;
        end if;
    end process;
end mult_16_arch;
```

# ANNEXE C

# STRATEGIE : QUADRATIQUE ET SUBDIVISION

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

------------------------------------------------------------------
--
-- Fichier : quad_subdiv.vhd
--
-- détermine la partie décimale du PTOA en
-- utilisant l'approximation quadratique et subdivision
------------------------------------------------------------------

entity  quad_subdiv  is
port    ( CLK , RESET : in std_logic;
          Y0        : in std_logic_vector(7 downto 0);
          Y1        : in std_logic_vector(7 downto 0);
          Y2        :  in std_logic_vector(7 downto 0);
          SEUIL     : in std_logic_vector(7 downto 0);
          RESULT    : out std_logic_vector(3 downto 0));
end;

architecture  comport  of  quad_subdiv  is

component coef_abc
  port ( CLK,RESET : in std_logic;
         Y0  : in std_logic_vector(7 downto 0);
         Y1  : in std_logic_vector(7 downto 0);
         Y2  : in std_logic_vector(7 downto 0);
         A,B : out std_logic_vector(9 downto 0);
         C   : out std_logic_vector(7 downto 0)
       );
end component;

component ax2bx
generic ( Nbits_A : integer := 10; Nbits_B : integer := 10);
port    ( CLK , RESET : in  std_logic;
          A         : in  std_logic_vector(Nbits_A - 1 downto 0);
          B         : in  std_logic_vector(Nbits_B - 1 downto 0);
          RESULT1   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT2   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT3   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT4   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT5   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT6   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT7   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT8   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT9   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT10  : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT11  : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT12  : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT13  : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT14  : out std_logic_vector( Nbits_A + 7 downto 0);
```

```
          RESULT15 : out std_logic_vector( Nbits_A + 7 downto 0);
end component;

component comp_n
generic ( N : integer := 8);
  port( A,B   : in std_logic_vector(N-1 downto 0);
        O       : out std_logic);
end component;


component prio8
port     ( CLK,RESET : std_logic;
           DIN       : in  std_logic_vector(7 downto 0);
           NUMERO    : out std_logic_vector(2 downto 0);
           V         : out std_logic);
end component;

signal NUM1,NUM2 : std_logic_vector(2 downto 0);
signal V1,V2     : std_logic;
signal X1        : std_logic_vector(179 downto 0);
signal X2        : std_logic_vector( 16 downto 0);
signal X3,X4     : std_logic_vector( 17 downto 0);
signal SEUIL_16     : std_logic_vector(17 downto 0);
signal SEUIL_Y0     : std_logic_vector(7 downto 0);
signal SEUIL_Y0_1   : std_logic_vector(7 downto 0);
signal SEUIL_Y0_2   : std_logic_vector(7 downto 0);
signal SEUIL_Y0_3   : std_logic_vector(7 downto 0);
signal SEUIL_Y0_4   : std_logic_vector(7 downto 0);
signal SEUIL_Y0_5   : std_logic_vector(7 downto 0);
signal PRD1 , PRD2  : std_logic_vector(17 downto 0);
signal PRD3 , PRD4  : std_logic_vector(17 downto 0);
signal PRD5 , PRD6  : std_logic_vector(17 downto 0);
signal PRD7 , PRD8  : std_logic_vector(17 downto 0);
signal PRD9 , PRD10 : std_logic_vector(17 downto 0);
signal PRD11, PRD12 : std_logic_vector(17 downto 0);
signal PRD13, PRD14 : std_logic_vector(17 downto 0);
signal PRD15        : std_logic_vector(17 downto 0);
signal C     : std_logic_vector(7 downto 0);
signal B, A  : std_logic_vector(9 downto 0);

begin
  SEUIL_Y0 <=  SEUIL - Y0;
  SEUIL_16 <= '0' & SEUIL_Y0_4 & "000000000"; -- 16*16*(SEUIL - Y0)*2

-- calcul des coefficients A,B et C
coefficients_abc :
    coef_abc port map(CLK,RESET,Y0,Y1,Y2,A,B,C);

-- calcul de Ax2+Bx : on calcule ici 2ax2 + 2bx
 multiplication :
       ax2bx generic map(10,10)
         port map (CLK , RESET, A , B, PRD1, PRD2, PRD3, PRD4,
PRD5,PRD6,
```

```
            PRD7, PRD8, PRD9,  PRD10,PRD11, PRD12, PRD13, PRD14,
PRD15);

-- comparaison :
    comp_00 : comp_n generic map(18) port map (PRD1 ,SEUIL_16, X2(1));
    comp_01 : comp_n generic map(18) port map (PRD2 ,SEUIL_16, X2(2));
    comp_02 : comp_n generic map(18) port map (PRD3 ,SEUIL_16, X2(3));
    comp_03 : comp_n generic map(18) port map (PRD4 ,SEUIL_16, X2(4));
    comp_04 : comp_n generic map(18) port map (PRD5 ,SEUIL_16, X2(5));
    comp_05 : comp_n generic map(18) port map (PRD6 ,SEUIL_16, X2(6));
    comp_06 : comp_n generic map(18) port map (PRD7 ,SEUIL_16, X2(7));
    comp_07 : comp_n generic map(18) port map (PRD8 ,SEUIL_16, X2(8));
    comp_08 : comp_n generic map(18) port map (PRD9 ,SEUIL_16, X2(9));
    comp_09 : comp_n generic map(18) port map (PRD10,SEUIL_16, X2(10));
    comp_10 : comp_n generic map(18) port map (PRD11,SEUIL_16, X2(11));
    comp_11 : comp_n generic map(18) port map (PRD12,SEUIL_16, X2(12));
    comp_12 : comp_n generic map(18) port map (PRD13,SEUIL_16, X2(13));
    comp_13 : comp_n generic map(18) port map (PRD14,SEUIL_16, X2(14));
    comp_14 : comp_n generic map(18) port map (PRD15,SEUIL_16, X2(15));

      X2(0) <= '0'; X2(16) <= '1';

-- decodage:

  deco1    : prio8 port map (CLK, RESET, X4( 7 downto 0), NUM1, V1);
  deco2    : prio8 port map (CLK, RESET, X4(15 downto 8), NUM2, V2);

-- selection:
   selec : process(NUM1,NUM2,V1,V2)
           begin
              if V2 = '1' then
                   RESULT <= ("1" & NUM2);
               else
                   RESULT <= ("0" & NUM1);
              end if;
           end process;

 X4 <= not X3;

-- registre de resynchronisation
  pp1 : process(CLK,RESET)
    begin
      if ( RESET = '1') then
         X3  <=  (others => '0');
         SEUIL_Y0_1 <= ( others => '0');
         SEUIL_Y0_2 <= ( others => '0');
         SEUIL_Y0_3 <= ( others => '0');
         SEUIL_Y0_4 <= ( others => '0');
         SEUIL_Y0_5 <= ( others => '0');

      elsif( CLK' Event and CLK = '1') then
         X3(17 downto 1)  <=  X2;
         SEUIL_Y0_1 <= SEUIL_Y0;
```

```
            SEUIL_Y0_2 <= SEUIL_Y0_1;
            SEUIL_Y0_3 <= SEUIL_Y0_2;
            SEUIL_Y0_4 <= SEUIL_Y0_3;
            SEUIL_Y0_5 <= SEUIL_Y0_4;
         end if;
      end process;

end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

------------------------------------------------------------------
--
-- Fichier : coef_abc.vhd
--
-- détermine les coefficients A,B et C
-- a partir des coordonnees de trois points de la parabole
------------------------------------------------------------------

entity  coef_abc  is
  port ( CLK,RESET : in std_logic;
         Y0        : in std_logic_vector(7 downto 0);
         Y1        : in std_logic_vector(7 downto 0);
         Y2        : in std_logic_vector(7 downto 0);
         A,B       : out std_logic_vector(9 downto 0);
         C         : out std_logic_vector(7 downto 0));
end;

architecture  comport  of  coef_abc  is

 signal A1,A2 : std_logic_vector(8 downto 0);
 signal A3    : std_logic_vector(8 downto 0);
 signal B1,B2 : std_logic_vector(8 downto 0);
 signal B21   : std_logic_vector(8 downto 0);
 signal AI,BI : std_logic_vector(9 downto 0);
 signal CI    : std_logic_vector(7 downto 0);

begin
-- on calcule : A = .5* (Y2 + Y0) - Y1  ,  C =  Y0

 A1 <= '0' & Y2 - Y1;  A2 <= '0' & Y1 - Y0;
 A3 <= signed(A1) - signed(A2);  AI <=  A3(8) & A3 ;

-- 2B = 4Y1 - Y2 - 3Y0 = 2( Y1-Y0 ) - (Y2 - Y1) + (Y1 - Y0)
   B1  <= '0' & Y1 - Y0;    B2  <= '0' & Y2 - Y1 ;
   B21 <= signed(B1) - signed(B2) ;
   BI  <= signed( B1 & '0') + signed(B21) ;

   CI  <=  Y0 ;

  pp1 : process(CLK,RESET)
    begin
      if ( RESET = '1') then
         A  <=  (others => '0');
         B  <= ( others => '0');
         C  <= ( others => '0');
      elsif( CLK' Event and CLK = '1') then
         A  <=  AI; B  <= BI; C  <= CI;
```

```
      end if;
    end process;
end;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_SIGNED.all;

--------------------------------------------------------------------
--
-- Fichier : ax2bx.vhd
-- Ce module effectue le calcul de ax2 + bx
-- x variant de 1 a 15
--
--------------------------------------------------------------------

entity ax2bx is
generic ( Nbits_A : integer := 10; Nbits_B : integer := 10);
port    ( CLK , RESET : in  std_logic;
          A          : in  std_logic_vector( Nbits_A - 1 downto 0);
          B          : in  std_logic_vector( Nbits_B - 1 downto 0);
          RESULT1    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT2    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT3    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT4    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT5    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT6    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT7    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT8    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT9    : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT10   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT11   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT12   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT13   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT14   : out std_logic_vector( Nbits_A + 7 downto 0);
          RESULT15   : out std_logic_vector( Nbits_A + 7 downto 0)
        );
end ax2bx;
architecture  comport  of  ax2bx  is

component mult_16
   generic ( N_bits : integer := 8);
   port (
         CLK,RESET: in STD_LOGIC;
         PENTE: in STD_LOGIC_VECTOR( N_bits - 1 Downto 0);
         P1  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P2  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P3  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P4  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P5  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P6  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P7  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P8  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P9  : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P10 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
         P11 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
```

```
        P12 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
        P13 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
        P14 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0);
        P15 : OUT STD_LOGIC_VECTOR( N_bits + 3 DOWNTO 0)
    );
end component;

component  AX2
    generic ( N_bits : integer := 8);
    port (
        CLK,RESET: in STD_LOGIC;
        AA  : in STD_LOGIC_VECTOR ( N_bits-1 downto 0);
        P1  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P2  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P3  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P4  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P5  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P6  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P7  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P8  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P9  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P10 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P11 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P12 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P13 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P14 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
        P15 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0));
end component;


signal A_PRD1,A_PRD2,A_PRD3  : std_logic_vector( Nbits_A + 3 downto 0);
signal A_PRD4,A_PRD5,A_PRD6  : std_logic_vector( Nbits_A + 3 downto 0);
signal A_PRD7,A_PRD8,A_PRD9  : std_logic_vector( Nbits_A + 3 downto 0);
signal A_PRD10,A_PRD11,A_PRD12 : std_logic_vector(Nbits_A +3 downto 0);
signal A_PRD13,A_PRD14,A_PRD15 : std_logic_vector(Nbits_A +3 downto 0);
signal B_PRD1,B_PRD2,B_PRD3  : std_logic_vector(Nbits_B +3 downto 0);
signal B_PRD4,B_PRD5,B_PRD6  : std_logic_vector( Nbits_B + 3 downto 0);
signal B_PRD7,B_PRD8,B_PRD9  : std_logic_vector( Nbits_B + 3 downto 0);
signal B_PRD10,B_PRD11,B_PRD12 : std_logic_vector(Nbits_B +3 downto 0);
signal B_PRD13,B_PRD14,B_PRD15 : std_logic_vector(Nbits_B +3 downto 0);
signal A2_PRD1,A2_PRD2,A2_PRD3 : std_logic_vector(Nbits_A +7 downto 0);
signal A2_PRD4,A2_PRD5,A2_PRD6 : std_logic_vector(Nbits_A +7 downto 0);
signal A2_PRD7,A2_PRD8,A2_PRD9 : std_logic_vector(Nbits_A +7 downto 0);
signal A2_PRD10,A2_PRD11,A2_PRD12:std_logic_vector(Nbits_A +7downto 0);
signal A2_PRD13,A2_PRD14,A2_PRD15:std_logic_vector(Nbits_A +7downto 0);
signal B1_PRD1,B1_PRD2,B1_PRD3: std_logic_vector(Nbits_B + 3 downto 0);
signal B1_PRD4,B1_PRD5,B1_PRD6: std_logic_vector(Nbits_B + 3 downto 0);
signal B1_PRD7,B1_PRD8,B1_PRD9: std_logic_vector(Nbits_B + 3 downto 0);
signal B1_PRD10,B1_PRD11,B1_PRD12: std_logic_vector(Nbits_B+3 downto0);
signal B1_PRD13,B1_PRD14,B1_PRD15: std_logic_vector(Nbits_B+3downto 0);
signal RES_1,RES_2,RES_3,RES_4 : std_logic_vector(Nbits_A +7 downto 0);
signal RES_5,RES_6,RES_7,RES_8 : std_logic_vector(Nbits_A +7 downto 0);
signal RES_9,RES_10,RES_11,RES_12:std_logic_vector(Nbits_A +7 downto0);
```

```
signal RES_13,RES_14,RES_15 : std_logic_vector( Nbits_A + 7 downto 0);

begin

 multi_B :
       mult_16 generic map (Nbits_b)
            port map (CLK,RESET,B,B_PRD1,B_PRD2,B_PRD3,B_PRD4 ,
                      B_PRD5,B_PRD6,B_PRD7,B_PRD8,B_PRD9,B_PRD10,
                      B_PRD11,B_PRD12,B_PRD13,B_PRD14,B_PRD15);

multi_ax2 :
       ax2 generic map (Nbits_A)
             port map (CLK,RESET,A,A2_PRD1,
                 A2_PRD2,A2_PRD3,A2_PRD4,A2_PRD5,A2_PRD6, A2_PRD7,
                 A2_PRD8,A2_PRD9,A2_PRD10,A2_PRD11,A2_PRD12,A2_PRD13,
                 A2_PRD14,A2_PRD15);


  RES_1  <= A2_PRD1  + (B_PRD1  & "0000");
  RES_2  <= A2_PRD2  + (B_PRD2  & "0000");
  RES_3  <= A2_PRD3  + (B_PRD3  & "0000");
  RES_4  <= A2_PRD4  + (B_PRD4  & "0000");
  RES_5  <= A2_PRD5  + (B_PRD5  & "0000");
  RES_6  <= A2_PRD6  + (B_PRD6  & "0000");
  RES_7  <= A2_PRD7  + (B_PRD7  & "0000");
  RES_8  <= A2_PRD8  + (B_PRD8  & "0000");
  RES_9  <= A2_PRD9  + (B_PRD9  & "0000");
  RES_10 <= A2_PRD10 + (B_PRD10 & "0000");
  RES_11 <= A2_PRD11 + (B_PRD11 & "0000");
  RES_12 <= A2_PRD12 + (B_PRD12 & "0000");
  RES_13 <= A2_PRD13 + (B_PRD13 & "0000");
  RES_14 <= A2_PRD14 + (B_PRD14 & "0000");
  RES_15 <= A2_PRD15 + (B_PRD15 & "0000");

  pp2 : process(CLK,RESET)
    begin
      if ( RESET = '1') then
        RESULT1 <= ( others => '0');
        RESULT2 <= ( others => '0');
        RESULT3 <= ( others => '0');
        RESULT4 <= ( others => '0');
        RESULT5 <= ( others => '0');
        RESULT6 <= ( others => '0');
        RESULT7 <= ( others => '0');
        RESULT8 <= ( others => '0');
        RESULT9 <= ( others => '0');
        RESULT10 <= ( others => '0');
        RESULT11 <= ( others => '0');
        RESULT12 <= ( others => '0');
        RESULT13 <= ( others => '0');
        RESULT14 <= ( others => '0');
        RESULT15 <= ( others => '0');
       elsif( CLK' Event and CLK = '1') then
```

```
        RESULT1  <= RES_1;
        RESULT2  <= RES_2;
        RESULT3  <= RES_3;
        RESULT4  <= RES_4;
        RESULT5  <= RES_5;
        RESULT6  <= RES_6;
        RESULT7  <= RES_7;
        RESULT8  <= RES_8;
        RESULT9  <= RES_9;
        RESULT10 <= RES_10;
        RESULT11 <= RES_11;
        RESULT12 <= RES_12;
        RESULT13 <= RES_13;
        RESULT14 <= RES_14;
        RESULT15 <= RES_15;
      end if;
    end process;
end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_SIGNED.all;

------------------------------------------------------------------
--
-- Fichier : ax2.vhd
-- Ce module effectue le calcul ax2
-- x variant de 1 a 15
--
------------------------------------------------------------------


entity  AX2  is
    generic ( N_bits : integer := 8);
    port (
         CLK,RESET: in STD_LOGIC;
         AA  : in STD_LOGIC_VECTOR (N_bits-1 downto 0);
         P1  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P2  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P3  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P4  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P5  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P6  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P7  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P8  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P9  : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P10 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P11 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P12 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P13 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P14 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0);
         P15 : OUT STD_LOGIC_VECTOR( N_bits+7 DOWNTO 0));
end;

architecture  ax2_arch  of  AX2  is

 signal PI1  : std_logic_vector( N_bits+7 downto 0);
 signal PI2  : std_logic_vector( N_bits+7 downto 0);
 signal PI3  : std_logic_vector( N_bits+7 downto 0);
 signal PI4  : std_logic_vector( N_bits+7 downto 0);
 signal PI5  : std_logic_vector( N_bits+7 downto 0);
 signal PI6  : std_logic_vector( N_bits+7 downto 0);
 signal PI7  : std_logic_vector( N_bits+7 downto 0);
 signal PI8  : std_logic_vector( N_bits+7 downto 0);
 signal PI9  : std_logic_vector( N_bits+7 downto 0);
 signal PI10 : std_logic_vector( N_bits+7 downto 0);
 signal PI11 : std_logic_vector( N_bits+7 downto 0);
 signal PI12 : std_logic_vector( N_bits+7 downto 0);
 signal PI13 : std_logic_vector( N_bits+7 downto 0);
 signal PI14 : std_logic_vector( N_bits+7 downto 0);
```

```
signal PI15 : std_logic_vector( N_bits+7 downto 0);
signal PI   : std_logic_vector( N_bits+7 downto 0);

signal PII1  : std_logic_vector( N_bits+7 downto 0);
signal PII2  : std_logic_vector( N_bits+7 downto 0);
signal PII3  : std_logic_vector( N_bits+7 downto 0);
signal PII4  : std_logic_vector( N_bits+7 downto 0);
signal PII5  : std_logic_vector( N_bits+7 downto 0);
signal PII6  : std_logic_vector( N_bits+7 downto 0);
signal PII7  : std_logic_vector( N_bits+7 downto 0);
signal PII8  : std_logic_vector( N_bits+7 downto 0);
signal PII9  : std_logic_vector( N_bits+7 downto 0);
signal PII10 : std_logic_vector( N_bits+7 downto 0);
signal PII11 : std_logic_vector( N_bits+7 downto 0);
signal PII12 : std_logic_vector( N_bits+7 downto 0);
signal PII13 : std_logic_vector( N_bits+7 downto 0);
signal PII14 : std_logic_vector( N_bits+7 downto 0);
signal PII15 : std_logic_vector( N_bits+7 downto 0);

signal PA1 : std_logic_vector( N_bits+7 downto 0);
signal PA2 : std_logic_vector( N_bits+7 downto 0);
signal PA3 : std_logic_vector( N_bits+7 downto 0);
signal PA4 : std_logic_vector( N_bits+7 downto 0);
signal PA5 : std_logic_vector( N_bits+7 downto 0);
signal PA6 : std_logic_vector( N_bits+7 downto 0);
signal PA7 : std_logic_vector( N_bits+7 downto 0);
signal PA8 : std_logic_vector( N_bits+7 downto 0);
signal PA9 : std_logic_vector( N_bits+7 downto 0);
signal PA10 : std_logic_vector( N_bits+7 downto 0);
signal PA11 : std_logic_vector( N_bits+7 downto 0);
signal PA12 : std_logic_vector( N_bits+7 downto 0);
signal PA13 : std_logic_vector( N_bits+7 downto 0);
signal PA14 : std_logic_vector( N_bits+7 downto 0);
signal PA15 : std_logic_vector( N_bits+7 downto 0);
signal PA   : std_logic_vector( N_bits+7 downto 0);

signal PII1_1 : std_logic_vector( N_bits+8 downto 0);
signal PII2_1 : std_logic_vector( N_bits+8 downto 0);
signal PII3_1 : std_logic_vector( N_bits+8 downto 0);
signal PII4_1 : std_logic_vector( N_bits+8 downto 0);
signal PI_1   : std_logic_vector( N_bits+8 downto 0);


signal S:  std_logic;

begin
     S <= AA( N_bits - 1);

     PI1 <=  S&S&S&S&S&S&S&S&AA ;      --    AA
     PI2 <=  S&S&S&S&S&S&AA&"00";      --  4*AA
     PI3 <=  S&S&S&S&S&AA&"000" + AA; --  9*AA
     PI4 <=  S&S&S&S&AA&"0000";        -- 16*AA
     PI5 <=  (S&S&S&S&AA&"0000")+  (AA &"000"); -- 24*AA
```

```
      PI6 <=  (S&S&S&AA&"00000" ) +  (AA &"00");  -- 36*AA
      PI7 <=  (S&S&S&AA&"00000")  +  (AA &"000"); -- 48*AA
      PI8 <=  S&S&AA & "000000";              -- 64*AA
      PI9 <=  (S&S&AA & "000000")+ (AA & "0000"); -- 80*AA
      PI10 <= S&S&AA & "000000"+ (AA & "00000"); -- 96*AA
      PI11 <= S&AA &   "0000000" + AA ;       -- 129*AA
      PI12 <= S&AA &   "0000000" + (AA & "0000"); -- 144*AA
      PI13 <= S&AA &   "0000000" + (AA & "00000"); -- 160*AA
      PI14 <= S&AA &   "0000000" + (AA * "000000");-- 192*AA
      PI15 <= S&S&S&AA&"00000" + AA ; -- 33*AA
      PI   <= S&S&S&S&S&AA & "000";

registres : process(CLK,RESET)
   begin
      if (RESET = '1') then
         PA1  <= ( others => '0');
         PA2  <= ( others => '0');
         PA3  <= ( others => '0');
         PA4  <= ( others => '0');
         PA5  <= ( others => '0');
         PA6  <= ( others => '0');
         PA7  <= ( others => '0');
         PA8  <= ( others => '0');
         PA9  <= ( others => '0');
         PA10 <= ( others => '0');
         PA11 <= ( others => '0');
         PA12 <= ( others => '0');
         PA13 <= ( others => '0');
         PA14 <= ( others => '0');
         PA15 <= ( others => '0');
         PA   <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         PA1  <= PI1;
         PA2  <= PI2;
         PA3  <= PI3;
         PA4  <= PI4;
         PA5  <= PI5;
         PA6  <= PI6;
         PA7  <= PI7;
         PA8  <= PI8;
         PA9  <= PI9;
         PA10 <= PI10;
         PA11 <= PI11;
         PA12 <= PI12;
         PA13 <= PI13;
         PA14 <= PI14;
         PA15 <= PI15;
         PA   <= PI;
      end if;
 end process;


    PII1  <= PA1;            --   1
```

```
    PII2  <= PA2;             --   4
    PII3  <= PA3;             --   9
    PII4  <= PA4;             --  16
    PII5  <= PA5 + PA1;       --  24 + 1
    PII6  <= PA6;             --  36
    PII7  <= PA7 + PA1;       --  48 + 1
    PII8  <= PA8;             --  64
    PII9  <= PA9  + PA1;      --  80 + 1
    PII10 <= PA10 + PA2;      --  96 + 4
    PII11 <= PA11 - PA ;      -- 129 - 8
    PII12 <= PA12;            -- 144
    PII13 <= PA13 + PA3;      -- 160 + 9
    PII14 <= PA14 + PA2;      -- 192 + 4
    PII15 <= PA14 + PA15;     -- 192 + 33


registr_01 : process(CLK,RESET)
   begin
      if (RESET = '1') then
         P1 <= ( others => '0');
         P2 <= ( others => '0');
         P3 <= ( others => '0');
         P4 <= ( others => '0');
         P5 <= ( others => '0');
         P6 <= ( others => '0');
         P7 <= ( others => '0');
         P8 <= ( others => '0');
         P9 <= ( others => '0');
         P10 <= ( others => '0');
         P11 <= ( others => '0');
         P12 <= ( others => '0');
         P13 <= ( others => '0');
         P14 <= ( others => '0');
         P15 <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         P1  <= PII1;
         P2  <= PII2;
         P3  <= PII3;
         P4  <= PII4;
         P5  <= PII5;
         P6  <= PII6;
         P7  <= PII7;
         P8  <= PII8;
         P9  <= PII9;
         P10 <= PII10;
         P11 <= PII11;
         P12 <= PII12;
         P13 <= PII13;
         P14 <= PII14;
         P15 <= PII15;
      end if;
 end process;
end;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
use IEEE.std_logic_arith.all;
--------------------------------------------------------------------
--
-- Fichier : mult_16.vhd
-- Ce module effectue le calcul de bx
-- x variant de 1 a 15
-- le calcul est effectue par des declage et addition
--------------------------------------------------------------------

entity mult_16 is
    generic ( N_bits : integer := 8);
    port (
         CLK,RESET: in STD_LOGIC;
         PENTE: in STD_LOGIC_VECTOR (N_bits-1 downto 0);
         P1   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P2   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P3   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P4   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P5   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P6   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P7   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P8   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P9   : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P10  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P11  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P12  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P13  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P14  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0);
         P15  : OUT STD_LOGIC_VECTOR( N_bits+3 DOWNTO 0)

    );
end mult_16;

architecture mult_16_arch of mult_16 is

 signal PI1  : std_logic_vector( N_bits+3 downto 0);
 signal PI2  : std_logic_vector( N_bits+3 downto 0);
 signal PI3  : std_logic_vector( N_bits+3 downto 0);
 signal PI4  : std_logic_vector( N_bits+3 downto 0);
 signal PI5  : std_logic_vector( N_bits+3 downto 0);
 signal PI6  : std_logic_vector( N_bits+3 downto 0);
 signal PI7  : std_logic_vector( N_bits+3 downto 0);
 signal PI8  : std_logic_vector( N_bits+3 downto 0);
 signal PI9  : std_logic_vector( N_bits+3 downto 0);
 signal PI10 : std_logic_vector( N_bits+3 downto 0);
 signal PI11 : std_logic_vector( N_bits+3 downto 0);
 signal PI12 : std_logic_vector( N_bits+3 downto 0);
 signal PI13 : std_logic_vector( N_bits+3 downto 0);
 signal PI14 : std_logic_vector( N_bits+3 downto 0);
 signal PI15 : std_logic_vector( N_bits+3 downto 0);
```

```
 signal PA1 : std_logic_vector( N_bits+3 downto 0);
 signal PA2 : std_logic_vector( N_bits+3 downto 0);
 signal PA3 : std_logic_vector( N_bits+3 downto 0);
 signal PA4 : std_logic_vector( N_bits+3 downto 0);
 signal PA5 : std_logic_vector( N_bits+3 downto 0);
 signal PA6 : std_logic_vector( N_bits+3 downto 0);
 signal PA7 : std_logic_vector( N_bits+3 downto 0);
 signal PA8 : std_logic_vector( N_bits+3 downto 0);
 signal PA9 : std_logic_vector( N_bits+3 downto 0);
 signal PA10 : std_logic_vector( N_bits+3 downto 0);
 signal PA11 : std_logic_vector( N_bits+3 downto 0);
 signal PA12 : std_logic_vector( N_bits+3 downto 0);
 signal PA13 : std_logic_vector( N_bits+3 downto 0);
 signal PA14 : std_logic_vector( N_bits+3 downto 0);
 signal PA15 : std_logic_vector( N_bits+3 downto 0);

 signal PI8_1 : std_logic_vector( N_bits+4 downto 0);
signal S1 : std_logic;

begin
   S1 <= PENTE( N_bits - 1); -- bit de signe
-- Multiplication par 1
     PI1 <=  S1&S1&S1&S1 & PENTE;
-- Multiplication par 2 = 2*P
     PI2 <= S1&S1&S1 & PENTE & '0';
-- Multiplication par 3
     PI3 <= PI1 + PI2 ;
-- Multiplication par 4
     PI4 <=  S1&S1 & PENTE & "00";
-- Multiplication par 5
     PI5 <= PI1 + PI4;
-- Multiplication par 8 = 8P
    PI8  <= S1 & PENTE & "000";
-- Multiplication par 9
     PI9 <= PI8 + PI1;

registres : process(CLK,RESET)
   begin
      if (RESET = '1') then
         PA1 <= ( others => '0');
         PA2 <= ( others => '0');
         PA3 <= ( others => '0');
         PA4 <= ( others => '0');
         PA5 <= ( others => '0');
         PA8 <= ( others => '0');
         PA9 <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         PA1 <= PI1;
         PA2 <= PI2;
         PA3 <= PI3;
         PA4 <= PI4;
         PA5 <= PI5;
```

```
            PA8 <= PI8;
            PA9 <= PI9;
         end if;
 end process;


-- Multiplication par 6 = 2 * P3
     PI6 <= PA3 + PA3 ;
-- Multiplication par 7
     PI7 <= PI6 + PA1;
-- Multiplication par 10 = 2*P5
     PI10 <= PA5 + PA5;
-- Multiplication par 11
     PI11 <= PI10 + PA1;
-- Multiplication par 12
      PI12 <= PI6 + PI6;
-- Multiplication par 13
     PI13 <= PI12 + PA1;
-- Multiplication par 14
    PI14 <= PI7 + PI7;
-- Multiplication par 15
     PI15 <= (PA3 + PA3) + PA9;

registr_01 : process(CLK,RESET)
   begin
      if (RESET = '1') then
         P1 <= ( others => '0');
         P2 <= ( others => '0');
         P3 <= ( others => '0');
         P4 <= ( others => '0');
         P5 <= ( others => '0');
         P6 <= ( others => '0');
         P7 <= ( others => '0');
         P8 <= ( others => '0');
         P9 <= ( others => '0');
         P10 <= ( others => '0');
         P11 <= ( others => '0');
         P12 <= ( others => '0');
         P13 <= ( others => '0');
         P14 <= ( others => '0');
         P15 <= ( others => '0');
      elsif (CLK' event and CLK = '1') then
         P1  <= PA1;
         P2  <= PA2;
         P3  <= PA3;
         P4  <= PA4;
         P5  <= PA5;
         P8  <= PA8;
         P9  <= PA9;
         P6  <= PI6;
         P7  <= PI7;
         P10 <= PI10;
```

```
            P11 <= PI11;
            P12 <= PI12;
            P13 <= PI13;
            P14 <= PI14;
            P15 <= PI15;
        end if;
    end process;

end mult_16_arch;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

--------------------------------------------------------------------
--
-- Fichier : ram_y0y1y2.vhd
--
-- Localise les deux points M0 et M1
--
--------------------------------------------------------------------

entity ram_y0y1 is
  port ( CLK,RESET : in std_logic;
       VAL       : in std_logic;
       P_END     : in std_logic;
       DATA      : in std_logic_vector(7 downto 0);
       NUMERO    : in std_logic_vector(4 downto 0);
         Y0,Y1,Y2  : out std_logic_vector(7 downto 0)
       );
end ram_y0y1;
architecture comport of ram_y0y1 is
signal DATA1, DATA2 : std_logic_vector(7 downto 0);

component ram_Y0
  port ( CLK,RESET : in std_logic;
       VAL       : in std_logic;
       P_END     : in std_logic;
       DATA      : in  std_logic_vector(7 downto 0);
       NUMERO    : in  std_logic_vector(4 downto 0);
       Y0        : out std_logic_vector(7 downto 0)
       );
end component;

begin

Z_RAM0 : ram_y0  port map(CLK,RESET,VAL,P_END,DATA2,NUMERO,Y0);
Z_RAM1 : ram_y0  port map(CLK,RESET,VAL,P_END,DATA1,NUMERO,Y1);
Z_RAM2 : ram_y0  port map(CLK,RESET,VAL,P_END,DATA,NUMERO,Y2);

  -- registre de synchronisation
  reg0 : process(CLK,RESET)
       begin
         if (RESET = '1') then
            DATA1    <= (others => '0');
            DATA2 <= (others => '0');
        elsif ( CLk'  event and CLK = '1') then
            DATA1 <= DATA;
            DATA2 <= DATA1;
        end if;
  end process;
end comport;
```

# ANNEXE D

# MÉMOIRES SYNTHETISÉES

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

**-- Fichier : ram_32xn.vhd**

```
entity ram_32xn is
generic (Nbit : integer := 8);

port (  CLK         : in std_logic;
        DATA_IN     : in  std_logic_vector(Nbit-1 downto 0);
        ADRESS      : in  std_logic_vector(4 downto 0);
        EN          : in  std_logic;
        DATA_OUT    : out std_logic_vector(Nbit-1 downto 0));
end ram_32xn;


architecture comport of ram_32xn is

   component RAM32X1S
     port (D,A4,A3,A2,A1,A0,WE,WCLK : in std_logic;
           O : out std_logic);
   end component;

   signal DATA_OUTI : std_logic_vector(Nbit-1 downto 0);

begin

   DATA_OUT <= DATA_OUTI;

   RAM : for i in 0 to Nbit-1 generate
      RAM32x1 : RAM32X1S port map(DATA_IN(i),ADRESS(4),ADRESS(3),
                ADRESS(2),ADRESS(1),ADRESS(0),EN,CLK,DATA_OUTI(i));
   end generate;
end comport;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

**-- Fichier : ra_y0.vhd**

```
entity ra_y0 is
generic (Nbit : integer := 8);
port (  CLK         : in std_logic;
        DATA_IN     : in  std_logic_vector(Nbit-1 downto 0);
        ADRESS_1    : in  std_logic_vector(4 downto 0);
        ADRESS_2    : in  std_logic_vector(4 downto 0);
        EN          : in  std_logic;
        DATA_OUT        : out std_logic_vector(Nbit-1 downto 0));
end ra_y0;


architecture comport of ra_y0 is

  signal DATA_OUT1,DATA_OUT2 : std_logic_vector(3 downto 0);
  signal DATA_IN1,DATA_IN2   : std_logic_vector(3 downto 0);

  component ram_32xn
    generic (Nbit : integer := 8);
    port (  CLK      : in std_logic;
            DATA_IN  : in  std_logic_vector(Nbit-1 downto 0);
            ADRESS   : in  std_logic_vector(4 downto 0);
            EN       : in  std_logic;
            DATA_OUT : out std_logic_vector(Nbit-1 downto 0));
  end component;


begin

     DATA_IN1 <= DATA_IN(3 downto 0);
     DATA_IN2 <= DATA_IN(7 downto 4);

     RAMA_1 : ram_32xn generic map (4)
              port map(CLK,DATA_IN1,ADRESS_1,EN,DATA_OUT1);
     RAMA_2 : ram_32xn generic map (4)
              port map(CLK,DATA_IN2,ADRESS_2,EN,DATA_OUT2);

     DATA_OUT <= DATA_OUT2 & DATA_OUT1;

end comport;
```

# ANNEXE E

# TEST BENCH

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- Fichier : tb_agc.vhd

entity tb_agc is
end tb_agc;

architecture Behave of tb_agc is

component agc
        port(CLK,RESET      : in  std_logic;
             DATA_PULSE     : in  std_logic_vector(7 downto 0);
             TH1, TH2       : in  std_logic_vector(7 downto 0);
             MH, ML         : in  std_logic_vector(1 downto 0);
             ECART          : in  std_logic_vector(7 downto 0);
             GAIN           : out std_logic_vector(7 downto 0);
             P_VAL          : out std_logic;
             P_DEBUT        : out std_logic;
             P_FIN          : out std_logic;
           LARGE          : out std_logic_vector(7 downto 0);
             PTOA_RES       : out std_logic_vector(23 downto 0));
end component;


component gene_par
port( CLK, RESET : in std_logic;
      TH1, TH2   : out std_logic_vector(7 downto 0);
      MH, ML     : out std_logic_vector(1 downto 0);
      DCO        : out std_logic_vector(7 downto 0);
      IPAD_OUT   : out std_logic_vector(7 downto 0));
end component;


signal CLK,RESET      : std_logic;
signal TH1, TH2       : std_logic_vector(7 downto 0);
signal MH, ML         : std_logic_vector(1 downto 0);
signal ECART_F        : std_logic_vector(7 downto 0);
signal GAIN_F         : std_logic_vector(7 downto 0);
signal PTOA_RES_F     : std_logic_vector(23 downto 0);
signal P_DEBUT_F      : std_logic;
signal P_FIN_F        : std_logic;
signal P_VAL_F        : std_logic;
signal LARGE_F        : std_logic_vector(7 downto 0);
signal DATA_P         : std_logic_vector(7 downto 0);

signal DATA1,DATA2,DATA3,DATA4,DATA5 : std_logic_vector(7 downto 0);
signal DATA6,DATA7,DATA8,DATA9 : std_logic_vector(7 downto 0);
signal P_VAL1,P_VAL2,P_VAL3,P_VAL4,P_VAL5  : std_logic;
signal P_VAL6,P_VAL7,P_VAL8,P_VAL9,P_VAL10 : std_logic;

begin
```

```
 a1 : agc port map (CLK, RESET, DATA_P, TH1, TH2,MH,ML,ECART_F,
      GAIN_F,P_VAL_F,P_DEBUT_F,P_FIN_F,LARGE_F,PTOA_RES_F);

 b1 : gene_par port map (CLK,RESET,TH1,TH2,MH,ML,ECART_F,DATA_P);

vv : process
begin
   clk <= '1'; wait for 10 ns;
   clk <= '0'; wait for 10 ns;
end process;

v1 : process
begin
   RESET <= '1';
   wait for 60 ns;
   reset <= '0' ;
   wait for 9000 ns;
   wait;
end process;

v2 : process(clk,RESET)
begin
if (RESET = '1') then
    DATA1 <= ( others => '0'); P_VAL1 <= '0'; P_VAL2 <= '0';
    DATA2 <= ( others => '0'); P_VAL3 <= '0'; P_VAL4 <= '0';
    DATA3 <= ( others => '0'); P_VAL5 <= '0'; P_VAL6 <= '0';
    DATA4 <= ( others => '0'); P_VAL7 <= '0'; P_VAL8 <= '0';
    DATA5 <= ( others => '0'); P_VAL9 <= '0';
    DATA6 <= ( others => '0');
    DATA7 <= ( others => '0');
    DATA7 <= ( others => '0');
    DATA8 <= ( others => '0');
 elsif ( CLK' event and CLK = '1')  then
    DATA1 <= DATA_P;P_VAL1 <= P_VAL_F; P_VAL2 <= P_VAL1;
    DATA2 <= DATA1; P_VAL3 <= P_VAL2 ; P_VAL4 <= P_VAL3;
    DATA3 <= DATA2; P_VAL5 <= P_VAL4 ; P_VAL6 <= P_VAL5;
    DATA4 <= DATA3; P_VAL7 <= P_VAL6 ; P_VAL8 <= P_VAL7;
    DATA5 <= DATA4; P_VAL9 <= P_VAL8 ;
    DATA6 <= DATA5;
    DATA7 <= DATA6;
    DATA8 <= DATA7;
 end if;
end process;
end;

configuration ctb_agc of tb_agc is
  for Behave
  end for;
end ctb_agc;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
USE IEEE.std_logic_arith.all;
```

**-- Fichier : gene_par.vhd**

```
entity gene_par is
port( CLK, RESET : in std_logic;
      TH1, TH2   : out std_logic_vector(7 downto 0);
      MH, ML     : out std_logic_vector(1 downto 0);
      DCO        : out std_logic_vector(7 downto 0);
      IPAD_OUT   : out std_logic_vector(7 downto 0));
end gene_par;

architecture comport of gene_par is

  signal ADDR     : std_logic_vector(7 downto 0);
  signal AADD     : std_logic_vector(1 downto 0);
  signal MHQ, MLQ : std_logic_vector(1 downto 0);

begin

   TH1  <=  "00001010";
   TH2  <=  "00011110";
   DCO  <=  "00000111";
   AADD <=  ADDR(7 downto 6);

-- *****************     genere une ROM et son contenu
*******************
 -- 01 05 0E 1C 21 32 24 06 05 01          IMPULSION I
 -- 0A 1B 1C 1D 23 1B 05 03 02 00          IMPULSION II
 -- 08 28 3C 50 50 5A 1C 0E 08 03          IMPULSION III
 -- 00 00
process(ADDR)
  subtype WORD is std_logic_vector(7 downto 0);
  TYPE ROM_TBL is array (0 to 31) of WORD;

 constant ROM_VAL : ROM_TBL := ROM_TBL'(
 "00000001" , "00000101" , "00001111" , "00011010" , "00100001" ,
 "00110010" , "00100100" , "00000110" , "00000101" , "00000001" ,
 "00001010" , "00011011" , "00011100" , "00011101" , "00100011" ,
 "00011011" , "00000101" , "00000011" , "00000010" , "00000000" ,
 "00001000" , "00101000" , "00111100" , "01010000" , "01010000" ,
 "01011010" , "00011100" , "00001111" , "00001000" , "00000011" ,
 "00000000" , "00000000");

begin
   IPAD_OUT <= ROM_VAL(conv_integer(unsigned(ADDR(4 downto 0))));
end process;
```

```
--
*************************************************************************

-- ********** compteur generant les adresses pour lire la ROM *********
process(RESET, CLK)
  begin
    if (RESET = '1') then
       ADDR <= "00000000";
     elsif (CLK'event and CLK = '1') then
       ADDR <= ADDR + "00000001";
    end if;
end process;
--
*************************************************************************


-- ************      genere Les parametres       **********************
process(CLK)
begin
case AADD is
 when "00" =>
   MHQ  <=  "11";
   MLQ  <=  "10";
 when "01" =>
   MHQ  <=  "10";
   MLQ  <=  "10";
 when "10" =>
   MHQ  <=  "01";
   MLQ  <=  "10";
 when others =>
   MHQ  <=  "00";
   MLQ  <=  "10";
end case;
end process;
--
*************************************************************************


-- ***********     synchronise les parametres   *********************
process(CLK,RESET)
begin
 if (RESET = '1') then
   MH <= (others => '0');
   ML <= (others => '0');
 elsif ( CLK' event and CLK = '1') then
   MH <= MHQ;
   ML <= MLQ;
 end if;
end process;
--
end comport;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
USE IEEE.std_logic_arith.all;
```

**-- Fichier : gene_par2.vhd**

```
entity gene_par is
port( CLK, RESET : in std_logic;
      TH1, TH2   : out std_logic_vector(7 downto 0);
      MH, ML     : out std_logic_vector(1 downto 0);
      DCO        : out std_logic_vector(7 downto 0);
      IPAD_OUT   : out std_logic_vector(7 downto 0));
end gene_par;

architecture comport of gene_par is

  signal ADDR     : std_logic_vector(7 downto 0);
  signal AADD     : std_logic_vector(1 downto 0);
  signal MHQ, MLQ : std_logic_vector(1 downto 0);

begin

   TH1 <=  "00011001";
   TH2 <=  "01001011";
   DCO <=  "00110010";
   AADD <=  ADDR(7 downto 6);
-- v = [5  9 43 74 100 80 90 66 23  1  0  0  12 33 76 90 85 50 82
--      1   0  0 68 80 85 97 76 10  15  3  1  0]

-- *********     genere une ROM et son contenu  ************************
--     5     9    43    74   100    80
--    90    66    23     1     0     0
--    12    33    76    90    85    50
--    82     1     0     0    68    80
--    85    97    76    10    15     3
--    1     0
process(ADDR)
  subtype WORD is std_logic_vector(7 downto 0);
  TYPE ROM_TBL is array (0 to 31) of WORD;
 constant ROM_VAL : ROM_TBL := ROM_TBL'(
 "00000000","00000101","00001001","00101011","01001010",
 "01100100","01010000","01011010","01000010","00010111",
 "00000001","00000000","00000000","00001100","00100001",
 "01001100","01011010","01010101","00110010","01010010",
 "00000001","00000000","00000000","01000100","01010000",
 "01010101","01100001","01001100","00001010","00001111",
 "00000011","00000001" );

begin
   IPAD_OUT <= ROM_VAL(conv_integer(unsigned(ADDR(4 downto 0))));
end process;
-- ********************************************************************
```

```
-- ********** compteur generant les adresses pour lire la ROM *********
process(RESET, CLK)
  begin
    if (RESET = '1') then
       ADDR <= "00000000";
     elsif (CLK'event and CLK = '1') then
       ADDR <= ADDR + "00000001";
    end if;
end process;
--
*****************************************************************************


-- ************      genere Les parametres       **********************
process(CLK)
begin
case AADD is
 when "00" =>
   MHQ  <=  "11";
   MLQ  <=  "10";
 when "01" =>
   MHQ  <=  "10";
   MLQ  <=  "10";
 when "10" =>
   MHQ  <=  "01";
   MLQ  <=  "10";
 when others =>
   MHQ  <=  "10";
   MLQ  <=  "10";
end case;
end process;
--
*****************************************************************************


-- ***********     synchronise les parametres   ***********************
process(CLK,RESET)
begin
 if (RESET = '1') then
   MH <= (others => '0');
   ML <= (others => '0');
 elsif ( CLK' event and CLK = '1') then
   MH <= MHQ;
   ML <= MLQ;
 end if;
end process;
--
*****************************************************************************


end comport;
```