

# Synthèse d'architecture sur FPGA sous contrainte de précision des calculs

Nicolas HERVÉ, Daniel MÉNARD et Olivier SENTIEYS

IRISA – Université de Rennes 1  
6, rue de Kérampont  
22300 Lannion, France  
{prenom}.{nom}@irisa.fr

---

## Résumé

Les FPGAs sont de plus en plus considérés comme une solution incontournable pour les applications de traitement du signal. Dans le même temps, les contraintes telles que le coût, la consommation et le temps de mise sur le marché des applications de traitement du signal exigent la mise en œuvre de méthodologies d'implantation automatique d'algorithmes spécifiés en virgule flottante au sein de FPGAs utilisant l'arithmétique virgule fixe. Dans cet article, une nouvelle méthodologie de synthèse d'architecture sous contrainte de précision est présentée. Cette approche s'appuie sur une bibliothèque d'opérateurs arithmétiques virgule fixe caractérisant le coût des opérateurs en fonction de leur largeur. Pour obtenir une implantation efficace, le processus d'optimisation est couplé avec la synthèse d'architecture. De plus, l'évaluation de la précision est réalisée par une approche analytique permettant ainsi d'obtenir des temps d'optimisation raisonnables.

**Mots-clés :** Arithmétique virgule fixe, précision finie, FPGA, Synthèse d'architecture, outils de CAO

---

## 1. Introduction

Les circuits reconfigurables, en particulier les FPGA, deviennent de plus en plus une cible architecturale privilégiée pour le développement des applications de multimédia ou de télécommunications basées sur le traitement numérique du signal. Leur capacité à s'adapter aux évolutions algorithmiques, conjuguée à des performances en constante évolution (quelques millions de portes équivalentes, capacité mémoire importante, opérateurs arithmétiques câblés, processeurs enfouis) leur permet de surpasser les processeurs généralistes hautes performances. Le problème principal freinant leur utilisation intensive est l'absence d'outils de développement de haut niveau permettant de cibler ces architectures à partir de langages de haut niveau.

L'implantation efficace des algorithmes de traitement numérique du signal dans les systèmes embarqués requiert l'utilisation de l'arithmétique virgule fixe afin de satisfaire les contraintes de coût, de consommation et d'encombrement exigées par ces applications. En effet, la largeur des opérateurs, des bus et des mémoires au sein des architectures virgule fixe étant plus faible, le prix et la consommation d'énergie de ces architectures sont moins importants. De plus, les opérateurs en virgule flottante doivent manipuler la mantisse et l'exposant associés à chaque donnée. En conséquence, ces opérateurs sont plus complexes par rapport à ceux en virgule fixe et conduisent à des temps d'exécution des opérations plus importants. Pour une implantation matérielle, le passage en virgule fixe de l'application est une tâche fastidieuse et source d'erreurs. Ce processus d'implantation matérielle nécessite de définir complètement l'architecture. Pour la partie opérative, la synthèse doit permettre de déterminer le nombre d'opérateurs à utiliser, mais également la largeur de ces opérateurs. Pour des architectures relativement complexes, l'espace de recherche des largeurs des opérateurs étant important, une exploration de cet espace manuellement n'est plus envisageable. La réduction du temps de mise sur le marché des applications exige donc l'utilisation d'outils de développement de haut niveau, permettant d'automatiser la synthèse de l'architecture en virgule fixe.

L'objectif du processus de conversion en virgule fixe est de déterminer, pour chaque donnée, la largeur et la position de la virgule. La première étape du processus de conversion correspond à l'évaluation de la dynamique des données. Ces résultats sont utilisés dans la seconde étape pour définir la position de la virgule pour chaque donnée. L'implantation efficace d'un algorithme au sein d'une plate-forme matérielle (ASIC, FPGA) nécessite de minimiser la surface et la consommation d'énergie. Ainsi, l'objectif de la troisième étape est de fixer la largeur des données en minimisant la surface de l'architecture tant que la contrainte de précision associée à l'application est satisfaite. Alors que les deux premières étapes du processus de conversion peuvent être déterminées une fois pour toute, l'étape de minimisation de la surface doit être effectuée par raffinements successifs. Pour obtenir une implantation efficace, le processus d'optimisation doit être couplé avec la synthèse d'architecture [6, 14]. Ceci est illustré par la figure 1 présentant le paradoxe entre la synthèse et l'optimisation de la largeur des données. D'une part, la synthèse d'architecture requiert la connaissance de la largeur des opérateurs pour pouvoir effectuer correctement les phases de sélection, d'ordonnancement et d'assignation [4]. D'autre part, cette optimisation nécessite de connaître l'assignation des ressources et l'ordonnancement des opérations pour pouvoir assigner efficacement les opérations sur les opérateurs. Ceci est un des problèmes cruciaux de la synthèse d'architecture en précision multiple.

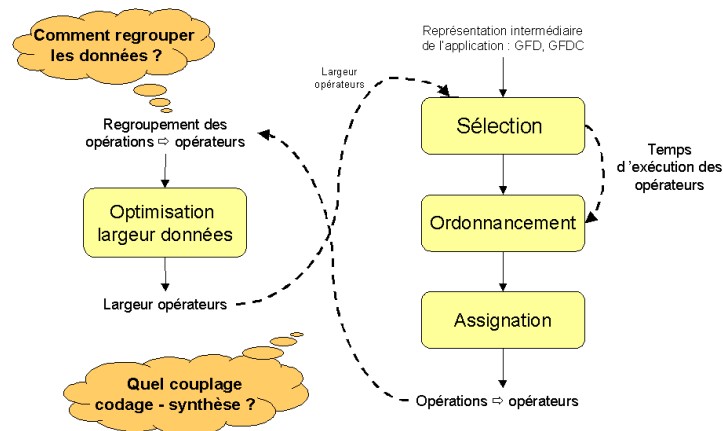


FIG. 1 – Le paradoxe optimisation-synthèse : l'optimisation nécessite de connaître le groupement des opérations obtenu par la synthèse d'architecture et la synthèse d'architecture requiert la connaissance des largeurs des opérateurs obtenues par le processus d'optimisation.

La majorité des méthodes proposées dans la littérature ne réalise pas de couplage entre les processus de synthèse d'architecture et d'optimisation de la largeur des données. Ceci est nécessaire pour obtenir une spécification virgule fixe réellement optimisée pour la contrainte de précision spécifiée. De plus, de nombreuses méthodes utilisent une technique d'évaluation de la précision uniquement basée sur la simulation, conduisant à des temps de simulation prohibitifs.

Dans cet article, une nouvelle méthodologie d'implantation automatique d'algorithmes spécifiés en virgule flottante au sein de FPGA utilisant l'arithmétique virgule fixe est proposée. Un processus itératif de synthèse d'architecture et d'optimisation de la largeur des données est mis en œuvre afin de coupler ces deux processus. De plus, l'évaluation de la précision est réalisée par une approche analytique permettant d'obtenir des temps d'optimisation raisonnables par rapport aux approches basées sur la simulation. Ce travail s'inscrit dans le cadre du projet RNTL OSGAR (Outils de Synthèse Génériques d'Architectures Reconfigurables). Cet article est organisé comme suit. La section 2 présente l'état de l'art des méthodologies existantes. Dans la section 3, le flot de la méthodologie proposée est décrit. La section 4 détaille le processus d'optimisation de la largeur des données. Finalement, la section 5 présente un exemple d'application et les résultats obtenus.

## 2. État de l'art des méthodologies de conversion en virgule fixe

### 2.1. Evaluation de la dynamique des données

Deux types de méthode peuvent être utilisés pour déterminer le domaine de définition des différentes données présentes au sein d'une application. La dynamique des données peut être estimée à partir des paramètres statistiques de chaque donnée obtenus par simulation de l'algorithme en virgule flottante [5]. Ces méthodes statistiques permettent d'estimer relativement précisément la dynamique des données en utilisant les informations liées aux caractéristiques du signal. Ces méthodes garantissent une probabilité de débordement très faible pour les signaux dont les paramètres statistiques sont proches de ceux des signaux d'entrée utilisés pour l'estimation. Cependant, l'absence de débordement n'est pas garantie pour des signaux ayant des caractéristiques nettement différentes. L'approche analytique [14, 2] représente l'alternative aux méthodes basées sur la simulation. Ces méthodes garantissent l'absence de débordement. En contrepartie, elles conduisent à des estimations souvent pessimistes car elles sont basées sur une analyse dans le cas le plus défavorable.

### 2.2. Evaluation de la précision

La métrique la plus utilisée pour évaluer la précision d'une implantation en virgule fixe est le Rapport Signal à Bruit de Quantification (RSBQ). Deux types de méthode peuvent être utilisés pour déterminer ce RSBQ. Les méthodes basées sur la simulation estiment la puissance du bruit de quantification de manière statistique à partir des signaux obtenus après simulation de l'algorithme en virgule fixe et en virgule flottante. Le signal obtenu en virgule flottante est considéré comme la référence, car l'erreur associée à ce codage est négligeable devant celle obtenue en virgule fixe.

La simulation de l'algorithme en virgule fixe nécessite d'émuler l'ensemble des mécanismes de l'arithmétique virgule fixe. En conséquence, le temps de simulation est nettement augmenté par rapport à une simulation traditionnelle en virgule flottante. De plus pour obtenir une estimation précise, il est nécessaire d'utiliser un nombre d'échantillons en entrée important. La combinaison de ces deux phénomènes conduit à des temps de simulation élevés. La conception matérielle nécessite d'optimiser la largeur des données sous contrainte de précision et ainsi d'explorer l'espace de conception des différentes largeurs de données. L'évaluation de la précision basée sur la simulation aboutit à un processus itératif nécessitant de nombreuses simulations. Ainsi pour les systèmes complexes, le temps d'optimisation devient prohibitif et des heuristiques doivent être utilisées afin de limiter l'espace de recherche.

Dans [7], une méthode de calcul analytique du bruit de quantification est proposée afin d'éviter une phase de simulation nécessitant des puissances de calcul élevées. Cette méthode est basée sur la propagation de modèles de bruit au sein de la description de l'algorithme. Afin d'obtenir des expressions des bruits générés et propagés indépendantes de la statistique des signaux utilisés en entrée des opérateurs, l'auteur a posé plusieurs hypothèses. Les variables en entrée de chaque opérateur sont considérées comme indépendantes et centrées. Les bruits générés lors d'un changement de format sont assimilés à des variables aléatoires centrées. Ainsi, ce modèle n'est valide que pour les opérateurs utilisant une loi de quantification par arrondi. Par défaut, une loi de quantification par troncature est utilisée car elle ne nécessite pas de matériel supplémentaire. L'utilisation d'une technique de propagation d'un modèle de bruit ne permet pas de traiter les graphes possédant des cycles. Ainsi, la méthode ne peut pas être appliquée aux systèmes récursifs (la sortie du système dépend des échantillons de sortie précédents).

### 2.3. Optimisation de la largeur des données

L'approche classique utilisée pour optimiser la largeur des données consiste à coder toutes les données dans un format unique [7]. Ceci réduit l'espace de recherche à une seule dimension et simplifie la synthèse car toutes les opérations s'exécuteront sur des opérateurs de même largeur. Cependant, dans le but d'obtenir une implantation efficace d'un algorithme de traitement du signal en virgule fixe, tout en satisfaisant la contrainte de précision de calcul imposée par l'environnement, il est nécessaire de considérer un format propre à chaque signal [2].

La méthodologie proposée par Constantinides et al [2] se compose de deux étapes principales. La première étape permet d'obtenir une spécification virgule fixe respectant une contrainte de précision. Pour cette phase, l'hypothèse d'une ressource dédiée à chaque opération est utilisée. Ainsi, la synthèse d'architecture

n'est pas considérée dans un premier temps dans le sens où le partage de ressources n'est pas géré.

Le modèle de coût pour le FPGA correspond à la surface utilisée en termes de cellules dans le cas d'une implémentation directe du graphe flot de signal (GFS). La multiplication par une constante est modélisée comme une succession d'additions, mais la valeur de cette constante n'est pas prise en compte. L'optimisation de la largeur des données est réalisée par un algorithme de programmation linéaire. Une solution sous-optimale peut être obtenue par une heuristique gloutonne. La méthode est purement analytique et ne traite que les systèmes linéaires et non variant dans le temps.

La seconde phase du processus correspond à la synthèse de l'architecture. Dans [1] une heuristique pour combiner l'ordonnancement, le partage de ressources est proposée dans le cas d'un graphe flot de signal intégrant des opérations de longueurs différentes tel que celui obtenu dans la première étape. Tout d'abord un ordonnancement avec des informations de largeur de données incomplètes est réalisé, ensuite le partage de ressources et la sélection sont combinés. Ce processus va affecter des opérations sur des opérateurs de largeur plus importante afin d'utiliser au maximum les ressources présentes.

Cette méthodologie réalise l'implantation d'une spécification virgule fixe de l'application conduisant à une précision des calculs supérieure ou égale à la contrainte de précision. Certes l'optimisation de la largeur des données dans la première phase est réalisée sous contrainte de précision, mais la seconde phase affecte des opérations sur des opérateurs de largeur plus importante et se traduit par une augmentation de la précision des calculs. En conséquence, la solution obtenue n'est pas optimisée exactement pour la contrainte de précision.

Dans [6], les auteurs proposent une méthode pour laquelle, la synthèse d'architecture est effectuée entre deux phases d'optimisation de la largeur des données. Le partage de ressources est pris en compte pour réduire à la fois le coût matériel et le temps d'optimisation. En effet, l'évaluation de la précision étant réalisée par simulation il est nécessaire de réduire l'espace de recherche par l'utilisation d'heuristiques afin d'avoir des temps d'optimisation raisonnables. Une première étape analyse le graphe flot de signal de l'application pour former des groupes de données. Une même largeur sera assignée aux données d'un même groupe. Ainsi, par exemple, les entrées d'additions successives pourront former un seul groupe. La seconde étape détermine la largeur de données minimale requise pour chaque partition. Ensuite, cette spécification virgule fixe est ordonnancée et assignée en utilisant la combinaison des largeurs de données minimales précédemment trouvée. Lors de l'ordonnancement, des opérations peuvent être affectées à des opérateurs de largeur supérieure. La dernière étape recherche la largeur des opérateurs optimale.

Le processus de synthèse et d'optimisation de la largeur des données doit être itératif et se terminer par une synthèse permettant d'implanter exactement la spécification virgule fixe optimisée pour la contrainte de précision donnée. En effet, la dernière étape de la méthodologie proposée par [6] optimise la largeur des opérateurs, mais ce processus peut remettre en cause les phases d'ordonnancement réalisées dans l'étape précédente.

Dans cet article, une nouvelle méthodologie de synthèse d'architecture sous contrainte de précision est proposée. Pour coupler les phases de synthèse d'architecture et d'optimisation de la largeur des données, un processus itératif est utilisé. De plus, l'évaluation de la précision est réalisée par une approche analytique conduisant à des temps d'optimisation raisonnables.

### 3. Arithmétique et synthèse sur FPGA

Pour des raisons de surface et de temps de propagation il est généralement préférable d'utiliser l'arithmétique virgule fixe sur un FPGA. Il existe cependant des travaux prônant l'utilisation d'une arithmétique virgule flottante [3]. Dans cette étude, uniquement la mise en œuvre de l'arithmétique virgule fixe est traitée.

La synthèse d'architecture s'appuie sur une bibliothèque d'opérateurs arithmétiques virgule fixe associée à une famille de FPGA donnée. Chaque opérateur est caractérisé en termes de surface, de temps et de consommation énergétique en fonction des largeurs des opérands d'entrée et du résultat. Ces informations sont utilisées dans les processus de synthèse d'architecture et d'optimisation des largeurs d'opérateurs. Cette caractérisation a été obtenue lors de synthèses de ces composants sur le FPGA à l'aide de l'outil Synplify Pro de Synplify [13]. Des composants d'interconnexion (tristate, multiplexeurs, de-

multiplexeurs) et de mémorisation (registres, RAM) ont également été caractérisés de la même façon. La puissance moyenne consommée par ces composants est caractérisée avec ISE/Xpower de Xilinx. Les informations telles que le nombre de ressources utilisées (cellules logiques, multiplieurs dédiés, etc.), le temps de traversée du composant ou sa consommation sont finalement sauvegardées sous la forme d'une base de données XML exploitée par notre méthodologie.

L'hypothèse, que le coût de plusieurs composants sur un FPGA est égal à la somme des coûts de chacun de ces composants est utilisée. Ainsi, l'effet du placement et routage et celui de la taille du FPGA sont négligés. Cette hypothèse est satisfaisante, car les estimations faites sont destinées à l'optimisation du coût global de tous ces composants et n'ont donc pas à prendre en compte ces autres critères.

Les résultats obtenus lors de la synthèse de l'opérateur de multiplication pour les FPGA Virtex-I et Virtex-II sont présentés à la figure 2. Le Virtex-I n'intègre pas de multiplieurs dédiés, ainsi, l'opérateur est implanté sous forme de LUT (Look-Up Table). La figure 2.a montre l'évolution du temps de propagation de l'opérateur en fonction de la largeur des entrées. Cette courbe présente des paliers montrant que la réduction d'un bit permet de gagner jusqu'à 2 ns. Ceci montre à quel point les processus de synthèse et de détermination des largeurs des opérateurs doivent être liés.

Les FPGAs récents, tels que le Virtex-II ou le Stratix, contiennent des ressources dédiées (multiplieurs, mémoire, chemins de données). Ces ressources doivent donc être intégrées dans le modèle de coût. Plutôt que d'essayer d'uniformiser le tout avec une seule valeur de coût, nous faisons le choix de compter ces ressources à part. De même que l'utilisateur pourra spécifier une borne pour le nombre de LUTs, il pourra autoriser un certain nombre pour chaque type de ressources dédiées et spécifier une borne pour le nombre de LUTs suivant le nombre de ressources dédiées effectivement utilisées.

Les résultats obtenus pour le Virtex-II sont présentés aux figures 2.b, 2.c, 2.d. Ce composant intègre des multiplieurs  $18 \times 18$ . Dans certains cas, l'outil de synthèse préfère ajouter des LUTs pour économiser le nombre de multiplieurs dédiés utilisés. Ces cas particuliers peuvent conduire notre processus d'optimisation à éviter d'utiliser ces largeurs.

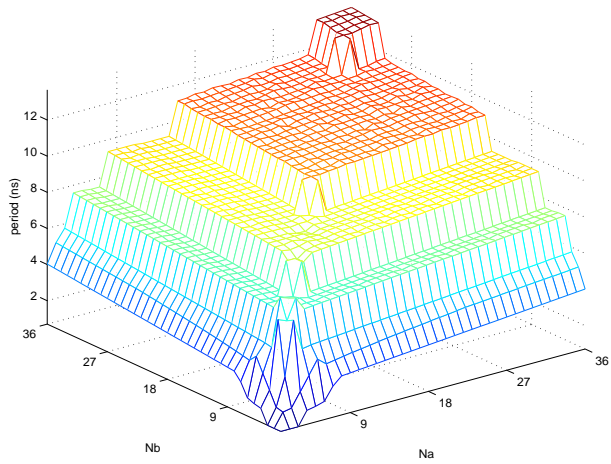
#### 4. Méthodologie de synthèse sous contrainte de précision

La méthodologie d'implantation automatique d'algorithmes spécifiés en virgule flottante au sein de FPGA utilisant l'arithmétique virgule fixe est détaillée à la figure 3. Cette approche se base sur la définition d'un environnement permettant d'effectuer conjointement la synthèse d'architecture et l'optimisation de la largeur des opérateurs. La méthodologie proposée se décompose en deux étapes distinctes. Tout d'abord, la position de la virgule associée à chaque donnée est déterminée. Cette étape doit permettre de définir pour chaque donnée la largeur de la partie entière permettant de représenter les données garantissant l'absence de débordement. Dans ce cadre, la dynamique de chaque donnée est évaluée et ces résultats sont utilisés pour définir la position de la virgule. Les opérations de recadrage nécessaires à l'obtention d'une spécification virgule fixe correcte, sont insérées. Cette étape est détaillée dans la partie 4.1.

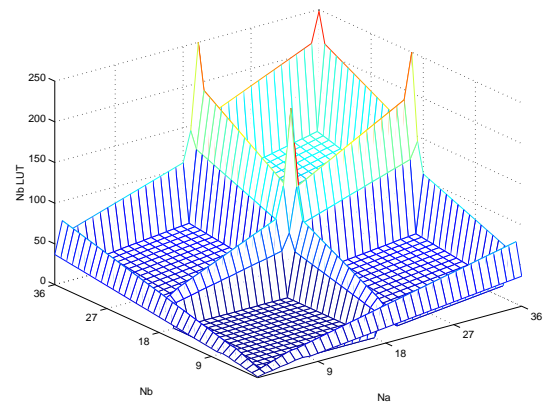
Ensuite, la synthèse de l'architecture à largeurs multiples est effectuée. Cette phase a pour objectif de minimiser la surface du circuit tant que la contrainte de précision des calculs associée à l'application est vérifiée. Pour cela différents regroupements d'opérations sur les opérateurs sont testés et les largeurs des opérateurs sont réduites. Ce processus doit évaluer la précision des calculs et la surface de la partie opérative de l'architecture pour chaque groupement et combinaison de largeur testés. Ces évaluations sont présentées dans les sections 4.2 et 4.3. La synthèse d'architecture à largeur multiple est un processus itératif. Elle fait l'objet de la section 5. Ce processus réalise successivement, le regroupement des données, l'optimisation de la largeur des groupes et la synthèse de l'architecture. Cette dernière phase est effectuée à l'aide de l'outil BSS [12].

##### 4.1. Détermination de la position de la virgule

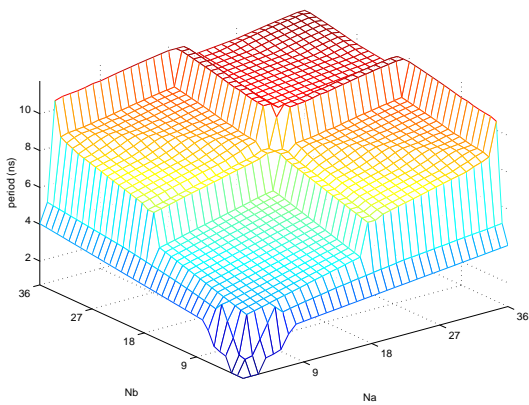
L'objectif de cette phase de la méthodologie est de déterminer la position de la virgule optimale de chaque donnée. Cette position de la virgule doit permettre de représenter toutes les valeurs prises par la donnée afin d'éviter d'éventuels débordements et minimiser le nombre de bits nécessaires pour coder la partie entière. Pour atteindre ce but, la dynamique de chaque donnée doit être évaluée avant de déterminer la position de la virgule.



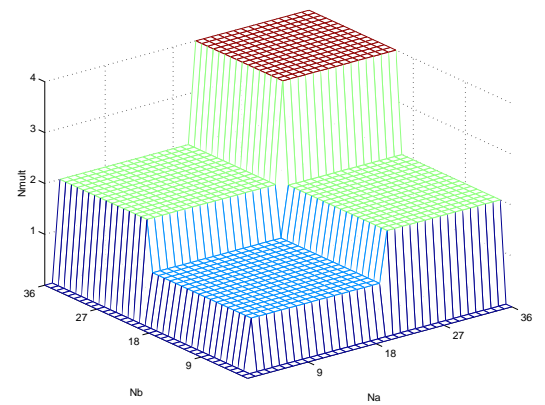
(a) - Temps de propagation pour Virtex-I



(b) - Nombre de LUTs utilisées pour le Virtex-II



(c) - Temps de propagation pour le Virtex-II



(d) - Nombre de multipliers dédiés ( $18 \times 18$ ) pour le Virtex-II

FIG. 2 – Temps de propagation et surface en fonction de la largeur des entrées pour un opérateur de multiplication.

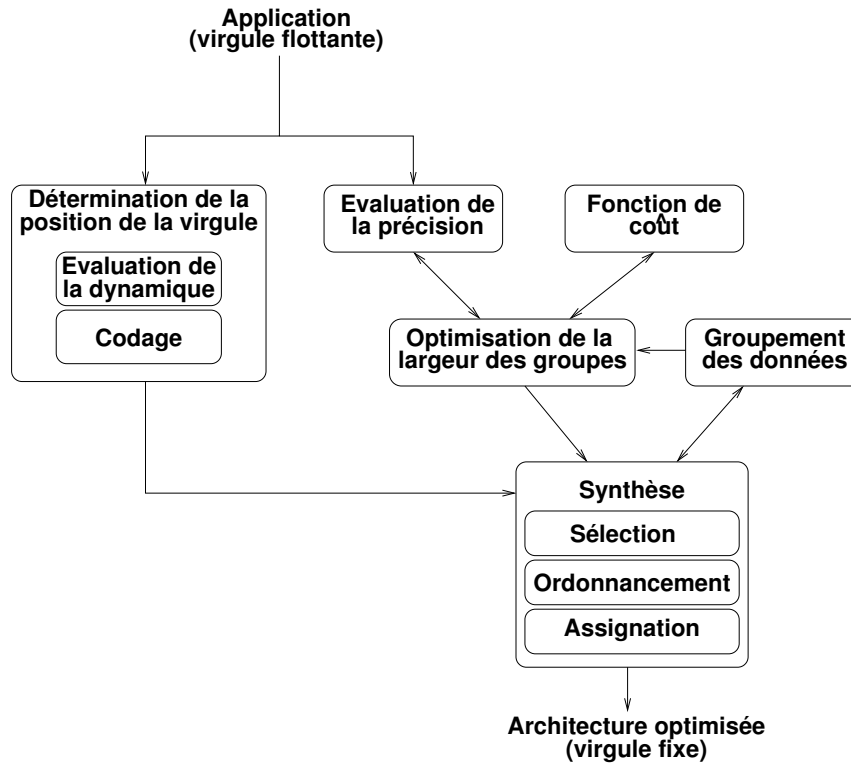


FIG. 3 – Environnement conjoint de synthèse et d'optimisation de la largeur des opérateurs

#### 4.1.1. Evaluation de la dynamique des données

Une approche analytique a été mise en œuvre pour obtenir une méthode garantissant l'absence de débordements. Cette technique permet de déterminer l'expression du domaine de définition d'une donnée à partir de celui des entrées. Pour les systèmes non-récurrents, l'arithmétique d'intervalle est utilisée. Le domaine de définition des entrées est propagé dans le graphe flot de données représentant l'application. Pour chaque opération le domaine de définition des données en sortie est déterminé en fonction de celui des opérandes en entrée selon une règle propre à chaque type d'opération. Ces règles donnent le domaine de définition dans le cas le plus défavorable. Dans le cadre des systèmes linéaires à coefficients constants, la dynamique des données est déterminée à l'aide de la norme L1 [11]. Cette norme est basée sur la connaissance de la fonction de transfert entre la donnée ciblée et les entrées. Ainsi, l'utilisation de la notion de fonction de transfert permet de traiter aussi bien les structures récurrentes que non récurrentes.

#### 4.1.2. Codage des données

L'étape précédente a permis d'évaluer la dynamique de chaque donnée. Ces résultats sont utilisés pour déterminer la position de la virgule (nombre minimal de bits requis pour coder la partie entière). L'objectif de cette phase est d'obtenir une spécification virgule fixe correcte de l'application, et garantissant l'absence de débordement. Ceci nécessite de respecter les différentes règles de l'arithmétique virgule fixe. En conséquence, des opérations de recadrage sont insérées afin d'adapter le format de codage de la donnée à sa dynamique ou pour adapter le format des entrées d'un additionneur. Pour déterminer le format des données, des règles de propagation du format sont définies pour chaque type d'opérateur. Ces règles sont appliquées lors d'un parcours ascendant du graphe représentant l'application.

#### 4.2. Evaluation de la précision

Une nouvelle méthode analytique d'évaluation de la précision a été définie afin d'obtenir des temps d'optimisation du codage des données raisonnables. Cette approche permet de déterminer automatiquement l'expression analytique de la puissance du bruit de quantification et de la puissance du signal dans le cas

des systèmes linéaires et des systèmes non-linéaires et non-récurrents.

Le bruit de quantification est modélisé par la somme pondérée des bruits issus des différentes sources de bruit. Ces différentes sources correspondent au bruit associé à chaque entrée et aux bruits générés lors d'un changement de format au sein de l'application. Les paramètres statistiques associés à une source de bruit dépendent du format de la donnée après quantification et du nombre de bits éliminés. Le gain entre la sortie et chaque source de bruit est déterminé de différentes manières en fonction du type de systèmes. Dans le cas des systèmes linéaires, le gain entre la sortie et chaque source de bruit est déterminé à partir de la fonction de transfert entre la sortie et la source [9]. La méthode utilisée pour déterminer automatiquement les différentes fonctions de transfert du système et l'outil développé pour implanter cette méthode sont présentés dans [10]. Pour les systèmes non-linéaires et non-récurrents, le gain est obtenu à partir des paramètres statistiques des différents signaux présents entre la sortie et la source de bruit [8]. Ces paramètres statistiques sont déterminés au travers d'une unique simulation en virgule flottante. Cette approche permet d'obtenir l'expression du RSBQ en fonction de la largeur des différentes données.

### 4.3. Fonction de coût

Par défaut notre fonction de coût correspond au nombre total de LUTs utilisées par les unités fonctionnelles sur le FPGA. Pour les FPGA intégrant des ressources dédiées, l'utilisateur doit fournir le nombre maximal de ressources pouvant être utilisées par l'application. Une fonction évalue pour chaque unité fonctionnelle le nombre de LUTs consommées en fonction du nombre de bits des entrées. Ces informations sont issues de la base de données obtenue lors du processus de caractérisation de la bibliothèque d'opérateurs.

Par la suite, il sera possible de modifier la fonction de coût pour prendre en compte aussi la consommation d'énergie par exemple. L'utilisateur fixera des coefficients de pondération pour indiquer l'importance relative de la consommation et de la surface dans le processus d'optimisation.

## 5. Optimisation d'architectures à largeur multiple

Le processus d'optimisation de l'architecture en fonction de la largeur des opérateurs que nous proposons est itératif. Il est décomposé en quatre étapes distinctes et permet de coupler efficacement la synthèse d'architecture et l'optimisation de la spécification en virgule fixe. Pour cette approche, la notion de groupe est définie. Un groupe rassemble les opérations d'un même type qui auront la même largeur. Pour la première itération, le point de départ correspond à une solution pour laquelle tous les opérateurs d'un même type possèdent la même largeur. Ainsi, un seul groupe est associé à chaque type d'opération arithmétique.

La première étape du processus d'optimisation permet de définir le nombre de groupes utilisés pour chaque type d'opérateur arithmétique (voir détail en section 5.1). Dans la seconde étape, un algorithme de groupement est appliqué afin d'affecter chaque opération à un groupe. La technique utilisée est présentée dans la section 5.2. La troisième étape correspond à l'optimisation de la largeur des différents groupes d'opérations. L'objectif est de minimiser la surface du circuit en respectant la contrainte de précision  $RSBQ_{min}$  tel que décrit dans l'équation 1 :

$$\min \left( S(\vec{b}) \right) \quad \text{avec} \quad RSBQ(\vec{b}) \geq RSBQ_{min} \quad (1)$$

où  $\vec{b}$  qui représente une instance de l'ensemble des largeurs  $b_i$  des données du graphe flot de signal représentant l'algorithme. Les termes  $S(\vec{b})$   $RSBQ(\vec{b})$  correspondent respectivement à la surface de l'architecture et au rapport signal à bruit de quantification obtenus pour la combinaison de largeur  $\vec{b}$  fixé.

L'évaluation de la précision est effectuée à l'aide de la méthode analytique présentée dans la section 4.2 et permettant d'obtenir l'expression analytique de  $RSBQ(\vec{b})$ . La surface de l'architecture est évaluée avec la fonction de coût présentée dans la section 4.3 et obtenue à partir de la bibliothèque d'opérateurs. L'algorithme d'optimisation est présenté dans la partie 5.2.

La quatrième étape, réalise une synthèse d'architecture avec la spécification virgule fixe obtenue à l'étape précédente. Cette synthèse va remettre en cause le nombre d'opérateurs utilisés pour chaque type d'opération. En effet, par rapport à la synthèse précédente, la largeur de certains groupes d'opérations a diminué



et ainsi leur temps d'exécution a été modifié. Cette diminution du temps d'exécution peut entraîner une réduction du nombre d'opérateurs et nécessite ainsi un processus itératif. Le nombre de groupes par type d'opérateur pour l'itération suivante correspond au nombre d'opérateurs nécessaires par type d'opérateur obtenu lors de cette phase de synthèse.

Cet algorithme a convergé lorsque plusieurs itérations successives conduisent à un résultat identique.

### 5.1. Groupement des données

Le groupement des données est réalisé à partir de l'analyse des résultats de la synthèse d'architecture. Pour chaque opération, une mobilité est calculée en effectuant un ordonnancement par liste dans le sens direct et dans le sens inverse. La mobilité d'une opération est définie comme la différence des indices du cycle d'exécution obtenus lors de l'ordonnancement par liste dans le sens directe et dans le sens inverse. Les opérations dont la mobilité est nulle sont traitées en premier. Ces opérateurs définissent la largeur initiale associée à chaque opérateur.

Le reste des opérations est traité par ordre croissant de mobilité. La mobilité est utilisée pour pouvoir associer l'opération à l'opérateur le plus adéquat. La mobilité des opérations est utilisée pour déplacer une opération à un cycle ou celle-ci utilisera l'opérateur le plus adéquat. Cette opération est placée sur l'opérateur dont la largeur est immédiatement supérieure. L'objectif de cette approche est d'obtenir pour chaque opérateur la largeur la plus faible.

Le groupement des données est effectué en considérant une largeur propre à chaque opération. Cette largeur a été obtenue dans le cas d'une optimisation de la largeur des opérateurs pour une implantation spatiale. Dans ce cas, chaque opération utilise un opérateur. Pour la première itération du processus d'optimisation, la synthèse est réalisée en considérant un seul groupe pour chaque type d'opération. La largeur associée à chaque groupe correspond à la largeur maximale des opérations de ce groupe. Pour les itérations suivantes, certaines opérations seront placées sur des opérateurs plus larges afin d'éventuellement réduire le nombre d'opérateurs utilisés.

### 5.2. Optimisation des groupements de données

Les algorithmes d'optimisation utilisés sont des procédures indépendantes qui, pour un certain groupement de données, vont rechercher une combinaison qui va minimiser la fonction de coût tout en satisfaisant la contrainte de précision. Pour chacune des combinaisons testées, la fonction d'évaluation du RSBQ et la fonction de coût sont évaluées. Leurs résultats déterminent la prochaine étape de l'algorithme d'optimisation.

La première étape consiste à déterminer la *combinaison des largeurs de groupe minimale*. Pour cela, toutes les largeurs de groupe sont réglées initialement au maximum permis. Le RSBQ doit donc être satisfait. Ensuite, pour chacun des groupes, la largeur minimale respectant le RSBQ est déterminée. Les autres groupes étant toujours réglés au maximum. Cette étape se termine en positionnant chacun des groupes à sa largeur minimale. La contrainte de précision n'est en principe alors plus satisfaite, mais cette combinaison ne sert que de point de départ pour l'algorithme. L'avantage étant qu'en partant de cette combinaison, la largeur des groupes ne va aller qu'en augmentant au fur et à mesure du déroulement de l'algorithme.

Les étapes suivantes vont consister à atteindre le RSBQ fixé tout en limitant le gain de coût. Tout d'abord, le rapport  $\Delta_{\text{RSBQ}}/\Delta_{\text{coût}}$  pour une augmentation de un bit de la largeur de groupe est calculé pour chaque groupe. La largeur dont ce rapport est le plus élevé est effectivement augmentée d'un bit. Ces opérations sont répétées jusqu'à ce que le RSBQ désiré soit atteint.

## 6. Résultats

L'application retenue pour montrer l'intérêt d'une technique d'optimisation de la largeur des opérateurs en fonction d'une contrainte de précision des calculs correspond à un filtre à réponse impulsionnelle infinie (IIR) d'ordre 8. Ce filtre est implanté sous forme de 4 filtres cascades d'ordre 2. Le graphe flot de signal de ce filtre récursif est présenté à la figure 4. Cette application est composée de 36 opérations correspondant à 20 multiplications et 16 additions.

La méthodologie présentée dans la section 4 a été utilisée pour obtenir la dynamique et la position de la

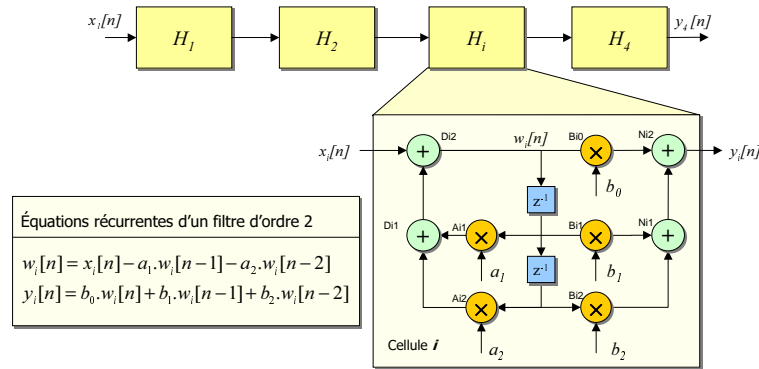


FIG. 4 – Synoptique du filtre IIR d'ordre 8 et graphe flot de signal d'une cellule d'ordre 2. Le filtre IIR d'ordre 8 est composé de 4 cellules d'ordre 2 cascadiées.

virgule de chaque donnée afin d'obtenir une spécification en virgule fixe correcte. De plus, l'expression analytique du rapport signal à bruit de quantification a été déterminée et la contrainte de précision a été fixée à 60 dB.

Tout d'abord, la largeur des différentes opérations a été optimisée en considérant une implantation spatiale. Dans ce cas, un opérateur est associé à chaque opération. Ces largeurs sont présentées pour chaque opération à la figure 5.a. Pour la première itération du processus global d'optimisation, un groupe est associé à chaque type d'opérateur et la largeur de celui-ci est fixée à la largeur maximale des opérations de ce groupe. Ainsi, les multiplications sont réalisées sur un multiplieur  $17 \times 17$  bits et les additions par un additionneur 20 bits. La synthèse d'architecture pour cette spécification virgule fixe conduit à l'ordonnancement présenté à la figure 5.a. La fréquence minimale de l'horloge du système est fixée à 200 MHz. Ainsi, la latence de chaque opérateur est un multiple de 5 ns. Pour une contrainte de temps de 70 ns, 5 multiplieurs et 2 additionneurs sont nécessaires. Suite à cette synthèse, 5 nouveaux groupes sont définis pour les multiplications et 2 pour les additions. Ces groupes, représentés à la figure 5.a, sont formés en fonction de la largeur obtenue pour une implantation spatiale et de la mobilité des opérations.

Une optimisation de la largeur des groupes sous contrainte de précision est effectuée pour ces 7 groupes d'opérations. Cette optimisation permet d'obtenir des groupes de largeur plus faible. La largeur des 5 groupes pour les multiplications est égale à 17, 16, 15, 14 et 14 bits et la largeur pour les 2 groupes d'addition est égale à 20 et 17 bits. La synthèse de l'architecture pour cette nouvelle spécification virgule fixe conduit à l'ordonnancement présenté à la figure 5.b. Les multiplieurs de largeur 14 à 16 bits ayant une latence plus faible (10 ns), quatre multiplieurs sont maintenant nécessaires. Ainsi, cette architecture utilise un multiplieur de moins que celle obtenue à la première itération. La réduction de la largeur de certains opérateurs combinée à la diminution du nombre d'opérateur, conduit à un gain au niveau de la surface des opérateurs de 35%.

La méthode proposée dans [7] réalise une architecture pour laquelle une largeur unique d'opérateur est utilisée. Ainsi, pour cette application, 5 multiplieurs  $19 \times 19$  et 2 additionneurs 19 sont utilisés. Par rapport à cette architecture, la surface des opérateurs obtenus avec notre approche est réduite de 47%. Ces résultats montrent l'intérêt d'utiliser des largeurs multiples pour les opérateurs et l'efficacité de notre approche.

## 7. Conclusion

Dans cet article, une nouvelle méthodologie de synthèse d'architecture sous contrainte de précision a été présentée. Un processus itératif d'optimisation de la largeur des données et de synthèse d'architecture

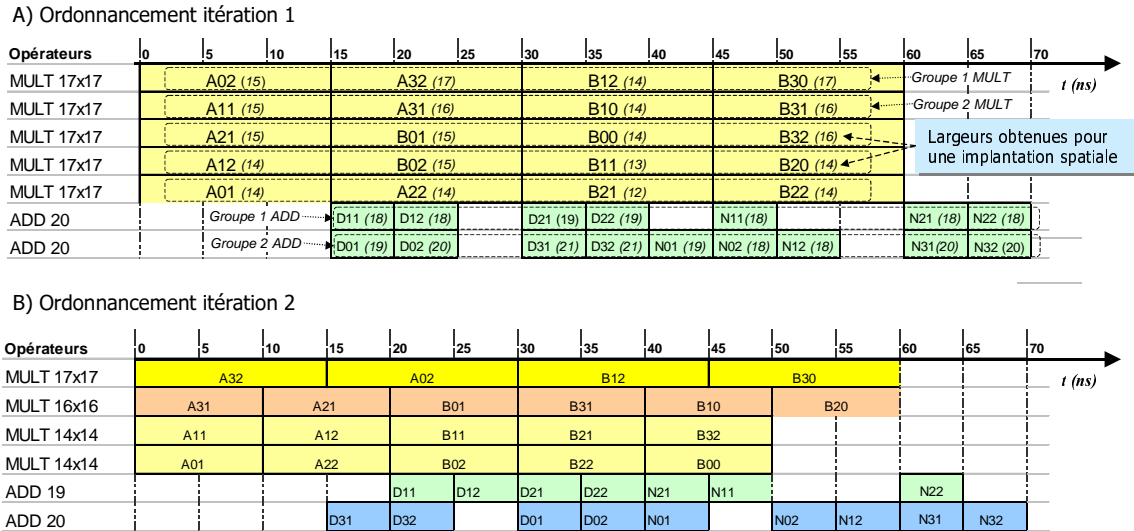


FIG. 5 – Ordonnements obtenus pour les deux premières itérations du processus global d'optimisation de l'architecture à largeur multiple.

est mis en œuvre pour coupler ces deux processus. De plus, l'évaluation de la précision est réalisée par une approche analytique permettant d'obtenir des temps d'optimisation raisonnables par rapport aux approches basées sur la simulation. L'outil permettant d'implanter cette méthodologie est en cours de développement. Les premiers résultats montrent l'intérêt d'une telle approche pour optimiser la surface de l'architecture par rapport à une architecture utilisant une largeur unique pour tous les opérateurs. Dans le cas de l'application considérée, la surface des opérateurs a été réduite de 47% par rapport aux méthodes existantes.

## Bibliographie

1. Constantinides (G.), Cheung (P.) et Luk (W.). – Heuristic Datapath Allocation for Multiple Word-length Systems. *In : Design Automation and Test in Europe (DATE 01)*, pp. 791–796. – Munich, Germany, mars 2001.
2. Constantinides (G. A.), Cheung (P. Y. K.) et Luk (W.). – *Synthesis and Optimization of DSP Algorithms*. – Kluwer Academic, 2004.
3. Dido (J.), Geraudie (N.), Loiseau (L.), Payeur (O.), Savaria (Y.) et Poirier (D.). – A flexible floating-point format for optimizing data-paths and operators in FPGA based DSPs. *In : International Symposium on Field-Programmable Gate Arrays*. pp. 50–55. – Monterey, California, USA, février 2002.
4. Gajski (Daniel), Dutt (Nikil), Wu (Allen) et Lin (Steve). – *High-Level Synthesis*. – Kluwer Academic, 1993.
5. Kim (S.) et Sung (W.). – Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, n8, December 1998, pp. 935–940.
6. Kum (K.) et Sung (W.). – Combined Word-Length Optimization and High-level Synthesis of Digital Signal Processing Systems. *In : IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 921–930. – août 2001.
7. Martin (E.), Tourelles (J.) et Nouet (C.). – Conception optimisée d'architectures en précision finie

- pour les applications de traitement du signal. *Traitement du Signal 2001*, vol. 18, n1, 2001, pp. 47–56.
8. Menard (D.), Quemerais (P.) et Sentieys (O.). – Influence of fixed-point DSP architecture on computation accuracy. *In : 11th European Signal Processing Conference (EUSIPCO 2002)*, pp. 587–590. – Toulouse, France, septembre 2002.
  9. Menard (D.) et Sentieys (O.). – A methodology for evaluating the precision of fixed-point systems. *In : IEEE International Conference on Acoustics, Speech and Signal Processing 2002 (ICASSP 2002)*, pp. III–3152– III–3155. – Orlando, USA, mai 2002.
  10. Menard (D.) et Sentieys (O.). – Automatic Evaluation of the Accuracy of Fixed-point Algorithms. *In : Design, Automation and Test in Europe 2002 (DATE-02)*, pp. 529–535. – Paris, France, March 2002.
  11. Parks (T.W.) et Burrus (C.S.). – *Digital Filter Design*. – Jhon Willey and Sons Inc, 1987.
  12. Sentieys (O.), Diguet (J.-P.) et Philippe. (J.L.). – GAUT : a High Level Synthesis Tool dedicated to real time signal processing application. *In : University Booth, EURO-DAC*. – Brighton, 18-22 september 1995.
  13. Synplicity corporation. – *Synplify Pro 7.3 Reference Manual*, juin 2003.
  14. Wadekar (S.) et Parker (A.). – Accuracy Sensitive Word-Length Selection for Algorithm Optimization. *In : IEEE/ACM International Conference on Computer Design (ICCAD'98)*, pp. 54–61. – San Jose, CA , USA, novembre 1998.