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

Nicolas Hervé, Daniel Ménard, Olivier Sentieys

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

Nicolas HERVÉ, Daniel MÉNARD et Olivier SENTIEYS
IRISA – Université de Rennes 1
{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 : Architecture Matérielle, Arithmétique virgule fixe, FPGA, 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 DSP hautes performances. Le problème principal freinant leur utilisation intensive est l'absence d'outils de développement 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* (i.e. le nombre de bits) des opérateurs, des bus et des mémoires au sein des architectures en virgule fixe étant plus faible, la surface et la consommation d'énergie de ces architectures sont moins importantes. 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 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 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 est important et des outils sont donc nécessaires.

L'objectif du processus de conversion en virgule fixe est de déterminer, pour chaque donnée, la largeur et la position de la virgule.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 [Kum, 2001, Wadekar, 1998]. En effet, ces deux taches sont interdépendantes : 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 [Gajski, 1993] et d'autre part, l'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.

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 comparée 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. Enfin, la section 5 présente un exemple d'application et les résultats obtenus.

2 ÉTAT DE L'ART

2.1 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), définit comme le rapport (la différence en dB) entre la puissance du signal qu'on désire représenter et la puissance du *bruit de quantification* (i.e. de l'erreur sur cette valeur due au codage). Deux types de méthode peuvent être utilisés pour déterminer ce RSQB : celles basées sur la simulation ou celles analytiques. 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. Pour les systèmes complexes, le temps d'optimisation devient prohibitif et des heuristiques doivent donc être utilisées afin de limiter l'espace de recherche.

Dans [Martin, 2001], 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 deux hypothèses : les variables en entrée de chaque opérateur sont considérées comme indépendantes et centrées, et 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, or par défaut, une loi de quantification par troncature est souvent 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. Cette méthode ne peut donc pas être appliquée aux systèmes récursifs (i.e. où la sortie du système dépend des échantillons de sortie précédents).

2.2 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 [Martin, 2001]. 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 [Constantinides, 2004].

La méthodologie proposée par Constantinides et *al* [Constantinides, 2004] 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. La synthèse d'architecture n'est alors 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 [Constantinides, 2001] une heuristique pour combiner l'ordonnancement et le partage de

ressources est proposée pour traiter le cas d'un graphe flot de signal intégrant des opérations de largeurs différentes tel que le graphe flôt 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 qui 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 spécifiée.

Dans [Kum, 2001], 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 groupe. 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 [Kum, 2001] 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 [Dido, 2002]. Dans cette étude, uniquement la mise en œuvre de l'arithmétique virgule fixe est traitée.

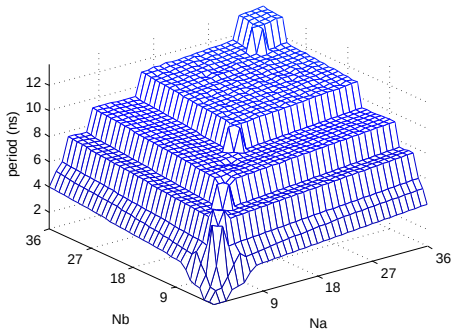
La synthèse d'architecture s'appuie sur une bibliothèque d'opérateurs arithmétiques virgule fixe associée à une famille de FPGA donnée. Chaque opérateur est caractérisé en termes de surface, de temps et de consommation énergétique, en fonction des largeurs des opérandes d'entrée et du résultat. Ces informations 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 Synplicity [SYN, 2003]. Des composants d'interconnexion (tristate, multiplexeurs, demultiplexeurs) 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. Le tableau 1 présente un extrait des informations enregistrées dans la base de données de composant pour la multiplication sur virtex-II.

Na	Nb	LUTs	Mult	delay (ns)
13	13	0	1	5.595
13	13	191	0	8.175
14	13	0	1	5.692
14	13	196	0	8.426
14	14	0	1	5.789
14	14	210	0	8.426
15	15	0	1	5.983
15	15	253	0	8.467
16	16	0	1	6.177
16	16	273	0	10.315
17	17	0	1	6.371
17	17	322	0	10.334
18	18	0	1	6.532
18	18	344	0	10.417
19	19	74	1	8.429

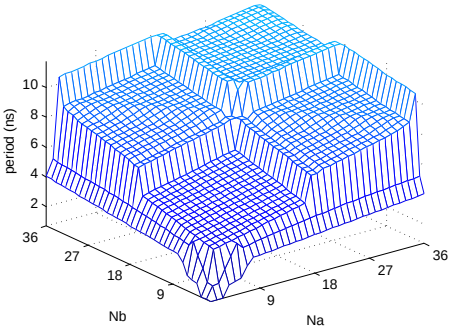
TAB. 1 – Extrait de la base de données de composants pour la multiplication sur virtex-II

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.

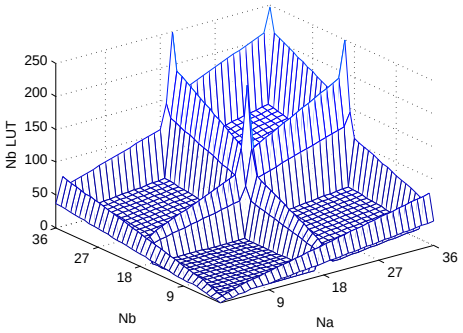
Les résultats obtenus lors de la synthèse de l'opérateur de



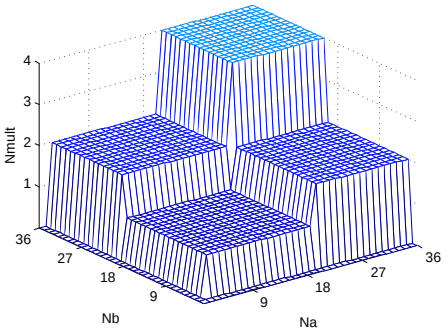
(a) - Temps de propagation (sans multiplieurs dédiés)



(b) - Temps de propagation (avec multiplieurs dédiés)



(c) - Nombre de LUTs utilisées



(d) - Nombre de multiplieurs dédiés 18×18

FIG. 1 – Temps de propagation et surface en fonction de la largeur des entrées pour un opérateur de multiplication sur virtex-II.

multiplication pour les FPGAs Virtex-II de Xilinx sont présentés à la figure 1. La figure 1.a montre l'évolution du temps de propagation de l'opérateur en fonction de la largeur des entrées, pour une implantation de la multiplication utilisant uniquement les ressources logiques. Avant le Virtex-II, les FPGAs ne comportaient pas de multiplieurs dédiés. 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 et leurs successeurs, contiennent des ressources dédiées (multiplieurs, mémoire, chemins de données, etc.). 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 de caractérisation de l'opérateur de multiplication sur virtex-II utilisant les multiplieurs câblés 18×18 bits sont présentés aux figures 1.b, 1.c et 1.d. Dans certains cas (multiplication par un nombre de 19 ou 20 bits), l'outil de synthèse préfère utiliser des LUTs pour économiser le nombre de multiplieurs dédiés utilisés, malgré le léger surcoût qui apparaît alors au niveau de la latence. 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 2. 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 fixer la position de la virgule (ou de manière équivalente le poids du bit le plus significatif) de chaque signal afin d'empêcher qu'un débordement puisse se produire. 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 [Sentieys, 1995].

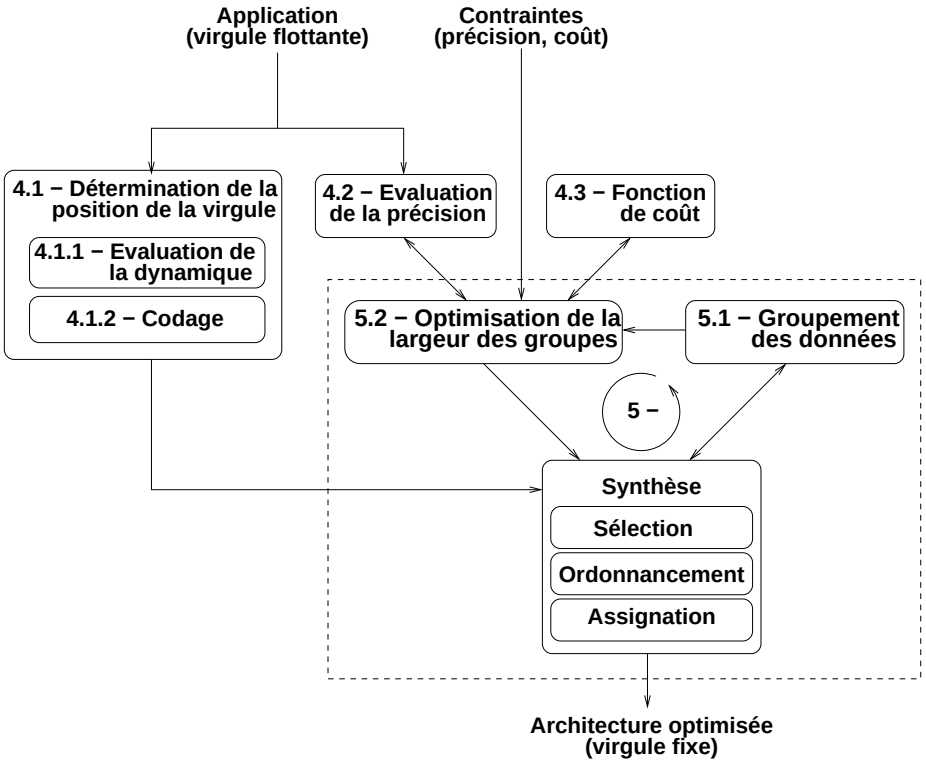


FIG. 2 – Environnement conjoint de synthèse et d'optimisation de la largeur des opérateurs

4.1 Détermination de la position de la virgule

4.1.1 Evaluation de la dynamique des données

Une approche analytique, basée sur l'arithmétique d'intervalle [Moore, 1963] et la norme L1 [Parks, 1987], a été mise en œuvre pour garantir l'absence de débordements. Ces techniques permettent de déterminer l'expression du domaine de définition d'une donnée à partir de celui des entrées. Pour les systèmes non-récursifs, 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 cas des systèmes linéaires à coefficients constants, la dynamique des données est déterminée à l'aide de la norme L1. 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 récursives que non récursives.

4.1.2 Codage des données

L'étape précédente a permis d'évaluer la dynamique de chaque donnée. Ces résultats sont ensuite utilisés pour déterminer la position de la virgule (nombre minimal de bits requis pour coder la partie entière). L'objectif de l'étape suivante 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, définies pour chaque type d'opération 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 [Menard, 2002] 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-récursifs.

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. La méthode utilisée pour déterminer automatiquement les différentes fonctions de transfert du système et l'outil développé pour implanter cette méthode sont présentés dans [Menard, 2002]. Pour les systèmes non-linéaires et non-

récursifs, le gain est obtenu à partir des paramètres statistiques des différents signaux présents entre la sortie et la source de bruit [Menard, 2004]. 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. 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 des opérations de même type qui auront la même largeur (mais qui ne seront pas nécessairement effectués sur le même opérateur). 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 initialement, 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 suivante :

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

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

L'évaluation de la précision est effectuée à l'aide de la méthode présentée dans la section 4.2 et permettant d'obtenir l'expression analytique de $RSBQ(\overrightarrow{b})$. 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. Pour un type d'opération donné, le nombre de groupes pour l'itération suivante correspond au nombre d'opérateurs nécessaires obtenu lors de cette phase de synthèse. Cet algorithme se termine lorsque deux 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* et une *largeur propre* sont calculées. La mobilité est définie comme la différence des dates d'exécution obtenus lors de deux ordonnancements par liste : un dans le sens directe et l'autre dans le sens inverse. La largeur propre est obtenue lors de l'optimisation pour une implantation spatiale, c'est à dire où chaque opération a un opérateur virgule fixe dédié. La largeur propre de l'opération correspond à la largeur de son opérateur dédié après optimisation. Les opérations sont traités par ordre croissant de mobilité. La mobilité est utilisée pour placer chaque opération dans le groupe le plus adéquat (i.e. celui de largeur immédiatement supérieur à la largeur propre de l'opération). L'objectif recherché est de privilégier les groupes de plus petites largeurs afin d'obtenir les plus petites largeurs possible pour les opérateurs.

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 propre maximale des opérations de ce groupe. Pour les itérations suivantes, le nombre de groupe est augmenté, de manière à transférer les opérations nécessitant moins de précision vers un nouveau groupe de plus petite largeur, et ainsi diminuer le nombre d'opérateur des groupes à grande largeur au profit de ces nouveaux groupes.

5.2 Optimisation de la largeur des groupes

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 combinai-

sons 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 (avec la largeur des autres groupes 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 $\Delta_{\mathrm{RSBQ}}/\Delta_{\mathrm{coût}}$ 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

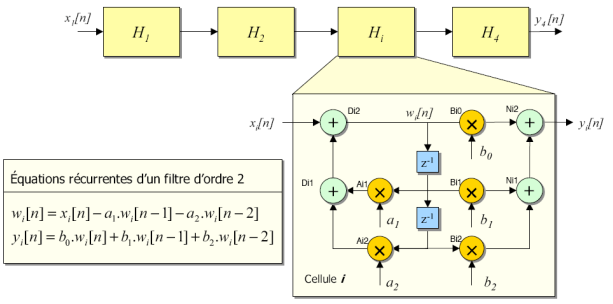


FIG. 3 – Graphe Flot de Signal du filtre IIR d'ordre 8.

à un filtre à réponse impulsionnelle infinie (IIR) d'ordre 8. Ce filtre est implanté sous forme de 4 filtres cascadés d'ordre 2. Le graphe flot de signal de ce filtre récursif est présenté à la figure 3. 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 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 4.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 propre maximale des opérations de ce groupe. Ainsi, les multiplications sont réalisées sur un multiplieur 17×17 bits et les additions par un additionneur 20 bits. L'architecture cible est un virtex-II, uniquement les ressources logiques sont utilisées. La synthèse d'architecture pour cette spécification virgule fixe conduit à l'ordonnancement présenté à la figure 4.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 4.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 4.b. Les multiplieurs de largeur 14 à 16 bits ayant une latence plus faible ($10\ ns$), quatre multiplieurs seulement sont finalement suffisants. 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 [Martin, 2001] réalise une architecture pour laquelle une largeur unique d'opérateur est utilisée. Ainsi, pour cette application, 5 multiplieurs 19×19 bits et 2 additionneurs 19 bits 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 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

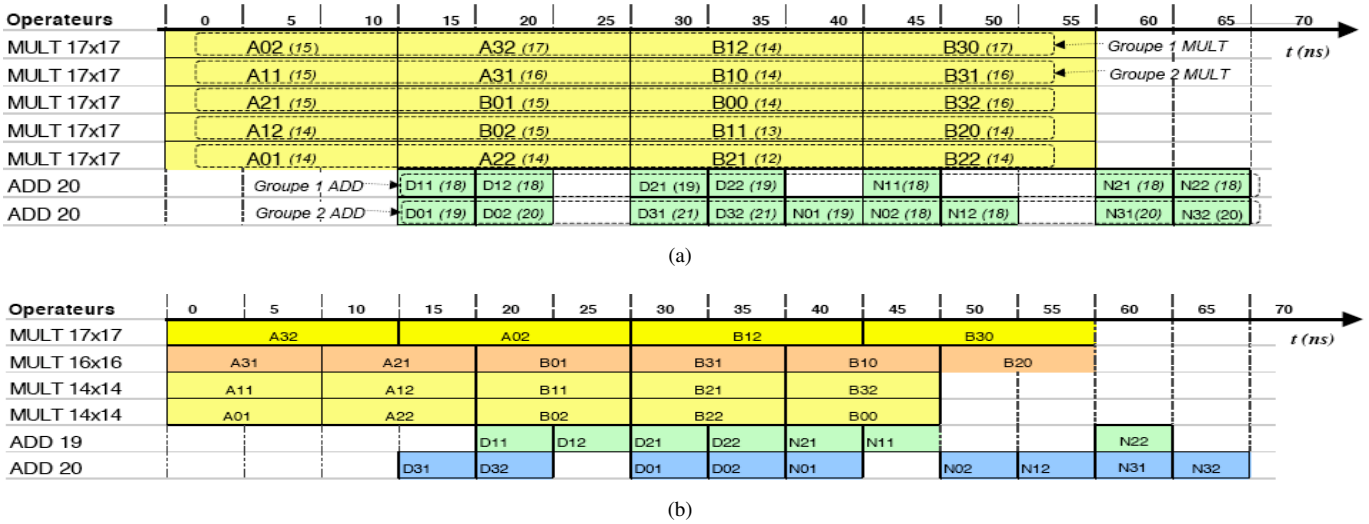


FIG. 4 – Ordonnancements obtenus pour deux étapes de l'optimisation de l'architecture à largeur multiple.

réduite de 47%.

Par rapport aux objectifs fixés dans le projet RNTL OSGAR, la méthode est maintenant complètement définie. Il reste à la mettre en oeuvre sur deux autres exemples d'application : un récepteur WCDMA et un algorithme d'estimation de mouvement. Les efforts actuels se portent essentiellement sur le développement de l'outil. Les expérimentation sur le WCDMA sont prévus pour la fin 2005.

Références

[SYN, 2003] *Synplify Pro 7.3 Reference Manual*. Synplicity corporation.

[Constantinides, 2001] Constantinides G., Cheung P. et Luk W., Heuristic Datapath Allocation for Multiple Wordlength Systems. *Design Automation and Test in Europe (DATE 01)*, pages 791–796, Munich, Germany.

[Constantinides, 2004] Constantinides G. A., Cheung P. Y. K. et Luk W., *Synthesis and Optimization of DSP Algorithms*. Kluwer Academic.

[Dido, 2002] 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. *International Symposium on Field-Programmable Gate Arrays*, pages 50–55, Monterey, California, USA. ACM Press.

[Gajski, 1993] Gajski D., Dutt N., Wu A. et Lin S., *High-Level Synthesis*. Kluwer Academic, University of California, Irvine, CA, USA. Introduction to Chip and System Design.

[Kum, 2001] Kum K. et Sung W., Combined Word-Length Optimization and High-level Synthesis of Digital Signal Processing Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 921–930.

[Martin, 2001] Martin E., Toureilles J. et Nouet C., Conception optimisée d'architectures en précision finie pour les applications de traitement du signal. *Traitement du Signal 2001*, 18(1) :47–56.

[Menard, 2004] Menard D., Rocher R., Scalart P. et Sentieys O., Sqnr determination in non-linear and non-recursive fixed-point systems. *XII European Signal Processing Conference (EUSIPCO 2004)*, pages 1349–1352, Vienna, Austria.

[Menard, 2002] Menard D. et Sentieys O., Automatic Evaluation of the Accuracy of Fixed-point Algorithms. *IEEE/ACM conference on Design, Automation and Test in Europe 2002 (DATE-02)*, pages 529–535, Paris, France.

[Moore, 1963] Moore R. E., *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis.

[Parks, 1987] Parks T. et Burrus C., *Digital Filter Design*. Jhon Willey and Sons Inc.

[Sentieys, 1995] Sentieys O., Diguet J. et Philippe. J., GAUT : a High Level Synthesis Tool dedicated to real time signal processing application. *University Booth, EURO-DAC*, Brighton.

[Wadekar, 1998] Wadekar S. et Parker A., Accuracy Sensitive Word-Length Selection for Algorithm Optimization. *IEEE/ACM International Conference on Computer Design (ICCAD'98)*, pages 54–61, San Jose, CA , USA.