Thèse de doctorat de l'université Pierre et Marie Curie - Paris VI
L'école Nationale d'ingénieurs de sfax - Université de Sfax

Présentée par : **Mariem TURKI**

Pour obtenir le grade de

Docteur de l'Université Pierre et Marie Curie

Spécialité Informatique et Micro-Electronique

ET

Docteur de l'Ecole Nationale d'Ingénieurs de Sfax

Spécialité Ingénierie des Systèmes Informatiques

Techniques de multiplexage pour un système d'émulation et de prototypage rapide à base de FPGA

Soutenue le 17 Septembre 2014, devant le jury composé de :

Pr.	Rached Tourki	FSM	Rapporteur
Pr.	Smail Niar	LAMIH	Rapporteur
Pr.	Bertrand Granado	LIP6	Examinateur
Pr.	Mohamed Masmoudi	ENIS	Examinateur
Pr.	Habib Mehrez	LIP6	Directeur de thèse
Pr.	Mohamed Abid	ENIS	Directeur de thèse
M.	Zied Marrakchi	Flexras	Invité

Résumé

De nos jours, la complexité de la conception des circuits intégrés et du logiciel croit régulièrement, faisant croître le besoin de la vérification dans chaque étape du cycle de conception. Le prototypage matériel sur une plateforme multi-FPGA présente le meilleur compromis entre le temps de conception d'un circuit et le temps d'exécution d'une application par ce circuit. Pour l'implémenter sur cette plateforme, une opération de partitionnement est effectuée afin de créer des partitions capables de s'intégrer dans chaque FPGA. Par conséquent, des signaux coupés à l'interface des partitions doivent passer d'un FPGA à un autre. Cependant, le nombre de traces physiques inter-FPGA est limité ce qui crée des problèmes de routabilité du circuit prototypé.

Cette thèse touche surtout la partie post-partitionnement et s'intéresse au problème de routage inter-FPGA. Ainsi, les principaux travaux de cette thèse sont les suivants :

•Dans un premier temps, nous nous intéressons au développement d'un générateur de benchmarks qui permet, à l'aide d'une description architecturale simple du benchmark, de générer un circuit modélisé avec le langage de description matérielle VHDL. Le générateur utilise un ensemble de composants ce qui donne aux benchmarks un aspect réel semblable à celui des circuits industriels. Ces circuits de tests nous serviront pour évaluer les performances des techniques développées dans cette thèse.

•Dans un deuxième temps, nous proposons de développer un outil spécifique qui intervient après le partitionnement pour prendre en compte la contrainte liée à la limitation du nombre de fils d'interconnexion entre les FPGAs. Cet outil est basé sur une approche itérative visant à réduire le taux de multiplexage (nombre de signaux qui partagent un seul fil physique). Le routage en lui même est assuré par l'algorithme de routage Pathfinder adapté. Cet algorithme servira comme point de départ pour les techniques de routage développées durant cette thèse. Des adaptations adéquates seront faites pour cibler un réseau de routage inter-FPGA. Dans une deuxième partie, nous essayons de déterminer la meilleure forme du signal à router (bi-points ou multi-points) ainsi que le graphe de routage utilisé. Pour cela, nous proposons des scénarios de test afin de sélectionner les critères qui donnent la fréquence de fonctionnement la plus performante.

•Par la suite, nous présentons une description détaillée des IPs de multiplexage utilisés. Ces IPs sont insérés dans les parties émettrices et réceptrices d'un canal de communication. Ces IPs incluent des composants spécifiques appelés SERDES pour assurer la sérialisation/déserialisation des données à transmettre. L'insertion de ces composants peut créer des problèmes de routabilité intra-FPGA. Ainsi, dans une deuxième partie, nous proposons un algorithme de placement basé sur l'estimation de la congestion afin d'améliorer la routabilité du circuit.

Mots-clés : FPGA (Field Programmable Gate Array), Prototypage, Routage, Pathfinder, Itératif, Multiplexage, Graphe de routage, Benchmarks.

Abstract

With the ever increasing complexity of system on chip circuits, the software and hardware developers can no longer wait for the fabrication phase to test their designs. Multi-FPGA-based prototyping is an important verification method since it presents a best compromise between the time of circuit design and the execution time of an application by the circuit. To implement a design into a multi-FPGA platform, a partitioning operation is performed to create partitions which can fit into each FPGA. Consequently, cut signals which appear at the interfaces are transmitted between pairs of FPGA. However, the number of physical traces inter-FPGA is limited, which creates problems of routability of the prototyped design.

This thesis mainly deals with the post-partitioning task and addresses the problem of inter-FPGA routing. Thus, the main contributions of this thesis are:

•Firstly, we focus on the development of a benchmark generator which, using a simple architectural description of the benchmark, generates a circuit modelled with the hardware description language VHDL. The generator uses a set of industrial components providing benchmarks with real behaviour similar to that of industrial circuits. These benchmarks are used to evaluate the performance of the techniques developed in this thesis.

•In a second step , we propose a specific tool which acts after the partitioning to handle the constraints related to the limited number of interconnection between FPGAs. This tool is based on an iterative approach and aims to reduce the multiplexing ratio (the number of signals that share the same physical wire). The routing task itself is operated by the Pathfinder routing algorithm which is widely used by academic and industrial researchers . This algorithm is used as a starting point for routing techniques developed in this thesis . In a second part , we try to identify the best shape of the routed signals and the appropriate routing graph. For this reason, we propose scenarios to select criteria that give the best system frequency.

•Finally, we present a detailed description of the architecture of the multiplexing IPs. These IPs are inserted in the transmitting and receiving FPGAs of a communication channel. These IPs include specific components called SERDES for serialization/deserialization of the data. The insertion of these IPs can create problems of intra-FPGA routability. Thus, in a second part, we propose a placement algorithm based on congestion estimation to improve the routability of the circuit.

Keywords: FPGA (Field Programmable Gate Array), Prototyping, Routing, Pathfinder, Iterative, Multiplexing, Routing graph, Benchmarks.

A mes parents pour leur amour et leur soutien
A mon mari pour sa patience et son encouragement
A toute ma famille et tous mes amis

Remerciements

Je voudrais exprimer ma plus grande gratitude à mes directeurs de thèse, Habib Mehrez, Professeur à Paris 6, et Mohamed ABID, Professeur à l'ENIS pour la confiance qu'ils m'ont accordée tout au long de ma thèse, ainsi que pour leur grande disponibilité, motivation et encouragement.

Je suis extrêmement reconnaissante à mon encadrant Zied Marrakchi pour ses conseils précieux et son support technique qui ont permis de me guider durant cette thèse. Zied Marrakchi a réussi, avec son dynamisme, son talent et surtout sa patience, à orienter les travaux de cette thèse vers le bon chemin.

J'adresse mes remerciements à monsieur Rached Tourki, professeur à la faculté des sciences à Monastir, et monsieur Smail Niar, professeur à l'université de valenciennes, qui m'ont fait l'honneur d'être rapporteurs de cette thèse.

Je tiens également à remercier messieurs Bertrand Granado, professeur au laboratoire Lip6 de Paris 6 et Mohamed Masmoudi, professeur à l'Ecole Nationale d'Ingénieurs de Sfax, qui ont bien voulu accepter d'évaluer ce travail en participant à mon jury de thèse.

Je voudrais aussi adresser mes sincères remerciements à monsieur Abdelmalek si-Mrabet pour sa gentillesse et ses réponses techniques.

Je souhaite ensuite saluer mes collègues au LIP6, Emna Amouri, Andi Drebes et Vinod Pangratious sans oublier mes amis à l'ENIS, Saoussen, Bouthaina, Nihel et Rania pour la bonne ambiance qu'ils ont créée durant la thèse.

Je témoigne ma plus profonde affection à mes parents et mes deux sœurs pour leur amour et leur perpétuel soutien. J'envoie également mes sentiments les plus tendres à mon mari, pour son soutien quotidien et son encouragement.

Pour terminer, je m'excuse auprès de ceux que j'aurais pu oublier, et à qui j'adresse évidemment mes sincères remerciements.

Table des Matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Avec la tendance mondiale vers le numérique, la complexité de la conception des circuits intégrés et du logiciel croit régulièrement tandis que la durée de vie des circuits et des produits se réduit. Par conséquent, les développeurs du matériel et du logiciel ne peuvent plus attendre jusqu'à la phase de fabrication pour tester leur circuit [3]. La vérification est une étape importante pour la création du produit final et c'est une composante clé pour la réussite de la commercialisation dans les délais prévus. Actuellement, la vérification occupe environ 60% à 80% du temps de conception d'un SoC [4]. Avant de produire le silicium réel, il n'y a que trois possibilités de vérification : une simulation, une émulation et un prototype sur FPGA.

1.1 Les buts de recherche et contributions

Le prototypage matériel sur plateforme FPGA présente le meilleur compromis entre le temps de conception d'un circuit et le temps d'exécution d'une application par ce circuit. L'avantage principal de l'utilisation d'une carte à base de circuit FPGA est sa capacité d'exécution à grande vitesse, proche des conditions réelles, un modèle à un cycle/bit précis [5]. D'autre part, la disponibilité des outils CAO pour les circuits FPGA ont simplifié le processus d'implémentation, ce qui rend le chemin entre la modélisation du SoC et son implémentation sur FPGA plus simple. Cependant, lorsque la taille d'un circuit dépasse la capacité logique d'un FPGA, une plateforme de prototypage multi-FPGA est utilisée pour supporter la totalité du circuit. La surface en silicium d'un circuit fabriqué avec la technologie FPGA est estimée à 40 fois plus grande qu'avec celle des ASIC [6]. Du coup, le circuit prototypé doit être partitionné sur une plateforme multi-FPGA. Le nombre de FPGA, dépendant de l'application à tester, varie entre quelques [7] et 60 FPGAs [1].
Une plateforme complète de prototypage propose une carte à base de circuits FPGA et un flot logiciel assurant l'implémentation du circuit à vérifier sur la carte. Cependant, rare sont les environnements de prototypage complets qui ont été proposés dans l'état de l'art. Les industriels tel que Dinigroup [8], Aldec [9] et Gidel [10] produisent une multitude de

cartes multi-FPGA à base de circuit Xilinx et/ou Altera sans proposer les outils CAO qui automatisent le flot de l'implémentation. Récemment, Synopsys a lancé son environnement de prototypage Certify [11], qui, à part les cartes Haps multi-FPGA [12], offre un flot logiciel permettant de gérer des systèmes contenant des millions de LUTs. Pareil pour S2C [13] et Cadence [14] qui offrent un environnement complet pour un prototypage rapide sur FPGA.
Vu l'importance de la vérification dans le cycle de vie d'un système sur puce et le nombre réduit des plateformes de prototypage matériel, un projet financé par System@tic [15] a été lancé, visant la conception d'un environnement complet de prototypage. Le projet couvre un aspect matériel (conception de carte multi-FPGA assuré par REFLEX_CES [16]) et un aspect logiciel assuré par Flexras [17], Adacsys [18] et le Laboratoire d'Informatique de Paris 6 (LIP6)(Laboratoire d'accueil des travaux de cette thèse). Les études faites durant cette thèse touchent surtout la partie post-partitionnement. En effet, lorsqu'un système est partitionné sur une carte multi-FPGA, des signaux coupés à l'interface des partitions doivent passer d'un FPGA à un autre. Cependant, le nombre de fils physiques inter-FPGA est limité ce qui crée des problèmes de routage du circuit prototypé. La solution proposée est de grouper un certain nombre de signaux ensemble et les faire passer à travers le même fil physique. Une fois le routeur détermine l'ensemble des groupes, des IPs de multiplexage(sérialisation/désérialisation) seront insérés dans les parties émettrices et réceptrices de chaque groupe de signaux. Cette étape, étant effectuée après le partitionnement et le routage, pourrait créer un problème de routage intra-FPGA à cause de la partie logique récemment insérée (les IPs de multiplexage).
Les solutions développées pour remédier à ces deux problèmes doivent être validées en utilisant des gros benchmarks de test ayant des caractéristiques spécifiques permettant de mettre en valeur les performances de ces techniques.
Les travaux de cette thèse s'intéressent à la résolution des problèmes suivants :

- Lorsque la taille d'un circuit à tester dépasse la capacité logique d'un seul FPGA, ce circuit est partitionné sur une carte multi-FPGA ayant un nombre limité de connexions physiques inter-FPGA. Dans ce cas, il s'agit d'un problème de routage inter-FPGA.
- Après l'insertion d'IPs de multiplexage dans les FPGAs sources et destinations, la congestion augmente dans les FPGAs et le circuit risque de ne pas pouvoir être routé. Dans ce cas, nous parlons d'un problème de routage intra-FPGA.
- Problème de limitation de benchmarks de test qui répondent à un certain nombre d'exigences requises par la nature des techniques à valider.

1.1.1 Problème de routage inter-FPGA

Comme nous l'avons expliqué, dans le cas où le nombre de cellules logiques d'un circuit dépassent largement le nombre de blocs logiques d'un seul FPGA, ce circuit devrait être partitionné sur une carte multi-FPGA.

La manière dont le circuit est découpé a un effet très important sur les performances du système de prototypage. L'outil de partitionnement permet d'obtenir une répartition du circuit objet du prototypage sur les FPGA de la carte. Cette répartition tente de tirer le meilleur profit de l'architecture du FPGA en tenant compte des contraintes imposées par celle-ci en termes de ressources logiques disponibles mais aussi du nombre de liens physiques inter-FPGA. En effet, l'étude dans [19] a montré que les FPGA disposent d'un nombre limité de ressources d'entrée-sortie. Or, a priori, cette ressource matérielle détermine le nombre de signaux qui peuvent apparaître à l'interface de deux parties et qui doivent passer d'un FPGA à un autre.

Face à cette limitation physique de ressources de routage inter-FPGA, des études dans la littérature ont proposé des approches basées sur le partage des ressources là où plusieurs signaux peuvent être transmis à travers le même fil physique. Le routage des signaux peut se faire à travers des algorithmes statiques qui ne dépendent pas de la variation de la valeur de chaque signal au cours du temps [20] [21] [22] [23]. Chaque signal est transmis durant un cycle donné indépendamment de son activité. Alternativement, une approche dynamique a été proposée dans [24] selon laquelle seuls les signaux qui ont changé de valeur sont transmis. Les approches dynamiques dépendent de l'activité du système à prototyper. Par contre, la disponibilité des ressources de routage est déterminée au cours de l'exécution de l'application, ce qui nécessite plus de ressources logiques dans chaque FPGA pour le contrôle des opérations de communication. En plus, pour des applications complexes avec une activité importante, les algorithmes dynamiques sont moins robustes face aux variations de la fréquence, et les risques de transmission de données erronées sont plus importants. Par conséquent, nous nous intéressons dans cette thèse aux méthodes statiques.

La technique des «fils virtuels» ou «virtual wires» [20] consiste à faire passer la valeur de plusieurs signaux logiques par le même canal physique (multiplexage). Les connexions sont établies via un réseau de communication pipeliné et routé statiquement. Le multiplexage permet d'augmenter l'efficacité de la bande passante tout en affectant plusieurs signaux à un pin. En effet, l'idée consiste à construire des canaux de communication logiques au-dessus des canaux de communication physiques qui sont les fils d'interconnexion dont on dispose entre deux FPGAs. Un canal de communication logique permet de virtualiser le transfert des signaux d'un FPGA à l'autre. Pour assurer cette virtualisation, il a fallu diviser la période de l'horloge d'émulation en un certain nombre de microcycles. Donc chaque signal multiplexé est transmis pendant un microcycle.

L'inconvénient de cette technique c'est qu'elle multiplexe tous les signaux sans tenir compte des signaux qui passent par le chemin critique. D'autre part, cette technique étant basée sur l'ordonnancement des signaux, est peu robuste du fait que la probabilité d'erreur est importante (erreurs dues au non-respect des contraintes temporelles). En cas d'erreur, il est nécessaire de retransmettre tous les signaux pendant une autre période d'émulation, et par conséquent dégrader la performance du système d'émulation.

Une autre technique a été proposée [21] dont le but est de déterminer, en utilisant la

programmation linéaire, quels sont les signaux qui doivent être multiplexés, et ceux qui ne le doivent pas. Une autre particularité de cette méthode, c'est que tous les signaux sont envoyés plusieurs fois dans la même période d'émulation, mais la valeur de chaque signal n'est évaluée qu'une seule fois. L'apport de cette méthode est qu'elle est facile à implémenter. En effet, toute erreur dans la transmission d'un signal est rectifiable pendant seulement quelques microcycles, contrairement à la méthode des virtual wires ou il faut toute une période d'émulation pour le faire.
Cependant, cette technique n'est pas basée sur l'ordonnancement, du coup, tous les signaux sont transmis plusieurs fois pendant la même période ce qui dégrade la performance du système de prototypage.
Dans cette étude, nous proposons de développer un outil spécifique qui intervient après le partitionnement pour prendre en compte la contrainte liée à la limitation du nombre de fils d'interconnexion entre deux FPGA.
L'objectif est donc de développer un outil de routage, basé sur le multiplexage des signaux entre les FPGA pour résoudre la contrainte limitant le nombre de connexions physiques disponibles. Un ensemble de signaux ayant la même source et les mêmes destinations sont routés à travers le même chemin. Pour tracer les propriétés de notre outil de routage, il a fallu examiner les inconvénients des techniques proposées dans l'état de l'art. En effet, la technique des fils virtuels et celle de la programmation linéaire utilisent le multiplexage pour transmettre les signaux d'un FPGA à un autre. Cette technique (le multiplexage) nous semble inévitable à cause de la grande limitation des ressources inter-FPGA. Bien évidemment, le multiplexage a un coût en termes de fréquence de fonctionnement, mais aussi en termes de surface logique (nombre de portes logiques). En effet, transférer la valeur de plusieurs signaux logiques à travers le même canal physique a un impact sur la performance du circuit. Il se traduit par une réduction de la fréquence de fonctionnement du circuit. Cette fréquence dépend essentiellement de deux paramètres : le taux de multiplexage et le nombre de stages par période. Le taux de multiplexage est le nombre de signaux transmis à travers le même lien physique. Le nombre de stage est le nombre de fois un signal est coupé entre deux registres.
Avec la technique des fils virtuels qui est basé sur l'ordonnancement, le nombre de stages par période est important contrairement à la technique de la programmation linéaire qui vise à diminuer le nombre de stages en sélectionnant les signaux qui ne doivent pas être multiplexés, mais en contrepartie, le taux de multiplexage est très grand du fait que tous les signaux sont transmis dans chaque stage. Pour remédier à ce problème, nous décidons de sélectionner, suivant des critères précis, les signaux qui ne doivent pas être multiplexés afin d'obtenir un nombre de stages réduit. Pour diminuer le taux de multiplexage, notre outil de routage effectuera plusieurs itérations. A travers ces itérations, le taux de multiplexage se décrémente jusqu'à ce qu'aucune solution de routage n'est trouvée.
D'autre part, pour trouver un chemin de routage entre la source et la destination d'un signal, les techniques proposées dans l'état de l'art utilisent un algorithme constructif basé sur l'évitement de congestion. Avec des taux de multiplexage réduits, cette méthode

conduit à un blocage rapide au niveau du routage qui ne se résout qu'en utilisant des taux de multiplexage plus grands, ce qui dégrade remarquablement la fréquence de fonctionnement du système à tester. Pour cette raison, nous proposons bâtir notre outil de routage en utilisant un algorithme basé sur la négociation de la congestion. Par conséquent, tous les signaux négocient leurs besoins d'utiliser une ressource afin de construire un chemin de routage entre la source et la destination de chaque signal.
Finalement, la performance du routeur dépend de la forme du signal à router. En effet, un signal peut être bi-points (ayant une source et une seule destination), mais aussi il peut avoir plusieurs destinations. Des plateformes matérielles proposent des traces physiques multi-points pour router les signaux multi-destinations. Les auteurs de [25] proposent une approche pour router ce type signaux. Mais comme la plupart des plateformes matérielles comportent uniquement des traces physiques bi-points, les techniques développées dans cette thèse ciblent uniquement ce type de plateformes. Les techniques de multiplexage dans [21] et [20] agissent sur des signaux bi-points. Même les signaux ayant plusieurs destinations sont décomposés sous forme de branches. Dans notre étude, nous allons définir plusieurs scénarios pour déterminer celui qui donne les meilleurs résultats de point de vue fréquence de fonctionnement. Dans chaque scénario, nous varions la forme du signal ainsi que le graphe de routage correspondant. Par conséquent, l'outil de routage essaie de définir le taux de multiplexage et les groupes de signaux qui sont transmis ensemble via les mêmes fils physiques.
Le choix des signaux à regrouper influe considérablement sur la performance de l'outil de routage. De l'autre côté, une fois les groupes de signaux sont déterminés, la gestion des canaux de communication logique doit être assurée par un matériel spécifique, appelé IPs de multiplexage. Ce matériel supplémentaire comporte des éléments mémoires et des portes combinatoires qui doivent être ajoutés de part et d'autre du canal de communication physique (sur le FPGA émetteur et sur le FPGA récepteur) ce qui peut causer des problèmes de routage à l'intérieur du FPGA lui-même.

1.1.2 Problème de routage intra-FPGA

La surface d'un FPGA inclut les ressources de routage et les ressources logiques. Contrairement à un circuit intégré à application spécifique (ASIC), 80% de la surface totale d'un FPGA est dédiée aux ressources de routage [26]. Cependant, il est parfois impossible de router tous les signaux d'un circuit malgré que le nombre de blocs logiques ne dépasse pas la surface logique disponible sur ce FPGA. En effet, Ce problème de routage est essentiellement dû à la congestion : une répartition non étudiée des blocs logiques, peut créer, d'une part, des zones dans lesquelles les ressources de routage sont fortement demandées et qui dépassent largement les ressources disponibles, et d'autre part des zones légèrement congestionnées la ou les ressources ne sont pas toutes exploitées.
Pour remédier à ce problème, des travaux ont été faits pour une meilleure gestion des

ressources de routage. Des solutions architecturales ont été proposées visant à agir sur la taille de la largeur du canal de routage tout en la rendant variable selon le besoin de l'application. Dans [27], les auteurs ont estimé que la congestion est généralement localisée au centre du FPGA. Pour cette raison, ils ont proposé d'augmenter la largeur du canal seulement au milieu du FPGA et d'utiliser des canaux qui sont moins larges dans reste du circuit. Cette technique n'avait pas donné des résultats suffisants pour justifier l'effort supplémentaire nécessaire pour la conception physique d'un tel FPGA. Une technique itérative qui a été proposée et qui vise à ajuster, durant un certain nombre d'itérations, le placement des blocs logiques selon l'information sur la congestion crée par le routage [28]. Cette technique est très gourmande de point de vue temps de compilation puisque le placement et le routage se refont à chaque itération.
Notre objectif est donc de développer une technique de placement intra-FPGA, qui permet de placer les blocs logiques en se basant sur l'estimation de la congestion mise à jour à fur et à mesure de l'évolution du placement. Cette technique permettra de répartir les blocs logiques dans toute la surface de l'FPGA tout en évitant de créer des zones congestionnées dans un temps relativement acceptable et ainsi, améliorer la routage du circuit.

1.1.3 Limitation des benchmarks de test

Les techniques de multiplexage développées dans le cadre de cette thèse seront évaluées à travers une série de tests en utilisant un ensemble de benchmarks. Par contre, le plus grand défi était de trouver des benchmarks spécifiques qui nous permettent de valider et évaluer ces techniques. En effet, nous cherchons des benchmarks suffisamment grands et dont la taille dépasse largement la capacité des FPGAs actuels, afin de pouvoir les tester sur une carte multi-FPGAs. D'autre part, nous nous intéressons aux benchmarks hétérogènes qui contiennent un mixe de différents composants. Ces circuits hétérogènes permettent de bien évaluer le degré d'adaptabilité et de flexibilité de nos algorithmes. La dernière caractéristique des benchmarks requis c'est la testabilité. En d'autres termes, nous devons toujours avoir la controlabilité et l'observabilité du circuit sous test (application software exécutée par le matériel).
En menant une recherche sur ce qui existe, nous n'avons pas trouvé de benchmarks qui répondent à nos besoins. En effet, la première initiative de génération de benchmarks de test a été faite par CBL [1] [29, 30] et MCNC [2][31]. Ces circuits sont des simples fonctions (pas d'application software) qui ne sont pas suffisamment grands pour cibler des opérations de partitionnement et de multiplexage. En fait, le plus grand circuit généré est le s38584 et il contient seulement 2904 blocs logiques(LUT) [32].
Plus récemment, des chercheurs ont développé un programme de génération de benchmark

1. Collaborative Benchmarking Laboratory, North Carolina State University,Raleigh, NC.
2. Microelectronics Center of North Carolina

GNL [3] [33] qui permet de créer des netlists ayant des comportements assez réalistes. Ce programme est basé sur la règle de Rent [34] pour contrôler la complexité des interconnexions entre les taches. En effet, l'utilisateur définit le nombre de blocs logiques, le nombre de flip flop, la profondeur combinatoire, le nombre d'entrées/sortie et aussi l'exposant rent. Par la suite, le programme GNL suit une approche bottom-up. Donc il commence par établir les connexions entre un certain nombre de blocs logiques ce qui en résulte la formation de clusters. Ces clusters eux-mêmes vont être connectés ensemble jusqu'à ce que tous les clusters seront combinés en un seul circuit. A chaque niveau, le programme vérifie si l'exposant rent fixé par l'utilisateur est respecté. L'inconvénient de ce générateur réside dans le fait que les circuits générés sont composés de ressources logique homogènes (pas de DSP, Ram..), en plus aucune application n'est générée pour tester ces circuits.
En 2005, la suite de benchmarks IWLS a été publiée dans le workshop IWLS (International Workshop on Logic and Synthesis)[35]. Elle contient des circuits déjà publiés dans des conférences précédentes, des circuits publiés par des communautés de développeurs de code libre et encore par des circuits industriels. Malgré la diversité et la taille relativement grande de ces circuits, mais ils n'offrent aucune solution de testabilité durant l'implémentation sur la carte multi-FPGA.
Notre objectif c'était donc de développer un générateur de benchmarks, qui permet, à l'aide d'une description architecturale simple du benchmarks, de générer le circuit demandé modélisé avec le langage de description matérielle VHDL. Le générateur utilise un ensemble de composant de la bibliothèque Soclib [36], ce qui donne aux benchmarks un aspect réel semblable à celui des circuits industriels.

1.2 Plan

Le présent manuscrit est réparti en chapitres. Le premier chapitre commence par une introduction sur le prototypage matériel ainsi que la problématique posée.
Dans le deuxième chapitre nous comparons les différentes méthodes de vérification : la simulation, l'émulation et le prototypage matériel. Comme ce dernier présente le meilleur compromis entre le temps de mise en œuvre et le temps d'exécution, nous présenterons quelques plateformes de prototypage commerciales qui existent sur le marché.
Le troisième chapitre décrit le flot de prototypage complet tout en mettant en œuvre les parties qui sont concernées par les travaux de cette thèse. Pour ces parties, nous donnerons une présentation détaillée sur les différentes techniques proposées dans la littérature.
Par la suite, nous détaillons l'environnement proposé pour la génération de netlists de circuits complexes qui serviront de benchmarks pour valider les outils et les techniques qui seront développés au cours de cette thèse. Cette étude couvrira les volets matériels et logiciels. Deux types d'architectures matérielles seront proposés : des architectures multi-processeurs et des architectures multi-coprocesseurs.

3. gnl is the acronym for Generate NetList

Le cinquième chapitre dresse un état de l'art des solutions envisageables pour assurer le multiplexage des signaux. Dans un premier temps, nous proposons un outil de routage itératif qui tente de réduire le taux de multiplexage durant un certain nombre d'itérations. Le routage en lui-même est assuré par l'algorithme de routage Pathfinder [37] qui est largement utilisé par les chercheurs académiques et industriels. Cet algorithme servira comme point de départ pour les techniques de routage développées durant cette thèse. Des adaptations adéquates seront faites pour cibler un réseau de routage inter-FPGA. Dans une deuxième partie, nous essayons de déterminer la meilleure forme de signal avec laquelle il doit être routé. Pour cela, nous proposons des scénarios de test afin de sélectionner celui qui donne la fréquence de fonctionnement la plus performante.
Le sixième chapitre présentera les résultats expérimentaux des techniques proposées. Nous mettrons en valeur la performance de ces techniques à travers des comparaisons avec avec les méthodes proposées dans l'état de l'art ainsi qu'avec celles de l'outil industriel Certify.
Dans le dernier chapitre, une description détaillée des IPs de multiplexage sera présentée. Ces IPs sont insérés dans les parties émettrices et réceptrices d'un canal de communication. Ces IPs inclus des composants spécifiques appelés SERDES pour assurer la sérialisation/déserialisation des données à transmettre. Dans une deuxième partie, nous proposons notre algorithme de placement basé sur l'estimation de congestion. Cet algorithme sera implémenté dans un outil de placement développé dans le LIP6 et destiné aux FPGAs de la CEA puisque nous n'avons pas la possibilité d'agir sur les outils de placement et routage de Xilinx ou Altera.
Enfin, une conclusion résumera les différents points traités dans ce manuscrit, et abordera les perspectives relatives à ces travaux.

Chapitre 2

État de l'art : Plateformes matérielles

Sommaire

2.1 Introduction

Le prototypage matériel a été proposé pour une vérification rapide des systèmes sur puce avant d'atteindre la phase de fabrication. Il permet de réduire remarquablement le temps de conception d'un circuit en guidant le concepteur à faire des choix stratégiques de son

système. Dans ce chapitre, nous allons commencer par une comparaison du prototypage matériel par rapport à d'autres méthodes de vérification. Par la suite, il est nécessaire d'examiner avec plus de détails l'architecture de l'élément de base des plateformes de prototypage qui est le "FPGA". La dernière partie de ce chapitre est dédiée à la présentation de quelques exemples de plateformes matérielles industrielles. Ces plateformes sont classées sous forme de deux types : plateformes complètes fournissant la carte multi-FPGA ainsi que le flot logiciel d'implémentation sur la carte. Mais aussi un second type qui regroupe tous les industriels qui commercialisent des plateformes matérielles, sur étagère ou sur mesure, sans aucun logiciel qui automatise le flot d'implémentation.

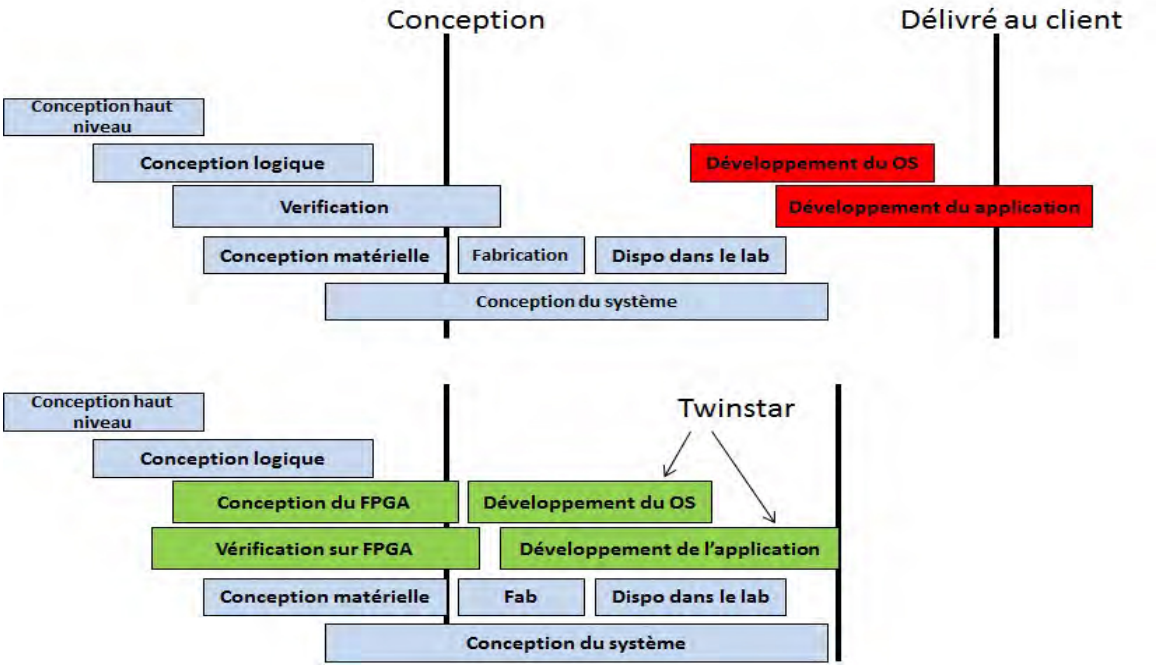


FIGURE 2.1 – Réduction du temps de conception d'un SoC en permettant la vérification matérielle et la validation logicielle[1]

2.2 Vérification des SoCs

Les concepteurs doivent s'assurer de la conformité de leur produit à ses spécifications à chaque étape de conception avant d'atteindre la phase de fabrication. Par conséquent, la vérification peut occuper jusqu'à 80% du cycle de réalisation du système sur puce [4] tout en diminuant le temps de mise sur le marché du produit final comme le montre la figure 2.1. En effet, traditionnellement, l'étape de validation ne peut commencer réellement qu'après la fabrication, le test et la mise en place du circuit dans le laboratoire. L'impacte de la vérification consiste à permettre la validation du matériel mais aussi du logiciel très tôt dans le cycle de développement du SoC ce qui permet d'accélérer le temps de la mise

sur le marché.
Le recours à la vérification a augmenté à partir des années 90 la où plusieurs erreurs de conception ont été découvertes et médiatisées, entre autres, celle du Pentium d'Intel en 1994 (bug FDIV du Pentium) [38], erreur du démarrage de la machine équipée d'un Pentium III [39], TLB bug du Phenom de AMD [40]. Ces erreurs ont coûtées des centaines de millions de dollars à l'époque. De ce fait, plus vite l'erreur est détectée, plus son coût de correction est faible. Généralement, il existe plusieurs méthodes de vérification matérielle que l'on peut citer essentiellement la simulation, l'émulation et le prototypage matériel.

2.2.1 Simulation logique

La simulation emploie des méthodes d'exécution et de calcul sur ordinateur. Cette technique offre une grande observabilité, une grande souplesse et une grande flexibilité. Elle peut intervenir dans plusieurs niveaux puisque pour un même système, plusieurs modèles à différents niveaux d'abstraction peuvent être utilisés. Plus le modèle est précis, plus les calculs pour la simulation sont nombreux et par conséquent, plus l'exécution est lente. Dans le domaine de conception des circuits monopuces, le système peut être modélisé selon 6 niveaux d'abstraction :

- Le niveau spécification fonctionnelle modélise le comportement global du système sans aucune précision vis à vis de sa réalisation finale. Ici, on travaille à un très haut niveau d'abstraction, une simulation à ce niveau permet de très rapidement simuler une spécification fonctionnelle et permettra, par son analyse, de mettre en évidence les besoins du système.
- Le niveau architectural modélise le système comme un ensemble de modules travaillant en parallèle et communiquant entre eux. A ce niveau, les différentes tâches du système sont allouées à des sous-systèmes. Chaque sous-système est modélisé au niveau fonctionnel. La granularité concernant les interactions est au niveau transactionnel (TLM). Ce type de simulation est particulièrement utile pour l'exploration d'architecture et le développement des parties logicielles du système.
- Le niveau micro-architecture correspond à la même simulation qu'au niveau architectural sauf qu'ici, les interactions entre sous-systèmes ne sont plus des transactions mais des signaux. La précision du modèle est donc au cycle d'horloge prêt au niveau de la communication entre sous-systèmes. Ce niveau de modélisation permet la réalisation de premières mesures de performances ainsi que le développement des pilotes de bas niveau («drivers») des logiciels embarqués.
- Le niveau RTL modélise un circuit comme un ensemble de registres et de relations logiques entre registres. Ce modèle est à bas niveau d'abstraction, le système entier est simulé au cycle d'horloge prêt. Ce niveau est particulièrement utilisé pour la mise au point des sous-ensembles matériels qui composent le système. De nos jours, plusieurs outils permettant de faire la simulation à ce niveau d'abstraction, à noter le Modelsim [41] et le Isim de Xilinx [42].

- Le niveau porte logique décrit le système complet comme un assemblage de portes logiques. Ce niveau est obtenu après synthèse. De nos jours, les outils permettant le passage du niveau RTL au niveau porte logique sont automatisés et suffisamment fiables pour ne pas avoir à travailler à ce niveau pour la conception d'un ASIC. Cependant, dans les flots d'émulation qui nécessitent eux aussi une phase de synthèse, le niveau de fiabilité est moins élevé et il s'avère parfois nécessaire d'effectuer des simulations à ce niveau pour trouver une erreur de synthèse.
- Le niveau analogique est le plus bas niveau d'abstraction utilisé en simulation. A ce niveau existent des outils d'extraction de paramètres électriques à partir du plan de masse. On travaille ici avec des modèles précis des transistors, dépendant de la technologie utilisée.

Pour conclure, la simulation est donc utile à toutes les phases de conception. Cette technique est très efficace de part sa grande souplesse, sa grande flexibilité, observabilité, contrôlabilité et un temps de mise en œuvre souvent court. Cependant, plus on avance dans le processus de conception et plus la précision du modèle nécessaire augmente, ce qui implique plus de calculs et donc une vitesse d'exécution moindre. La simulation trouve donc ses limites lorsqu'il faut jouer de longues séquences de tests à un bas niveau d'abstraction.

2.2.2 Émulation matérielle

L'émulation utilise des modèles physiques qui imitent le comportement matériel. Les modèles utilisés offrent une grande observabilité du système, proches de celles des simulateurs, mais leur vitesse d'exécution est beaucoup plus rapide. Un émulateur pourra fonctionner à 1MHz là ou une simulation tourne à 10Hz. Les émulateurs puissants sont souvent à base de processeurs, mais il existe encore des machines qui sont à base de circuits FPGAs.

2.2.2.1 Emulateurs à base de processeurs

Le marché de ce type d'émulateurs est dominé par Cadence avec son émulateur Palladium [43]. Présent sur le marché depuis Janvier 2002, l'émulateur Palladium est basé sur l'utilisation parallèle de plusieurs dizaines de milliers de microprocesseurs spécialement conçus pour les besoins de l'émulation. La plus petite machine a une capacité d'émulation relativement grande. En combinant plusieurs machines, on peut étendre la capacité d'émulation jusqu'à deux billions de portes logiques. Cette machine offre des fréquences de fonctionnant allant jusqu'à 1GHz. Toutes les techniques d'émulation et de co-émulations sont supportées. La capacité de déboguage est excellente, comparable à celle des solutions concurrentes. Un des gros avantages de Palladium est d'être une machine multi-domaines et multi-utilisateurs, c'est à dire que plusieurs circuits peuvent être émulés en même temps sur la même machine. Cela permet d'optimiser l'utilisation de la machine. La figure 2.2 représente la machine Palladium capable de supporter jusqu'à 256M de portes logiques. A

part sa grande taille, cette machine est très onéreuse avec un prix de l'ordre de 12 millions de dollars.

FIGURE 2.2 – Emulateur Palladium supportant 256M de portes

2.2.2.2 Emulateurs à base de circuits FPGA

Deux grandes industries qui domine le marché des émulateurs à base de circuits FPGA : Synopsys avec son émulateur Zebu [44] et Mentor graphics avec son émulateur Veloce [45].

La solution Zebu de Synopsys : La société Eve a proposé une famille d'émulateurs nommée ZeBu (Zero Bug) avant d'être achetée par Synopsys. Cette famille d'émulateurs est un ensemble de circuits FPGA standards. La première machine fut le modèle ZeBu-ZV utilisant deux FPGAs Xilinx Virtex2, la carte s'installant sur le bus PCI d'un PC. Les dernières machines apparues sont utilisées pour la vérification des SoCs ayant une taille logique qui varie entre 20 et 200 millions de portes. Quelle que soit la machine Zebu considérée, ces machines ont toutes, les mêmes capacités de déboguage, à savoir, une visibilité sur l'ensemble des registres et mémoires du circuit. L'utilisateur a le choix entre effectuer un déboguage dynamique (sélection à la volée des signaux observés) et lent (fréquence à quelques kilohertz) ou un déboguage statique (sélection à la compilation des signaux observés). En mode statique, le déboguage n'altère pas la vitesse d'exécution de la plateforme mais, seule une courte fenêtre temporelle est observable. Les machines ZeBu supportent l'ensemble des techniques d'émulation et de co-émulation et offrent une fréquence d'émulation très élevée, allant jusqu'à 100MHz.

La solution Veloce 2 de Mentor qui propose des machines d'émulation à base de circuit FPGAs spécifiques. La capacité logique de ces émulateurs varie entre 16 millions et 2 billions de portes logiques. Cet émulateur cible les SoCs multi-puces ainsi que les grands

processeurs et GPU. Pour mettre un œuvre un tel SoC, des outils de partitionnement automatiques sont disponible ainsi que des outils de déboggage offrant une bonne visibilité pour les signaux au cours de l'exécution. La fréquence de fonctionnement peut aller jusqu'à 100MHz.

2.2.3 Prototypage matériel

Le prototypage est une technique proche de l'émulation, consistant à réaliser un prototype du système avec des modèles physiques très rapides (FPGA) mais qui n'offrent quasiment pas d'observabilité. Le prototypage vise à vérifier la fonctionnalité du système final, à aider au développement des logiciels embarqués. Cette technique est celle qui offre la meilleure vitesse d'exécution (jusqu'à plusieurs dizaines de méga hertz). Elle intervient, en général, lorsque le matériel a atteint un certain niveau de maturité. De plus, il arrive que l'environnement du circuit soit impossible à modéliser. Dans ce cas, il faut vérifier le système dans son environnement, en temps réel. Le prototypage est la seule technique offrant une vitesse d'exécution suffisante pour couvrir ce besoin. La nuance entre émulation et prototypage se situe au niveau de la mise en œuvre et de la capacité de déboguage des machines. Les émulateurs sont avant tout des machines conçues pour réaliser un déboguage matériel rapide et efficace. Elles ont des flots de mise en œuvre assez rapides, de l'ordre de quelques heures. Par contre, pour un flot de prototypage n'est pas automatisé, le partitionnement d'un circuit entre les différents FPGAs de la plateforme de prototypage n'est pas aisé et engendre des temps de mise en œuvre assez longs, pouvant atteindre plusieurs mois.
Au cours de la conception d'un système sur puce, la question qui peut se poser est : pourquoi faire le prototypage ? Une partie de la réponse peut être présentée dans les points suivants :

- Seul le prototypage à base de circuits FPGA fournit à la fois la vitesse et la précision nécessaire pour tester correctement de nombreux aspects du design. Le logiciel peut être validé sur un système virtuel avec une performance encore plus élevée, mais ça serai au dépend de la précision qui vient du fait d'employer le RTL réel.
- La vérification d'un SoC est souvent très compliquée parceque l'état actuelle du système dépend de plusieurs variables, entre autres, son état précédent, ses entrées. Le prototypage permet d'exécuter le système en temps réel ce qui permet d'apercevoir les effets immédiats des conditions temps réel, les entrées et le changement des sorties.
- Au début d'un projet, les concepteurs doivent prendre des décisions fondamentales à propos du choix de la technologie de conception de la puce, la performance, la consommation d'énergie... Certains de ces choix sont mieux réalisés en utilisant des outils de modélisation algorithmique, mais quelques expériences supplémentaires pourraient également être effectuée en utilisant les FPGA. Par conséquent, un circuit FPGA est utilisé pour prototyper une idée à partir de sa modélisation en RTL. Une fois dans le FPGA, les informations préliminaires peuvent être recueillies pour aider à conduire

l'optimisation de l'algorithme et l'architecture éventuelle du SoC.

- Un système de prototypage est considéré comme autonome. En effet, un FPGA peut être configuré, par exemple à partir d'une carte mémoire flash EEPROM ou autre support autonome, sans supervision par un PC hôte. Le prototype peut donc tourner de façon autonome et être utilisé pour tester la conception de SoC dans des situations tout à fait différentes de celles fournies par d'autres techniques de vérification, comme l'émulation, qui reposent sur l'intervention d'une machine hôte. Dans les cas extrêmes, le prototypage pourrait être pris complètement hors du laboratoire et dans les environnements de la vie réelle sur le terrain pour servir comme étant une démonstration .

2.2.4 Synthèse : Comparaison entre les différents types de vérification

La figure 2.3 présente les caractéristiques des 3 méthodes de vérification matérielle précédemment citées. La simulation offre une grande souplesse, simplicité d'utilisation, grande

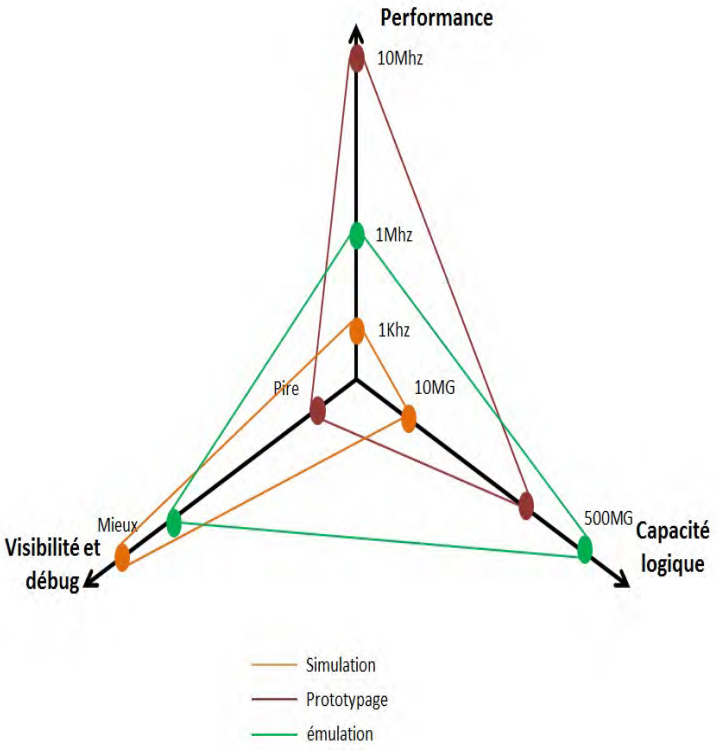


FIGURE 2.3 – Caractéristiques des différentes techniques de vérification matérielle

observabilité, un temps de mise en place assez court. De part ses qualités, la simulation est l'outil fondamental dans la conception des systèmes sur puce. Cependant, cette technique a une faible vitesse d'exécution au niveau RTL ce qui limite son utilisation.
L'émulation, quant à elle, offre une souplesse, observabilité et une vitesse d'exécution supérieure par rapport à la simulation moyennant un flot de mise en œuvre plus ou moins

complexe et un important budget de fonctionnement.
Enfin, le prototypage matériel, avec un temps modéré de mise en œuvre et d'exécution, offre la meilleure solution aux développeurs pour réussir la conception de leurs produits dans les plus courts délais. En plus, une plateforme de prototypage est beaucoup moins chère qu'un émulateur matériel. Par conséquent, et dans un stade avancé dans la conception d'un système sur puce, le prototypage matériel permet d'anticiper le développement du logiciel et le circuit est prêt rapidement. En plus la capacité logique de la plateforme de prototypage est assez grande et peut atteindre 500 millions de portes logiques. Dans le reste de ce manuscrit, nous nous intéressons au prototypage matériel ainsi qu'aux plateformes de prototypage qui existent sur le marché. Mais en premier lieu, nous allons nous attarder sur la description de l'élément de base des plateformes matérielles qui est le FPGA

2.3 Architecture FPGA

Les plateformes de prototypage matériel sont conçues à base de circuits FPGA. Deux grands fabricants s'imposent sur le marché des FPGA qui sont Altera et Xilinx. L'architecture d'un FPGA, qui a beaucoup progressé ces dernières années, varie de l'un à l'autre tout en gardant l'aspect reconfigurable de ces circuits. En effet, un FPGA (Field Programmable Gate Array) est un circuit logique intégré qui peut être reprogrammé après sa fabrication, permettant ainsi d'implémenter plusieurs applications.
Un FPGA est formé d'un réseau de blocs logiques configurables connectés par un réseau d'interconnexion configurable. Récemment, un FPGA peut intégrer d'autres composants permettant d'améliorer sa performance.

2.3.1 Bloc logique de base du FPGA

Le bloc logique de base d'un FPGA, nommé BLE (Basic Logic Element) est composé d'une table de transcodage (LUT : Look Up Table) à K entrées et d'une bascule suivies d'un multiplexeur. Le BLE peut fonctionner soit en mode combinatoire si sa sortie est fournie par la LUT, soit en mode séquentiel si sa sortie vient de la bascule.
Une table de transcodage à K entrées (K-LUT) contient 2^K bits de configurations. Elle est capable d'implémenter n'importe quelle fonction booléenne ayant au plus K entrées. Des études ont été menées sur le nombre d'entrées des LUTs. Les auteurs de [46] [47] ont montré que le choix de 4 entrées est un bon compromis entre les performances du circuit et les contraintes des algorithmes de placement-routage, leur complexité et leur efficacité. La figure 2.4 montre la structure d'un bloc logique de base ayant 4 entrées. La 4-LUT utilise 16 bits SRAM (static random access memory) pour implémenter n'importe quelle fonction booléenne à 4 entrées.

Les éléments logiques de base peuvent être rassemblés en clusters hiérarchiques afin

de réduire la connectivité globale et favoriser une connectivité locale et rapide. Un cluster est caractérisé par le nombre de BLEs (N) qu'il contient et le nombre de ses entrées (I). Le nombre de BLEs N est typiquement entre 4 et 10 dans les FPGAs modernes. Chaque entrée d'un BLE peut être connectée à n'importe quelle entrée parmi les I entrées du cluster ou à n'importe quelle sortie des BLEs contenus dans le cluster. Le schéma d'un cluster est illustré par la figure 2.5.

Les FPGAs ont été longtemps réalisés avec seulement des blocs logiques configurables à base de LUT. Aujourd'hui, elles peuvent contenir aussi des macro-blocs tels que de larges mémoires RAM et des cœurs de microprocesseurs.

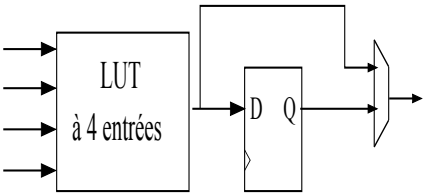


FIGURE 2.4 – Bloc logique de base (BLE)

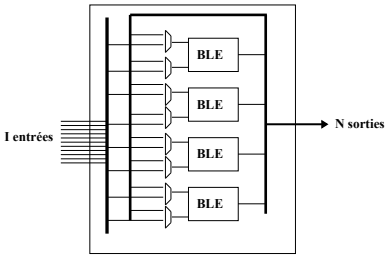


FIGURE 2.5 – Cluster de N blocs logiques

2.3.2 Ressources de routage

Les ressources d'interconnexions sont utilisées pour connecter les blocs logiques pour définir ainsi la topologie d'un FPGA. Il existe plusieurs topologies entre autres l'architecture arborescente et l'architecture matricielle. Cette dernière est une architecture populaire largement employée dans les FPGAs commerciaux tels que les FPGAs Altera Stratix-V [48] et Xilinx Virtex- II [49], et largement utilisée dans la recherche académique. Pour cette raison, nous nous somme intéressée à la description de cette topologie.
Une architecture FPGA matricielle (en anglais : Mesh) [2], appelée aussi architecture de type îlot (en anglais : Island style FPGA), est composée de blocs logiques configurables placés régulièrement en deux dimensions et formant une structure matricielle, comme l'illustre la figure 2.6. Les lignes d'interconnexion, entourant les blocs logiques, sont organisées en lignes et en colonnes, formant ainsi des canaux de routage horizontaux et verticaux.

L'architecture de routage est composée de trois éléments essentiels : les boîtes de connexion (connection blocks), les boîtes de commutation (switch box) et les canaux de routage.

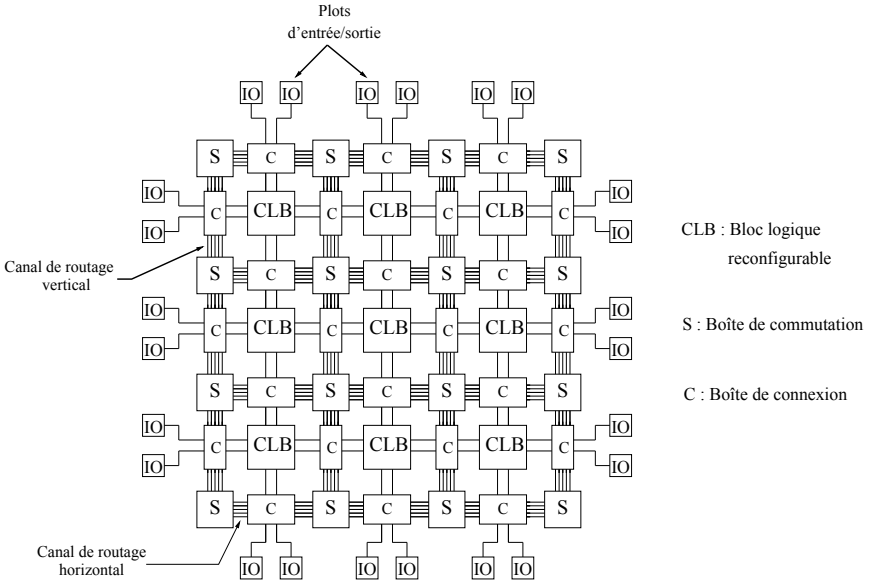


FIGURE 2.6 – Architecture FPGA matricielle

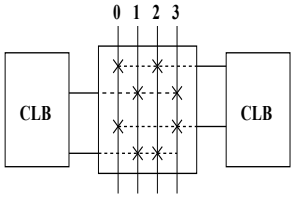


FIGURE 2.7 – Boîte de connexion avec une flexibilité Fc = 2

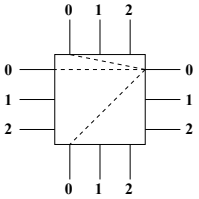


FIGURE 2.8 – boîte de commutation avec une flexibilité Fs = 3

Les boîtes de connexion sont utilisées pour connecter les entrées et les sorties d'un bloc logique aux canaux de routage qui lui sont adjacents via des connexions programmables. Elles diffèrent par leurs topologies et leurs flexibilités. La flexibilité est déterminée par deux paramètres $F_{c_{in}}$ et $F_{c_{out}}$ qui définissent le nombre de fils de routage de la boîte de connexion auxquels les entrées et les sorties d'un bloc logique peuvent être connectées respectivement.
La figure 2.7 montre un exemple de boîte de connexion, dans laquelle une connexion programmable est représentée par une croix. Dans cet exemple, chaque entrée ou sortie d'un bloc logique peut être connectée à deux fils de routage verticaux. La flexibilité F_c est ainsi égale à 2.

Les boîtes de commutation sont placées à l'intersection des canaux de routage verticaux et horizontaux. Elles permettent de connecter des fils de routage appartenant à deux canaux de routage adjacents. Il existe plusieurs topologies de boîte de commutation, à

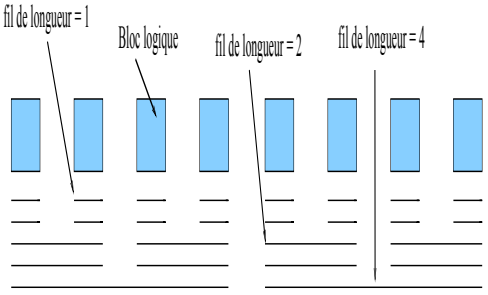


FIGURE 2.9 – Canal de routage composé de fils de différentes longueurs [2]

savoir les topologies disjoint [50], wilton [51] et universel [52]. Selon cette topologie, chaque fil de routage se trouvant sur un côté de la boîte de commutation peut être connecté à quelques fils ou à tous les fils de routage des trois autres côtés.
La flexibilité de la boîte de commutation est déterminée par le paramètre F_s. Ce paramètre définit le nombre de fils de routage auxquels un fil de routage incident à la boîte de commutation peut être connecté. Un exemple de boîte de commutation est illustré par la figure 2.8, dans laquelle chaque segment interrompu représente une connexion possible. Dans cet exemple, la flexibilité F_s est égale à 3.

Chaque canal de routage est formé de W fils de routage parallèles (cf. fig. 2.6), où W est appelé largeur de canal. La même largeur de canal est utilisée pour tous les canaux de routage. L'outil de routage a la responsabilité d'assurer que le nombre maximal des signaux routés à travers chaque canal de routage ne dépasse pas la largeur du canal. Si cette largeur est dépassée, le circuit qu'on cherche à implémenter sur le FPGA est dit non routable.
Un canal de routage peut contenir des segments de routage de différentes longueurs. La longueur d'un segment de routage est déterminée par le nombre de blocs logiques qu'il longe. La figure 2.9 montre un exemple d'un tel canal de routage. Les segments d'une longueur supérieure à 1 requièrent moins de commutateurs (switches) programmables, réduisant ainsi la surface et le délai du routage, mais réduisant également la flexibilité du routage ce qui peut rendre la tâche du routage du circuit plus difficile. Les FPGAs commerciaux tels que Xilinx utilisent des segments de routage de différentes longueurs.

2.3.3 Éléments spécifiques

En plus du tissu logique et celui de routage, un FPGA récent peut incorporer d'autres composants à citer :

- Des blocs de mémoire RAM synchrones simple ou double ports, qui peuvent être utilisés pour effectuer des opérations de traitement du signal, mais ils peuvent aussi être utilisés avec profit pour réaliser des opérations logiques.

- Des blocs DSP qui opèrent à des fréquences relativement élevées et représentent la solution la plus adéquate pour exécuter les applications multimédias. Les DSP dans les circuits FPGA sont câblés ce qui offre plus de rapidité.

2.4 Exemples de FPGA industriels

Dans cette partie, nous allons présenter quelques exemples d'architectures industrielles présentes sur le marché. Deux FPGA de Xilinx et un exemple de FPGA Altera seront décrits dans cette section.

2.4.1 Architecture du Virtex VI

La famille Virtex VI vient avec plus de performance, plus de capacité d'intégration et moins de consommation d'énergie par rapport aux familles plus anciennes de Xilinx.

2.4.1.1 Les blocs logiques

Un design synthétisé est implémenté dans des blocs logiques appelés "slices". Les slices contiennent des LUTs, des registres et des éléments de cascade.
Dans un FPGA Virtex VI, les LUTs sont utilisés pour implémenter des générateurs de fonctions ayant 6 entrées indépendantes et 2 sorties. Ces générateurs sont capables d'implémenter n'importe quelles fonctions booléennes ayant jusqu'à 6 entrées pour la première sortie et une deuxième fonction ayant jusqu'à 5 entrées pour la deuxième sortie du même LUT.
Deux slices sont combinées dans un bloc logique configurable (CLB). Les CLB sont arrangés en matrice dans le FPGA et ils sont connectés entre eux à travers des ressources d'interconnexion. Deux types de slices dans le Virtex VI :

- SLICEM est une slice capable d'implémenter des fonctions combinatoires, des petits blocs RAM ou un registre à décalage.
- SLICEL : Dans cette slice les LUTs sont utilisés pour implémenter seulement de la logique combinatoire.

La figure 2.10 montre la structure d'un SLICEM qui intègre 4 LUTs à 6 entrées (à gauche), 8 éléments de stockage (4 FFs et 4 FF/latches) et un élément de cascade.

2.4.1.2 Les blocs mémoire

Dans le Virtex VI, les ressources mémoires, appelées BlockRAM, varient entre 156 et 1064 blocs distribués sur tout le FPGA. La figure 2.11 montre l'architecture d'un BlockRAM du Virtex VI. Chaque BlockRAM est caractérisé par :

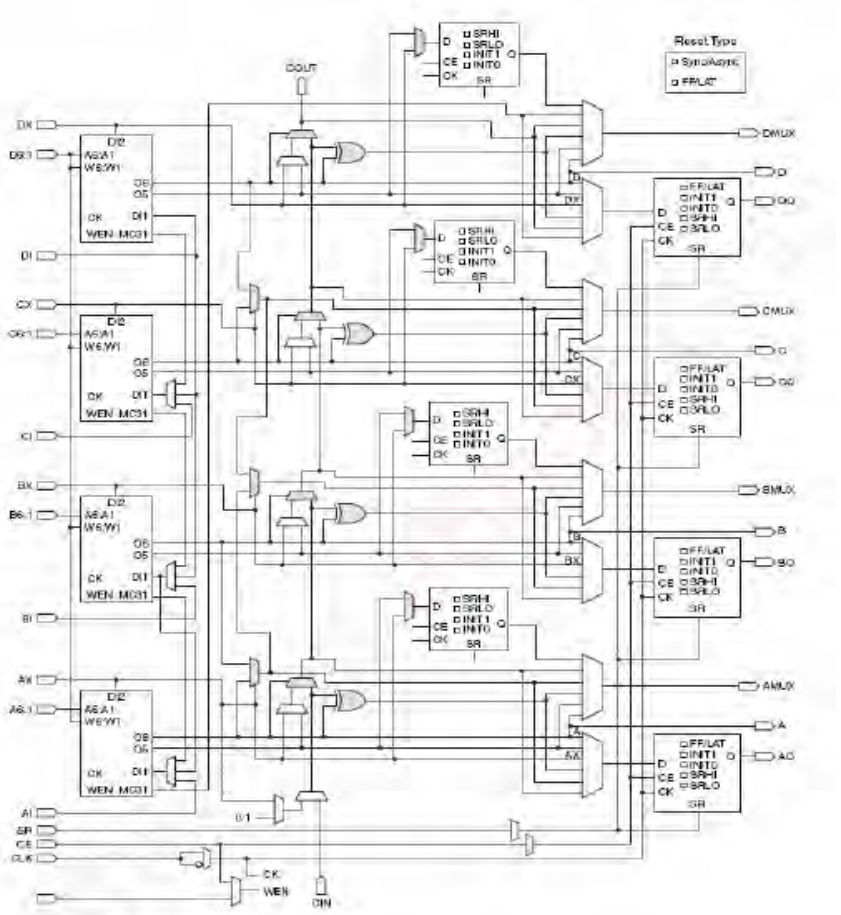


FIGURE 2.10 – Architecture d'une SLICEM

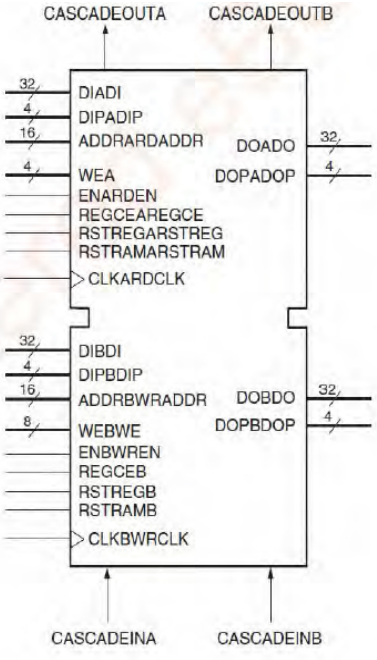


FIGURE 2.11 – Architecture d'un BlockRAM du Virtex VI

- La reconfigurabilité : les blocs sont synchrones et double ports de taille 36 Kbits. Ils peuvent être configurés en 32K x 1, 16K x 2, 8K x 4, 4K x 9 (ou8), 2K x 18 (ou 16), 1K x 36 (ou32), ou bien 512 x 72 (ou 64). Chaque bloc peut être configuré indépendamment des autres.
- Des opérations synchrones : Chaque bloc est capable d'implémenter n'importe quelle mémoire synchrone simple ou double ports. Dans le cas où il est configuré en RAM double ports, chaque port peut fonctionner à une horloge indépendante.
- Élément FIFO : sont utilisés en combinaison avec les BlocRAMs pour implémenter des points d'adresse et des drapeaux de prise de handshaking. La largeur et la profondeur des éléments FIFO peuvent être reconfigurables mais les parties de lecture et d'écriture doivent avoir la même largeur.

2.4.1.3 Les ressources DSP

Pour supporter les fonctions arithmétiques, le Virtex VI contient un nombre fini de blocs DSP appartenant à la famille DSP48E1. Ces DSP sont dédiés, configurables, performant et de petite taille pour garder la flexibilité du design. L'architecture d'un DSP consiste à un ensemble d'un accumulateur 48 bits et un multiplieur à deux entrées 25x18 bits. Les deux opèrent à une fréquence de 600MHz.

2.4.1.4 La gestion d'horloge

Dans le FPGA Virtex VI, les horloges sont générées par des unités configurables appelées CMT (Clock Management Tile) qui incluent deux modules de gestion mixte des horloges (MMCM). Il s'agit d'un module ayant plusieurs sorties de fréquence et basé sur des PLLs ayant des fonctionnalités plus avancées. L'architecture d'un CMT est représentée par la figure 2.12.

Pour garantir une meilleure distribution des horloges sur le circuit FPGA, le Virtex VI fourni 5 lignes d'horloge pour satisfaire les exigences reliées à la réduction du skew, propagation du délai...

2.4.1.5 Les ressources d'entrées/Sorties

Les FPGAs Virtex VI contiennent entre 240 et 1200 pins tout dépendant du package. Les pins sont organisés en bancs de 40 pins chacun. Un block SERDES réside dans la structure de chaque IO. Les entrées ont accès à un désérialisateur (convertisseur série-parallèle) ayant une largeur programmable de 2, 3, 4, 5, 6, 7, 8 ou 10 bits. Les sorties sont connectées aux sérialisateurs (convertisseur parallèle-série) ayant une largeur programmable de 8 bits et plus pour un transfert SRD(Single Data Rate) et plus que 10 bits pour une transmission DDR (Double Data Rate).

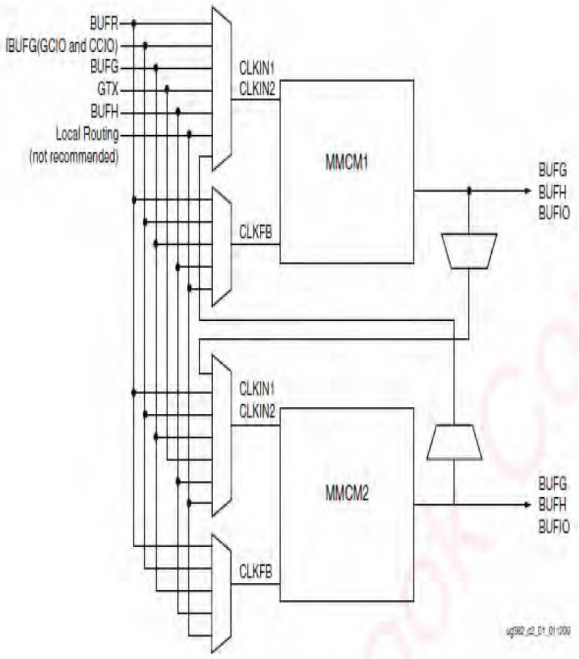


FIGURE 2.12 – Architecture d'un Block CMT du Virtex VI

2.4.1.6 Les transceivers

La transmission série Ultra-Rapide dans les Virtex VI est assurée par la présence des blocks transceivers nommés GTX. Chaque GTX est un block combiné de récepteur et transmetteur capables d'opérer à un flux de données entre 155Mb/s et 6.5 Gb/s. Le transmetteur et le récepteur sont indépendants et utilisent des PLL pour multiplier la fréquence d'entrée par un nombre variable entre 2 et 25.

2.4.2 Architecture du Virtex 7

Le Virtex 7 vient pour remédier au problème de la limitation technologique et annoncer une nouvelle technologie innovante. Avec son architecture de 2.5D, ce FPGA représente le compromis entre la haute intégration et la performance. Fabriqué en technologie CMOS 28 nm basse consommation, il se distingue par la présence de 80 blocs d'émission-réception GTX gérant des flux allant jusqu'à 13,1 Gbit/s, soit plus de 2 Tbit/s de bande passante sur la puce. Ces transceivers sont intégrés sur une puce distincte du cœur FPGA, ce qui améliore leur isolation et l'intégrité des signaux en entrée/sortie. La figure 2.13 montre une coupe de l'architecture d'un Virtex7.

La puce comporte 4 dies déposées sur un interposeur passif, chacun permet l'implémentation de 500.000 blocs logiques ainsi que d'autres ressources tel que les blocs RAM, DSP, les IOs. Ainsi, la capacité totale d'un Virtex7 est de l'ordre de 2 millions de blocs logiques (un bloc logique est l'équivalent d'un Lut 6 bits). Toutes ces ressources sont organisées sous forme de colonnes. Chaque die a sa propre horloge et son propre circuit de configuration.

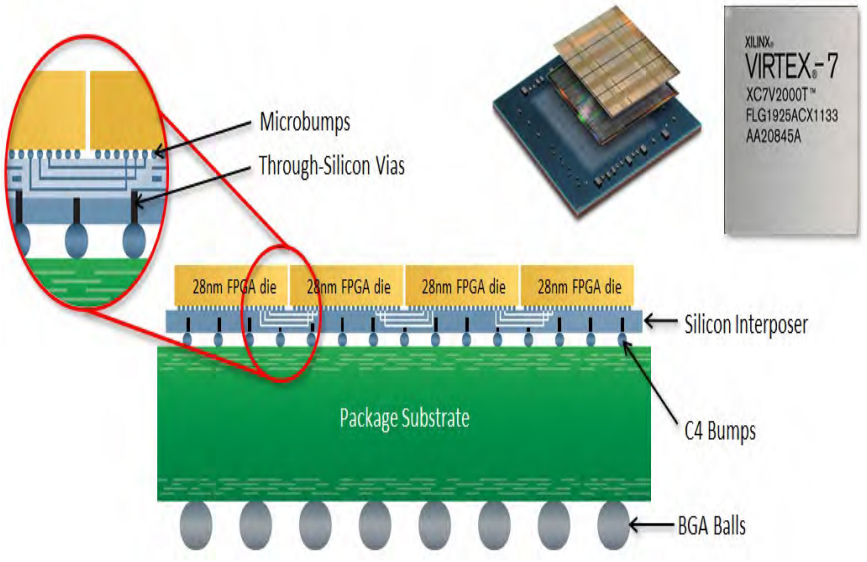


FIGURE 2.13 – Architecture d'un Virtex 7

L'interposeur fournit des dizaines de milliers de connexions die-à-die permettant une interconnexion ultra-haute avec une faible consommation d'énergie et une latence égale à 1/5 de celle des blocs IO standards. Ces connexions sont assurées à travers les micro-billes des différents dies. En effet, chaque die subit des étapes supplémentaires de traitement pour fabriquer des micro-billes qui fixent la matrice sur le substrat de silicium.
Des TSV sont combinés avec des perles de soudure (C4) permettent de monter le FPGA sur un substrat haute-performance en utilisant des techniques d'assemblage flip-chip.

2.4.3 Architecture du Stratix V

Dans cette section nous allons présenter une description de l'architecture d'un FPGA Stratix V GT de Altera. Contrairement aux trois autres familles de stratix V, notamment Stratix V GX, GS et E, ce FPGA est dédié aux applications nécessitant principalement une bande passante performante et une grande surface. Cet FPGA est équipé de transceivers de 28.05-Gbps et 12.5-Gbps. Le Stratix V GT est fabriqué avec la technologie 28nm permettant ainsi une haute densité d'intégration allant jusqu'à 622K éléments logiques et 939K registres. La figure 2.14 représente l'architecture d'un Stratix V. Le Stratix V d'Altera est un exemple industriel d'architecture matricielle. La structure logique se compose de LAB (réseau logique de blocs), blocs de mémoire, et des blocs DSP. Les LAB sont utilisés pour implémenter des blocs logiques, et sont symétriquement répartis en rangées et en colonnes à travers le tissu du circuit. Les blocs DSP sont conçus pour mettre en œuvre des multiplicateurs nécessitant une grande précision, et sont regroupés en colonnes. Les pins d'entrée-sortie(IOs) représentent l'interface externe du circuit. Les IOs sont situés tout au long de la périphérie du dispositif.
Chaque LAB consiste à 8 ALMs (Adaptive Logic Modules). Un ALM comporte deux Luts

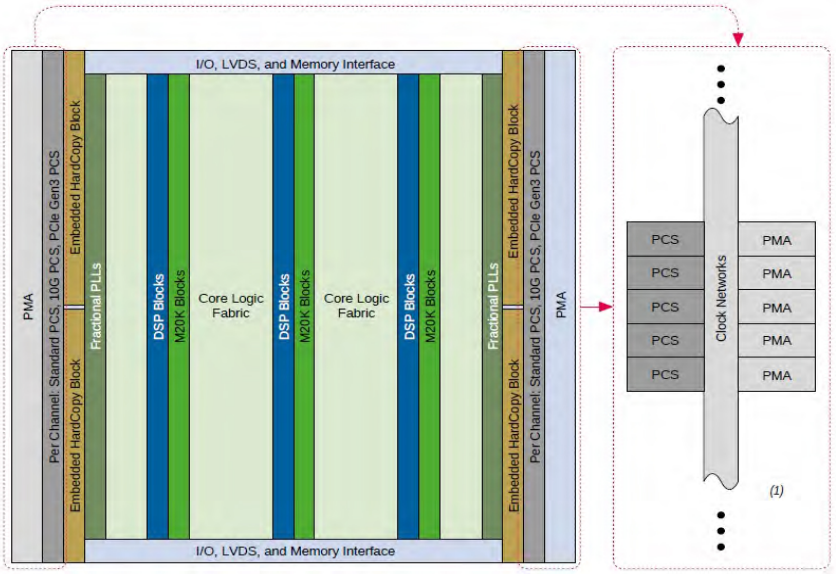


FIGURE 2.14 – Architecture d'un Stratix V

Adaptive (ALUT) ayant 8 bits en entrée. La construction d'un ALM permet l'implémentation de deux fonctions booléennes ayant chacune 4 bits. Un ALM peut encore être utilisé pour implémenter des fonctions ayant 6 et 7 entrées. Un ALM contient en plus deux registres programmables, deux additionneurs, une chaine de retenues et une chaine de registres. Les additionneurs et les chaines de retenues sont utilisés pour implémenter les opérations arithmétiques, quant à la chaine de registre, elle est utilisée pour construire les registres à décalage. La figure 2.15 représente l'architecture d'un LAB. Les interconnexions entre

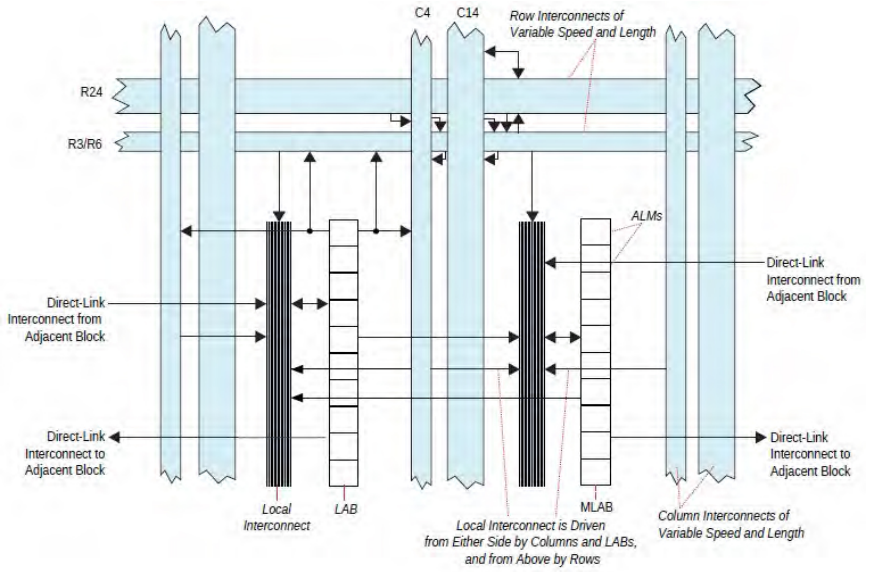


FIGURE 2.15 – Architecture d'un LAB de Stratix V

les LABs, RAMs, DSP et les IOs sont établies en utilisant une structure de connections multi-pistes. Cette structure consiste à un ensemble de segments de différentes longueurs

et vitesse. Les segments sont étendus sur des distances fixes dans les directions horizontales (les interconnexions des lignes) et verticales (les interconnexions des colonnes).

2.5 Plateformes industrielles de prototypage

Une plateforme de prototypage comporte deux volets : Un volet matériel représentant la carte multi-FPGAs et un volet logiciel correspondant à l'ensemble des outils CAO qui automatisent l'implémentation d'un système sur puce modélisé au niveau RTL sur une carte multi-FPGAs. Selon la nature de la solution présentée, les plateformes de prototypages sont décomposées en deux types : Plateformes complètes et plateforme matérielle uniquement.

2.5.1 Plateformes matérielles

Durant les années précédentes, des plateformes multi-FPGA sont de plus en plus proposées sur le marché du prototypage matériel. Deux familles de plateformes sont disponibles : des plateformes sur mesure et des plateformes sur étagère.

2.5.1.1 Plateformes matérielles sur mesure

Ce type de plateformes est extensible selon le besoin d'application. La taille et le nombre de FPGA par plateforme peut être modifié au cours du test. TWINSTAR est un exemple plateforme sur mesure qui a été proposé par IBM [1]. L'architecture de la plateforme consiste à un ensemble de plus de 28 cartes FPGA montées dans un backplane (base) actif pour former un système reconfigurable fortement couplé. Deux types de cartes qui existent : des cartes ajoutant de l'espace mémoire et des cartes fournissant de l'espace logique. La taille logique de la plateforme lorsque toutes les cartes "logiques" sont couplées est équivalente à 60 FPGA Virtex 5 LX330.
Le backplane offre une flexibilité d'interconnexion entre les différentes cartes. Le circuit de contrôle de tout le système est localisé au centre du backplane. Les cartes sont groupées dans 4 quadrants : est, ouest, nord et sud. La manière dont elles sont connectées définie la topologie du système (en étoile, point à point..).

2.5.1.2 Plateformes matérielles sur étagère

Ces plateformes matérielles sont standards et ne ciblent aucune application. Chaque utilisateur essaie de choisir la carte qui satisfait le plus les exigences de son système à prototyper en terme de taille logique, espace mémoire, connectivité...

DiniGroup est une compagnie américaine fournissant des plateformes de prototypage standards avec les FPGAs les plus performants et les plus grands de Xilinx ou Altera.

La plus grande plateforme peut supporter jusqu'à 130 millions de portes ASIC. D'autre part, Dinigroup ne propose aucun outil logiciel permettant l'automatisation du flot d'implémentation sur la carte matériel. Par contre, Dinigroup recommande l'utilisation de l'environnement ACE Compiler de Auspy [53] qui supporte toute les cartes comportant des FPGAs StratixIII, StratixIV et StratixV de Altera, Virtex5, Virtex6, et Virtex7 de Xilinx. ACE Compiler permet de mapper le système à vérifier sur la plateforme de prototypage cible. Il permet également de résoudre les problèmes liés au partitionnement, les contraintes timing..

Aldec Propose deux plateformes matérielles de prototypage :

- HES-7 avec ses deux virtex-7 2000T, offre plus que 24 millions de portes ASIC à part la capacité mémoire et les DSP. L'implémentation du circuit à prototyper sur cette carte ne demande pas trop d'effort de partitionnement puisque le nombre de FPGA est limité. Avec l'utilisation d'un backplane, HES-7 peut facilement accroître la capacité de prototype jusqu'à 96 millions de portes ASIC et peut inclure l'expansion des cartes filles.
- Aldec et Microsemi se sont réunis, offrant une solution innovante qui est la carte ProASIC. Cette carte est basée sur des FPGA Actel fabriqués avec la technologie flash au lieu de la technologie anti-fusible, qui est reprogrammable qu'une seule fois, offrant ainsi une capacité de reprogrammabilité infinie.

Gidel propose des plateformes intégrant un, deux ou quatre FPGAs Stratix de Altera. Aucune carte à la base de FPGA xilinx n'est fournie. En tant qu'un partenaire de Altera, l'utilisateur peut utiliser les outils de débogage fournis par Altera, à citer Quartus II et l'analyseur logique Signal Tap II, pour débogguer son circuit.

2.5.2 Plateformes de prototypage complètes

Récemment, les entreprises Synopsys et Cadence, leaders dans le domaine du prototypage, ont proposé des flots complets de prototypage matériel en addition avec leurs cartes matérielles multi-FPGAs. Des petites entreprises tels que S2C essaient de marquer leurs traces dans le même domaine.

2.5.2.1 La plateforme Confirma de Synopsys

Synopsys, leader mondial en logiciel et en IP pour la conception et la fabrication de semi-conducteur, présente sa plateforme de prototypage rapide étendue : Confirma. Cette plateforme, est une suite complète de produits pour le prototypage rapide incluant : des systèmes et des cartes de prototypage basées sur FPGA, des cartes d'interface et de mémoire, et le logiciel d'implémentation et de débogage. La plateforme Confirma fournit tous les éléments nécessaires pour mettre en œuvre un prototype rapide.

Les cartes HAPS La famille HAPS (High-performance ASIC Prototyping Systems) de cartes de prototypage rapide présente une diversité de cartes multi-FPGAs de plus haute performance pour la validation de système et le développement de logiciel embarqué. Cette famille est constituée d'un ensemble de cartes intégrant des FPGAs Xilinx. Ces cartes sont utilisées individuellement ou s'interconnectant les unes avec les autres. Cette solution est la plus flexible du marché, celle qui offre le plus de souplesse quant à l'interconnexion entre FPGAs. Cela se traduit par de très hautes performances en fréquence, jusqu'à 200MHz. Étant donné que l'on peut interconnecter autant de cartes que l'on veut, la capacité de la machine est illimitée. Cependant, plus on ajoute des cartes et plus les performances en vitesse vont diminuer.

Solution Logicielle d'implémentation Synopsys propose une suite d'outil CAO pour l'implémentation du système sur la carte multi-FPGA afin d'accélérer la mise en œuvre d'une plateforme de prototypage fonctionnelle.

- Certify : Il s'agit d'un logiciel de partitionnement fournissant une méthode rapide et facile pour implémenter les plus grands ASICs sur une carte de prototypage multi-FPGA. L'outil Certify fonctionne parfaitement avec les cartes HAPS et offre des solutions de partitionnement et de routage adaptées à l'aspect paramétrable et flexible du réseau d'interconnexion de cette famille de cartes. En effet, étant donné que le nombre de connexions entre les cartes/les FPGAs peut être paramétré à l'aide de connecteurs, le partitionnement effectué tient compte de cette flexibilité tout en offrant des solutions qui supposent l'augmentation du nombre de liaisons physiques entre les cartes/FPGAs déterminés. Cette fonctionnalité donne des solutions optimisées pour une implémentation sur les carte HAPS, par contre elle risque de donner des solutions irréalisables sur des cartes FPGA présentes sur le marché et ayant un réseau d'interconnexion figé.
- Identify Pro : C'est un logiciel de débogage avec la technologie de visibilité améliorée TotalRecall(TM).
- Synplify : Il s'agit d'un outil de synthèse logique pour des circuits FPGAs.

2.5.2.2 La plateforme de Cadence : Rapid Prototyping Platform

Contrairement aux cartes Haps, les cartes de prototypage proposées par Cadence sont basées sur des FPGAs Stratix-4 d'Altera supportant jusqu'à 30M de portes logiques avec une fréquence de fonctionnement allant jusqu'à 600MHz. Pour implémenter ce nombre d'éléments logiques, une solution logicielle est proposée afin d'accélérer la mise en œuvre. Son outil de partitionnement automatique permet de diviser le système prototypé sous forme de plusieurs partitions tout en tenant compte de la capacité de chaque FPGA.

2.5.2.3 La solution S2C

S2C propose des plateformes matérielles à base de Virtex 6 et Virtex 7 de Xilinx, mais aussi à base de FPGA de Altera, permettant ainsi une large marge de choix pour l'utilisateur [13]. D'autre part, il propose encore une solution logicielle permettant d'automatiser l'implémentation du circuit sur la carte matérielle. Cette solution est donnée par l'outil : TAI Player Pro Compile Module qui facilite considérablement le prototypage des SoC en particulier pour les circuits qui nécessitent un partitionnement sur plusieurs FPGAs. Avec une interface graphique simple d'utilisation, des tests peuvent être effectués dans une fraction du temps. TAI Player Pro est un front-end pour Altera, Xilinx et d'autres outils de synthèse ou de débogage. L'utilisateur peut rester dans un environnement commun tout en se déplaçant entre les flux d'Altera et Xilinx. Les signaux à analyser sont définis avant la synthèse de telle sorte que noms sont conservés à travers le flux de la compilation, même si le design est partitionné sur plusieurs FPGAs.

2.6 Conclusion

Ce chapitre a présenté en premier lieu le rôle du prototypage matériel et l'efficacité qu'il offre pour réduire le temps de conception d'un système en puce. Il permet d'anticiper le développement du logiciel et le circuit est prêt plus rapidement. Ensuite nous avons donné un aperçu sur les plateformes de prototypage qui existent sur le marché. Les solutions complètes ne sont pas nombreuses. Seuls les grands industriels fournissent des cartes matérielles multi-FPGA et l'ensemble des outils qui permettent l'implémentation du circuit sur la carte. Nous nous sommes attardés dans la description de la plateforme Confirma de Synopsys puisque nous comptons nous en servir dans la validation du flot proposé dans le cadre du projet PPR.

Chapitre 3

État de l'art : Flot logiciel pour plateforme Multi-FPGA

Sommaire

3.1 Introduction

A part la plateforme de prototypage matérielle, un ensemble d'outils CAO sont indispensables pour automatiser le flot entre la modélisation du système jusqu'à son implémentation sur la carte matérielle.
Dans ce chapitre, nous allons présenter le flot complet de prototypage tout en se focalisant sur les parties de routage inter-FPGA et de placement intra-FPGA. Nous allons dégager les contraintes relatives à ces deux parties tout en s'attardant sur les travaux de recherche dont l'objectif a été de résoudre les problèmes relatifs.

La figure 3.1 représente le flot traditionnel complet d'implémentation allant de la modélisation d'un système au niveau RTL jusqu'à son implémentation sur carte multi-FPGAs. Un autre flot plus spécifique a été proposé dans [54]. La différence entre le flot de la figure 3.1 (que nous allons détailler dans la section suivante) et le flot présenté dans [54] réside dans le fait que ce dernier utilise un partitionnement hybride qui résout le problème de partitionnement en considérant la netlist sous forme de portes logiques, mais applique la solution trouvée sur le circuit modélisé au niveau RTL . Ainsi, il atteint les deux avantages suivants :

- Une estimation exacte de la taille des partitions synthétisées au niveau de portes logiques
- la génération des partitions au niveau RTL.

Dans la section suivante, nous détaillons le flot représenté par la figure 3.1.

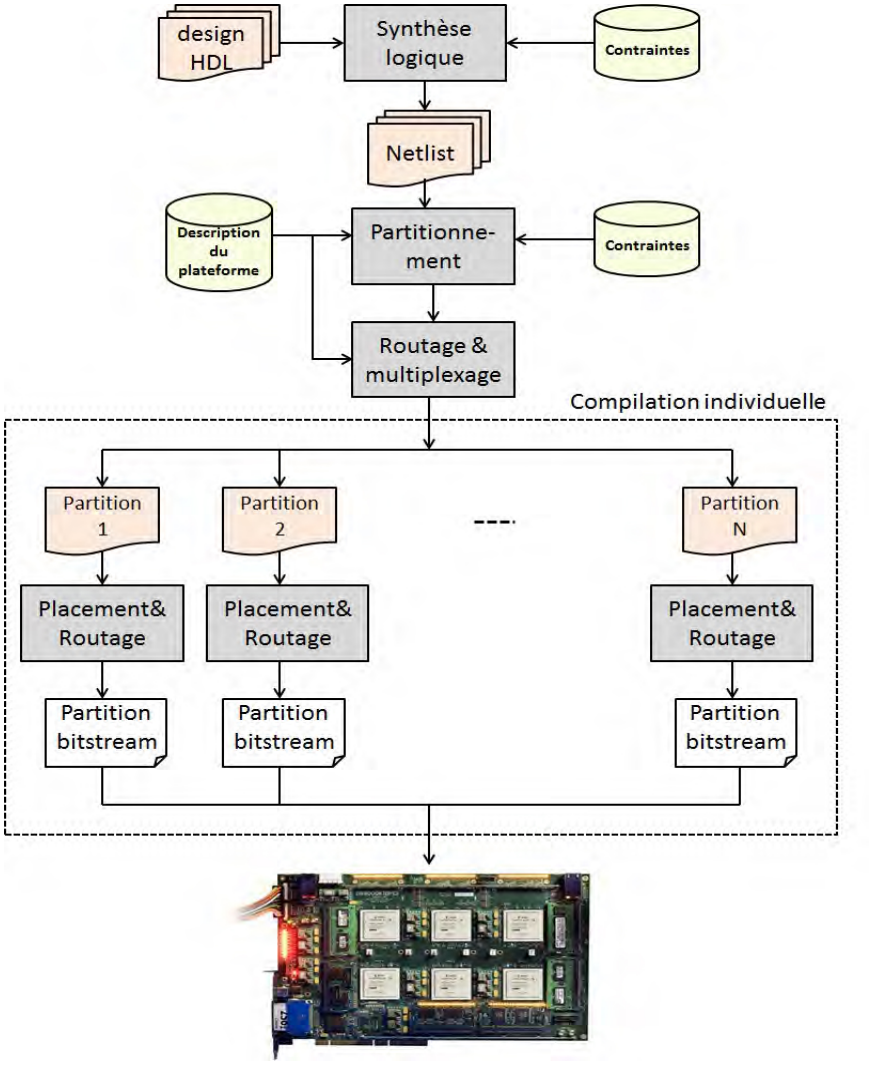


FIGURE 3.1 – Flot de prototypage matériel

3.2 Modélisation RTL du système sur puce

Dans un stade avancé de la conception d'un système sur puce, ce système est modélisé au niveau registre (RTL) pour décrire l'aspect comportemental de ce circuit. La description comportementale est généralement constituée d'un ensemble d'affectations concurrentes et d'un ensemble de processus séquentiels communicants. Elle définit le comportement d'éléments matériels interconnectés, tels que des automates d'états, des mémoires, des blocs combinatoires, des opérateurs arithmétiques et des registres. Cette description est écrite avec un langage de description de matériel (Hardware Description Langage) tel que VHDL ou Verilog.

3.3 Synthèse logique et mapping

L'étape de la synthèse consiste à transformer la description comportementale en une interconnexion de portes logiques et de bascules [55] [56]. Cette étape est indépendante de l'architecture FPGA. Par la suite une étape de mapping est effectuée et qui consiste à transformer cette représentation du circuit en une interconnexion de composants élémentaires d'une bibliothèque de la technologie cible.

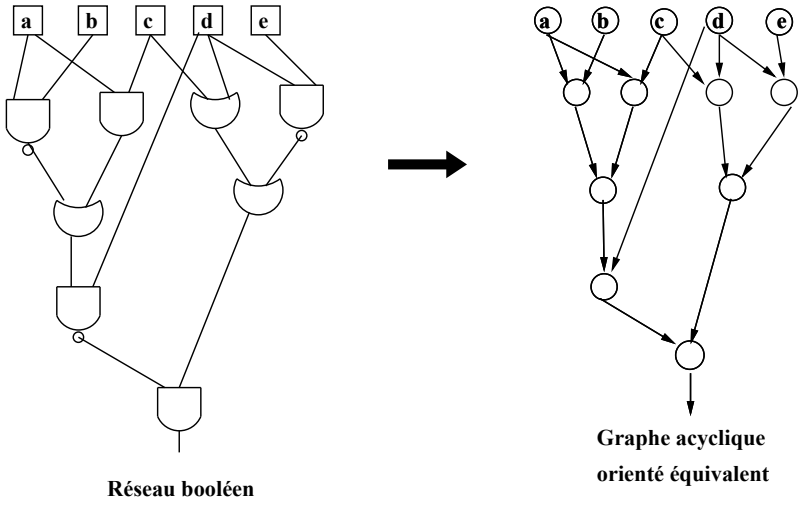


FIGURE 3.2 – Représentation du circuit avec un graphe acyclique orienté

Par la suite, une étape de projection structurelle succède l'étape de la synthèse. Son entrée est une description définissant le comportement d'un circuit sous la forme d'un réseau booléen.
Le circuit peut être représenté par un graphe acyclique dirigé (DAG : Direct Acyclic Graph), dans lequel chaque sommet représente une porte logique, un registre, une entrée primaire ou une sortie primaire du circuit. Les arcs du graphe représentent les connexions entre les différents sommets. Les entrées des registres sont considérées comme des sorties primaires

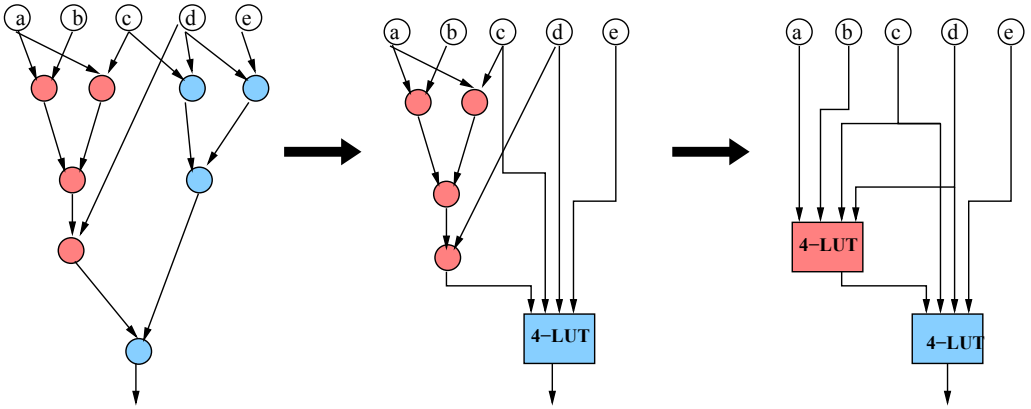


FIGURE 3.3 – Exemple de projection structurelle

du graphe et les sorties des registres sont considérées comme des entrées primaires du graphe. Un exemple de graphe acyclique dirigé est illustré par la figure 3.2.
La projection structurelle doit transformer cette représentation du circuit en une interconnexion de composants élémentaires d'une bibliothèque de la technologie cible. Dans le cas d'un FPGA à base de LUTs (Look Up Table), la projection structurelle consiste à transformer la représentation booléenne du circuit en une représentation à base de LUTs et de bascules. La figure 3.3 montre un exemple de projection structurelle d'un réseau booléen sur une librairie de LUTs à 4 entrées.
Plusieurs méthodologies de synthèse logique ont été proposées dans l'état de l'art. Traditionnellement, les outils de synthèse utilisent le flot Top-Down qui consiste à traiter le circuit à partir des composants appartenant au plus haut niveau d'hiérarchie. Cette propagation du haut vers le bas permet à l'outil de synthèse d'identifier l'arbre de dépendance entre les modules ce qui en résulte plus de d'itérations d'optimisation et par conséquent des résultats plus performants [57]. Par contre, cette méthodologie est très gourmande de point de vue temps d'exécution à cause du grand nombre d'itérations. Et pire, dans le cas des gros circuits, elle peut ne pas être possible à cause de la limitation de la capacité des mémoires des machines hôtes. Pour remédier à ce problème, Synopsys a introduit une nouvelle méthodologie, point de compilation ou compile point, à son outil de synthèse Synplify Premier [58]. Le nouveau flot consiste à découper le circuit sous forme de plusieurs partitions, ou points, qui peuvent être synthétisés séparément. Un design peut avoir n'importe quel nombre de points de compilation, et ces points peuvent être imbriqués dans d'autres points de compilation. Durant la synthèse, tout le design est compilé, puis synthétisé à partir du point situé au niveau d'hiérarchie le plus bas. Après avoir compilé tous les points de compilation, une dernière itération du haut vers le bas est effectuée. Cette méthodologie est plus rapide qu'un flot top-down traditionnel, mais reste encore insuffisante pour les circuits de plus en plus complexes.

3.4 Partitionnement

Généralement, les circuits complexes à développer dépassent la capacité logique d'un seul FPGA, d'où la nécessité de les découper sur différents FPGA. La manière dont le circuit est découpé a un effet très important sur les performances et le comportement du système de prototypage. En effet, la communication entre les FPGA, assurée par les pistes gravée sur la carte, est très coûteuse par rapport à une communication intra-FPGA. Par conséquent, il est important de réduire au maximum les communications externes lors de la phase de partitionnement en absorbant le maximum de signaux à l'intérieur des parties. En plus de cet objectif il y a des contraintes à respecter lors du partitionnement comme le nombre de ressources logiques disponibles et le nombre de connecteurs d'entrée/sortie de chaque FPGA. L'algorithme le plus adapté à cet objectif est celui proposé par Fiduccia et Mattheyses et dont l'objectif est de minimiser la coupe. L'algorithme *FM* [59] s'applique aux hypergraphes et effectue des déplacements de sommets d'une partie vers l'autre au lieu des échanges des sommets entre deux parties. L'algorithme FM est un algorithme itératif. Son pseudo code est présenté par l'algorithme 1.

algorithme 1 Pseudo code de l'algorithme de Fiduccia-Mattheyses [60]

Partitionnement = solution initiale ;
Tantque (la qualité de la solution du partitionnement est améliorée) **faire**
  Initialisation : calculer le gain de chaque sommet ;
  coût = coût de la solution du partitionnement ;
  Tantque (il existe des sommets non verrouillés) **faire**
    /* Itération-FM */
    1. Sélectionner le sommet v_j de plus grand gain et dont le déplacement satisfait la contrainte d'équilibre ;
    2. Déplacer v_j et verrouiller ce sommet ;
    3. Mettre à jour le gain des voisins du sommet v_j ;
    4. Mettre à jour le coût de la solution du partitionnement.
  Fin Tantque
  Retourner à la meilleure solution vue au cours de l'itération ;
  déverrouiller tous les sommets.
Fin Tantque

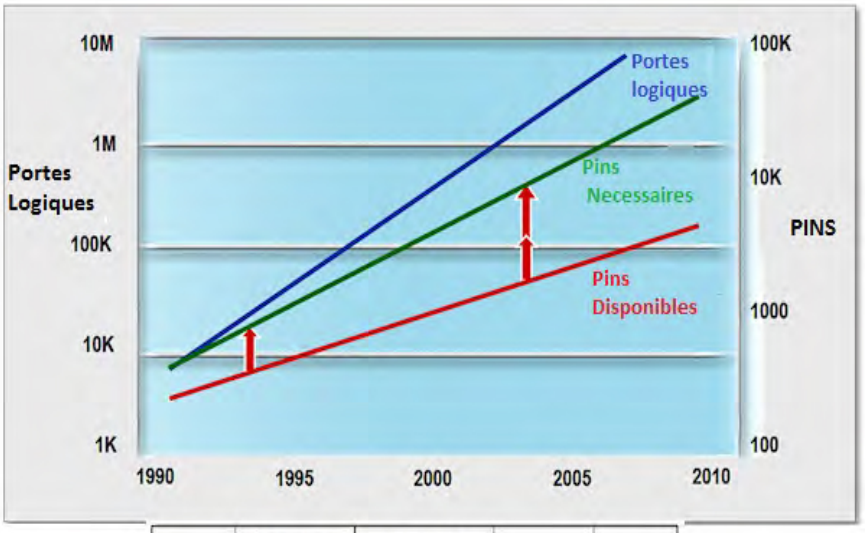
L'algorithme FM démarre avec une solution de bi-partitionnement initiale (par exemple une solution aléatoire), et affine par la suite cette solution en effectuant plusieurs itérations. Au début de chaque itération, tous les sommets sont libres de se déplacer d'une partie à une autre. Le gain du mouvement de chaque sommet est calculé et il lui est attribué. Le gain du mouvement d'un sommet est le coût de son déplacement vers une autre partie. Un gain positif réduit le coût de la solution de partitionnement, et un gain négatif augmente le coût de la solution.
Ensuite, d'une façon successive, le sommet qui présente le meilleur gain, et dont le déplacement est conforme aux contraintes de la taille des deux parties, est sélectionné. Ce sommet est déplacé vers la nouvelle partie, et ensuite verrouillé, c-à-d, il n'a plus le droit

d'être déplacé pendant l'itération en cours.
Vu que le déplacement d'un sommet peut affecter le gain de déplacement des sommets adjacents, les gains de ces derniers doivent être mis à jour.
La sélection et l'exécution du déplacement présentant le meilleur gain, suivies de la mise à jour des gains, sont répétées tant qu'il y'a des sommets non verrouillés. Une itération se termine lorsque tous les sommets sont déplacés. A la fin d'une itération, la meilleure solution vue pendant l'itération est choisie comme la solution de départ de l'itération suivante. L'algorithme FM s'arrête lorsqu'une itération n'améliore pas la solution par rapport à l'itération précédente.

En sortie de l'outil de partitionnement nous obtenons donc N sous-netlists qui seront placées et routées séparément sur chaque FPGA de la carte. La qualité de la solution finale est mesurée par la fréquence à laquelle tourne le circuit implémenté sur la carte. Le temps d'exécution de gros circuits (qui font l'objet de prototypage) sur la carte est très important et pourrait varier d'heures en jours d'une implémentation à une autre. Ceci pourrait avoir une influence néfaste et engendrer des délais sur le cycle de développement de ces circuits et par conséquent retarder leurs mises sur le marché.

3.5 Routage des signaux inter-FPGA

Après avoir découpé le design sous forme de plusieurs partitions, des signaux coupés entre les différentes partitions vont apparaitre à l'interface. Généralement, le nombre de signaux communiquant entre deux FPGAs est largement supérieur au nombre de traces physiques disponibles entre les deux. En effet, les études [61],[19] ont montré que l'évolution du nombre des portes logiques dans les FPGA est beaucoup plus rapide que celui des pins. Ce déséquilibre d'évolution cause une mauvaise exploitation des ressources logiques dans les FPGAs. En effet, les portes logiques disponibles dans un FPGA sont inexploitables à cause de la limitation des pins qui doivent leur supporter. Cette observation est démontrée par les courbes de la figure 3.4. Dans cette figure, il est à noter que pour une certaine partition avec un certain nombre de pins nécessaires, les FPGA avec le même nombre de portes n'ont qu'un nombre de pins largement inférieur que celui du nombre requis.
Pour remédier au problème de limitation du nombre de pins, plusieurs solutions ont été proposées dans l'état de l'art. Initialement, les fabricants ont essayé de déterminer la meilleure topologie d'interconnexion entre les différents FPGAs. Une topologie particulière a été proposée dans [62]. Cette nouvelle architecture utilise des composants spécifiques appelés FPIC (Field-Programmable Interconnect Chips). Comme le montre la figure 3.5, toutes les interconnexions passent à travers ces FPICs en formant un réseau de Crossbar. Cette architecture offre une flexibilité au niveau du routage ainsi qu'une prédictibilité des délais des longues connexions. Par contre, pour les circuits de plus en plus grands, cette solution ne résout pas définitivement le problème de limitation du nombre de pin, ce qui rend in-



Année	Famille de FPGA	Capacité Logique	Max I/Os	Capacité Par I/Os
2003	Virtex-II Pro	1,230,000	1040	1,183
2004	Virtex-4	2,490,000	960	2,594
2006	Virtex-5	3,320,000	1200	2,767
2009	Virtex-6	9,105,000	1200	7,588
2011	Virtex-7	14,000,000	1200	11,667

FIGURE 3.4 – Nombre de Pin par rapport à la taille logique d'un FPGA

utile le recours à ces composants, relativement chers. La topologie traditionnelle consiste à placer les FPGAs les uns à côté des autres pour former une sorte de matrice uniforme. Cette architecture est la plus utilisée de nos jours grâce à son coût de fabrication qui n'est pas cher. Par contre, son réseau d'interconnexion pas flexible a amené les développeurs à chercher des solutions logicielles pour remédier au problème de routage. Parmi les solutions proposées les plus utilisées, nous citons, la technique des fils virtuels ou Virtual Wires [20] et la technique basée sur la programmation linéaire (ILP) [21].

3.5.1 Technique des fils virtuels : Avec ordonnancement

Généralement, le fait d'assigner un pin à un seul signal ne permet pas d'exploiter au max toute la bande passante pour plusieurs raisons :

- La fréquence de l'horloge d'émulation est nettement plus faible que la fréquence du FPGA.
- Tous les signaux du design ne sont pas activés simultanément, donc il y aura nécessairement des temps d'inactivité de quelques fils physiques en attendant l'évaluation des signaux dont ils dépendent.

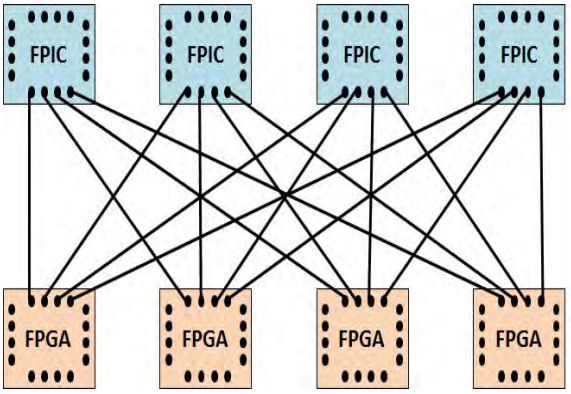


FIGURE 3.5 – Interconnexions à travers des circuits FPICs

Donc l'idée était de transférer plusieurs signaux à travers le même fil physique par la fréquence maximale du FPGA. D'où vient l'appellation de "fils virtuels" qui consiste à allouer le même fil physique pour multiplexer et transmettre plusieurs signaux logiques. La figure 3.6 montre la différence entre le transfert multiplexé et celui non-multiplexé de six signaux logiques. Dans le cas du multiplexage, les signaux sont commandés par deux boucles shift pipelinées placées de part et d'autre du fil physique. A des instants bien déterminés, le registre à décalage dans le FPGA source laisse passer un signal qui va être reçu par un autre registre dans le FPGA destination.

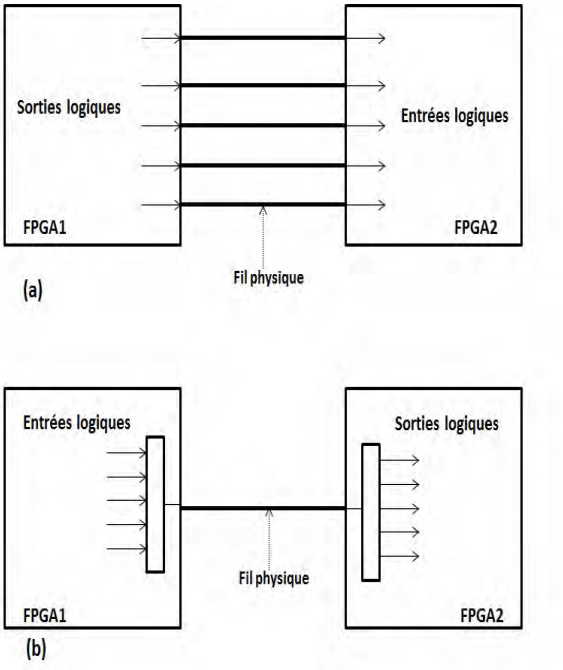


FIGURE 3.6 – (a) Transfert de signaux non multiplexés, (b) Transfert de signaux multiplexés

3.5.1.1 La période de l'horloge d'émulation

C'est la période d'émulation du circuit logique à tester (Design Under Test : DUT). La plus petite unité de temps de la période est appelée «microcycle» qui est générée par une horloge de très haute fréquence. Un ensemble de microcycles forme ce qu'on appelle «une phase» ou «un stage» comme le montre la figure 3.7. Chaque phase à son tour est divisée

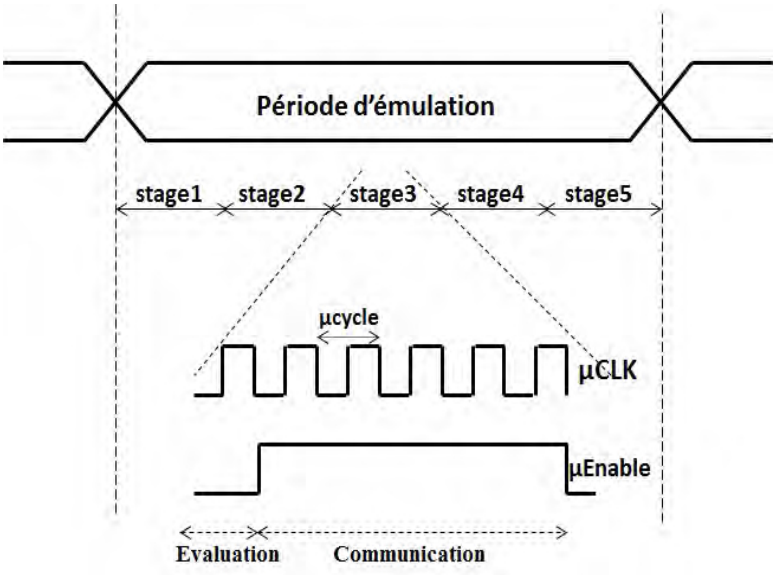


FIGURE 3.7 – Domaine d'horloge

en deux parties : une partie d'évaluation de la logique et une partie de communication inter-FPGA

- Partie évaluation : Elle se fait au début du stage. Les entrées de chaque partition sont propagées dans le FPGA considéré jusqu'à arriver aux sorties dépendants de ces entrées. Ces sorties doivent être regroupés dans des registres de décalage pour enfin être transmises à travers les fils physiques. Non toutes les entrées sont disponibles au début de chaque stage, et non toutes les sorties vont être produites. La transmission de chaque entrée ou sortie se fait pendant un seul stage.
- Partie communication : Elle se fait après la phase d'évaluation où les sorties vont être transmises vers d'autres FPGA. Cette transmission est rythmée à la fréquence maximale de l'FPGA.

Tous les stages nécessaires doivent être exécutés avant la fin de la période d'émulation. Dans un premier temps, les signaux asynchrones ne devaient pas être multiplexés, donc chaque signal asynchrone devait être assigné à un pin seul à lui. En [63], kadulghi a présenté une étude qui permet de multiplexer les signaux asynchrones. Vu que tous les signaux ne peuvent pas être transportés en même temps, donc il a fallu fixer des paramètres pour évaluer la priorité de transmission de chaque signal.

3.5.1.2 Paramètres d'ordonnancement

Deux paramètres d'émulation sont nécessaires pour faire l'ordonnancement. Ils ne sont calculés que pour les signaux inter-FPGA.

- Dépendance : pour déterminer ce paramètre, il faut dégager la liste des sorties (fils) qui sont dépendants d'une certaine entrée. Nous disons qu'une sortie est le fils d'une certaine entrée si tout changement dans cette dernière conduit directement à un changement de cette sortie. Un exemple pour le calcul de la dépendance est présenté par la figure 3.8 Dans cette figure, tout changement de la valeur de l'entrée Int2

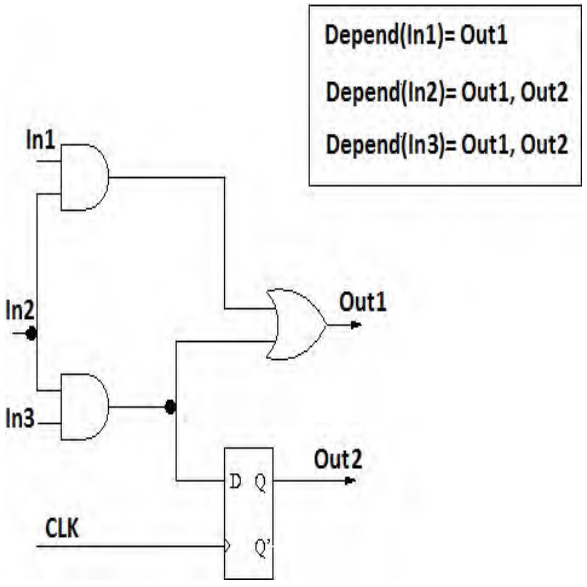


FIGURE 3.8 – Calcul de la dépendance

implique un changement des sorties Out1 et Out2. Ainsi, ces deux sorties dépendent de cette entrée.

- Le depth : le depth d'un signal c'est le plus grand nombre de partition qui existent dans le chemin commençant pas ce signal. Le depth est utilisé pour prioriser le routage du chemin critique. Durant l'ordonnancement, le signal qui a le plus grand depth sera routé en premier. Un exemple de calcul du depth est présenté dans la figure 3.9

3.5.1.3 Algorithme de routage

Le but de l'algorithme est d'assigner chaque signal à un couple ressource (pin), temps (microcycle), tout en contrôlant la disponibilité des ressources de routage de l'FPGA. Pour router un signal, l'algorithme utilise le chemin le plus court tout en essayant de router le plus grand nombre de signaux. Une fois il n'y a plus de signal à router, l'algorithme passe au stage suivant. Le passage au stage suivant se fait aussi s'il n'y a plus de chemin de routage disponible, ou bien encore si les signaux parents ne sont pas encore routés. Cet algorithme est basé sur un aspect constructif puisqu'une ressource de routage utilisée ne

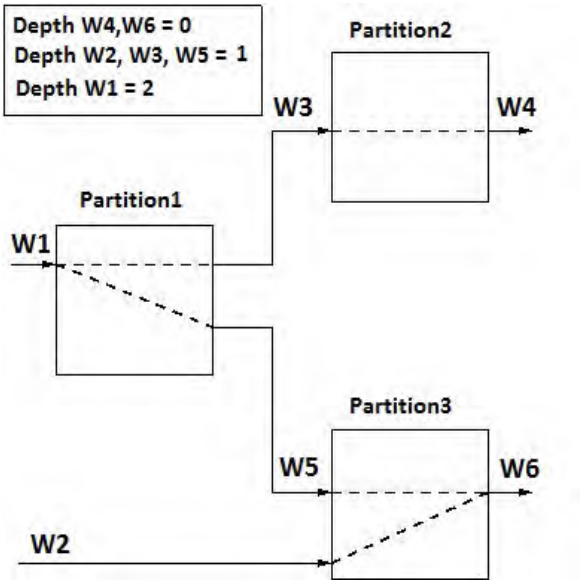


FIGURE 3.9 – Calcul du Depth

peut plus être libérée pour l'allouer à un groupe de signaux plus prioritaire. Cette approche n'est pas flexible. Et du coup, le nombre de solutions de routage est limité.

3.5.1.4 Génération des modules de multiplexage

Pour assurer la sérialisation des signaux sur le support physique, l'algorithme du virtual wire est implémenté sans matériel spécifique. Autrement dit, le multiplexage est assuré par des composants logiques qui seront ajoutés à la netlist de chaque partition. La nouvelle partie logique ajoutée est un ensemble de registres de décalage ainsi que des FSM. L'algorithme parcoure l'ensemble des groupes de signaux (shiftgroup) pour dégager 3 paramètres : les L signaux qui appartiennent à ce shiftgroup, le numéro du stage associé, ainsi que les FPGA par lesquels passe le chemin de routage de ces signaux. En parcourant tous les FPGA qui appartiennent au chemin de routage, une structure logique bien spécifique est insérée à chacun.

- Si le FPGA contient la source du signal, alors il lui est associé une architecture de sortie ayant L cases qui représentent les signaux appartenant à ce shiftgroup. Et puis assigner la sortie de cette architecture à un seul pin.
- Si le FPGA est un FPGA intermédiaire ou «hop», alors une architecture de saut ayant un seul bit lui sera attribuée.
- Enfin si ce FPGA contient la destination du signal, une architecture d'entrée ayant également L cases contenant les signaux d'entrée de ce shift group lui sera associée.

Enfin, comme le montre la figure 3.10, la nouvelle partie synthétisée de chaque FPGA est ajoutée au netlist initiale pour obtenir la netlist complète de chaque FPGA. Cette netlist complète va être l'entrée d'un stage de placement et routage de chaque FPGA à fin de créer le bitstream final. Malgré la grande performance de cet algorithme, mais il a quelques

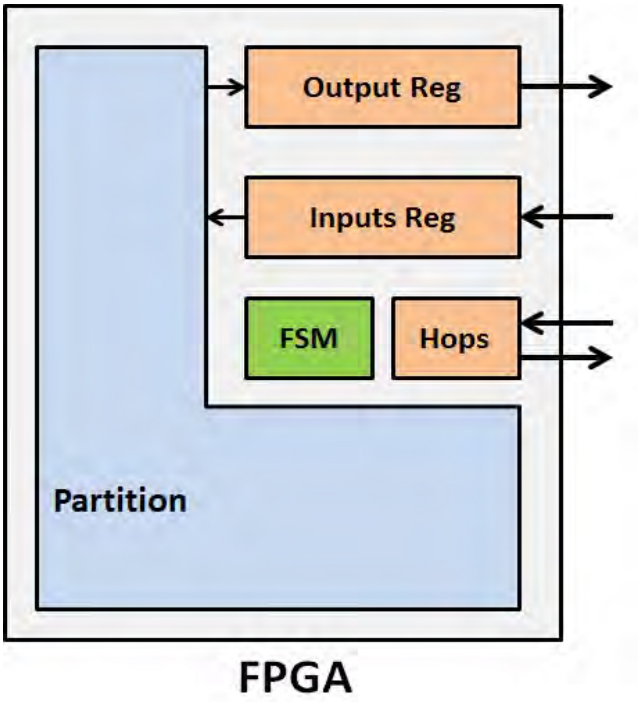


FIGURE 3.10 – Les différentes parties de la netlist finale

inconvénients qui doivent être améliorés à savoir :

- Dans le cas où un signal est prêt à être routé ou évalué, il faut impérativement attendre jusqu'au stage suivant ce qui dégrade la performance de l'algorithme.
- A un instant bien déterminé, et pour tous les pins, il n'y a à la fois qu'une évaluation ou une communication, or il est intéressant de chevaucher entre les deux quand est-ce que c'est nécessaire pour gagner de point de vue temps d'exécution.
- Il est intéressant de calculer cas par cas le nombre de microcycles nécessaire pour chaque stage au lieu d'appliquer le «pire des cas» pour tous les stages.
- L'algorithme doit être modifié pour cibler une fine granularité en visant l'ordonnancement au niveau des microcycles et non pas au niveau des stages.

Certains points ont fait l'objet de la recherche dans [21],[22]. L'étude effectuée en [21],[22] est basée sur des méthodes de la recherche opérationnelle. Elle s'intéresse à optimiser la fréquence du système de prototypage en utilisant la programmation linéaire ou ILP (integer linear programming)

3.5.2 Technique de routage sans ordonnancement

L'objectif principal de cette méthode est d'identifier les signaux qui vont être multiplexés et ceux qui ne seront pas multiplexés. Donc il était intéressant d'utiliser une méthode de recherche qui donne la sélection optimale. Pour cela, les auteurs dans [21], ont proposé une technique d'optimisation basée sur la programmation linéaire en nombre entiers(ILP :

Integer Linear Programming).

3.5.2.1 Définitions

Pour implémenter cette nouvelle technique, la netlist doit être décrite par un hypergraphe G(V,E) ou V est l'ensemble des nœuds qui représentent les portes logiques, les flip-flops et les ports d'entrée/sortie. E est l'ensemble des arrêtes qui représentent les signaux. Tout comme dans [20], un signal multiplexé est transmis entre deux FPGA pendant une unité de temps qui s'appelle microcycle ou slot. Un ensemble de signaux sont transmis pendant un certain nombre de slots appelé stage. Donc la période d'émulation T est calculée comme suit

$$T = \sharp stage * \sharp slot * T_{slot} \tag{3.1}$$

Avec $\sharp stage$ c'est le nombre de stage par période, $\sharp slot$ est le nombre de slot par stage et T_{slot} c'est la période d'un slot. L'objectif donc est de diminuer la période d'émulation pour accélérer le système. Diminuer T revient à minimiser $\sharp stage$ et $\sharp slot$. Donc, avant de le faire, il faut préciser l'expression de chacun de ces deux paramètres.

3.5.2.2 Multiplexage des signaux

Soit N_{depth} le nombre de signaux multiplexés qui assurent une connexion entre deux flip flop. Donc nécessairement il faut suffisamment de stage pour garantir la transmission des signaux entre deux flip flop. La figure 3.11 montre la technique de transmission des signaux par la méthode de ILP. Le signal 4 est transmis du FPGA1 vers le FPGA2 pendant

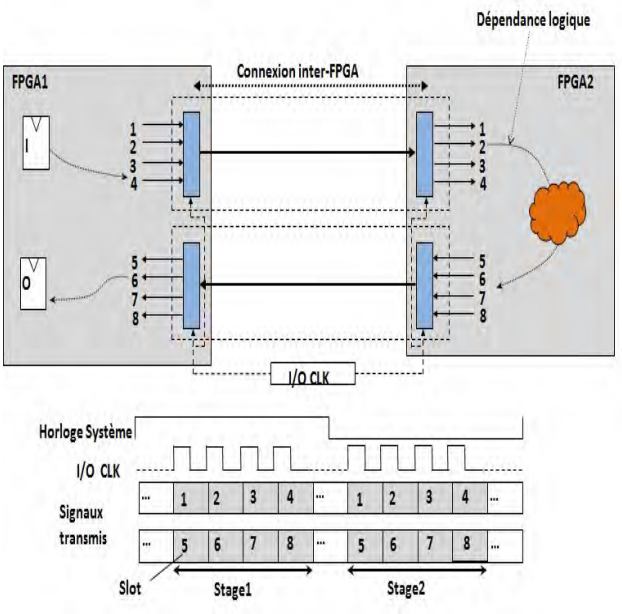


FIGURE 3.11 – Multiplexage des signaux

le premier stage et puis du FPGA2 vers le FPGA 1 via le signal 7 pendant le deuxième

stage. Donc le nombre de stage est représenté par l'expression 3.2

$$\sharp stage = max_{p \in paths}(S_{mux}) \tag{3.2}$$

C'est le nombre maximal de signaux multiplexés appartenant à un chemin p entre deux flip flop. D'autre part, le nombre de slots par stage est égale au nombre de signaux multiplexés divisé par le nombre de pins disponibles pour assurer le multiplexage. Cette relation est traduite par l'expression suivante

$$\sharp slot = \frac{S_{mux}}{P_{mux}} \tag{3.3}$$

Or le nombre de pins disponibles pour le multiplexage est égale au nombre total de pins moins le nombre de pins transmettant des signaux non multiplexés

$$P_{mux} = P - (S - S_{mux}) = P - S + S_{mux} \tag{3.4}$$

Donc l'équation 3.3 est représentée comme suit

$$\sharp slot = \frac{S_{mux}}{P - S + S_{mux}} \tag{3.5}$$

D'après l'équation 3.5, on remarque que le nombre de slot diminue lorsque S_{mux} augmente. D'autre part, d'après l'équation 3.2 le nombre de stage augmente avec l'augmentation de S_{mux} Donc il est nécessaire de trouver un compromis entre les variations de $\sharp slot$ et de $\sharp stage$ afin d'obtenir la période d'émulation la plus optimisée. Ceci revient à formuler un problème d'optimisation en lui indiquant les données et les contraintes à respecter.

3.5.2.3 Algorithme de sélection des signaux à multiplexer

Soit l'entier x_{vi} est un entier associé au signal i. il prend la valeur 0 si le signal i n'est pas multiplexé, et il est égale à 1 si le signal i est multiplexé. Donc la somme de tous ces entiers représente le nombre de signaux multiplexés.

$$S_{mux} = \sum_{v \in V_{io}} x_v \tag{3.6}$$

Avec V_{io} c'est l'ensemble des signaux inter-FPGA. Soit y_v l'entier qui représente le nombre de signaux multiplexés depuis un flip flop donné jusqu'au signal v. y_v peut être calculé d'une manière récursive comme suit

$$y_v = x_v + max_{v' \in V_{fi}(v)}(y_{v'}) \tag{3.7}$$

Avec $V_{fi}(v)$ est l'ensemble de signaux inter FPGAs depuis lesquels il y a nécessairement un chemin qui conduit vers le signal v.

L'algorithme 2 représente le problème d'optimisation développé dans [21]

algorithme 2 Pseudo code de l'algorithme de sélection de signaux à multiplexer

Constantes
N : nombre de stages
Variables
x_{v1}, x_{v2}, x_{v3}... x_{vm}, : entiers de valeurs 0 ou 1
y_{v1}, y_{v2}, y_{v3}... y_{vm}, : entiers
Contrainte
$0 \leq y_v \leq$ N pour tout v $\in V_{io}$
pour tout v $\in V_{io}$
$y_v \geq x_v$ **si** $V_{fi} \neq \emptyset$
$y_v \geq x_v + y_{v'}$ Pour tout v' $\in V_{fi}(v)$ **si** $V_{fi} \neq \emptyset$
fin pour
objectif
maximiser $\sum_{v \in V_{io}} x_v$

Pour résoudre ce problème d'optimisation il suffit de varier la valeur de $\sharp stage$ et trouver la valeur minimale de $\sharp slot$ associée. Le meilleur couple ($\sharp stage$, $\sharp slot$) est déduit depuis la valeur de la période la plus optimisée.

3.6 Placement intra-FPGA

Le placement consiste à attribuer des éléments spécifiques du FPGA aux cellules logiques du circuit. La qualité du placement influe considérablement sur le résultat du routage du circuit. Plusieurs techniques de placement ont été définies dans la littérature.

3.6.1 Placement par le recuit simulé

Le placement recuit simulé (Simulated Annealing) [64] [65] est l'approche de placement la plus performante et la plus utilisée aujourd'hui pour les FPGAs. La méthode du recuit simulé tend à optimiser la longueur totale des fils de routage (wirelength) utilisés pour router le circuit. Elle consiste à effectuer plusieurs itérations pour trouver le meilleur placement des blocs logiques en se basant sur une fonction coût qui englobe les contraintes du placement.
Une solution initiale S est choisie en plaçant aléatoirement les blocs logiques du circuit sur le FPGA. Ensuite, un certain nombre d'itérations est effectué. Dans une itération, une instance de cellule logique du circuit et un bloc physique du FPGA sont choisis aléatoirement. Le déplacement de l'instance vers ce nouveau bloc physique va conduire le circuit à une solution de placement voisine $S + 1$. Si cette solution présente une amélioration, elle est retenue. Mais, si elle implique une dégradation du coût de la solution actuelle, elle est retenue avec une probabilité de $\exp(-\frac{\Delta C}{T})$, où ΔC est la différence entre le coût de la

solution S et le coût de la solution $S+1$, et T représente la température.
Au début du processus du placement, la température est assez élevée. Il y a donc une forte probabilité que les déplacements des instances de cellules logiques soient acceptés même s'ils dégradent la qualité du placement. Ensuite, la température décroît progressivement et la probabilité de retenir un mauvais déplacement décroît également. La température T est diminuée à une température $r \cdot T$, où r est appelé le taux de refroidissement. A chaque température, de nouvelles itérations sont effectuées.

3.6.2 Fonction coût

Les auteurs de [66] ont défini un outil de placement et de routage pour les FPGAs, nommé VPR (Versatile Packing, Placement and Routing) et dont la méthode de placement est basée sur l'approche du recuit simulé. L'objectif est de minimiser le coût de routage. La fonction de coût utilisée est une fonction de la longueur totale des fils de routage (wirelength) utilisés par le placement actuel, et qui est estimée en fonction du périmètre des rectangles englobant les signaux du circuit, comme le montre l'équation 3.8. N est le nombre de signaux du circuit et $q(i)$ est un facteur de correction. $bb_x(i)$ et $bb_y(i)$ sont respectivement la largeur et la longueur du rectangle englobant le signal i.

$$cout = \sum_{i=1}^{N} (q(i) \cdot (bb_x(i) + bb_y(i))) \tag{3.8}$$

Le rectangle englobant un signal est défini comme étant celui qui englobe tous les connecteurs (pins) du signal. La figure 3.12 montre le rectangle englobant un signal connectant cinq blocs logiques.
La minimisation de cette fonction coût permet de rapprocher tous les blocs qui appartienne

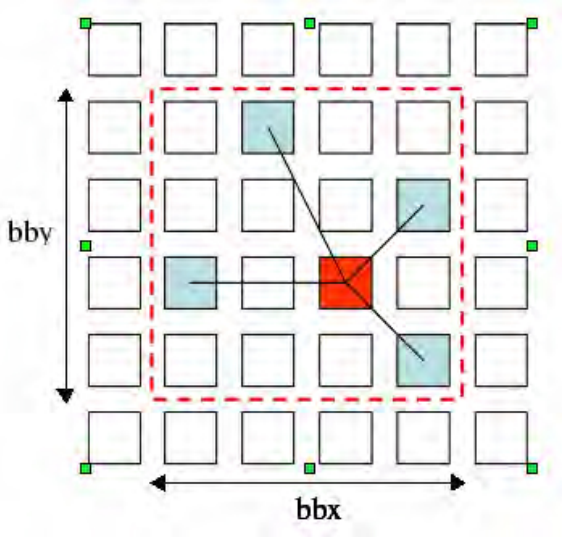


FIGURE 3.12 – Rectangle englobant un signal connectant 5 blocs logiques [2]

à la même équipotentiel ce qui implique la création des zones de congestion dans le circuit FPGA. Les zones congestionnées nécessitent des ressources de routage supplémentaires, ce qui rend parfois la solution de routage inexistante.

3.6.3 Réduction de la congestion

Pour remédier au problème de la congestion, plusieurs travaux ont été faits. Les auteurs de [28] ont proposé une approche itérative qui consiste à insérer des blocs logiques vides dans les zones fortement congestionnées. En effet, si l'outil de routage n'arrive pas à trouver une solution de routage, une carte de congestion est tracée pour identifier les zones les plus congestionnées et qui nécessitent des ressources de routage supplémentaires. A chaque bloc logique est assigné un indice de congestion qui représente le nombre de signaux routés à travers les ressources de routages adjacentes à ce bloc logique. Par la suite, les zones ayant les indices les plus importants seront dépeuplées, regroupées puis étalées sur une surface supplémentaire du circuit FPGA. Ainsi, la congestion moins centralisée, le router essayera encore une fois de router le circuit. S'il n'arrive pas encore à trouver une solution acceptable avec les ressources disponibles, les étapes précédemment décrites seront refaites de nouveau. L'avantage de cette technique est qu'elle donne une information réelle sur les zones congestionnées. Par contre, le fait de refaire le flot de prototypage dans chaque itération (placement et routage), cause un temps de mise en œuvre très long.
Pour cette raison, des travaux qui ont été faits et qui consistent à utiliser un placement basé sur l'estimation de congestion, mais sans passer par le routage. De ce fait, l'information sur la congestion n'est qu'une estimation calculée en se basant sur différents paramètres.
Dans [67], l'idée était d'estimer la congestion avec la règle de Rent [68]. En effet, le circuit FPGA est divisé en parties élémentaires de surface égales. Pour chaque partie, l'exposant Rent est calculé, et le résultat est assigné à chaque bloc logique de la surface élémentaire. Cette information est transmise à l'outil de placement à travers la fonction coût de telle façon que tout déplacement de blocs qui cause une augmentation de la fonction coût sera rejeté.
Dans [69], un coefficient de congestion a été introduit dans la fonction coût de l'équation 5.3 pour forcer l'outil de placement à éviter la création de zones congestionnées. Pour calculer ce coefficient, il a fallu attribuer à chaque bloc logique une valeur de congestion qui est le nombre de bounding box auxquels appartient ce bloc. Cette estimation ne donne pas une information précise sur le nombre des ressources de routage utilisés. Par conséquent, il a fallu trouver d'autres métriques pour donner une estimation plus proche de la valeur réelle, en plus il était nécessaire d'adapter l'expression du coefficient de congestion à l'architecture du FPGA utilisé.
Les auteurs de [70] ont présenté leur outil visuel permettant d'estimer la congestion au court du placement. Plusieurs métriques telles que l'exposant de Rent, le nombre de signaux par région et le chevauchement du rectangle englobant, ont été utilisées afin de trouver le meilleur paramètre qui donne une information sur la congestion la plus proche de la réalité.

Cette dernière métrique fera l'objet d'un travail que nous avons réalisé pour améliorer la routabilité du circuit FPGA. Nous comptons l'introduire dans le coefficient de congestion proposé dans [69]. L'expression de ce coefficient sera adaptée à l'architecture que nous avons utilisée. Et par conséquent, nous allons développer avec plus de détails la métrique proposée ainsi que l'expression du coefficient de congestion dans le chapitre 7.

3.7 Routage Intra-FPGA

Le routage est la dernière étape du flot de configuration du FPGA précédant la génération du bitstream. C'est l'une des étapes les plus fastidieuses, mais aussi les plus importantes dans le flot de configuration du FPGA. L'objectif du routage consiste à créer des connexions physiques entre les cellules logiques du circuit en optimisant la longueur totale des fils de routage, en minimisant le délai du chemin critique du circuit ou en combinant les deux critères. Jusqu'à ce jour, l'outil de routage le plus efficace et détaillé est l'outil VPR [66] [2]. Cet outil repose sur l'algorithme de routage PathFinder [37].
Dans cette thèse, nous allons proposer un routage inter-FPGA basé sur l'algorithme PathFinder. Nous allons donc nous attarder sur son explication dans la section suivante.

3.7.1 Algorithme de routage PathFinder

L'algorithme PathFinder [37] est un algorithme itératif, basé sur une approche de négociation afin de réussir à router tous les signaux d'un circuit. Cet algorithme agit sur un graphe direct G(V,E) qui représente les ressources de routage disponible. L'ensemble des nœuds V représentent les pins de chaque bloc logique ainsi que les segments de routage. Un arc entre deux nœuds représente une connexion possible entre eux.
La figure 3.13 présente la transformation de l'architecture de routage dans un FPGA en un graphe direct.
Le pseudo code de l'algorithme de PathFinder est présenté par l'algorithme 4.

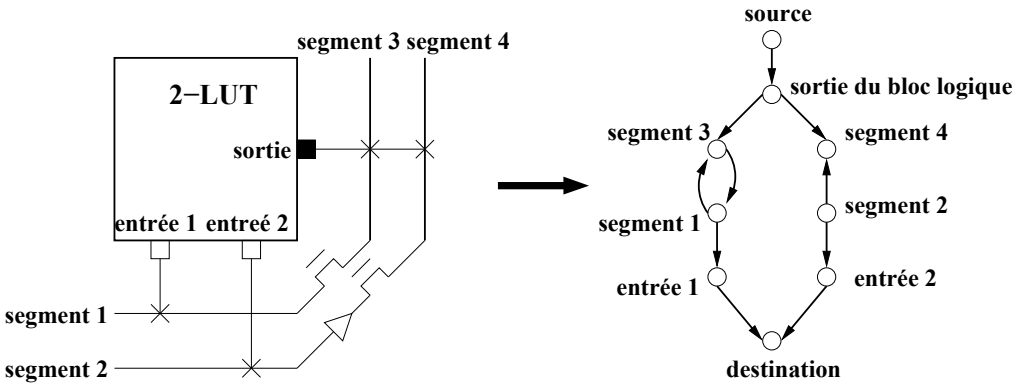


FIGURE 3.13 – Modélisation d'une architecture FPGA par un graphe de routage

Le routeur PathFinder implémente plusieurs itérations jusqu'à ce qu'il n'y ait plus de

conflits de ressources. A chaque itération, il route tous les signaux du circuit en mettant à jour dynamiquement le coût des ressources de routage utilisées. Au cours d'une itération, les signaux sont déroutés et routés de nouveau en tenant en compte de la congestion actualisée. Cette procédure est connue sous le nom de "*Rip-up and retry*". Utiliser cette méthode et permettre aux signaux de partager des ressources permet de résoudre les conflits entre les différents signaux et permet aussi d'éliminer la dépendance entre l'ordre des signaux à router et le résultat du routage.

algorithme 3 Pseudo code de l'algorithme PathFinder [37]

```
Soit RT_i l'arbre de routage du signal i Durant l'itération en cours ;
Tantque (il existe des ressources de routage partagées) faire
  /*routeur global*/ ;
  pour tout signal i faire
    /*routeur d'un signal*/ ;
    vider l'arbre de routage RT_i /* Rip-up*/ ;
    RT_i = s_i ; s_i = source du signal i ;
    pour tout destination t_ij du signal i faire
      Initialiser la file de priorité PQ à RT_i avec un coût = 0 ;
      Tantque ( destination t_ij non atteinte ) faire
        enlever le noeud m ayant le coût minimal de PQ ;
        pour tout noeud voisin n du noeud m faire
          insérer n dans PQ avec un coût PathCost(n) = c(n) + PathCost(m) ;
        fin pour
      Fin Tantque
      pour tout noeud n appartenant au chemin de t_ij à s_i faire
        /*remonter le chemin des prédécesseurs de t_ij à s_i*/ ;
        Metre à jour p(n) ;
        Ajouter n à RT_i ;
      fin pour
    fin pour
  fin pour
  mettre à jour hn() pour toutes les ressources n ;
Fin Tantque
```

Dans la première itération, tous les signaux sont routés de telle façon qu'ils peuvent partager des ressources de routage. Le routage de chaque signal repose sur l'algorithme Dijkstra [71] [72]. Cet algorithme cherche à router une connexion avec le plus court chemin. Il garantit de trouver le chemin de coût minimal s'il existe. A la fin de la première itération, certaines ressources peuvent se trouver congestionnées, car elles ont été partagées par plusieurs signaux. Durant les itérations suivantes, la pénalité de partage des ressources est incrémentée progressivement. Le coût d'une ressource congestionnée est incrémenté en fonction du nombre de signaux qui l'utilisent et de son historique de congestion.
Ainsi, les signaux sont forcés à négocier pour les ressources partagées. Si une ressource est fortement congestionnée ce qui se traduit par un coût très élevé, les signaux qui la partagent vont essayer d'utiliser d'autres ressources moins coûteuses et donc moins congestionnées.

Mais, si toutes les autres alternatives sont plus congestionnées que la ressource en question, les signaux continuent à utiliser cette dernière.
La fonction de coût implémentée par l'algorithme PathFinder ayant pour objectif la routabilité du circuit est donnée par l'équation 5.3.

$$c(n) = (b(n) + h(n)) \cdot p(n) \tag{3.9}$$

où n est la ressource de routage utilisée. Le terme $b(n)$ est le coût de base d'une ressource n. Le terme $h(n)$ reflète l'historique de congestion de la ressource n pendant les itérations précédentes. L'objectif de ce terme est d'empêcher les signaux d'utiliser des ressources qui ont conduit à une mauvaise solution de routage dans des itérations passées. Le terme $p(n)$ représente le coût de partage d'une ressource n. Il est proportionnel au nombre de signaux qui utilisent la ressource durant l'itération en cours.
Dans la version originale de PathFinder [37], le coût de base $b(n)$ est égal au délai intrinsèque de la ressource n. Dans la version de VPR [2], il a été noté que ce choix ne donne pas les meilleurs résultats en termes de nombre de ressources de routage utilisées. Le coût de base adopté est égal à 1 pour les segments de routage, à 0.95 pour les pins d'entrée des blocs logiques et à 0 pour les destinations finales du circuit. Les deux dernières valeurs sont choisies afin d'accélérer le temps d'exécution de l'algorithme de routage. L'utilisation d'un coût de base égal à 0 pour les destinations finales ne risque pas de compromettre la réussite du routage parce qu'il n'y a aucune congestion possible à leur niveau.
La pénalité de congestion $p(n)$ est une fonction de la différence entre la capacité de la ressource n et le nombre de signaux qui la partagent. La capacité d'une ressource est égale à la largeur du canal de routage dans le cas d'un routage global, et égale à 1 dans le cas d'un routage détaillé. La valeur de $p(n)$ est mise à jour à la fin d'une itération, mais aussi au cours de l'itération après le routage de chaque signal afin d'éviter le partage excessif d'une ressource. Le calcul de $p(n)$ dans VPR [2] est illustré par l'équation 3.10, où p_{fac} est un facteur qui s'incrémente à chaque itération.

$$p(n) = 1 + max(0, [1 + occupation(n) - capacité(n)] \cdot p_{fac}) \tag{3.10}$$

$h(n)$ est le coût de l'historique de congestion de la ressource n. Dans VPR [2], sa valeur est égale à 1 au cours de la première itération. Ensuite, elle est calculée suivant l'équation 3.11.

$$h^i(n) = h^{i-1}(n) + max(0, [1 + occupation(n) - capacité(n)] \cdot h_{fac}) \tag{3.11}$$

où h_{fac} est une constante entre 0.2 et 1. Le terme $h(n)$ est mis à jour à la fin de chaque itération.

Le routeur PathFinder implémente plusieurs itérations jusqu'à ce qu'il n'y ait plus de conflits de ressources. A chaque itération, il route tous les signaux du circuit en mettant à

jour dynamiquement le coût des ressources de routage utilisées. Au cours d'une itération, les signaux sont déroutés et routés de nouveau en tenant en compte de la congestion actualisée. Cette procédure est connue sous le nom de "*Rip-up and retry*". Utiliser cette méthode et permettre aux signaux de partager des ressources permet de résoudre les conflits entre les différents signaux et permet aussi d'éliminer la dépendance entre l'ordre des signaux à router et le résultat du routage.

3.8 Conclusion

Nous avons détaillé l'ensemble des étapes du flot de prototypage matériel. Les algorithmes les plus utilisés dans la littérature, en relation avec les travaux de cette thèse ont été expliqués. En effet, nous avons présenté les techniques des Virtuals Wires et la programmation linéaire en nombres entiers pour le routage inter-FPGA, l'heuristique du recuit simulé et les techniques d'évitement de congestion pour le placement ainsi que l'algorithme PathFinder pour le routage intra-FPGA. Dans les chapitres suivants, nous nous baserons sur ces algorithmes dans nos outils de placement intra-FPGA et de routage inter-FPGA, et nous les adapterons dans le but d'obtenir les meilleurs résultats de point de vue fréquence de prototypage.

Chapitre 4

Générateur de benchmarks

Sommaire

4.1 Introduction

Pour mener les expériences tentant à prouver l'efficacité des méthodes proposées dans cette thèse et en particulier de vérifier l'aspect dynamique de ces méthodes, il a été nécessaire de trouver des circuits de test stimulés par des patterns afin de garantir un niveau de contrôlabilité par l'utilisateur.
Dans ce chapitre, nous présentons un générateur de benchmarks qui permet de générer des circuits de test qui répondent à un certain nombre de critères. Ces circuits sont décrits par un langage de description matériel (VHDL) afin de faciliter leur implémentation sur une carte multi-FPGA.

Nous proposons de bâtir les benchmarks autour des architectures multiprocesseurs, multi-coprocesseurs. Cette hétérogénéité permettra de mettre en œuvre la performance des techniques développées dans cette thèse, mais aussi celles des autres partenaires du projet PPR, à citer, les techniques de partitionnement. La construction de ce démonstrateur comporte donc deux volets : un volet matériel qui concerne l'architecture elle-même, et un volet logiciel pour le développement de l'application exécutée par le circuit.

4.2 Les architectures multiprocesseurs

L'évolution très rapide des technologies de fabrication de circuits intégrés sur silicium permet aujourd'hui de réaliser des systèmes numériques complets intégrés sur une même puce et contenant un grand nombre de processeurs (MP-SOC) afin d'atteindre les performances requises par l'application. La taille du marché des MP-SoC devient de plus en plus importante (plus que 1600 circuits pas an), ce qui rend de ces circuits la cible directe des plateformes de prototypage. Par conséquent, notre intérêt était de chercher des benchmarks ayant un comportement semblable à celui des circuits industriels. Notre première tentative était de récupérer des circuits industriels auprès des grandes industries des systèmes sur puces. Mais pour des raisons de confidentialité, cette tentative a échoué. Par la suite, nous avons essayé de récupérer les benchmarks "open source". Les circuits proposés dans [29, 30, 31] ne sont pas suffisamment grands pour être partitionnés sur des plateformes multi-FPGA. La taille des benchmarks générés par un programme GNL utilisant la formule de Rent [33] peut dépasser les millions de LUT, par contre, le comportement aléatoire de ces circuits n'obéit pas aux exigences du marché.
Pour toutes ces raisons, nous avons décidé de concevoir un générateur de benchmarks qui permet de générer des circuits multiprocesseurs ayant un comportement réel et des caractéristiques semblables à celles des circuits industriels. En effet, ces benchmarks sont conçus à l'aide d'une base de données contenant un ensemble de composants virtuels dont la fonctionnalité pourra correspondre à un processeur, une mémoire ou même un réseau sur puce.

4.2.1 Caractéristiques des architectures multiprocesseurs

Les benchmarks générés doivent répondre à un certain nombre d'exigences afin de valider l'ensemble des techniques développées dans le cadre du projet PPR.

4.2.1.1 Architectures hétérogènes

A part les composants processeurs, les benchmarks doivent contenir d'autres composants répartis d'une manière asymétrique sur l'architecture. Ces composants inclus des accélérateurs, des coprocesseurs, des mémoires et des périphériques. L'hétérogénéité et

l'asymétrie permettent d'évaluer "l'intelligence" de l'outil de partitionnement et sa capacité de créer des partitions qui respectent les contraintes de surface sur les différents FPGA de la carte.

4.2.1.2 Horloges multi-domaines

Les circuits intégrés récents sont contrôlés par plusieurs horloges (multi-domaines). Le nombre d'horloges dépend de l'application logicielle. Lors des étapes de partitionnement et de routage, les outils CAO doivent gérer convenablement les différentes horloges en y appliquant une attention considérable. Les benchmarks de test doivent être multi-domaines afin de bien évaluer le comportement des outils de partitionnement et de routage envers ces horloges.

4.2.1.3 Architectures de grande taille

Les architectures générées doivent être assez grandes pour occuper toutes la surface logique d'une carte multi-FPGA. Ces architectures permettront d'évaluer la performance de l'outil de partitionnement, mais aussi celle des techniques de multiplexage proposées dans cette thèse. La plateforme matérielle fabriquée dans le cadre du projet PPR contient deux FPGAs Virtex 7, chacune de taille logique qui dépasse les 2 millions de LUT. Par conséquent, il est nécessaire de concevoir un outil de génération de benchmarks qui génère en un temps acceptable une telle taille.

4.2.2 Caractéristiques du générateur de benchmarks

Le générateur de benchmarks comportera un volet logiciel et un volet matériel. Le volet matériel consiste à développer une plateforme matérielle générique décrite dans un langage de description de matériel et synthétisable. Cette plateforme est construite par un générateur permettant d'obtenir un système multiprocesseur avec un nombre variable de processeurs. On peut également envisager de faire varier les caractéristiques des caches et les moyens de communication entre les processeurs. L'intérêt de cette plateforme générique est de faire varier simplement la complexité du système à travers les paramètres du générateur. Ainsi, on obtient un démonstrateur permettant à la fois la mise au point des méthodes proposées que leur évaluation à travers une série d'expériences.
Le volet logiciel est basé sur le développement d'une ou de plusieurs applications. L'expérimentation consiste à compiler l'application et à enregistrer le code exécutable obtenu dans les mémoires du système multiprocesseur avant de demander au système d'exécuter l'application. Les applications doivent être écrites sous forme de plusieurs tâches communicantes (des threads). Cette écriture permet de distribuer facilement l'application sur la plateforme multiprocesseur. L'objectif est d'obtenir des applications comportant un nombre variable

de tâches qui s'exécutent sur une plateforme ayant elle-même un nombre variable de processeurs.
Une application qui répond à ces caractéristiques a été développée dans le laboratoire d'Informatique de Paris 6 (LIP6). Cette application est basée sur l'outil DSX [73] qui permet la co-conception de plateformes matérielles-logicielles destinées au traitement de flux basées sur des architectures multiprocesseurs sur puces. Cet outil génère des netlists décrites avec le langage de description SystemC. Ces netlists ne sont pas implémentables sur carte FPGA à moins de passer par une étape de traduction qui risque d'être impossible.
Nous proposons donc de mener des modifications au environnement DSX_SystemC afin de générer des netlists synthétisables et implémentables sur une plateforme de prototypage. Avant de présenter notre nouvel environnement, nous donnons d'abord une description des fonctions principales et de la chaine de compilation de l'environnement DSX_SystemC.

4.3 L'environnement DSX_SystemC

L'outil DSX (Design Space eXploration)[73] assiste la conception d'applications logicielles décrites de manière statique sous forme d'un graphe de taches et du matériel sous-jacent. Pour le concepteur d'un système intégré, il est impératif d'avoir un contrôle maximum sur le système sur puce construit, tant au niveau de l'architecture, que du placement des objets physiques. Un objectif essentiel est de permettre au concepteur de l'application de contrôler de façon très fine, non seulement le placement des taches sur les processeurs, mais également le placement des différents objets logiques (piles d'exécution des taches, canaux de communication, barrières de synchronisation, verrous,..) sur les bancs mémoire physiques.

4.3.1 Chaine de compilation de DSX_SystemC

La figure 4.1 représente un schéma simplifié des flux d'entrée et de sortie classiques de DSX.
Le développement d'un SoC est décomposé en trois étapes :

- La conception d'une application décrite sous forme d'un graphe de tâches communicantes par des canaux de communication spécifiques appelés MWMR (Multi Writer Multi Reader). Chaque tâche doit être définie au préalable. Cette définition comporte la façon dont la tache est intégrée dans le graphe (entrées, sorties, ressources) et encore l'ensemble de ses implémentations matérielles et logicielles. La description se fait à travers une API en python.
- La conception d'une plateforme matérielle pour héberger cette application : il s'agit d'une architecture conçue à l'aide des IP-core de la bibliothèque SoCLIB [36]. C'est une librairie ouverte conçue pour le prototypage virtuel des systèmes sur puce. Il s'agit d'un ensemble de modèles de simulation system C des composants matériels,

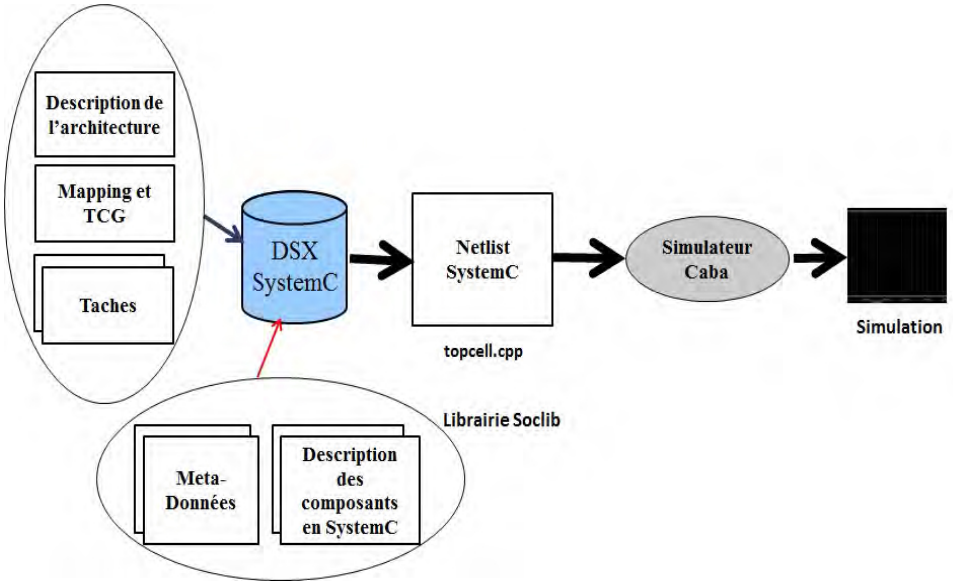


FIGURE 4.1 – Schéma descriptif des flux d'entrée et de sortie de DSX_SystemC

qui peuvent être utilisés lors de la description architecturale des systèmes intégrés, notamment sous DSX.

- Enfin le déploiement de l'application logicielle sur la plateforme matérielle.

Les plateformes sont simulées soit au niveau CABA (cycle accurate bit accurate, cycle précis bit précis) soit au niveau transactionnel (TLM). Tous les composants sont intégrés sur une seule puce. L'architecture générique contient un nombre variable de processeurs RISC 32 bits programmables et de bancs mémoire embarqués. Elle contient du matériel pour l'affichage et la gestion des verrous, et des coprocesseurs spécifiques à l'application. Il y a deux types de composants : initiateurs (typiquement des processeurs et des coprocesseurs) et cibles (typiquement, les bancs memoire, terminaux..).

4.3.2 Bibliothèque SoCLib de composants

SoCLib est une plateforme ouverte, conçue pour le prototypage des systèmes multi-processeurs sur puces. Il s'agit d'une bibliothèque de modèles de simulation SystemC des composants matériels, qui peuvent être utilisés lors de la description architecturale de systèmes intégrés, notamment DSX.
L'outil DSX_SystemC et SoCLib ont été conçus pour cibler la simulation comportementale des ASICs comme une étape de vérification avant la fabrication effective du circuit intégrés. Pour profiter des avantages du prototypage matériel, il a été nécessaire de cibler une autre plateforme matérielle (au lieu de la station de travail) qui n'est autre qu'une plateforme à base de circuits FPGA. Par conséquent, notre but c'était de modifier l'outil DSX_SystemC pour générer, en plus du simulateur system C caba, une plateforme VHDL synthétisable et implémentable sur carte FPGA.

4.4 Environnement DSX_VHDL

Le but de cet environnement est de générer une netlist en VHDL qui représente la description de l'architecture matérielle. Pour cela, il a fallu étudier l'organisation interne de DSX_SystemC et comprendre le mécanisme de construction de la netlist SystemC.

4.4.1 Génération de la netlist synthétisable

En étudiant les sources de DSX_SystemC, écrits en Python, nous avons pu suivre les différents appels aux classes et fonctions, depuis la lecture des composants instanciés et les différentes connexions entre eux jusqu'à l'écriture proprement dite du fichier de la netlist .cpp. Nous avons pu donc à partir de ce cheminement, construire un cheminement parallèle qui, en combinant les objets déjà présents dans l'outil avec des nouvelles fonctions et classes propres aux spécificités du langage VHDL, permet de générer le fichier netlist .vhd. L'écriture de ce fichier est faite sur plusieurs étapes représentées par la figure 4.2.

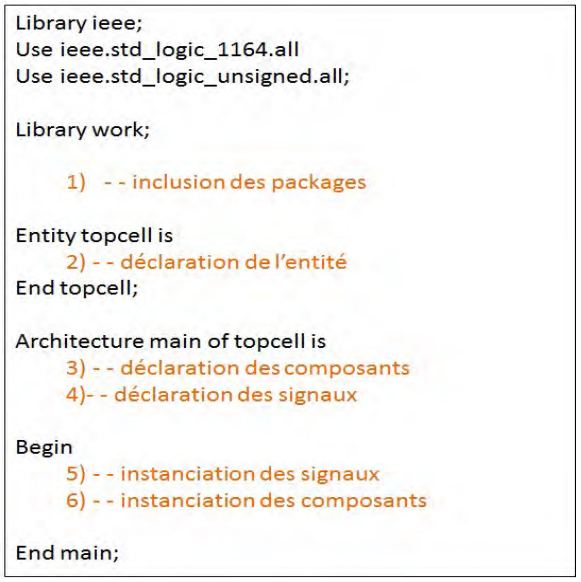


FIGURE 4.2 – Les différentes parties de la netlist .vhd

4.4.1.1 Inclusion des packages

Lorsqu'un composant contient un port composé, c'est à dire un tableau de largeur L de vecteurs de taille N, il est impératif de déclarer ce type composé à part, dans ce qu'on appelle les packages. En général, lorsque l'architecture matérielle décrite par l'utilisateur sous DSX contient des composants qui comprennent des ports de types composés, un fichier de packages est automatiquement crée et sera instancié dans la netlist .vhd.

4.4.1.2 Déclaration de l'entité

L'entité de la netlist générée est appelée par défaut "topcell". Elle contient les ports servant à connecter cette entité à son environnement extérieur (autres entités ou pins de la carte FPGA). Pour permettre à l'utilisateur de définir ces ports externes, une fonction est utilisée et qui a pour but de rendre "externe" certain ports et de les connecter aux signaux de l'architecture.

4.4.1.3 Déclaration des composants

Lorsque l'utilisateur décrit sa plateforme matérielle sous DSX, un driver est utilisé pour retrouver des informations nécessaires à la construction de la netlist, à savoir :

- Les noms des composants.
- les ports des composants.
- Les divers paramètres d'instance de ces composants.
- Les diverses connexions entre ces composants.

Cet objet est construit à partir de deux choses :

- La description de l'architecture passée par l'utilisateur sous DSX.
- Les fichiers de métadonnées qui donnent les propriétés de chaque composant : noms et types de ports, paramètres générique...

Le driver recense donc tous les composants instanciés par l'utilisateur, et construit à partir des informations recueillies dans les fichiers de métadonnées, la déclaration des composants dans la netlist .vhd. Dans les fichiers de métadonnées, les ports d'un composants peuvent avoir n'importe quel nom et n'importe quel type, à condition de définir ce type dans un fichier de métadonnées pour qu'il soit reconnu par DSX.
Les ports d'un composants sont de deux types :

- Type simple : un vecteur ou un bit : les types simples doivent avoir une syntaxe particulière pour être reconnus et interprétés par le driver de DSX. Par exemple, une déclaration telle que "clock rtl :bit_in" dans le fichier de métadonnées correspond au type "clock : in std_logic" dans le fichier VHDL.
- Type composé : un port de type composé ou métaport représente en fait un ensemble de ports de type simple regroupés. Ce regroupement est fait pour faciliter la connexion entre des composants ayant plusieurs ports identiques.

4.4.1.4 Déclaration des signaux

Sous DSX, chaque port d'un composant est associé à un signal. Le signal associé à un port est donné lors de la description du port en question dans les fichiers de métadonnées. Et tout comme les ports, il existe deux types de signaux : simple et composé ou métasignal. A chaque fois que l'utilisateur crée une connexion entre deux ou plusieurs ports simples ou deux ou plusieurs métaports, le signal ou le métasignal associé est créé et instancié dans la netlist VHDL générée.

4.4.1.5 Instanciation des signaux

La connexion avec les ports de l'entité se fait automatiquement lorsqu'un port est déclaré par l'utilisateur de DSX comme "externe". Un signal est alors crée et connecté au port de l'entité.

4.4.1.6 Instanciation des composants

Cette partie est construite en fonction des connexions entre les composants que l'utilisateur a défini dans son architecture matérielle.

Un exemple complet d'architecture matérielle décrite sous DSX, des métadonnées utilisés et la netlist VHDL générée sont représentés dans l'annexe de ce manuscrit.

4.4.2 Chaine de compilation de DSX_VHDL

Après avoir intégré le driver VHDL dans DSX et valider la génération d'une topcell VHDL synthétisable, il était pertinent de repenser à l'organisation de la chaine de compilation de DSX pour organiser plus ergonomiquement les flux de sortie relatif au VHDL, mais aussi de recharger l'exécutable de l'application logicielle dans les segments mémoire. L'idée était d'externaliser l'appel des outils de compilation et de synthèse en dehors de DSX et de restreindre le rôle de ce dernier à la génération de la topcell VHDL et d'un fichier Makefile qui contient les règles de la compilation et de synthèse. La nouvelle chaîne de compilation est présentée par la figure 4.3.

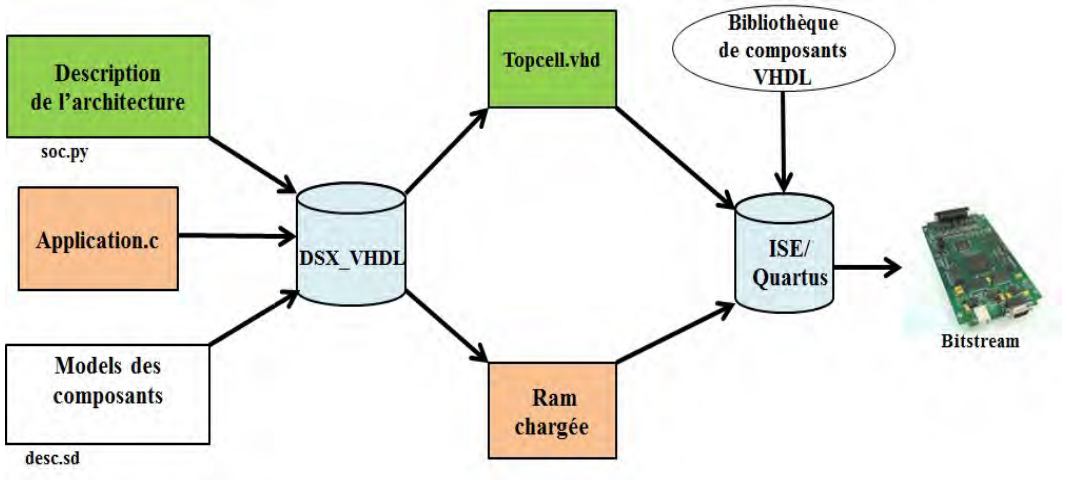


FIGURE 4.3 – Chaîne de compilation de DSX_VHDL

4.4.2.1 Fichiers d'entrée de DSX

Pour générer une architecture avec un certain nombre de caractéristiques, plusieurs fichiers doivent être présents à l'entrée de l'outil DSX.

Description de l'architecture : Il s'agit d'un fichier écrit en python dans lequel l'utilisateur décrit son architecture matérielle en utilisant les composants de la librairie VHDL. Ce fichier contient l'instanciation des composants nécessaires pour l'application ainsi que la définition des paramètres de chacun d'eux.
Après avoir instancié tous les composants, il est nécessaire de les connecter selon la description de l'architecture.

Les fichiers de metadonnées : L'utilisateur peut décrire les connexions entre les composants avec des commandes simples grâce à un fichier de métadonnées .sd. Ce fichier contient les propriétés de chaque composant : noms et types des ports, paramètres génériques... Les ports d'un composant peuvent avoir deux types : simple ou composé.
Prenons l'exemple du protocole VCI qui utilise plusieurs signaux de contrôle pour assurer le transfert de données entre le bus et le reste des composants. Dans le fichier de métadonnées, il suffit de définir un port VCI comme étant un port composé. Donc le fait de connecter les deux ports VCI des deux composants, implique une connexion automatique du reste des sous ports de ce protocole (les signaux d'acquittement, d'adresse..). La figure 4.4 montre la déclaration d'un port composé ainsi que la connexion entre les deux ports de deux composants.

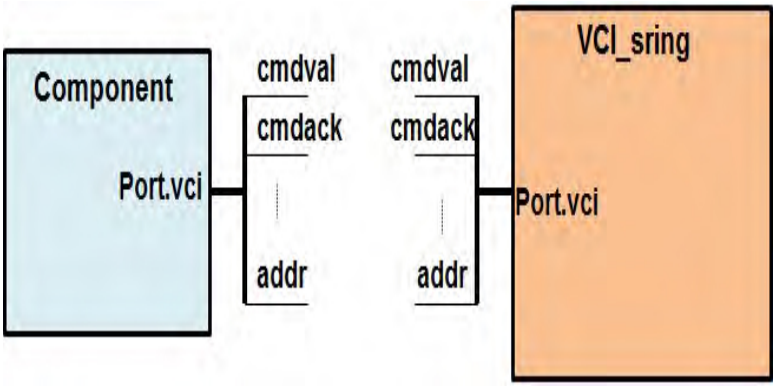


FIGURE 4.4 – Exemple de port composé

Description de l'application logicielle : Le volet logiciel est basé sur le développement d'une ou de plusieurs applications. L'expérimentation consiste à compiler l'application et à enregistrer le code exécutable obtenu dans les mémoires du système multiprocesseur avant de demander au système d'exécuter l'application.
Nous avons validé notre environnement par un modèle simple d'application basé sur l'idée

suivante :
Dans une architecture monoprocesseur contenant un coprocesseur : Un processeur envoie des données au coprocesseur 1. Le coprocesseur 1 fait le traitement de ces données et calcule des résultats. Le coprocesseur 1 envoie ces résultats au processeur. Ce dernier à son tour, compare les résultats obtenus avec ceux de référence, enregistrés dans la mémoire. Le processeur envoie un OK ou un KO vers l'UART suivant le résultat de comparaison. L'application est écrite en langage C et puis elle est compilée par une moulinette à fin de générer le fichier VHDL de la Ram chargée.

4.4.2.2 Fichiers de sortie de DSX

Après avoir préparé tous les fichiers d'entrée nécessaires à la création de l'architecture, DSX permet de générer les fichiers de sortie relatifs.

- Description de la netlist : Il s'agit de la description en VHDL de l'architecture matérielle.
- Un fichier Makefile est généré automatiquement et contient toutes les règles de compilation de l'application logicielle et de la synthèse FPGA.

4.4.2.3 Chaine de compilation de l'application logicielle

Pour générer le fichier VHDL qui contient le code de l'application logicielle, le fichier de description de l'application est compilé par une moulinette comme le montre la figure 4.5. La fonction main est compilée à l'aide du compilateur croisé relatif au processeur présent

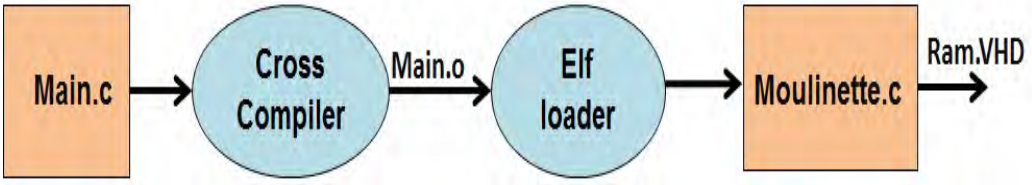


FIGURE 4.5 – Chaîne de compilation de l'application

dans l'architecture matérielle (c'est le MIPS dans notre cas). Il en résulte un exécutable en code objet, qui sera traduit en langage de description matériel à l'aide d'un elfloader. Le elfloader extrait le contenu de chaque partie (reset, boot, text..) et les charge dans un fichier VHDL contenant un ensemble de segments mémoire du vendeur FPGA : Altera/Xilinx. Un exemple de fichier de mémoire a la structure représenté par la figure 4.6.
La Ram obtenue est composée de 4 segments mémoire : boot, text, données, et except. La taille de chaque segment est paramétrable suivant les demandes de l'application.
Suivant la famille du FPGA cible, des bancs mémoire de petite taille sont instanciés dans les segments de la RAM. Le RAM16_S9 est un exemple de bloc mémoire de la famille Xilinx, sa taille est de 2Ko, donc pour avoir un segment de 8Ko, il suffit d'instancier 4 fois le RAM16_S9.

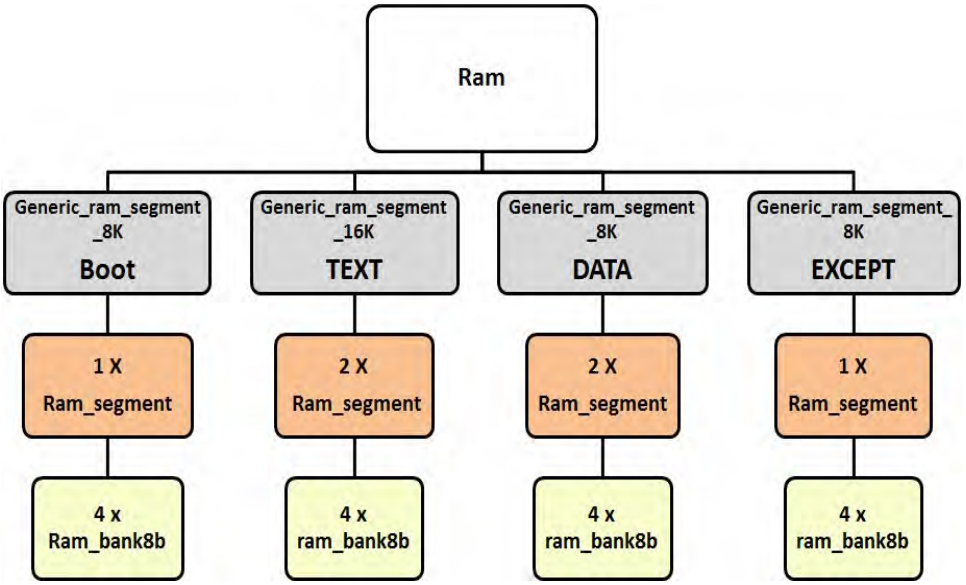


FIGURE 4.6 – La structure d'un exemple de Ram générée

4.4.2.4 Bibliothèque de composants génériques

Pour implémenter l'architecture sur une carte FPGA, il est nécessaire de trouver, dans la bibliothèque VHDL, les différents composants de la plateforme matérielle conçue, et encore les différents coprocesseurs nécessaires.
La bibliothèque existante propose un certain nombre de composants tel que :

Processeur MIPS : Le processeur MIPS R3000 est un processeur 32 bits industriel conçu dans les années 80. Son jeu d'instructions est de type RISC. Il communique avec le bus à travers le composant Xchache qui joue le rôle d'un pont entre les deux.

Réseau sur puce : SRING : Il s'agit d'un réseau d'interconnexion virtuel utilisant de protocole de communication VCI. Ce protocole définit plusieurs niveaux de service permettant l'échange de données entre les différents composants.

Générateur de coprocesseur UGH : L'outil UGH (User Guided High Level Synthesis) est l'outil de synthèse d'architecture permettant de transformer une tâche de l'application - décrite comme un algorithme séquentiel - en un coprocesseur matériel spécialisé, capable d'accéder aux mêmes canaux de communication que les autres tâches (logicielles) de l'application.

D'autre part, notre objectif est de créer des netlists suffisamment grandes pour cibler une carte de prototypage multi-FPGA et par la suite, tester l'efficacité des techniques de partitionnement et de multiplexage développées.

Pour cela nous avons proposé d'utiliser un coprocesseur spécifique qui a une taille paramétrable, ce qui permet d'augmenter considérablement la taille selon les besoins. Il s'agit d'un FPGA embarqué ou embedded-FPGA (e-FPGA).

FPGA embarqué : Pour ajouter plus de souplesse à notre architecture et encore la rendre plus réaliste, nous avons choisi d'intégrer un FPGA embarqué parmi l'ensemble des coprocesseurs [74]. En effet, les systèmes sur puce les plus récents contiennent une partie de cellules reconfigurables pour réutiliser une partie de la puce et pour y ajouter d'autres fonctionnalités même après la fabrication du système.
D'autres parts, les fabricants de FPGA et les développeurs d'IP proposent actuellement des composants matériels de FPGA embarqués qui peuvent être ajoutés dans le système sur puce [75],[76].
Le composant qu'on a ajouté dans la bibliothèque VHDL a été développé dans le laboratoire LIP6 [77]. Pour intégrer ce FPGA dans une architecture, il a fallu en ajouter une interface de communication. La figure 4.7 représente les détails de l'interface de connexion entre le e-FPGA et le composant multififos.
Pour communiquer avec le FPGA embarqué, nous avons besoin seulement d'une fifo de lecture et une fifo d'écriture. La première permet de configurer le FPGA par le bitstream envoyé par le processeur, et elle permet encore de sélectionner les entrées. La fifo d'écriture sert à récupérer les résultats à la sortie du FPGA.
La fifo de lecture est accessible depuis 3 ports :

- Dout : pour la donnée à lire
- Read : pour faire une demande de lecture
- ROK pour accepter la demande de lecture

Le même principe pour la fifo d'écriture. Elle est accessible depuis 3 ports également

- Din pour y écrire la donnée
- Write pour signaler une demande d'écriture
- WOK pour accepter la demande d'écriture.

La machine d'état de la figure 4.8 décrit l'automate de configuration et de fonctionnement de l'e-FPGA. Le e-FPGA attend jusqu'à la réception de la commande qui correspond à la configuration ou au mode fonctionnel. Si config est mis à '1', alors le loader récupère les mots du bitstream un par un à travers le port de données du fifo de lecture. Le bitstream est groupé en mots de 18 bits chacun, et qui peuvent être chargés aléatoirement. Dans notre FPGA embarqué, nous utilisons la technique du registre à décalage, donc le loader a besoin de 18 cycles d'horloge pour décoder les adresses du ligne et du colonne et pour charger les données dans les SRAMs convenables.
Si config = '0', il s'agit du mode fonctionnel, et donc les entrées du e-FPGA sont mises aux valeurs envoyées par le processeur. Si la fifo de lecture n'est pas pleine (traduit par un WOK = '1'), le e-FPGA écrit les résultats à travers le port de données du fifo d'écriture.

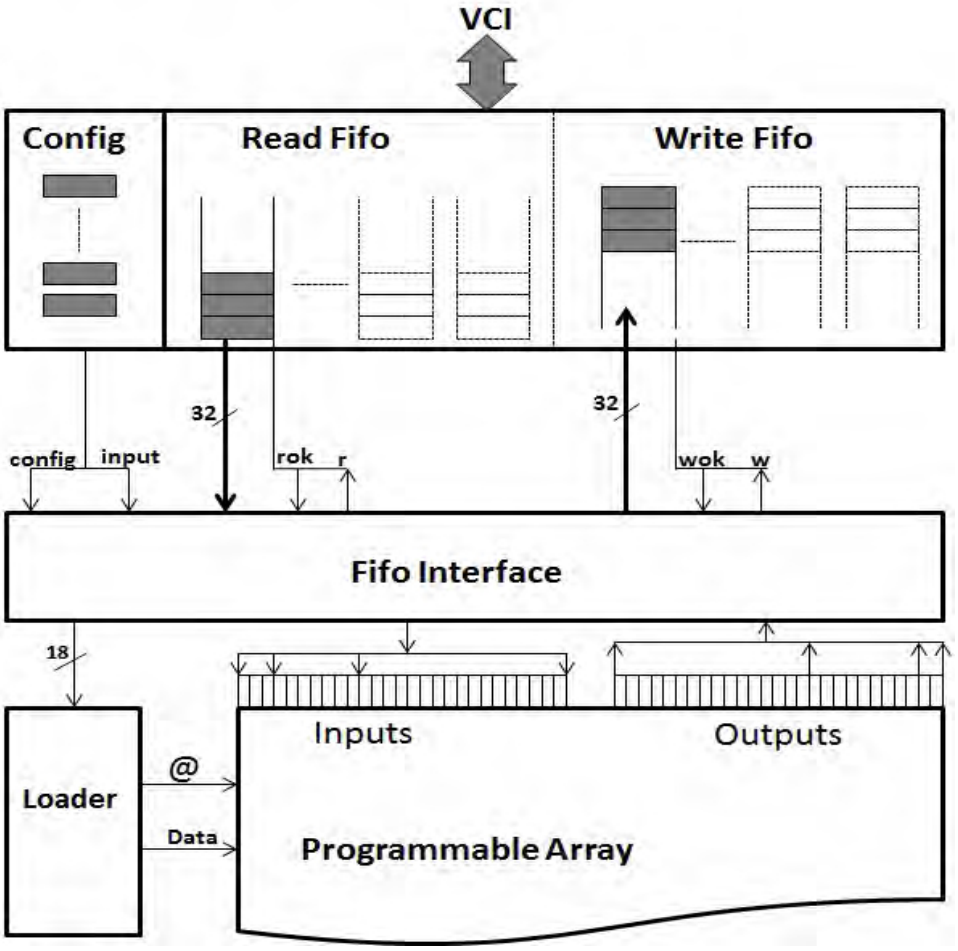


FIGURE 4.7 – Interface du FPGA embarqué

Système de synchronisation Multi-horloges : La plupart des systèmes sur puce récents sont des circuits multi-domaines. Pour cette raison nous avons pensé à intégrer d'autres horloges dans les benchmarks proposés pour les rendre plus réalistes.
Comme l'UART est le composant le plus sensible aux variations de fréquence, l'idée était de l'isoler (de point de vue fréquence) du reste du circuit, et de lui affecter une fréquence fixe, qui peut être différente à la fréquence du reste du circuit. Pour réaliser cette idée, nous avons pensé à insérer une fifo bi-synchrone entre le bus local et le composant UART [74]. La fifo bi-synchrone doit contenir deux compteurs pour contrôler les données écrites et lues depuis la fifo. Les compteurs binaires sont adaptés à l'adressage des mémoires synchrones. Néanmoins, le fait d'utiliser deux horloges différentes pour la lecture et l'écriture peut poser problème.
La meilleure solution pour passer les pointeurs entre les deux domaines d'horloge est d'utiliser des compteurs à codage Gray pour les deux pointeurs de notre fifo. Les compteurs à codage gray changent un seul bit à la fois. Donc, si le signal d'horloge de l'opération d'écriture (respectivement lecture) arrive au milieu de la transition du compteur de lecture (respectivement écriture), alors la valeur synchronisée peut être ou bien la valeur correcte,

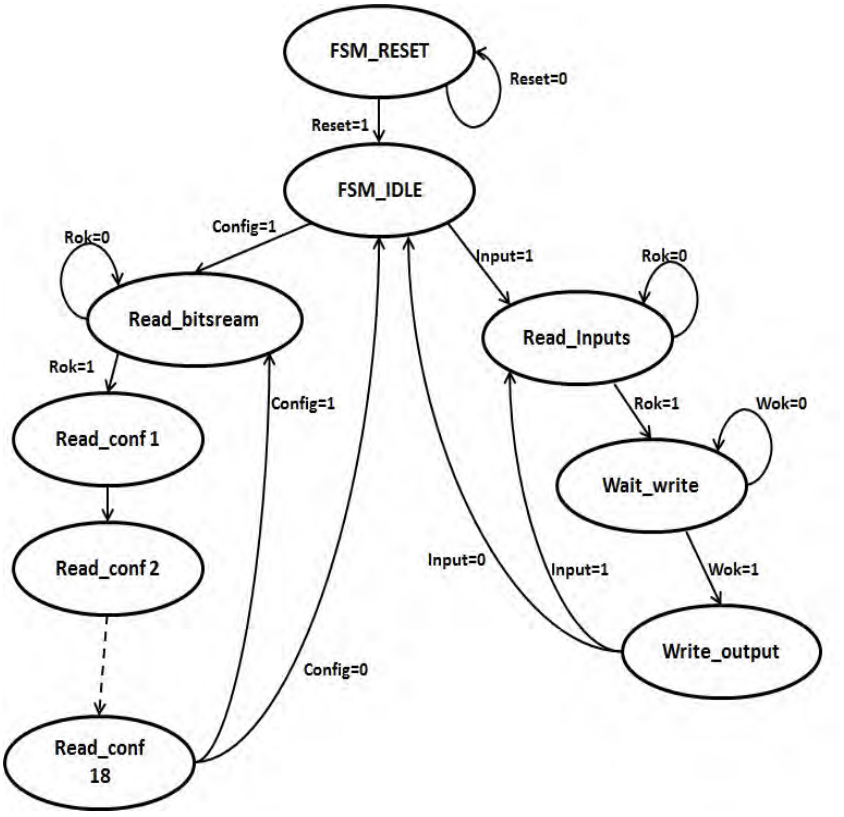


FIGURE 4.8 – Automate de configuration et de fonctionnement de l'e-FPGA.

ou l'ancienne valeur, puisqu'un seul bit qui peut changer à la fois.
En réalité, nous avons décidé de garder l'interface VCI du composant UART. Alors nous avons essayé de faire passer tous les signaux de contrôle du protocole VCI à travers la fifo bi-synchrone.
La figure 4.9 montre la connexion entre le bus et la fifo d'une part et entre la fifo et l'uart de l'autre part.

4.4.3 Exemple de benchmark généré

Un certain nombre de benchmarks générés par l'environnement décrit ci-dessus ont été utilisés dans [78]. Dans ce papier, les auteurs ont proposé une architecture multiprocesseur pour effectuer le calcul de l'intra-prédiction de la chaine de décodage du H.264/AVC.

4.5 Méthodologie de synthèse logique rapide

La plupart des outils commerciaux de synthèse logique utilisent l'approche traditionnelle qui est basée sur le flot descendant "top-down". Malgré que cette approche donne les meilleurs résultats de point de vue taille logique et timing, mais les concepteurs ont intérêt à chercher d'autres alternatives de synthèse à cause de la limitation du cycle de conception,

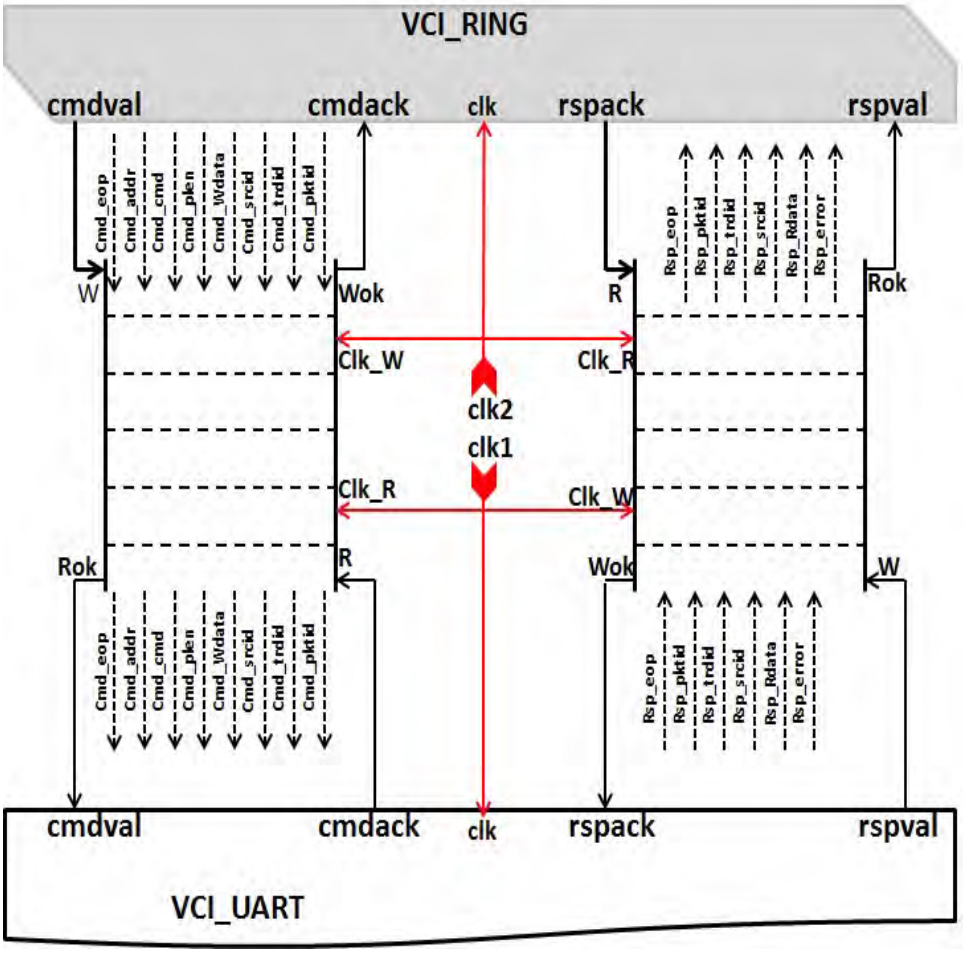


FIGURE 4.9 – Intégration d'une fifo asynchrone

et encore à cause de la limitation de mémoire.

Notre méthode proposée est essentiellement adaptée aux circuits développés par des groupes de concepteurs où le circuit est divisé en plusieurs petits blocs qui sont développés indépendamment [79]. Donc chaque partie validée du système peut être isolée pendant que les autres groupes continuent à travailler sur le reste du circuit. A chaque fois le système est synthétisé, seuls les blocs qui ont été modifiés seront synthétisés ce qui réduit considérablement le temps de synthèse.

L'approche que nous avons proposée est représenté par le graphe de la figure 4.10 La première étape consiste à compiler tous les fichiers du projet. Comme le système peut contenir plusieurs composants, notre générateur de benchmark permet de générer une liste de tous les composants chacun avec les composants qu'il instancie (appelés fils). Par la suite, tous les composants qui doivent être synthétisés sont énumérés dans une liste S. Deux types de composants doivent être éliminés. Le premier type inclus les composants ayant des paramètres génériques. En effet, chaque composant est synthétisé indépendamment de celui dans lequel il est instancié. Donc, au moment du synthèse, les paramètres génériques ne sont pas encore définis. Le second type inclus les composants qui n'ont été pas modifiés

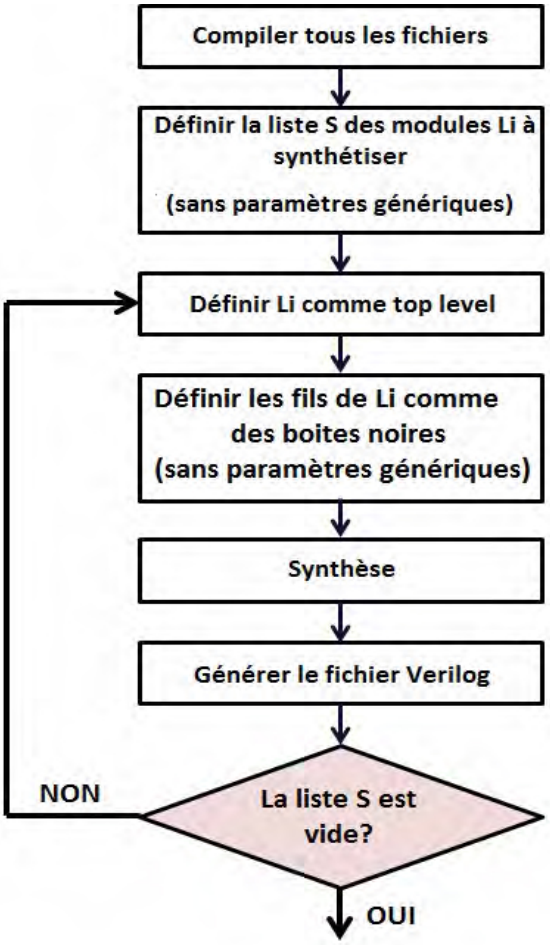


FIGURE 4.10 – Graphe de la méthode de synthèse logique proposée

depuis la dernière synthèse.
Une fois la liste S est définie, l'outil de synthèse sélectionne un composant "li" de cette liste qui sera définit comme étant le haut niveau du projet précédemment crée. Par la suite, tous les fils de ce composant sont sélectionnés comme étant des boites noires. En effet, à chaque composant "li", est assignée une liste de "fils". Cette liste contient tous les composants instanciés dans "li" sauf ceux qui ont des paramètres génériques. Ceux-ci sont remplacés par leurs fils à l'exception de ceux qui ont des paramètres génériques et ainsi de suite.
Dans l'exemple de la figure 4.11, nous considérons que, sauf le composant l12 possède des paramètres génériques. Donc la liste de fils relative au composant "l2" est définie comme suit :
set l2 {l10 l11 l02 l03}.
Une fois tous les fils sont sélectionnés comme étant des boites noires, "li" est synthétisé et le fichier Verilog post-synthèse est généré. Finalement, "li" est supprimé de la liste S. Tant que la liste S n'est pas encore vide, l'outil de synthèse refait toutes les étapes précédentes pour tous les composants restants. Toutes les étapes décrites ci-dessus sont développées dans un fichier script donné à l'entrée de l'outil de synthèse. A la fin, nous obtenons les

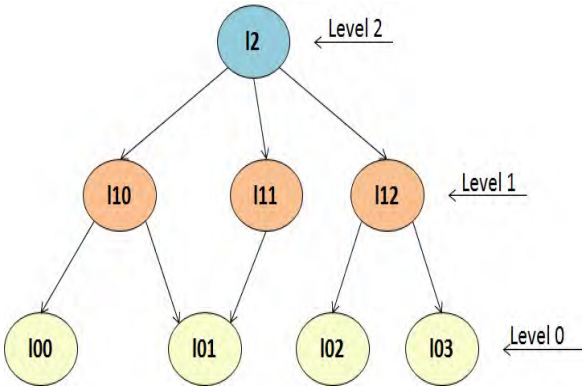


FIGURE 4.11 – Les différents niveaux d'un design hiérarchique

netlists de tous les composants qui ont été synthétisés. Ces netlists seront à l'entrée de l'outil de partitionnement.

4.6 Résultats expérimentaux

Nous avons fait quelques expérimentations pour évaluer notre technique de synthèse proposée par rapport au flot traditionnel de quelques outils commerciaux comme l'outil de Xilinx, XST [80] et celui de Synopsys : Synplify premier [58]. Par contre, il était impossible de synthétiser par XST nos circuits générés et dont la taille dépasse les quelques centaines de milliers de LUTs. Le processus de synthèse fini toujours par un dépassement de mémoire.

En réalisant les expérimentations, nous avons comparé le temps de synthèse avec notre technique proposée, et le temps de synthèse avec l'outil de Synplify premier. Dans la table 4.1, les résultats montrent que notre méthode apporte un gain moyen de 56,78%. En effet, dans les architectures multiprocesseurs, le composant MIPS est synthétisé une seule fois, mais il est appelé autant de fois que nécessaire. Donc, la même netlist est utilisée à chaque fois que ce composant est appelé par le top-level là ou il est considéré comme étant une boite noire. Contrairement à l'outil Synplify qui synthétise le Composant MIPS autant de fois qu'il est instancié dans le design.

D'autre part, nos benchmarks ont été testés par les outils de partitionnement de Flexras technology [17]. Les résultats de partitionnement de quelques circuits sur une carte multi-FPGAs sont présentés dans la table 4.1. La carte utilisée contient 6 FPGA virtex-6 (xc6vsx475tff1759)[8].

TABLE 4.1 – Durée de synthèse et résultats de prototypage de quelques benchmarks générés

Bench	LUTs	RAMLUTs	DSP	RAM	REG	NB FPGA	MUX Ratio	Freq (MHZ)	Synplify	Approche proposée	Imp
CPU_20	143217	6192	2	21	66937	1	1	80	518s	399s	22,97%
CPU_30	213524	9272	2	21	99588	3	9	27,78	864s	419s	51,5%
CPU_50	353697	15432	2	21	164587	4	12	20,83	1712s	454s	73,48%
CPU_125	879897	38532	2	21	408712	5	17	14,70	3023s	629s	79,19%

4.7 Conclusion

Dans ce chapitre, nous avons présenté notre générateur de benchmarks. Ce générateur permettant d'obtenir des architectures ayant des caractéristiques semblables à celles des circuits réels.

L'intérêt de cette plateforme générique est de faire varier simplement la complexité du système à travers les paramètres du générateur. Ainsi, on obtient un démonstrateur permettant à la fois la mise au point des méthodes proposées que leur évaluation à travers une série d'expériences.

Dans la deuxième partie, nous avons proposé une approche de synthèse logique permettant de réduire le temps de synthèse. Cette approche, donnant des netlists pas trop optimisés, est surtout adaptée au problème de prototypage là ou la taille logique et les optimisations ne sont pas aussi critiques.

Chapitre 5

Techniques de multiplexage proposées pour une plateforme de prototypage multi-FPGA

Sommaire

5.1 Introduction

La limitation du nombre de pins d'Entrées/Sorties dans les circuits FPGA constitue le problème principal du prototypage matériel sur une plateforme multi-FPGA. Cette limitation rend très difficile, voire impossible, le routage de tous les signaux qui apparaissent à l'interface de deux partitions et qui doivent passer d'un FPGA à un autre.
C'est dans le but de résoudre cette problématique que nous allons proposer dans ce chapitre une approche de routage basée sur un aspect itératif. Cette approche repose sur l'algorithme de routage PathFinder qui a été utilisé précédemment pour le routage des signaux intra-FPGA. L'aspect itératif de cette approche a pour objectif de bien exploiter

les ressources de routage disponibles et de donner les meilleurs résultats de point de vue fréquence de prototypage.

5.2 Multiplexage des signaux

Nous avons montré dans le chapitre 2 que le prototypage matériel présente la meilleure solution au concepteur pour valider son design dans un stade avancé du cycle de conception. Parmi les clés de succès du prototypage matériel, est sa fréquence de fonctionnement relativement grande par rapport à l'émulation et plus considérablement la simulation. Le multiplexage des signaux inter-FPGA, étant une solution inévitable pour remédier à la limitation du nombre de pins, a un coût considérable sur la fréquence de prototypage. En effet, la communication multiplexée entre les FPGAs, assurée par les pistes gravées sur la carte, est très coûteuse par rapport à une communication intra-FPGA. Par conséquent, il est important d'optimiser les paramètres de multiplexage pour réduire l'effet sur la performance et le comportement du système de prototypage.
La transmission des signaux est cadencée par une horloge de multiplexage multiplie de celle du système prototypé. La valeur de la période de prototypage dépend du taux de multiplexage (nombre de signaux transmis à travers le même fil physique) mais aussi, elle dépend du nombre de hops combinatoires (nombre de fois un chemin est coupé entre deux registres). En effet, le délai de transmission d'un signal est le temps nécessaire pour récupérer en O (Output) la bonne information envoyée par I (Input). Dans la figure 5.1, 4 signaux logiques (1,2,3,4) se partagent un même fil physique : ce nombre de signaux représente le taux de multiplexage. Le chemin de l'information envoyée par I et récupérée en O est coupé deux fois : une fois du $FPGA1 \rightarrow FPGA2$ et une deuxième fois du $FPGA2 \rightarrow FPGA1$. Le nombre de fois que ce chemin est coupé représente le nombre de hops combinatoires (autant de stages que de hops). Le délai de transmission est alors représenté par l'équation simplifiée 5.1 :

$$T_{SYS\_CLK} = mux\_ratio * NB\_combi - hop * T_{I/O\_CLK} \tag{5.1}$$

Le but recherché consiste à optimiser les paramètres de multiplexage, notamment le taux de multiplexage et le nombre de hops combinatoire afin d'améliorer la fréquence de fonctionnement.

5.2.1 Signaux non qualifiés pour le multiplexage

Afin d'augmenter la robustesse du circuit prototypé, l'idée est de sélectionner les signaux non qualifiés pour le multiplexage et de les router chacun seul à travers un fil physique.

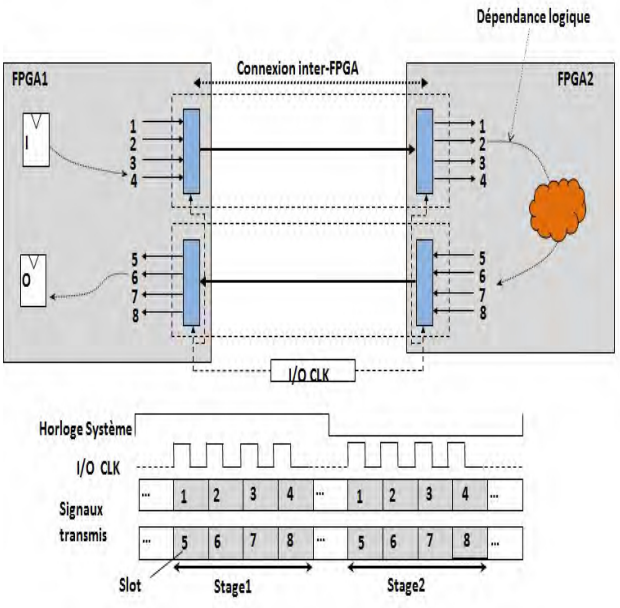


FIGURE 5.1 – Exemple de hops combinatoires

5.2.1.1 Signaux appartenant à des hops combinatoires

La présence des hops combinatoires dépend du design et de la qualité de partitionnement. Nous revenons plus tard avec plus de détails sur les contraintes qui doivent être prises en considération par l'outil de partitionnement. Une de ces contraintes consiste à réduire la longueur des chemins combinatoires afin de réduire le nombre de hops. La détermination de ces chemins nécessite l'implémentation d'un outil d'analyse de timing qui, à partir des caractéristiques des cellules et des connexions (délais intrinsèques), évalue les délais sur les différents chemins combinatoires. Certains de ces chemins ne sont pas fonctionnels (impossibles logiquement, chemins de test etc.) et l'outil d'analyse doit les identifier et les ignorer en suivant les consignes de l'utilisateur. En général les gros systèmes représentés par les netlists comportent plusieurs modules qui peuvent avoir différents domaines d'horloge (différentes fréquences de fonctionnement). La criticité pourrait être différente d'un domaine à un autre ce qui nécessite des traitements spécifiques pour les chemins combinatoires suivant leur appartenance. Ceci impose à l'outil de partitionnement de pouvoir attribuer des priorités différentes à ces chemins en favorisant certains (ne pas les couper) par rapport aux autres. L'élimination de tous les hops combinatoires est difficile, voire impossible. Ainsi, pour ne pas pénaliser la fréquence du système, notre idée consiste à fixer le nombre de stage à 1. Autrement dit, les signaux appartenant à un chemin combinatoire avec un nombre de hop ≥ 2 ne seront pas multiplexés.
Dans la figure 5.2, 4 signaux doivent être routés. D'après l'hypothèse que nous avons considérée, seuls les signaux S1 et S2 peuvent être multiplexés. Les signaux S3 et S4 ne sont pas qualifiés pour le multiplexage du fait que le premier ne finit pas dans un registre et le deuxième ne sort pas d'un registre. Par conséquent, le routage des signaux présents

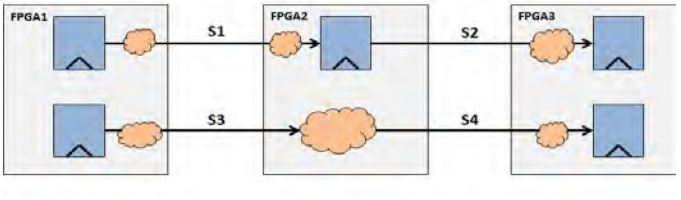


FIGURE 5.2 – Sélection des signaux non qualifiés pour le multiplexage

dans la figure 5.2 peut se faire dans un seul stage. Les signaux non qualifiés pour le multiplexage inclus aussi les signaux du top-level et les signaux critiques tel que le signal reset, les signaux d'horloge...

5.2.1.2 Les signaux appartenant à un chemin demi-cycle

Certains signaux appartiennent à ce qu'on appelle chemin demi-cycle ou (half cycle path). Autrement dit, ces signaux sont déclenchés puis détectés à des fronts d'horloge inversés. Dans la figure 5.3, le signal S1 n'est pas qualifié pour multiplexage.

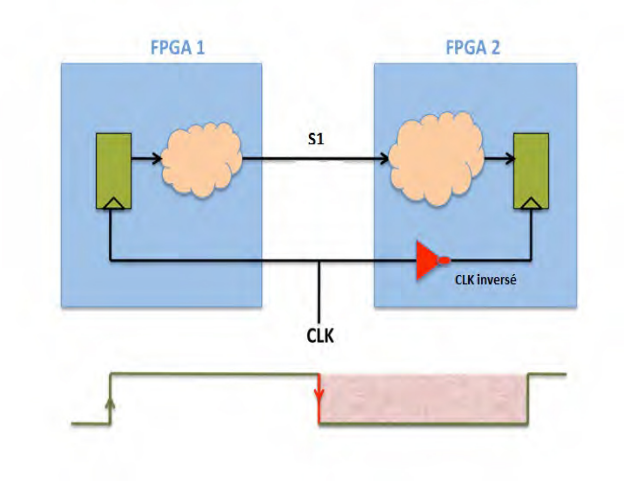


FIGURE 5.3 – Signal appartenant à un chemin demi-cycle

5.2.1.3 Les signaux d'horloge

Dans certains cas, les signaux d'horloge traversent une partie logique avant d'atteindre les flip flop qu'ils cadencent. Le multiplexage de ces signaux peut créer des problèmes de skew qui pouvant infecter le fonctionnement du circuit teste. Ce type de signaux ne doit pas être multiplexé. Dans les meilleurs cas, la partie logique traversée par les signaux d'horloge peut être répliquée évitant ainsi l'apparition d'un signal d'horloge coupé. Dans la figure 5.4, le signal S1 n'est pas qualifié poyr multiplexage. Dans le cas où la partie logique est

répliquée dans le FPGA destination, le signal S1 n'est plus coupé et les problèmes de skew sont éliminés.

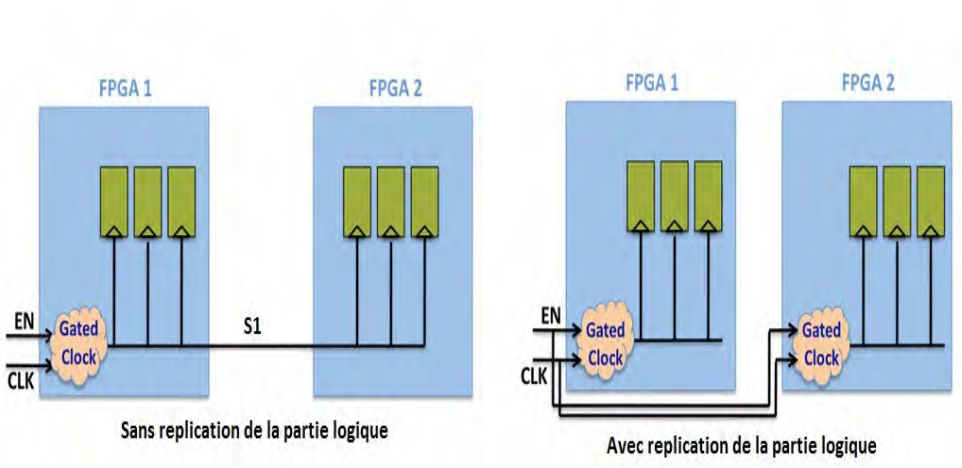


FIGURE 5.4 – Signal d'horloge traversant une partie logique

5.2.2 Routage des signaux multiplexés

En considérant l'hypothèse posée dans la section précédente, la figure 5.5 présente une période de prototypage ainsi que ces différentes composantes. L'expression de cette période

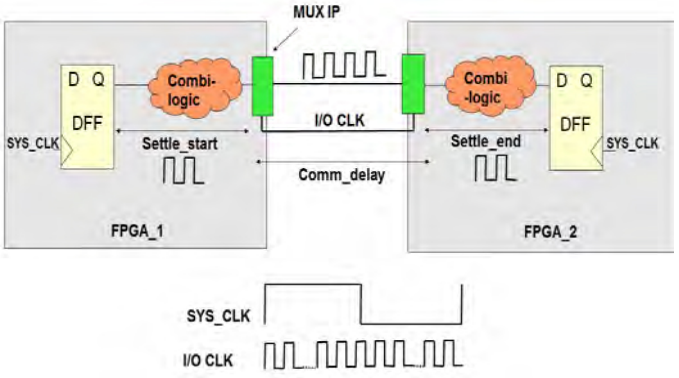


FIGURE 5.5 – Domaine d'horloge du système prototypé

est donnée par l'équation 5.2 :

$$T_{SYS\_CLK} = settle\_start + comm\_delay + settle\_end \tag{5.2}$$

Settle_start and settle_end correspondent respectivement aux délais de propagation interne du FPGA source et destination. Comm_delay correspond au délai de communication inter-FPGA. Elle dépend du taux de multiplexage qui n'est autre que le nombre de signaux transmis à travers le même fil physique. Pour réduire ce taux, nous proposons une approche itérative comme le montre la figure 5.6. Le routeur prend en entrée la description de la

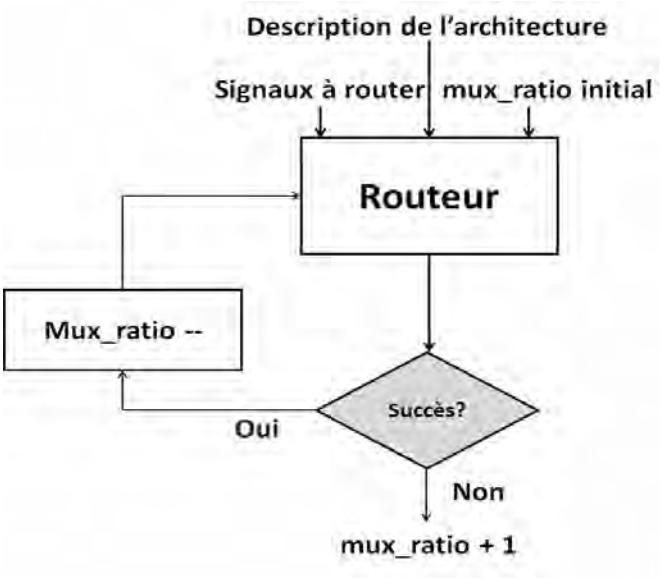


FIGURE 5.6 – Approche itérative pour la réduction du taux de multiplexage

carte multi-FPGA ainsi que la liste des signaux à router. Chaque signal est identifié par son nom et par sa source et ses destinations. Un autre paramètre qui doit être connu par le routeur qui est le taux de multiplexage initial (mux_ratio initial). Ce paramètre est calculé comme étant le plus grand ratio entre le nombre de signaux à multiplexer par rapport au nombre de traces physiques disponibles de toutes des paires FPGAs.
le routeur essaie de trouver une solution de routage avec un taux de multiplexage égal à mux_ratio initial. Cette solution existe puisque ce paramètre n'est que le pire cas de tous les taux de multiplexage entre toutes les paires de FPGA. Par la suite, tous les signaux sont déroutés, le taux de multiplexage est décrémenté, et le routeur va tenter à nouveau de trouver des chemins de routages non chevauchant avec le nouveau mux_ratio. Le routeur s'autorise de traverser des FPGA pour atteindre le FPGA destination (création de hop de routage). S'il arrive à router tous les signaux alors le taux de multiplexage va être décrémenté de nouveau, sinon le routeur gardera la valeur précédente de mux_ratio.
La figure 5.7 illustre l'utilité de notre approche. En effet, la valeur du mux_ratio initiale est égale à 4. Elle est donnée par le rapport entre le nombre de signaux de $F1 \rightarrow F3$ (égale à 4) et le nombre de traces physiques entre la même paire (égale à 1). La première itération représente la solution de routage avec un mux_ratio égale à 4. Autrement dit, chaque fil physique est partagé au plus par 4 signaux. Par la suite, la valeur de mux_ratio passe à 3. La solution de routage est donnée par la deuxième itération : des signaux entre $F1 \rightarrow F3$ sont routés en passant par F4. De même pour la troisième itération là ou un fil physique est partagé par au plus 2 signaux. Une dernière tentative de routage avec une valeur de mux_ratio égale à 1. Avec cette valeur, aucune solution n'existe, et par conséquent, le routeur retient la dernière valeur ayant une solution de routage acceptable qui est égale à 2.
En plus de l'aspect itératif de cette méthode, le routeur joue un rôle très important pour

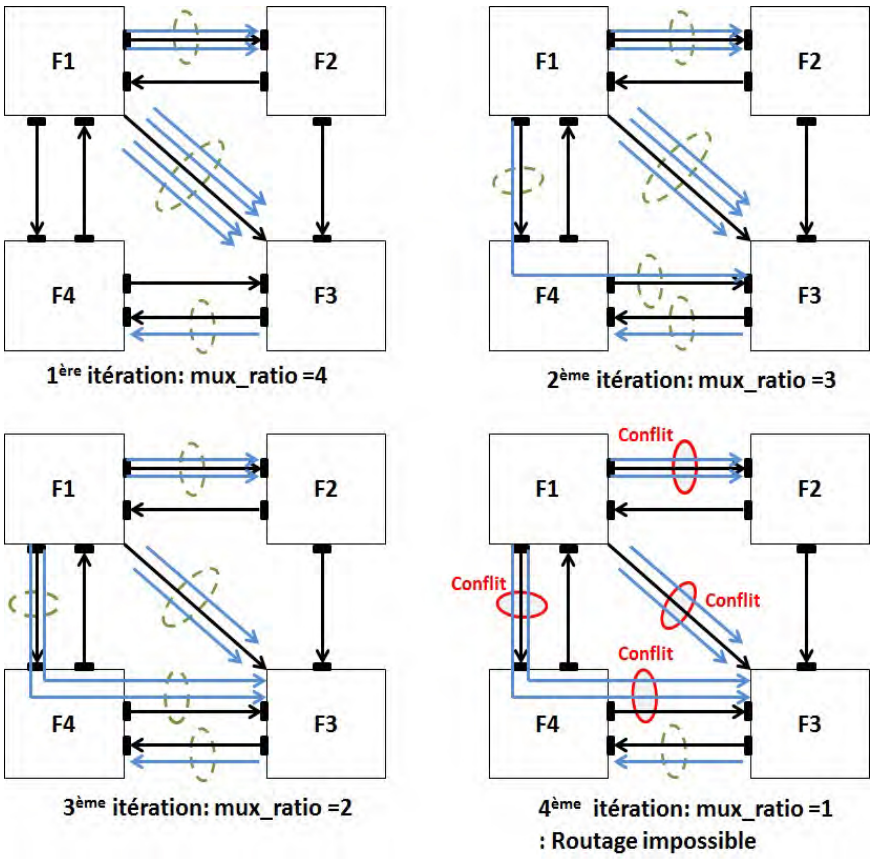


FIGURE 5.7 – Réduction du taux de multiplexage par une approche itérative

trouver des chemins de routage non chevauchant et répondant aux contraintes imposées par la valeur du mux_ratio. De ce fait, il est nécessaire de trouver un algorithme performant qui permet d'assigner, d'une manière optimisée, les signaux aux ressources de routage disponibles.

5.3 Routage Inter-FPGA : Constructif OU Itératif ?

L'algorithme de routage le plus utilisé dans l'état de l'art repose sur un aspect constructif [22], [20]. En effet, avant de router, une matrice de réservation est initialisée avec le nombre de connexions physiques entre les FPGAs i et j de la carte. Par la suite, le routeur sélectionne un signal aléatoirement et applique l'algorithme Dijkstra [71] pour déterminer le plus court chemin entre la source et la destination de ce signal tout en veillant à respecter les ressources disponibles. Si le chemin existe, alors la matrice de réservation est mise à jour en décrémentant 1 de chaque élément qui appartient au chemin sélectionné. Ainsi, une ressource utilisée pour router un signal à la phase N, ne peut plus être utilisée pour router un signal à la phase N+1. Par conséquent, l'inconvénient de cet algorithme réside dans deux aspects :

- L'algorithme est basé sur l'évitement des obstacles. Par conséquent, durant le routage

d'un signal, ce dernier ne peut utiliser que les ressources disponibles qui n'ont pas été utilisées avant par d'autres signaux. D'où la possibilité de tomber dans des solutions non routables qui correspondent à des minimas locaux.

- Le routage est constructif, autrement dit, les arbres de routages des signaux sont construits successivement pendant une seule itération. Un fil physique étant utilisé dans l'arbre de routage d'un signal, ne peut, en aucun cas être libéré pour l'exploiter dans le routage d'un autre signal. Et comme l'ordre de routage des signaux est aléatoire, aucune négociation ne peut se faire pour déterminer le signal qui a plus besoin d'une ressource de routage.

Pour mieux comprendre les limitations du routage constructif, nous avons modéliser un exemple représenté par la figure 5.8. Dans cet exemple, deux signaux $S1$ et $S2$ ayant

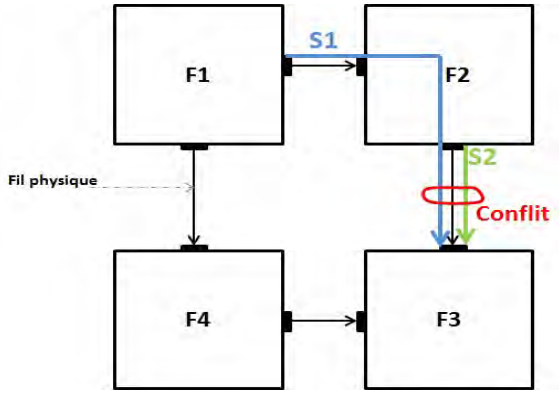


FIGURE 5.8 – Exemple montrant l'inconvénient d'un algorithme de routage constructif basé sur l'évitement d'obstacles

respectivement comme sources $F1$ et $F2$. $F3$ est la destination commune de ces deux signaux. La solution de ce problème de routage sans multiplxage existe : le signal $S1$ routé à travers $F1$, $F4$ et $F3$ et le signal $S2$ utilise le chemin $F2$,$F3$. Par contre, puisque l'ordre de routage des signaux est aléatoire, il se peut que le signal $S1$, sélectionné en premier, est routé à travers $F1$ - $F2$ -$F3$. Par conséquent, aucune solution de routage acceptable de $S2$ ne sera trouvée puisque la seule ressource entre $F2$ et $F3$ est déjà utilisée par $S1$. Dans le cas d'un algorithme constructif, l'ordre du routage des signaux est très important. Par contre, la détermination de cet ordre avant de commencer le routage est très compliquée voire impossible, ce qui rend la solution de routage donnée par un algorithme constructif très loin d'être optimale.

Pour remédier à ce problème, l'idée est d'appliquer un algorithme de routage itératif basé sur la négociation de congestion [81]. En effet, durant un certain nombre d'itérations, les signaux vont négocier lequel parmi eux a le plus besoin d'une ressource donnée. Cette méthode, nécessitant plus de temps, donne une solution de routage optimisée.

Si nous reprenons l'exemple de la figure 5.8, le problème posé peut être résolu à l'aide d'un algorithme itératif basé sur la négociation de congestion. En effet, la figure 5.9 montre les différentes itérations nécessaires pour éliminer le conflit. Avant de commencer le routage,

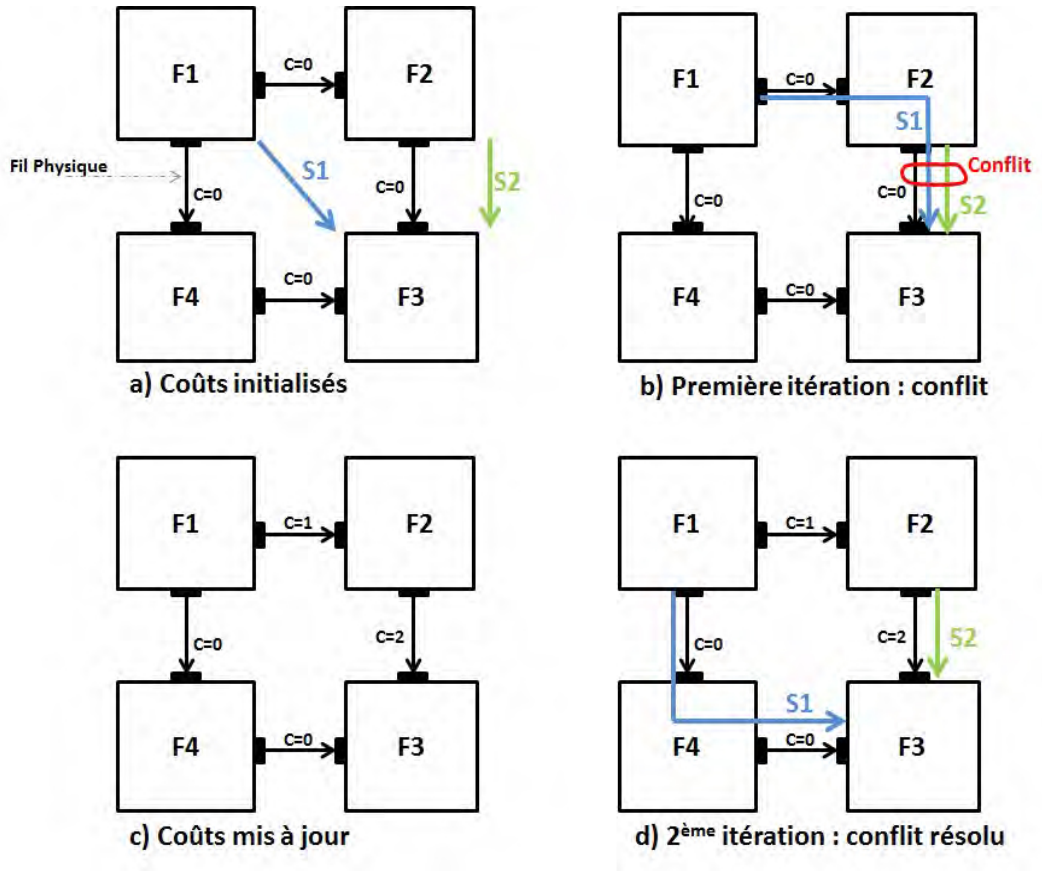


FIGURE 5.9 – Résolution d'un conflit de routage durant deux itérations

un coût égal à 0 est attribué à chaque fil physique : (F0 ;F1),(F1 ;F2),(F2 ;F3),(F0 ;F2). Dans chaque itération, s'il y a un partage de ressources, tous les signaux seront déroutés et le coût des ressources est mis à jour par la valeur qui correspond au nombre de signaux qui ont utilisé chaque fil dans l'itération précédente. Dans la deuxième itération de la figure 5.9, les coûts des fils (F1 ;F2) d'une part et (F2 ;F3) d'autre part sont mis respectivement à 1 et 2. Dans la troisième itération, si le signal $S1$ est sélectionné en premier, il va être routé à travers $F4$ puisque le coût du fil (F1 ;F4)=0 est inférieur à celui du fil (F1 ;F2)=1.

5.4 Algorithme de routage Pathfinder

Parmi les techniques qui existent dans l'état de l'art, Pathfinder [37] semble être l'algorithme le plus adapté à la problématique posée puisqu'il réalise un compromis entre les objectifs de performance et de routabilité. Comme indiqué dans le chapitre précédent, cet algorithme est utilisé principalement pour assurer le routage des signaux à l'intérieur d'un seul FPGA. Nous l'avons adapté pour un problème de routage inter-FPGA. Dans la section suivante, nous allons reprendre brièvement les principes de l'algorithme pathfinder qui peuvent résoudre le problème de routabilité inter-FPGA.
Généralement, une architecture (ensemble de FPGAs avec leurs connexions) est modélisée

algorithme 4 Pseudo code de l'algorithme de routage PathFinder[37]

```
Soit RT_i l'arbre de routage du signal i Durant l'itération en cours ;
Tantque (il existe des ressources de routage partagées) faire
  /*routeur global*/ ;
  pour tout signal i faire
    /*routeur d'un signal*/ ;
    vider l'arbre de routage RT_i /* Rip-up*/ ;
    RT_i = s_i ; s_i = source du signal i ;
    pour tout destination t_ij du signal i faire
      Initialiser la file de priorité PQ à RT_i avec un coût = 0 ;
      Tantque ( destination t_ij non atteinte ) faire
        enlever le noeud m ayant le coût minimal de PQ ;
        pour tout noeud voisin n du noeud m faire
          insérer n dans PQ avec un coût PathCost(n) = c(n) + PathCost(m) ;
        fin pour
      Fin Tantque
      pour tout noeud n appartenant au chemin de t_ij à s_i faire
        /*remonter le chemin des prédécesseurs de t_ij à s_i*/ ;
        Mettre à jour p(n) ;
        Ajouter n à RT_i ;
      fin pour
    fin pour
  fin pour
  mettre à jour h(n) pour toutes les ressources n ;
Fin Tantque
```

sous forme d'un graphe direct. Par conséquent, le problème de routage d'un signal donné se réduit à trouver un chemin direct dans G qui permet de connecter la source de ce signal à chacune de ses destinations. Après plusieurs itérations, il essai de converger vers une solution dans laquelle tous les signaux sont routés. Or, puisque les ressources de routage sont limitées, alors, le fait de trouver des chemins, uniques et non-chevauchant pour tous les signaux dans la netlist risque être un problème difficile.
Pathfinder utilise une approche itérative basée sur la négociation de congestion à fin de router tous les signaux dans la netlist. La routabilité est obtenue en forçant les signaux à négocier pour une ressource et par la suite détermine quel signal a le plus besoin de cette ressource. En utilisant une fonction de coût, il résout les congestions en augmentant petit à petit le coût des ressources de routage trop convoitées et à la fin il prend les chemins les plus courts avec aucune congestion. Le pseudo code de l'algorithme de PathFinder est présenté par l'algorithme 4.

Durant la première itération de routage, les signaux sont routés librement sans tenir compte du partage des ressources entre plusieurs signaux. A la fin de la première itération, plusieurs ressources peuvent être congestionnées du fait qu'elles sont utilisées par plusieurs signaux. Durant les itérations suivantes, le coût d'utilisation d'une ressource de routage augmente en se basant sur deux paramètres : le premier c'est le nombre de signaux uti-

lisant cette ressource, et le deuxième paramètre c'est l'historique de congestion de cette ressource. Donc, si la ressource est très congestionnée, les signaux qui ont des alternatives vers des ressources moins congestionnées sont forcés à choisir ces alternatives. Sinon, si les alternatives sont plus congestionnées, alors le signal peut être routé par cette ressource. Le coût d'utilisation d'une ressource "n" durant une itération quelconque est donné par l'équation 5.3

$$c(n) = (b(n) + h(n)) \cdot p(n) \tag{5.3}$$

Avec b(n) c'est le coût de base de la ressource n, h(n) est le coût historique d'utilisation de n durant les itérations précédentes, et p(n) est le coût actuel qui est proportionnel au nombre de signaux partageant la ressource n. En tenant compte du coût historique, les ressources congestionnées apparaissent avec un coût élevé même si elles sont légèrement occupées dans l'itération actuelle.
Au cours d'une itération, chaque signal est routé individuellement en utilisant l'algorithme de Dijkstra pour calculer plus court chemin [71], [82]. Cet algorithme cherche à router une connexion avec le plus court chemin. Il garantit de trouver le chemin de coût minimal s'il existe. Pour un signal possédant K destinations, l'algorithme Dijkstra est appliqué K fois pour réaliser le routage de toutes les connexions, en commençant par les connexions les plus proches de la source. Dans la première itération, l'algorithme Dijkstra explore le graphe de routage en partant de la source du signal et s'arrête quand il atteint la destination recherchée. Ensuite, toutes les ressources de routage utilisées pour router cette destination sont ajoutées à l'arbre de routage du signal, avec un coût égal à 0. Dans les applications suivantes de l'algorithme Dijkstra, l'exploration du graphe de routage commence à partir de toutes les ressources de l'arbre de routage et non pas seulement à partir de la source du signal. Ainsi, la prochaine destination est routée en utilisant l'arbre de routage des destinations appartenant au même signal.
L'algorithme PathFinder est actuellement largement utilisé par les chercheurs académiques et industriels. Le principe itératif pour résoudre la congestion et la négociation des signaux pour les ressources utilisées rendent cet algorithme très performant et attractif. Par ailleurs, l'algorithme PathFinder est indépendant de l'architecture FPGA. Il est en effet appliqué à un graphe de routage orienté. Il peut donc être utilisé pour n'importe quelle architecture qui pourrait être modélisée par un graphe de routage. Nous nous reposons sur cet aspect pour l'appliquer à l'architecture multi-FPGA.

5.5 Adaptation de l'algorithme de routage Pathfinder

Avant de présenter l'adaptation de l'algorithme Pathfinder au problème de routage inter-FPGA, nous formulons le problème du routage multiplexé dans les points suivants :

Contraintes :

- Les signaux multiplexés doivent avoir la même direction

- Le nombre de signaux multiplexés doit être inférieur à la capacité d'un fil physique.

Objectif : Router les signaux multiplexés avec le plus court chemin (minimiser le nombre de FPGA traversés)

5.5.1 Modélisation en graphe de routage

L'algorithme de routage Pathfinder est appliqué sur un graphe de routage. Par conséquent, notre but est de modéliser l'architecture multi-FPGA sous forme d'un graphe de routage orienté G(V,E). Pour cela, nous avons choisi la modélisation suivante :
L'ensemble des nœuds $V = \{v_1, ..., v_n\}$ représente les pins de tous les FPGAs ainsi que les FPGAs eux même. Autrement dit, chaque FPGA est représenté par un nœud top. L'ensemble des arrêtes $E = \{e_1, ..., e_m\}$ représente les connexions physiques entre les pins. Une Connexion unidirectionnelle est modélisée par une arrête orientée, tandis qu'une connexion bidirectionnelle telle celle entre un nœud top et un pin correspondant est représentée par deux arrêtes orientées.
La figure 5.10 montre un exemple de graphe de routage correspondant à une architecture à 3 FPGAs.

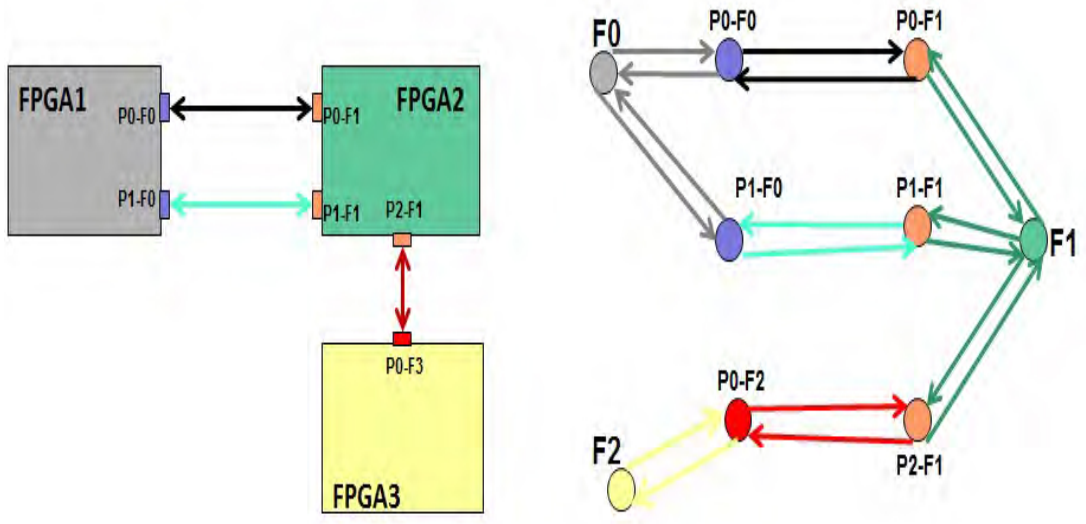


FIGURE 5.10 – Modélisation des ressources inter-FPGAs par un graphe de routage

5.5.2 Conflit de direction

L'algorithme de routage Pathfinder agit sur un graph direct. Chaque nœud de ce graphe possède une capacité qui ne doit pas être dépassée. La valeur de cette capacité reflète le nombre maximal de signaux qui peuvent partager cette ressource. Tout dépassement de cette capacité est appelé conflit. Ce conflit doit être résolu dans les prochaines itérations

de l'algorithme de routage sinon, et à l'absence d'une solution faisable, le routage est inachevé et abandonné. Par conséquent, la capacité de chaque nœud, n'est autre que le taux de multiplexage imposé. Dans le cas du routage inter-FPGA, les fils physiques sont bidirectionnels. Par conséquent, une ressource ayant une capacité supérieure à 1, peut être partagée par des signaux ayant des sens différents ce qui n'est pas acceptable lors du multiplexage qui impose que tous les signaux multiplexés aient la même direction. Ce problème est critique du fait que l'objectif principal de Pathfinder est de résoudre le problème de congestion, mais pas de direction. Donc avec la nouvelle contrainte, il a fallu trouver des solutions pour gérer le problème de direction.

Convention Les différents FPGA sont indexés d'une manière séquentielle, commençant par 0. On dit qu'un signal a un sens direct si l'index de son FPGA source est inférieur à celui de son FPGA destination. Les signaux de sens indirect se sont les signaux qui sont dirigés vers le sens inverse.

Terminologie

- Fils$[F_i; F_j]$: Nombre de fils total entre F_i et F_j
- Fils$[F_i\ rightarrow F_j]$: Nombre de fils physiques allant de F_i vers F_j
- Sig$[F_i; F_j]$: Nombre de signaux total à router entre F_i et F_j
- Sig$[F_i \rightarrow F_j]$: Nombre de signaux à router allant de F_i vers F_j

5.5.2.1 Graphe de routage unidirectionnel

Une première solution propose de router les signaux dans un graphe unidirectionnel. En effet, avant de procéder au routage des signaux, nous comptons affecter à chaque lien bidirectionnel de la carte un sens bien déterminé selon l'ensemble des signaux qui vont être routés. Le flot que nous proposons est représenté par la figure 5.11. La première étape consiste à créer le graphe unidirectionnel. L'idée était d'attribuer, selon des critères à détailler par la suite, un sens bien déterminé à tous les fils physiques. Dans le graphe de routage, ceci est traduit par une seule arrête dirigée entre chaque couple de nœuds.
Le nombre de fils physiques qui transmettent des signaux dans le sens direct entre deux FPGA, est proportionnel au nombre de signaux qui ont un sens direct entre ces deux FPGA. Et de même pour les connexions indirectes. Pour calculer le nombre de fils physiques ayant une direction donnée, il suffit d'appliquer l'équation 5.4

$$Fils[F1 \rightarrow F2] = \frac{Sig[F1 \rightarrow F2]}{Sig[F1; F2]} * Fils[F1; F2] \tag{5.4}$$

Avec $Sig[F1 \rightarrow F2]$ et $Sig[F1, F2]$ représentent respectivement le nombre de signaux entre F1 vers le F2 et le nombre signaux total entre les deux FPGAs. $NB_{fils}[F1, F2]$ représente le nombre de fils physiques entre les deux FPGAs.

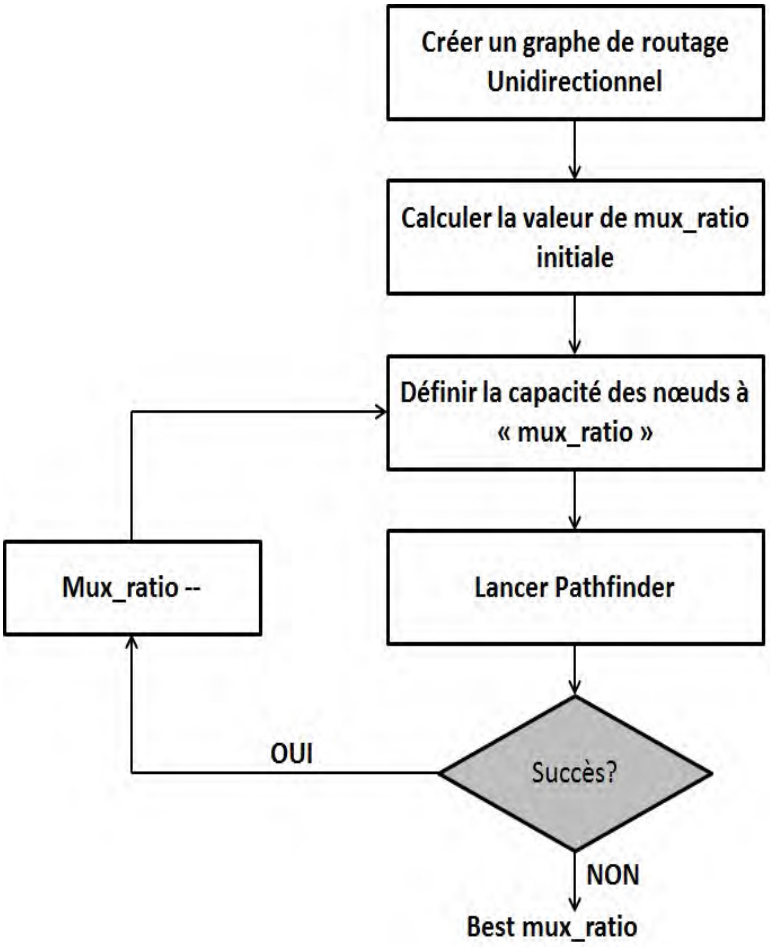


FIGURE 5.11 – Les étapes de routage sur un graphe unidirectionnel

La figure 5.12 montre un exemple de calcul du nombre de fils physiques dans chaque sens.

Dans la figure 5.12, le nombre de fils bidirectionnels entre le FPGA_1 et le FPGA_2 est égal à 5. Le Nombre de signaux transmis dans le sens direct est égal à 4. Et 3 c'est le nombre de signaux transmis dans le sens inverse.

Le nombre de fils physiques unidirectionnels dans le sens direct est égale à

$NB_{filsdirect} = \frac{4}{4+3} * 5 = 3$

Par conséquent, le nombre de fils physiques transmettant des signaux dans le sens indirect est égale à

$NB_{filsindirect} = NB_{filstotal} - NB_{filsdirect}$ = 5 - 3 =2.

Après avoir attribué une direction pour chaque fil physique, la deuxième étape du flot proposé consiste à calculer le paramètre mux_ratio initial comme nous l'avons expliqué dans la section 5.2. La capacité de tous les nœuds (à l'exception des nœuds top) du graphe est fixée à la valeur de mux_ratio. Cela veut dire que chaque nœud peut être partagé par un nombre maximal de signaux qui est égale à mux_ratio. La capacité des nœuds top est infinie. En effet, le nœud top ne représente pas réellement une ressource de routage, il

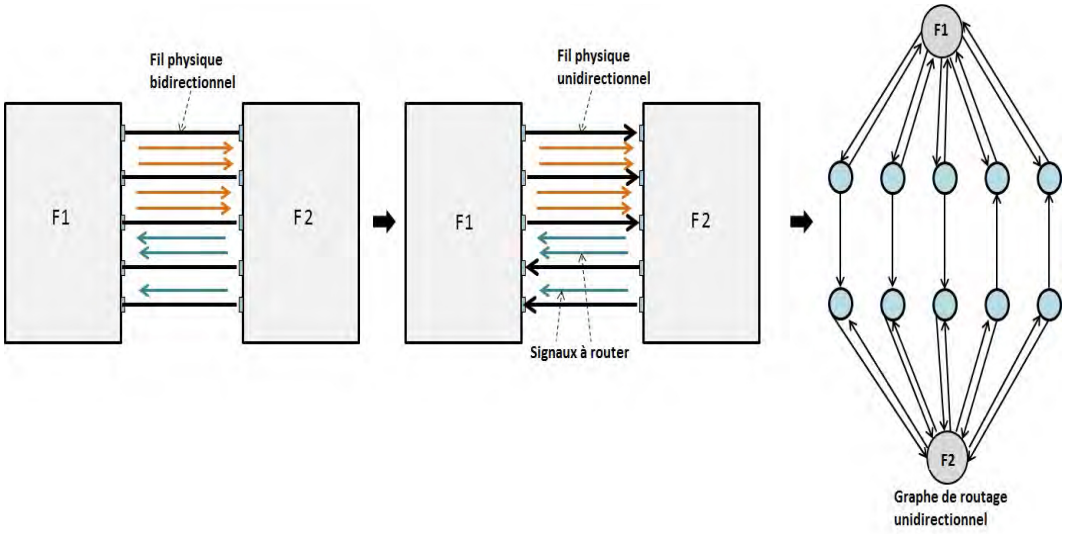


FIGURE 5.12 – Sélection des fils unidirectionnels proportionnellement aux signaux

représente le FPGA lui-même. Il est considéré comme le point de départ/ou d'arrivée/ou de passage des signaux. Ainsi tous les nœuds d'un même FPGA ont des chances égales d'être choisis par l'algorithme de routage.
Par la suite l'algorithme de routage Pathfinder est lancé pour chercher une solution qui vérifie deux contraintes :

- Tous les signaux doivent être routés.
- Aucun nœud ne soit partagé par un nombre supérieur à mux_ratio.

Si ces deux contraintes sont vérifiées, le paramètre mux_ratio est décrémenté et les deux dernières étapes sont refaites. Par contre, si la solution trouvée ne vérifie pas les contraintes fixées, alors, la recherche s'arrête en gardant meilleur solution de routage trouvée, c'est à dire celle avec mux_ratio = mux_ratio + 1.
Bien que l'algorithme de présélection des fils unidirectionnels résout considérablement le problème de conflit de sens, mais ça reste toujours une solution non optimisée du fait que des signaux peuvent être routés en utilisant des chemins différents à ceux considérés lors du calcul du nombre de fils unidirectionnels. Pour résoudre ce problème nous avons décidé de garder la forme bidirectionnelle du graphe de routage.

5.5.2.2 Graphe de routage bidirectionnel

Le graphe bidirectionnel permet une meilleure exploitation des fils physiques disponibles dans la carte de prototypage multi-FPGAs. Le flot de routage sur ce graphe est représenté par la figure 5.13. La première étape consiste créer ce graphe en utilisant deux arcs de sens opposé pour représenter chaque fil physique. Tout comme l'algorithme unidirectionnel, la deuxième étape consiste à calculer le paramètre mux_ratio initial. Dans le cas du graphe bidirectionnel, la capacité des nœuds est mise à 1 pour éviter le partage des ressources par des signaux ayant des sens opposés. Par contre, cette contrainte ne permet pas le routage

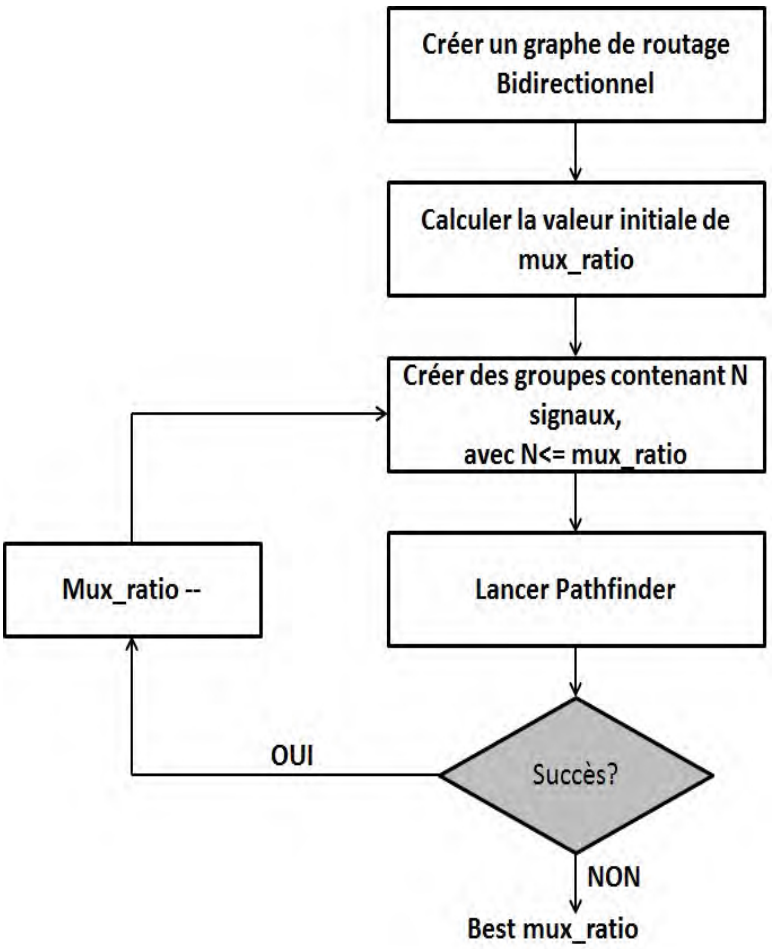


FIGURE 5.13 – Les étapes de routage sur un graphe bidirectionnel

d'un nombre de signaux qui dépasse généralement le nombre des fils physiques sur la carte. L'idée était de regrouper les signaux qui ont la même source et les mêmes destinations ensemble. La taille de chaque groupe ne dépasse pas la valeur de mux_ratio. La figure 5.14 représente des ensembles de signaux ayant un nombre maximal de 3 éléments. Les signaux appartenant au même groupe ont la même couleur. Un groupe de signaux est considéré comme étant un seul signal. Ainsi, l'hypergraphe représentant le design initial à router est remplacé par un hypergraphe compressé comme l'indique la figure 5.14. Par la suite, l'algorithme Pathfinder est lancé pour chercher une solution qui vérifie deux contraintes :

- Au bout de toutes les itérations, tous les signaux doivent être routés.
- Dans le cas d'un graphe bidirectionnel, chaque nœud doit être partagé par uniquement un seul groupe de signaux, autrement dit, un nombre de mux_ratio signaux.

Enfin, si la solution trouvée vérifie bien ces contraintes, alors le mux_ratio initial est décrémenté et les deux dernières étapes sont refaites. Sinon, la boucle s'arrête avec le meilleur taux de multiplexage trouvé et qui est égale à mux_ratio + 1.
Ce choix permet d'éviter les conflits de direction, puisque l'algorithme Pathfinder s'arrange à résoudre la congestion, et à la fin de toutes les itérations, aucun nœud ne sera utilisé par

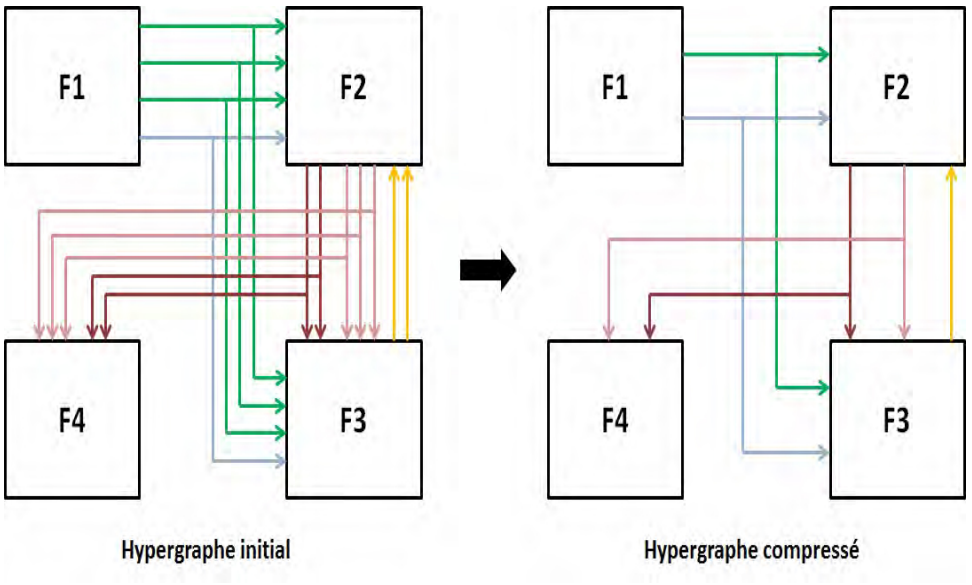


FIGURE 5.14 – Exemple de regroupement de signaux/branches

plus qu'un seul groupe de signaux, qui ont tous la même direction.

5.5.3 Modélisation d'un signal

Les signaux inter-FPGAs peuvent avoir plusieurs destinations. Ce signal peut être routé avec sa forme multi-points, ou décomposé sous forme de signaux bi-points.

5.5.3.1 Signal multi-points

Après le partitionnement du design à prototyper, un signal peut avoir plusieurs destinations réparties sur des FPGAs différents. L'algorithme de routage Pathfinder est capable de router un signal multi-points. Il sélectionne la source de ce signal et l'ensemble de toutes ses destinations. Puis il commence par trouver un chemin entre la source et la première destination. Le choix de cette destination est complètement arbitraire et ne se repose sur aucun critère. Le reste des destinations sont routées successivement tout comme la première et en utilisant en partie l'arbre de routage établi. Un signal multi-destination peut être routé de deux manière comme le montre la figure 5.15. Dans l'exemple de la figure 5.15, 3 partitions part 0, part 1, part 2 seront placées respectivement dans le F0, F1, F2. Le signal à router a comme source le F0, et deux destinations : F1 et F2. Deux solutions existent pour router ce signal :
La première solution consiste à utiliser deux chemins non chevauchant pour router les deux destinations. En effet, si aucune ressource disponible n'existe entre F1 et F2, le chemin vers part2 ne va pas intersecter celui vers part1. Cette solution est représentée par la figure 5.15-(a). La deuxième solution consiste à router la deuxième destination en utilisant une partie de l'arbre de routage de la première destination puisque les deux appartiennent au même

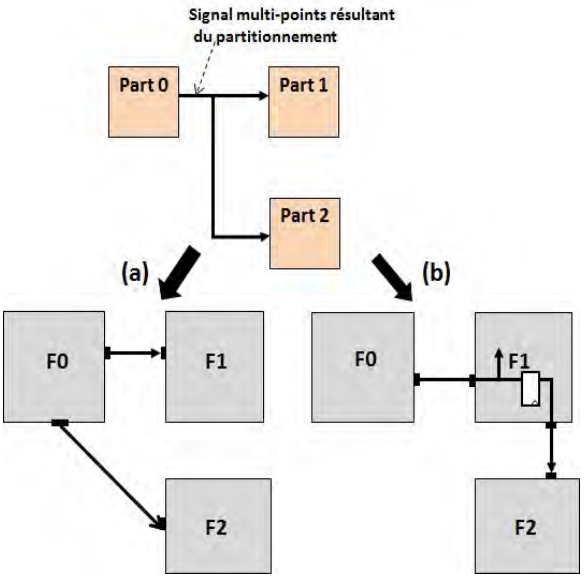


FIGURE 5.15 – Possibilités de routage d'un signal multi-destinations

équipotentiel. Cette solution est représentée par la figure 5.15-(b). Cette modélisation du signal semble être optimale surtout de point de vue nombre de pins utilisés. Par contre, cette forme du signal multi-points n'est pas du tout flexible surtout dans le cas d'un graphe bidirectionnel. En effet, lorsqu'il s'agit de grouper des signaux ensemble, le nombre de signaux qui ont la même source et les mêmes destinations est parfois largement inférieur à la valeur de mux_ratio. Par conséquent, sur un chemin entre une source S et une destination D, le nombre de signaux routés est inférieur au taux de multiplexage fixé, ce qui implique une mauvaise exploitation des ressources de routage disponibles.

5.5.3.2 Signal bi-points ou branche

Pour remédier à la non flexibilité de la forme multi-points d'un signal, l'idée est de décomposer un signal multi-destinations sous forme de branches ayant chacune une seule source et une seule destination. Chaque branche est routée indépendamment des autres à travers des chemins non chevauchant. Cette décomposition offre plus de flexibilité au niveau du choix des signaux à multiplexer, et par contre, elle masque le fait que certaines branches appartiennent au même équipotentiel.

5.6 Conclusion

Dans ce chapitre, nous avons présenté notre approche itérative de routage des signaux. Les différentes itérations ont pour but de mieux exploiter les connexions physiques disponibles et de réduire le taux de multiplexage des signaux inter-FPGA à fin d'optimiser la fréquence de prototypage. Le routage est basé essentiellement sur l'algorithme Pathfinder

adapté de façon qu'il gère le partage des ressources par plusieurs signaux tout en évitant les conflits de direction.
Dans le cas d'un routage sur un graphe unidirectionnel, les signaux sont routés, séparément, à travers des fils physiques unidirectionnels. Le sens de chaque fil est défini à l'avance proportionnellement au nombre de signaux entre chaque paire de FPGA. Dans le cas d'un routage sur un graphe bidirectionnel, la capacité de chaque ressource est fixée à 1, mais les signaux sont regroupés sous forme de groupe de signaux de taille égale au taux de multiplexage. Par conséquent, chaque ressource est utilisée par uniquement un seul groupe de signaux qui contient lui-même un nombre de signaux égale à mux_ratio.
La modélisation du signal joue un rôle très important dans la résolution du problème de routage. Nous avons proposé deux modélisations différentes : la forme multi-points et la forme bi-points.

Chapitre 6

Résultats expérimentaux

Sommaire

6.1 Introduction

Dans ce chapitre, nous évaluons les performances des techniques développées dans cette thèse. En effet, dans le chapitre 5, nous avons présenté notre approche pour router les si-

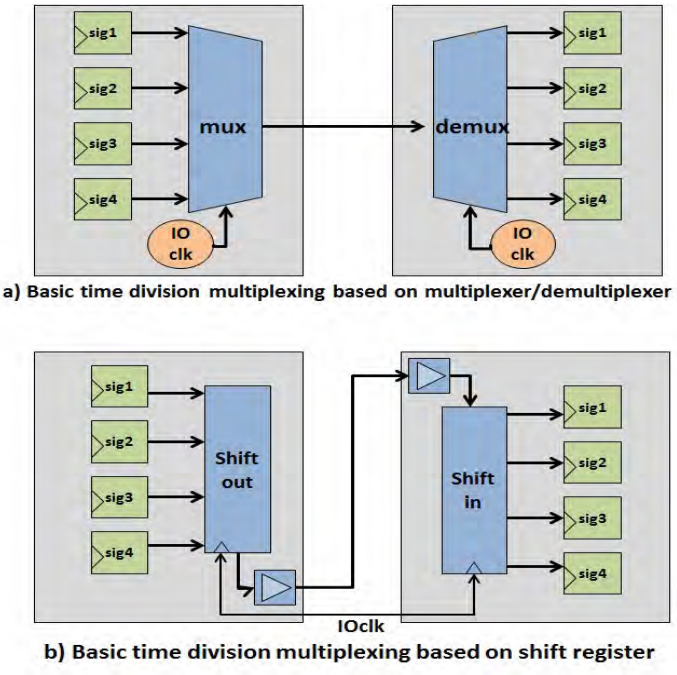


FIGURE 6.1 – Exemples d'architectures d'IPs de multiplexage

gnaux inter-FPGA sur une carte multi-FPGA. L'idée est basée sur le routage de ces signaux par un algorithme itératif Pathfinder sur un graphe de routage. Cet algorithme définit l'ensemble des signaux qui vont être transmis ensemble ainsi que le même chemin de routage. Par la suite, des IP de multiplexage sont insérés dans les FPGA source et destination afin d'acheminer en série les signaux sur le lien physique. Ces IP ajoutent des latences et des délais à considérer dans le calcul de la fréquence du système. Dans ce chapitre, nous allons comparer les résultats de routage des scénarios proposés. Pour chaque scénario, nous avons défini la forme des signaux (bi-points ou multi-points) ainsi que les paramètres du graphe de routage. Par la suite, nous allons comparer l'effet des outils de partitionnement sur les performances. Finalement, nous allons comparer notre approche itérative par rapport à une approche constructive utilisée par la plupart des techniques proposées dans l'état de l'art. Cette comparaison nous est utile pour mettre en évidence la performance des techniques développées dans cette thèse. Dans cette partie expérimentale, nous avons utilisé les circuits de test générés par le générateur de benchmarks décrit précédemment dans ce manuscrit.

6.2 Modèle de calcul de la période d'émulation

Pour comparer les performances des différentes approches développées, il a été nécessaire d'identifier les paramètres qui interviennent dans le calcul de la fréquence d'émulation.

6.2.1 Architecture des IPs de multiplexage

Dans le chapitre suivant, nous allons présenter l'architecture des IPs de multiplexage que nous avons conçu pour les intégrer dans la plateforme de prototypage contenant des Virtex 7 comme l'exige le cahier de charge du projet PPR : Plateforme de Prototypage Rapide. La construction et le test de cette carte ont été achevés au mois de mars 2013. Comme nous étions obligés de valider les techniques de multiplexage avant cette date, des IP de multiplexage ciblant une carte contenant des FPGAs Virtex 6 ont été utilisés.

6.2.1.1 Architecture traditionnelle des IPs de multiplexage (sans SERDES)

Les signaux multiplexés sont transmis à travers le même fil physique par le biais d'IPs de multiplexage insérés dans les FPGAs source et destination. Comme le montre la figure 6.1, les travaux précédents utilisent des IPs contenant des multiplexeurs ou des registres à décalage [22], [19]. Ces composants ne sont pas suffisamment pour les gros circuits là ou le taux de multiplexage peut atteindre des grandes valeurs. En plus, le débit inter-FPGA est considérablement limité.
Si nous prenons l'exemple d'IPs basés sur des multiplexeurs, la fréquence du système dépend du taux de multiplexage et le nombre de hops de routage. En effet, le nombre de hops de routage est le nombre de FPGA à traverser pour router un signal d'une source vers une destination. La figure 6.2 représente un exemple de signal routé à travers 2 hops de routage (F3 et F4). Un hop de routage est différent par rapport au hop combinatoire. Le premier existe lorsqu'un signal S est routé à travers un FPGA qui ne contient aucun élément logique auquel dépend le signal S. Un hop combinatoire existe lorsqu'un FPGA (autre que la source et la destination) contient des éléments logiques auxquels dépend le signal S. Lorsqu'un

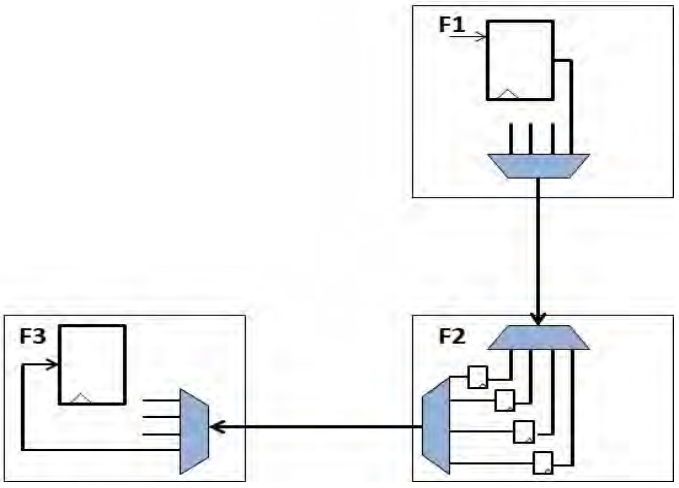


FIGURE 6.2 – Exemple de signal routé à travers deux hops de routage

signal est routé à travers un ou plusieurs hops de routage, des registres sont insérés dans chaque FPGA intermédiaire afin de récupérer la donnée avant d'être acheminée vers le

FPGA suivant. Ainsi 1 cycle d'horloge de sérialisation (I/O_CLK) est nécessaire pour traverser un hop. Par contre, l'outil de placement et routage peut ne pas satisfaire cette contrainte. Par conséquent, et afin de détendre l'étape de placement et routage, nous fixons la propagation intra-FPGA à 3 cycles d'horloge rapide, ainsi le cas pour traverser un hop de routage.
En utilisant les multiplexeurs, la fréquence de transmission inter-FPGA est fixée à 100MHz. Ainsi, la fréquence du système est donnée par l'équation 6.1

$$Sys\_freq = \frac{100}{mux\_ratio + 3 * NB\_hop} \tag{6.1}$$

6.2.1.2 Structure matérielle des IPs à base de primitives SERDES

Dans les FPGAs modernes (tel que le Virtex VI et Virtex7 de Xilinx), des nouveaux composants, appelés IOSERDES, sont utilisés pour accélérer la transmission inter-FPGA (Voir chapitre 2). La figure 6.3 représente les architectures des modules de conversion OSERDES et ISERDES utilisés respectivement pour une conversion parallèle vers série et série vers parallèle dans un FPGA Virtex VI. La taille de donnée sérialisée/désérialisée par une paire de IOSEDES est variable peut atteindre 10 bits pour un Virtex VI en connectant un ISERDES(ou OSERDES) maitre par un ISERDES(ou OSERDES) esclave comme le montre la figure 6.3. En utilisant les IOSERDES, le débit inter-FPGA est nettement

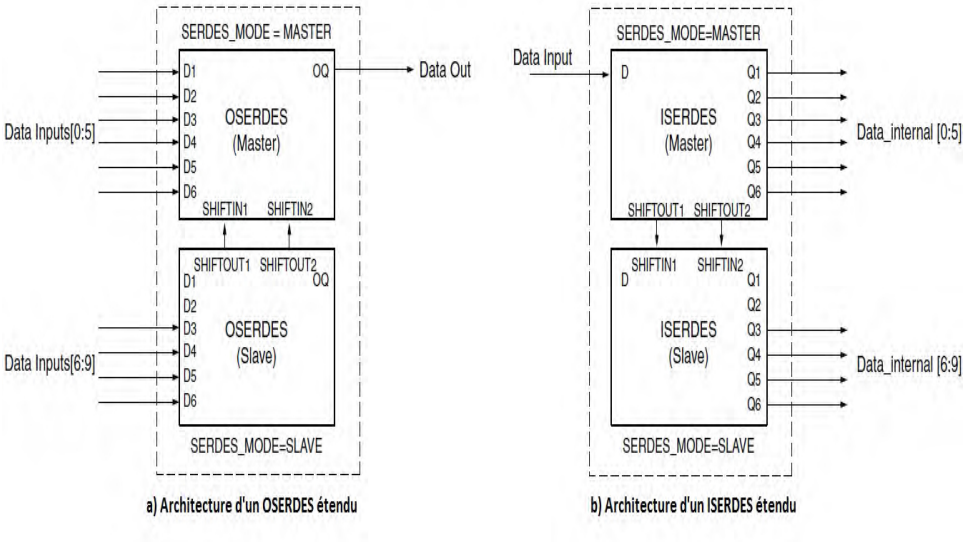


FIGURE 6.3 – Architecture matérielle des composants SERDES

augmenté. Pour cette raison, une paire différentielle (LVDS) est utilisée pour transférer les données entre les SERDES.
Dans une première étape, nous avons fixé la taille de la donnée sérialisée/désérialisée par une paire de IOSERDES à 4 bits. Par conséquent, lorsque le nombre de signaux dépasse le nombre de fils physiques disponibles, l'outil de partitionnement insère des SERDES 4 bits.

Autrement dit, les signaux sont transmis par des groupes de 4 entre un OSERDES et un ISERDES comme le montre la figure 6.4. Néanmoins, si le nombre de signaux inter-FPGA n'est pas un multiple de 4, certaines entrées de OSERDES (resp. sorties de ISERDES) sont laissées libres. Le mux_ratio est défini comme étant le nombre maximum de signaux transmis entre une paire ISERDES/OSERDES avec : $2 \leq mux\_ratio \leq 4$.

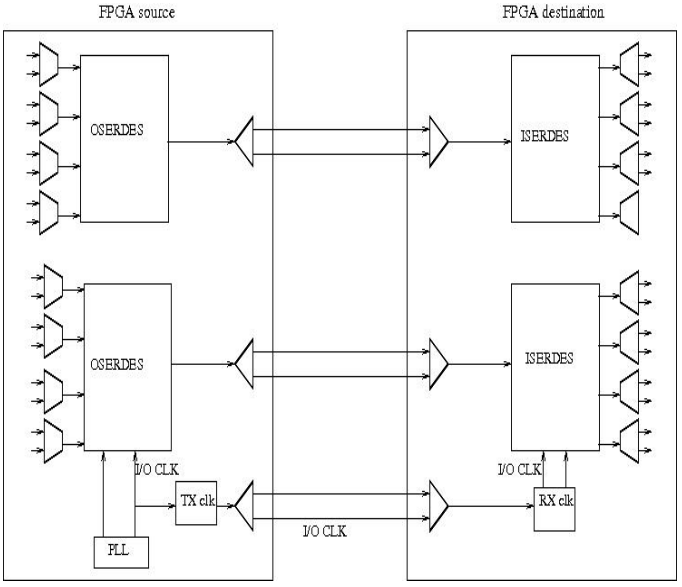


FIGURE 6.4 – Insertion des IPs de multiplexage dans les FPGAs source et destination

Dans les designs ayant une très grande connectivité, le nombre de signaux peut encore dépasser la capacité de transmission entre une paire de FPGA, même après la mise en œuvre des convertisseurs SERDES. Dans ce cas, des multiplexeur (respectivement des démultiplexeur) de largeur égale à n bits sont ajoutés à l'entrée de la OSERDES (resp. à la sortie de la ISERDES). Le paramètre n représente le nombre des mots de taille 4 bits qui sont transmis entre une paire de SERDES. Lorsque le paramètre mux_ratio est inférieur ou égal à 4, alors n=1. D'autre part, si le paramètre $mux\_ratio \geq 5$, alors $n \geq 2$. L'ensemble d'un SERDES de taille 4 avec un Mux ou Demux de largeur n représente l'IP de multiplexage.

6.2.2 Période d'émulation

Dans le chapitre 5, nous avons établi l'expression de la période d'émulation que nous allons reprendre dans cette section. Comme le montre la figure 6.6, l'expression de la période d'émulation est représentée par l'équation 6.2

$$T_{SYS\_CLK} = settle\_start + comm\_delay + settle\_end \tag{6.2}$$

Settle_start and settle_end correspondent respectivement aux délais de propagation interne du FPGA source et destination. Comm_delay correspond au délai de communication inter-FPGA. Elle dépend du taux de multiplexage qui n'est autre que le nombre de signaux

transmis à travers le même fil physique.

6.2.2.1 Communication entre les IPs basés sur des composants SERDES

L'IP de multiplexage gère la communication inter-FPGA. L'expression du temps de communication entre la partie TX et la partie RX est représentée par l'équation 6.3

$$Comm\_delay = T_{mux} + T_{routing\_hop} + T_{latencies} \tag{6.3}$$

T_{mux} est le temps de transfert de tous les signaux via le même fil physique et il est proportionnel au taux de multiplexage mux_ratio. $T_{latencies}$ représente les latences au niveau des SERDES. $T_{routing\_hop}$ est le temps de transfert des signaux à travers les hops de routage. La partie émettrice de l'IP envoie les données ainsi que le pattern de début de taille égale à 4 bits (utilisé pour la synchronisation) et enfin, le paramètre de vérification ou checksum de taille égale à 4 bits. La donnée de synchronisation est utilisée pour réordonner la donnée envoyée au niveau du FPGA de réception. En effet, lorsque l'IPs est utilisé pour la première fois, la donnée est reçue d'une manière décalée. Par conséquent, des opérations de décalage sont effectuées pour récupérer l'ordre initial. Comme la donnée est de taille 4, et en tenant compte des latences au niveau de l'IP, 8 cycles d'horloge I/O_{CLK} sont nécessaires pour finir l'opération de synchronisation. D'autre part, le transfert d'un mot de 4 bits est effectué durant 2 cycles de l'horloge de sérialisation. Ainsi, la figure 6.5 représente le chrono-gramme de transfert de la donnée de synchronisation ($S_1S_2S_3S_4$), 3 mots (A, B, et C) de taille égale à 4 bits chacun est le checksum (CS). En tenant compte de tous ces

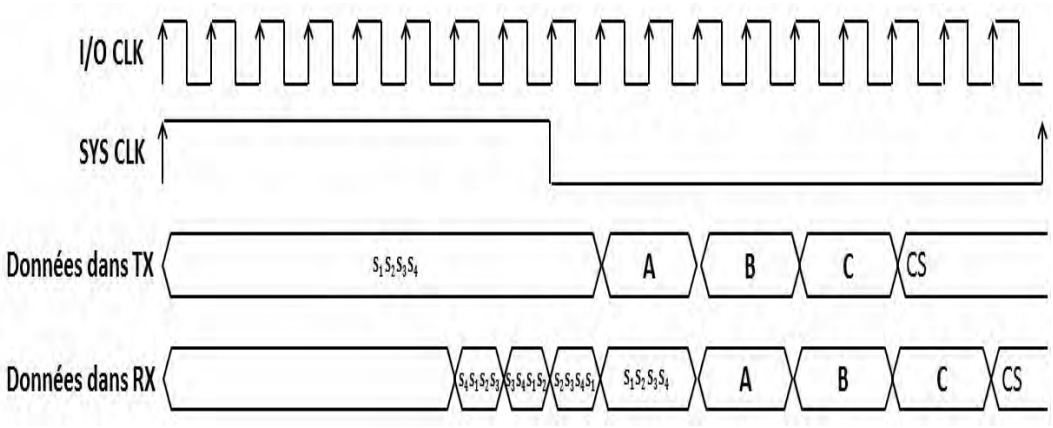


FIGURE 6.5 – Chronogramme du fonctionnement d'un IP de multiplexage

paramètres, le temps de communication inter-FPGA est représenté par l'équation 6.4

$$Comm\_delay = (12 + 2 * n) * T\_I/O_{CLK} \tag{6.4}$$

Avec n est le nombre de mots de 4 bits transmis à travers une paire de SERDES. Ainsi n n'est autre que le taux de multiplexage divisé par 4. Par conséquent, l'équation 6.4 devient

comme suit :

$$Comm\_delay = (12 + 2 * \frac{mux\_ratio}{4}) * T\_I/O_{CLK} \quad (6.5)$$

Pour assurer le routage à travers les hops de routage, des registres sont instanciées pour récupérer une donnée de taille 4 bits avant de l'envoyer au FPGA suivant. Comme dans le cas des IPs de multiplexage à base de multiplexeurs, 3 cycles d'horloge de sérialisation sont nécessaires pour traverser un hop de routage.
Après avoir défini les paramètres nécessaires, l'expression du comm_delay est représentée par l'équation 6.6

$$Comm\_delay = (NB_{R\_hop} * 3 + 12 + \frac{mux\_ratio}{2}) * T\_I/O_{CLK} \quad (6.6)$$

Avec NB_{R_hop} représente le nombre de hops de routage, mux_ratio est le taux de multiplexage qui est égale à n/4.

6.2.2.2 Délais intra-FPGA

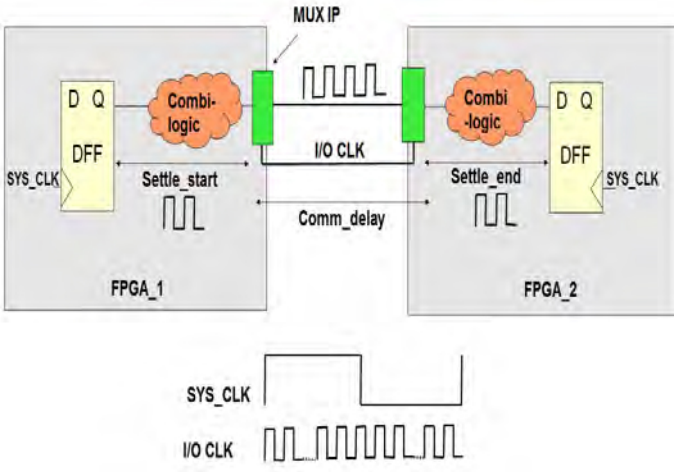


FIGURE 6.6 – Domaine d'horloge du système prototypé

Les délais de propagation interne sont déterminés après le placement et le routage intra-FPGA. En effet, en absence de contrainte utilisateur, l'outil de placement et routage opère individuellement sur chaque FPGA. Il essaie de trouver une solution qui satisfait les contraintes de l'horloge la plus rapide. Autrement dit, dans notre cas, le temps de propagation entre deux FF à l'intérieur du FPGA doit être inférieur ou égale à une période de l'horloge inter-FPGA (I/O CLK). Cette contrainte, difficile à satisfaire, oblige l'outil de placement et routage à effectuer plusieurs itérations et à faire un effort considérable. Pour relaxer cette opération, nous fixons une contrainte utilisateur appelée "multi cycle path". Cette contrainte définit le délai de propagation intra-FPGA à 3 cycles d'horloge de sérialisation (I/O CLK) au lieu d'un seul. Ainsi, la recherche d'une solution de mapping

intra-FPGA est plus flexible.

Ainsi, l'expression totale de la période d'émulation est représentée par l'équation 6.7

$$T_{SYS\_CLK} = 3+NB_{R\_hop}*3+12+\frac{mux\_ratio}{2}+3 = NB_{R\_hop}*3+18+\frac{mux\_ratio}{2} \quad (6.7)$$

Compte tenu de cette expression, la fréquence du système prototypé est représentée comme suit :

$$Sys\_freq = \frac{I/Ofrequency}{NB_{R\_hop} * 3 + 18 + \frac{mux\_ratio}{2}} \quad (6.8)$$

6.3 Environnement d'expérimentation

Pour réaliser l'ensemble des expériences détaillées dans les sections suivantes, nous avons utilisé un nombre de circuits de test générés par le générateur de benchmarks décrit dans le chapitre 4. Ces benchmarks, représentant des architectures multiprocesseurs, sont hiérarchiques pour permettre à l'outil de partitionnement WASGA d'agir à un haut niveau d'hiérarchie et de réduire ainsi le temps d'exécution et le nombre d'éléments traités.

6.3.1 Outil de partitionnement WASGA

WASGA est un outil de partitionnement multi-objectifs qui a pour but d'avoir la fréquence maximale du système prototypé [17]. En effet la fréquence du système dépend essentiellement de la façon avec laquelle le design (circuit) est réparti (découpé) entre les FPGA. Pour cette raison, au cours du partitionnement, WASGA considère les objectifs suivants :

- Optimisation des chemins combinatoires : WASGA découpe le design de sorte à réduire le nombre de sauts entre FPGA sur un chemin combinatoire.
- Optimisation du taux de multiplexage : WASGA découpe le design de façon à réduire le dépassement (overhead) entre le nombre de signaux communiquant entre 2 FPGA et le nombre de fils physiques disponibles sur la carte.
- Prise en compte des distances entre FPGA : Nombre de FPGA à traverser pour atteindre un FPGA destination depuis un FPGA source.
- Prise en compte des différents domaines d'horloge présents dans le circuit à partitionner.

Mis à part son rôle de partitionnement, WASGA permet encore de générer un ensemble de sous-design à partir du design original en se basant sur les assignations des instances aux différents FPGAs. Il assure donc les fonctionnalités suivantes :

- Unification des modèles dans le cas de synthèse séparée de différents modules du circuit à implémenter

- Réplication de certains modules dans les parties pour utiliser des ressources locales des FPGA (exemple : générateurs de constantes)

6.3.2 Plateforme matérielle de prototypage

Pour mapper l'ensemble des benchmarks de test, nous avons utilisé la carte DNV6F6PCIe de Dini group. Comme le montre la figure 6.7, cette carte contient 6 Virtex 6 LX550T utilisant tous le même package FF1759, autrement dit ils ont tous le même nombre de pins. La fréquence inter-FPGA (I/O CLK) est fixée à 500Mhz pour un transfert assuré par les composants SERDES et la normalisation LVDS. Par contre, dans le cas d'un multiplexage assuré par des multiplexeurs, le débit inter-FPGA est limité et il est fixé à 100MHz.

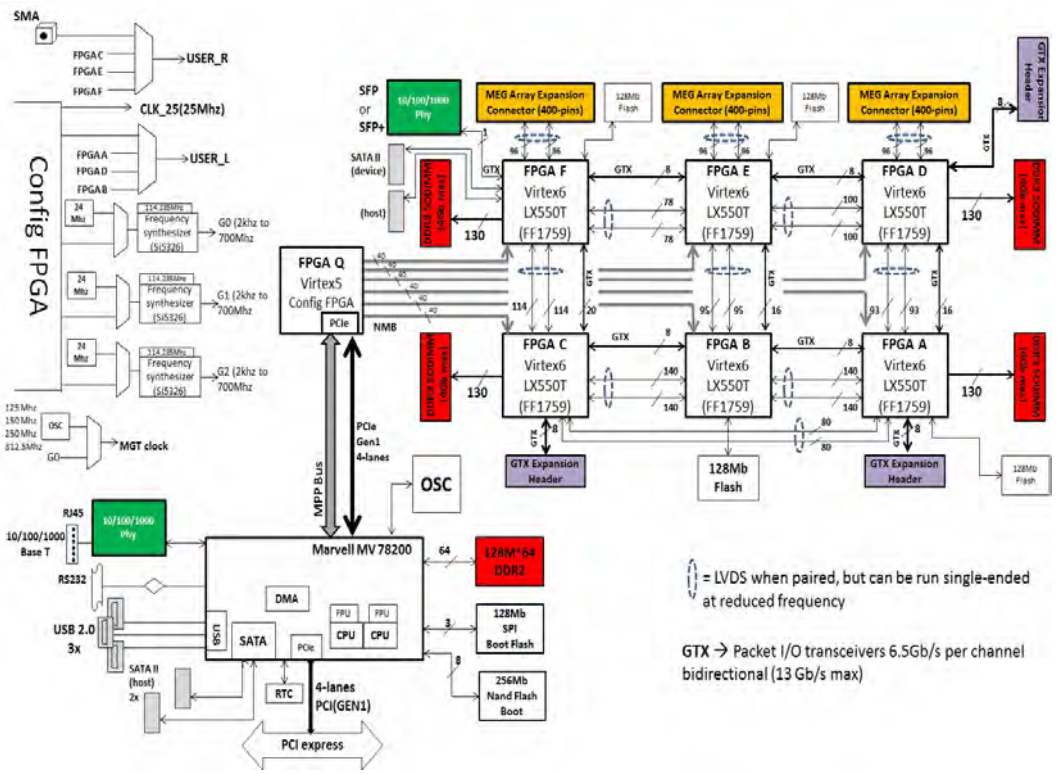


FIGURE 6.7 – Carte Dini à base de 6 Virtex 6

6.3.3 Comparaison des performances des IPs de multiplexage avec et sans SERDES

Nous comparons les performances des IPs de multiplexage avec et sans SERDES. Comme le montre la figure 6.8, un IP de multiplexage à base de primitives IOSERDES donne les meilleurs résultats de points de vu fréquence de fonctionnement. L'impact de ces composants est de plus en plus important pour les gros designs là ou le taux de multiplexage

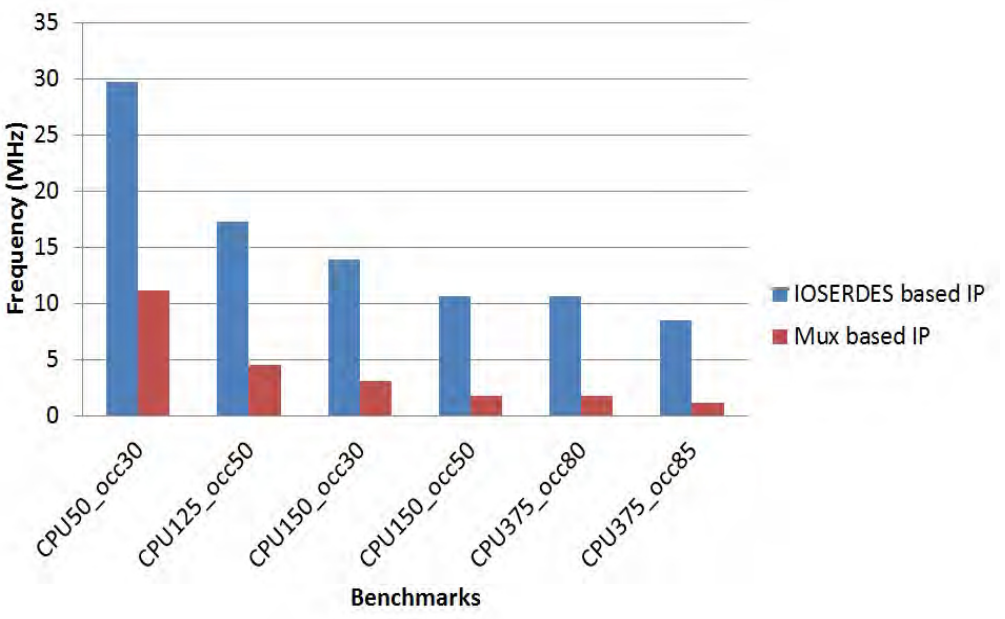


FIGURE 6.8 – Comparaison des performances de deux architectures d'IPs de multiplexage

est grand. L'architecture basée sur les multiplexeurs est flexible, facile à implémenter et peut être utilisée pour toutes les plateformes de prototypage. Cependant, cette architecture ne peut pas satisfaire à l'objectif de prototypage rapide qui consiste à permettre un transfert inter-FPGA très rapide. Par conséquent, l'architecture des IPs à base des IOSERDES offre une plus grande performance, mais l'effort de mise en œuvre est plus élevé et nécessite plus d'attention à gérer les délais sur les traces physiques.
Dans le reste ce cette partie expérimentale, nous utilisons des IPs de multiplexage basés sur des primitives IOSERDES.

6.4 Comparaison des résultats de routage : Branche ou Signal ?

Un signal peut avoir une ou plusieurs destinations. Ainsi, il peut être routé sous sa forme multi-points ou divisé sous forme de branche bi-points. Pour identifier la meilleure forme pour router un signal, nous avons proposé 4 scénarios différents. Dans chaque scénario, nous avons choisi une forme de signal ainsi que le graphe de routage adapté.

6.4.1 Scénario 1 : Routage de signaux multi-points sur graph unidirectionnel

Pour ce scénario, tous les signaux sont routés en gardant leurs formes multi-points. Le routeur agit sur un graphe unidirectionnel ou tous les arcs ont des directions définies à l'avance. La sélection de la direction de ces arcs se fait proportionnellement par rapport au

nombre de signaux transmis d'un FPGA à un autre. D'autre part, la capacité de chaque nœud est mise à une valeur supérieure ou égale à 1. Cette valeur correspond au taux de multiplexage de telle sorte qu'une ressource de routage peut transmettre un certain nombre de signaux. Si à la fin de chaque itération il existe une ressource qui est partagée par un nombre de signaux supérieur à la valeur de mux_ratio, un conflit est détecté, et le routeur tente de le résoudre dans les prochaine itérations.

6.4.2 Scénario 2 : Routage de branches sur un graphe unidirectionnel

Dans ce scénario, nous gardons toujours le graphe unidirectionnel avec des directions fixées à l'avance et dans lequel la capacité des ressources de routage peut être supérieure à 1 (Voir chapitre 5). Par contre tous les signaux multi-terminaux sont divisés sous forme de branches bi-points ou chacun n'est qu'une seule destination.

6.4.3 Scénario 3 : Routage de signaux multi-terminaux sur un graphe bidirectionnel

Dans ce scénario, nous avons gardé la forme multi-terminaux des signaux. D'autre part, ces signaux sont routés sur un graphe de routage bidirectionnel. Contrairement à un graphe unidirectionnel, le sens des liens physique n'est pas déterminé à l'avance. Seul le routeur est capable d'attribuer un groupe de signaux, ayant tous la même direction, à une ressource de routage bidirectionnelle. En effet, la capacité de chaque nœud est mise à 1 pour empêcher les signaux ayant des sens différents de partager la même ressource. Par contre, les signaux sont regroupés sous forme d'ensembles comportant chacun un nombre de signaux égale au taux de multiplexage. Le routeur traite un groupe de signaux comme étant un seul signal, et une fois affecté à chemin de routage, tous les signaux appartenant à ce groupe vont partager toutes les ressources de routage qui font partie de ce chemin.

6.4.4 Scénario 4 : Routage de branches sur un graphe bidirectionnel

Dans ce scénario, le routeur agit sur un graphe de routage bidirectionnel tout comme celui détaillé dans le scénario 3. Par contre, les signaux sont divisés sous forme de branches. Ces branches sont groupés sous forme d'ensemble ayant chacun un nombre égale au taux de multiplexage. Ainsi chaque ressource de routage va être utilisée par toutes les branches appartement au même ensemble

6.4.5 Comparaison des résultats de routage de chaque scénario

Pour comparer les résultats de routage de chaque scénario, nous avons utilisé un ensemble de benchmarks ayant chacun des signaux multi-terminaux qui représentent 70% du nombre total des signaux. La table 6.1 représente tous les résultats de routage relatifs aux

TABLE 6.1 – Comparison des résultats de routage des différents scénarios

Benchmark	scenario1			scenario2			scenario3			scenario4		
	mux_ratio	R_hop	Freq (MHz)	mux_ratio	R_hop	Freq (MHz)	mux_ratio	R_hop	Freq (MHz)	mux_ratio	R_hop	Freq (MHz)
Circuit A	12	2	17.85	15	2	16.66	4	2	20.83	4	1	26.31
Circuit B	18	3	13.88	24	2	14.7	4	3	17.24	7	1	23.8
Circuit C	24	3	12.82	44	2	11.36	11	3	15.15	11	1	21.73
Circuit D	50	3	9.61	50	2	10.63	15	3	14.28	20	1	18.51
Circuit E	119	6	4.9	116	4	5.55	57	2	9.8	56	4	8.33
Circuit F	160	3	4.67	168	3	4.5	68	3	8.19	68	1	9.8
Circuit G	220	5	3.4	256	1	3.44	89	2	7.46	86	3	7.14

4 scénarios. Les résultats représentés par ce tableau montrent que le routage sur un graphe bidirectionnel est considérablement meilleur que celui sur un graphe unidirectionnel. En effet, comme les directions ne sont pas prédéfinies d'avance sur un graphe bidirectionnel, le routeur choisi avec plus de flexibilité le chemin de routage de chaque signal. En plus, sur un graphe unidirectionnel, le sens de chaque fil physique est déterminé proportionnellement au nombre de signaux dans chaque direction ce qui impose, indirectement, un chemin de routage pour chaque signal. Dans la figure 6.4.5, le meilleur taux de multiplexage utilisé est égale à 4. Il est conditionné par le routage des 4 signaux de F1 à F3 qui ne peuvent être routés qu'à travers le fil qui existe entre F1 et F3. Par contre, les ressources entre F2 et F1 d'une part et entre F3 et F2 d'autre part sont mal exploités du fait qu'ils sont utilisés par seulement 3 signaux. Le taux de multiplexage aurait peut être passé à 2 si le graphe est bidirectionnel puisque deux signaux de F1 à F3 peuvent être routés à travers le chemin $F1 \rightarrow F2 \rightarrow F3$. La table 6.1 compare encore les résultats de routage des deux

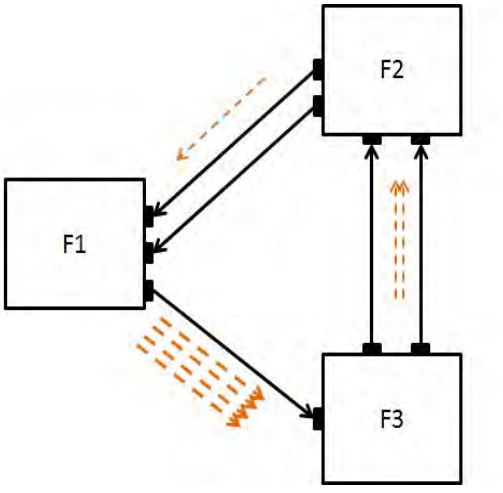


FIGURE 6.9 – solution de routage non optimisée sur un graphe unidirectionnel

formes des signaux : multi-points et branche. Sur un graphe bidirectionnel, le routage de branches donne souvent la meilleur fréquence de fonctionnement même si le taux de multiplexage est supérieure à celui du routage des signaux multi-terminaux. En effet, quand

TABLE 6.2 – Caractéristiques des benchmarks

Benchmark	LUTs	RAMLUTs	DSP	RAM	REG
CPU50_occ30	353697	15432	25	54	164587
CPU125_occ50	879897	38532	28	130	408712
CPU150_occ30	995750	43280	23	152	449210
CPU150_occ50	995750	43280	23	152	449210
CPU375_occ80	2236584	102590	35	257	1079621
CPU375_occ85	2236584	102590	35	257	1079621
CPU700_occ80	3988631	200991	41	281	1866433

le taux de multiplexage est décrémenté, le routeur tente de router les signaux à travers des chemins plus longs en utilisant des hops de routage. Comme nous utilisons des registres pour traverser chaque hop, ceci peut pénaliser la fréquence du système. Du coup, il est préférable de trouver le meilleur compromis entre la réduction du taux de multiplexage et l'augmentation du nombre de hop de routage.
Pour répondre à cet objectif la solution la plus simple était de calculer la fréquence de fonctionnement au cours du routage pour chaque taux de multiplexage. En effet, le paramètre mux_ratio est décrémenté après chaque itération. Avant de passer à l'itération suivante, le routeur doit calculer la fréquence du système à partir des résultats de routage trouvés (mux_ratio, et nombre de hop de routage). A la fin de toutes les itérations, les paramètres qui donnent la meilleure fréquence de fonctionnement sont maintenus.

6.5 Comparaison des techniques de routage : itératif ou constructif ?

Dans la section précédente, nous avons conclu que le routage de groupe de branche sur un graphe bidirectionnel donne les meilleurs résultats en termes de fréquence de fonctionnement. Nous allons donc utiliser ce scénario pour comparer l'algorithme de routage itératif par rapport à l'algorithme constructif utilisé dans l'état de l'art. Autrement dit, nous allons comparer les résultats de routage des signaux sous forme de branche sur un graphe bidirectionnel par l'algorithme itératif (NCR) que nous proposons par rapport à l'algorithme constructif (OAR). Comme son nom l'indique, NCR (Negotiated Congestion Routing) est basée sur la négociation de congestion puisque tous les signaux négocient, au cours de plusieurs itérations, leur besoin pour une ressource de routage. OAR (Obstacle Avoidance Routing) est basé sur l'évitement des obstacles. Autrement dit, si un signal utilise une ressource de routage, celle-la est éliminée de la matrice de réservation, et du coup, aucun autre signal ne peut l'utiliser pour éviter tout conflit.
La table 6.3 montre les résultats de routage pour les deux algorithmes. Les caractéristiques des benchmarks utilisés sont présentées dans la table 6.2. Les résultats montrent l'impact de l'algorithme itératif et son efficacité pour améliorer la performance du système d'émulation. La fréquence est augmentée par un taux moyen de 12,8% et l'impact du nouvel

TABLE 6.3 – Comparaison de la fréquence du système après routage par les algorithmes itératif et constructif

Benchmarks	OAR			NCR			Gain
	R_hop	mux_ratio	Freq(MHz)	R_hop	mux_ratio	Freq(MHz)	
CPU50_occ30	0	9	29.41	0	9	29.41	0%
CPU125_occ50	2	16	16.66	1	16	20	20.04%
CPU150_occ30	3	24	12.82	1	29	15.62	21.84%
CPU150_occ50	2	51	10.41	1	51	11.62	11.65%
CPU375_occ80	2	51	10.41	1	51	11.62	11.65%
CPU375_occ85	2	79	8.06	2	69	8.77	8.8%
CPU700_occ80	2	134	5.61	2	109	6.49	15.68%

algorithme est important pour les benchmarks les plus complexes. En effet, grâce à son aspect itératif, NCR permet d'éviter les minimums locaux et réduit le chemin de routage entre la source et la destination. En plus, notre stratégie de routage permet d'obtenir un compromis entre le taux de multiplexage et le nombre des hops de routage.

6.6 Comparaison des résultats de prototypage des flots Wasga et Certify

Pour mettre en œuvre la performance du flot de prototypage développé dans le cadre du projet PPR, nous avons utilisé l'outil de prototypage commercial de Synopsys qui est Certify. Comme nous l'avons indiqué dans le chapitre 2, cet outil permet de partitionner un design sur une plateforme multi-FPGA, puis multiplexer les signaux inter-FPGA avant de générer une netlist individuelle pour chaque FPGA.

6.6.1 Objectifs de partitionnement

Le partitionnement a un grand impact sur la performance du système de prototypage. Si un design est convenablement partitionné sur une carte multi-FPGA, le routage est plus optimisé et la fréquence de fonctionnement est meilleure. Pour cette raison, il a été convenu de diriger l'outil de partitionnement Wasga pour qu'il tienne en considération les contraintes qui permettent d'améliorer les paramètres dont dépend notre outil de routage inter-FPGA [83]. Avant de détailler les contraintes imposées à Wasga, nous donnons une petite description sur le comportement de Certify durant le partitionnement.

6.6.1.1 Caractéristiques du partitionnement de Certify

Avant de lancer Certify, l'utilisateur doit déterminer les contraintes relatives à l'occupation de chaque FPGA, mais aussi au taux de multiplexage. Au cours du partitionnement, Certify essaie, durant plusieurs itérations, de satisfaire les contraintes imposées. S'il trouve

une solution de partitionnement qui respecte la surface de chaque FPGA et le taux de multiplexage fixés, ce taux est décrémenté et l'outil tente de trouver une solution acceptable avec la nouvelle valeur.
Cette démarche exige à l'utilisateur d'avoir une connaissance considérable de son design pour pouvoir choisir un taux de multiplexage initial. Dans le cas où aucune solution de partitionnement n'existe, le designer doit modifier les paramètres choisis ce qui dégrade le temps de mise en œuvre du circuit à prototyper.

6.6.1.2 Les objectifs de partitionnement de Wasga

Contrairement à Certify, Wasga essaie de trouver une solution de partitionnement qui présente un compromis entre les différents paramètres. La qualité du partitionnement est mesurée par le degré de satisfaction des objectifs suivants :

Chemins combinatoires : La fréquence de fonctionnement du système sera imposée par le délai du plus long chemin combinatoire (entre 2 registres). Le délai sur un chemin combinatoire est fortement corrélé au nombre de fois ce chemin traverse la frontière d'un FPGA (nombre de hops combinatoires). Ceci est dû au fait que les délais des connexions inter-FPGA (utilisant les pistes de la carte) sont beaucoup plus importants que les délais des connexions locales aux FPGA. Par conséquent il est essentiel, dans la phase d'optimisation, d'absorber les signaux appartenant à des chemins combinatoires critiques. La détermination de ces chemins nécessite une analyse de timing qui, à partir des caractéristiques des cellules et des connexions (délais intrinsèques), évalue les délais sur les différents chemins combinatoires. Comme nous l'avons expliqué dans le chapitre 5, les signaux appartenant à un chemin combinatoire coupé plus qu'une fois, ne sont pas multiplexés lors du routage inter-FPGA

Ressources logiques hétérogènes et limitées : Aujourd'hui les FPGA comportent des blocs hétérogènes (LUT, Ram, DSP, Multiplier, Adder etc.). Le nombre de ces ressources est limité et représente la capacité logique du circuit. Lors du partitionnement, l'outil doit prendre en compte les ressources disponibles et en respecter le nombre. Contrairement à Certify qui applique un partitionnement sur des netlists niveau RTL, Wasga opère sur une netlist synthétisée au niveau portes logiques ce qui donne une visibilité exacte sur le nombre total de ressources et permet de respecter les contraintes imposées. En effet, l'étape de synthèse logique est importante du fait qu'une netliste non synthétisée offre plus de liberté à l'outil de partitionnement et moins de visibilité sur la vraie taille des parties. Dans ce cas, la synthèse logique est appliquée après le partitionnement sur chaque sous-netlist et engendre en général un dépassement des ressources logiques autorisées par l'FPGA. Par conséquent, une phase de légalisation (post-partitionnement) devient impérative au risque de dégrader la qualité de partitionnement et augmenter le temps d'exécution.

Nombre d'entrées/sorties limité : Le nombre de connecteurs d'entrée/sortie d'un FPGA est limité. Cette contrainte ne peut en général être satisfaite, car le découpage de la description fonctionnelle du circuit n'admet aucune solution, au niveau des entrées/sorties de chacune des parties, qui soit susceptible de respecter les contraintes des FPGA auxquelles ces parties seront affectées. La solution que nous avons détaillée dans le chapitre 5 est donc de procéder à un post-traitement qui permet de multiplexer certains signaux pour partager le même connecteur dans des fractions de temps différentes. L'insertion de ces multiplexeurs augmente les délais sur les chemins combinatoires. Ces délais sont corrélés au nombre de signaux multiplexés (taux de multiplexage) d'où l'intérêt de réduire leur nombre. Par conséquent, il est important de réduire le nombre de signaux entrant et sortant des différentes parties à fin d'obtenir de faibles taux de multiplexage. Mais le plus important c'est d'équilibrer ce taux entre toutes les paires de FPGAs. En effet, la figure 6.6.1.2

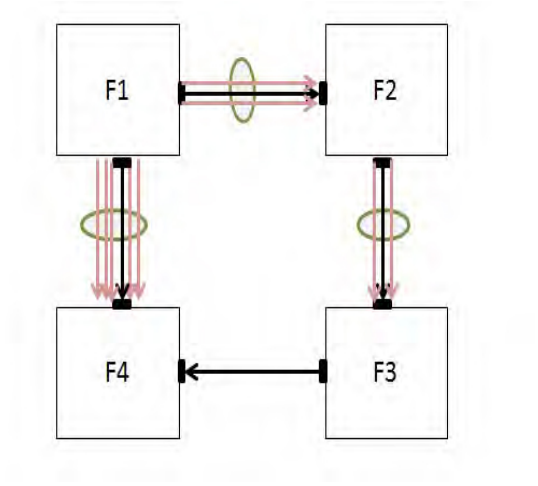


FIGURE 6.10 – Taux de multiplexage non équilibré entre les paires de FPGA

représente les différents taux de multiplexage d'une plateforme comportant 4 FPGAs. Bien que le taux entre les paires F1, F2 et F2, F3 sont faible, mais le calcul de la fréquence de fonctionnement dépend du pire cas (worst case) qui est celui entre F4 et F1. La fréquence du système aurait pu être meilleure si le taux de multiplexage était bien équilibré entre les différentes paires. Pour cette raison, l'objectif de Wasga au cours du partitionnement est de minimiser l'expression du paramètre C_p représentée par l'équation 6.9 :

$$C_p = \sqrt{\sum_{p=0}^{N} (\frac{Sp}{Tp})^2} \tag{6.9}$$

Avec N est le nombre de paires de FPGA sur la plateforme de prototypage, S_p est le nombre de signaux entre une paire p, et T_p est le nombre de fils physiques disponible entre la même paire p.

TABLE 6.4 – Caractéristiques des benchmarks

Benchmark	LUTs	RAMLUTs	DSP	RAM	REG
CPU20_occ10	143217	6192	2	21	66937
CPU20_occ20	143217	6192	2	21	66937
CPU30_occ20	213524	9272	12	33	99588
CPU30_occ30	213524	9272	12	33	99588
CPU50_occ30	353697	15432	25	54	164587
CPU50_occ50	353697	15432	25	54	164587
CPU125_50occ	879897	38532	28	130	408712
CPU125_occ65	879897	38532	28	130	408712

TABLE 6.5 – Comparaison entre la fréquence de fonctionnement résultante du prototypage par les flots de WASGA et CERTIFY

Benchmarks	WASGA				CERTIFY			
	cut signals	NB_FPGA	R_hop	mux_ratio	cut signals	NB_FPGA	R_hop	mux_ratio
CPU20_occ10	1545	6	0	3	3316	6	0	10
CPU20_occ20	1002	4	0	3	1634	4	0	4
CPU30_occ20	1710	4	0	3	3076	4	0	6
CPU30_occ30	1487	4	0	4	2521	3	0	7
CPU50_occ30	2819	4	0	5	5279	4	0	11
CPU50_occ50	2202	4	0	6	4019	3	0	9
CPU125_occ50	7809	6	1	11	NR	NR	NR	NR
CPU125_occ65	7644	5	0	12	NR	NR	NR	NR

Différents types de signaux : En général dans une netlist il y a différents types de signaux : Logiques et Globaux. Les signaux logiques connectent les portes entre elles et définissent la fonctionnalité du circuit. Les signaux globaux sont des signaux de contrôle (horloge, reset, etc.). Ces signaux n'ont pas la même criticité et ne doivent pas être traités de la même façon. En effet, les signaux de contrôle sont très critiques et ne doivent pas être ralentis en traversant différentes parties. Wasga permet d'identifier ces signaux et leur attribut une priorité élevée.

6.6.2 Résultats de prototypage des flots Wasga et Certify

Dans cette partie, nous allons implémenter 8 benchmarks sur la même plateforme multi-FPGA en utilisant deux flots :

- Flot Certify qui permet dans un premier lieu de partitionner le design, puis, effectuer une étape de routage inter-FPGA avant d'insérer des modules de multiplexage.
- Flot Wasga qui permet de partitionner le design par l'outil de partitionnement de Flexras. Cet outil génère la liste des signaux inter-FPGA. Ces signaux sont routés par le routeur que nous proposons dans cette thèse (Routage de branches par un algorithme itératif basé sur la négociation de congestion).

La table 6.5 montre les résultats d'implémentation effectuée par les deux flots précédemment cités. Les caractéristiques des benchmarks utilisés sont présentées dans la table 6.4. Le nombre des signaux coupés par Wasga est considérablement inférieur à celui obtenu par Certify. Du coup, pour tous les circuits testés, le taux de multiplexage est inférieur à celui de l'outil commercial. La table montre encore des résultats relatifs au nombre de hops de routage qui correspondent au nombre de FPGA traversés par un signal depuis sa source jusqu'à sa destination. L'outil Certify n'utilise pas la notion de hops de routage ce qui explique peut-être le taux de multiplexage relativement élevé. Pour les deux derniers benchmarks, la solution de routage n'existe pas avec Certify (NR :Not Routable). En effet, le nombre de signaux inter-FPGA est important, ce qui explique l'augmentation du taux de multiplexage. Par contre, Certify utilise une bibliothèque de module de multiplexage avec une taille qui ne dépasse pas les 32 bits [84]. Ainsi, les circuits ayant un taux de multiplexage supérieur à 32 sont non routable.
Les résultats présentés montrent l'importance de l'effet du partitionnement sur la performance du système. Bien que Synopsys propose une solution complète pour un prototypage rapide sur cartes multi-FPGA. Néanmoins, l'outil de partitionnement Certify est dédié principalement pour les plateformes HAPS (cartes multi-FPGA de Synopsys). Ces cartes sont extensibles de telle sorte qu'on pourrait augmenter la capacité logique de la plateforme tout en connectant les cartes entre elles. Des connecteurs pouvant être ajoutés pour améliorer la connectivité inter-FPGA suivant le résultat de partitionnement. De ce fait, la solution de partitionnement donnée par cet outil n'est pas optimisée et sa qualité dépend de la flexibilité que l'on puisse introduire sur la carte matérielle.
De son côté, Wasga propose une solution de partitionnement dédiée à la plateforme donnée par l'utilisateur. Cette solution répond au contraintes exigées, et par l'utilisateur (taux d'occupation par exemple) et par les caractéristiques de la carte multi-FPGA. Ainsi, tel le montre les résultats donnés par le flots Wasga, l'outil de partitionnement de Flexras en addition de l'outil de routage proposé dans cette thèse, présentent les meilleurs fréquences de fonctionnement sur une plateforme de prototypage.

6.7 Conclusion

Dans ce chapitre nous avons tout d'abord présenté notre modèle de calcul de la fréquence de fonctionnement sur une plateforme matérielle. Notre modèle dépend des IP de multiplexage insérés dans les FPGAs source et destination. Les IPs sont dédiés pour des FPGA Virtex 6 ce qui nous a permis d'implémenter réellement des designs sur une plateforme multi-FPGA. Les paramètres de ces IP sont optimisés afin de réduire la fréquence de fonctionnement.
Par la suite, nous avons donné une description sur la plateforme matérielle ainsi sur l'outil de partitionnement que nous avons utilisé pour valider les travaux de cette thèse.
A travers plusieurs scénarios, nous avons déterminé la meilleur forme des signaux à router

ainsi que le graphe de routage utilisé. Le scénario sélectionné a été utilisé pour comparer notre approche itérative par rapport à un algorithme constructif largement utilisé dans l'état de l'art. En utilisant l'algorithme de routage Pathfinder durant plusieurs itérations a permis une amélioration de la fréquence de fonctionnement par une moyenne de 12,8%. La performance de notre outil de routage est encore mise en œuvre en l'introduisant dans le flux Wasga et en le comparant par rapport à l'environnement de prototypage de Synopsys.

Chapitre 7

Insertion des IPs de multiplexage et gestion de la congestion

Sommaire

7.1 Introduction

Dans le chapitre 5, le but était de grouper les signaux qui vont être transmis ensemble via le même lien physique. Le transfert de chaque groupe de signaux est fait à travers une paire d'IPs de multiplexage insérée dans le FPGA source et le FPGA destination. Dans ce chapitre, nous allons détailler l'architecture de l'IP d'émission (TX) et celle de l'IP de réception (RX). Ces IPs sont conçus principalement pour mettre en œuvre des moyens de communication inter-FPGA sur une plateforme de prototypage multi-FPGAs, et aussi

pour assurer le multiplexage des signaux. Par contre, l'insertion de ces IP après la phase du routage inter-FPGA, consomme des ressources logiques supplémentaires du FPGA, mais aussi sur le succès du routage intra-FPGA.
Dans la deuxième partie de ce chapitre, nous présentons une technique de placement intra-FPGA basée sur l'estimation de congestion afin d'améliorer la routabilité du circuit. En effet, le fait de bien répartir les blocs logiques sur la surface du FPGA, a une grande influence sur la performance de l'opération de routage.

7.2 Spécification des IPs de multiplexage

Les IPs instancient des modules de la bibliothèque Xilinx. Par conséquent, ils ne seront utilisés que sur des plateformes contenant des FPGAs du même vendeur. Les modules instanciés sont des blocs SERDES(SERialisation/ DESerialisation) et des blocs LVDS pour une exploitation maximale de la bande passante entre les FPGA émettrice et réceptrice. En effet, le LVDS (Low Voltage Differential Signaling) est une norme de transmission de signaux à une fréquence élevée basée sur la différence du potentiel sur une ligne différentielle. L'IP étant paramétrable, permettra à l'utilisateur de l'adapter selon les besoins de son application. La figure 7.1 présente l'environnement de l'IP dans une carte contenant 2 FPGAs.

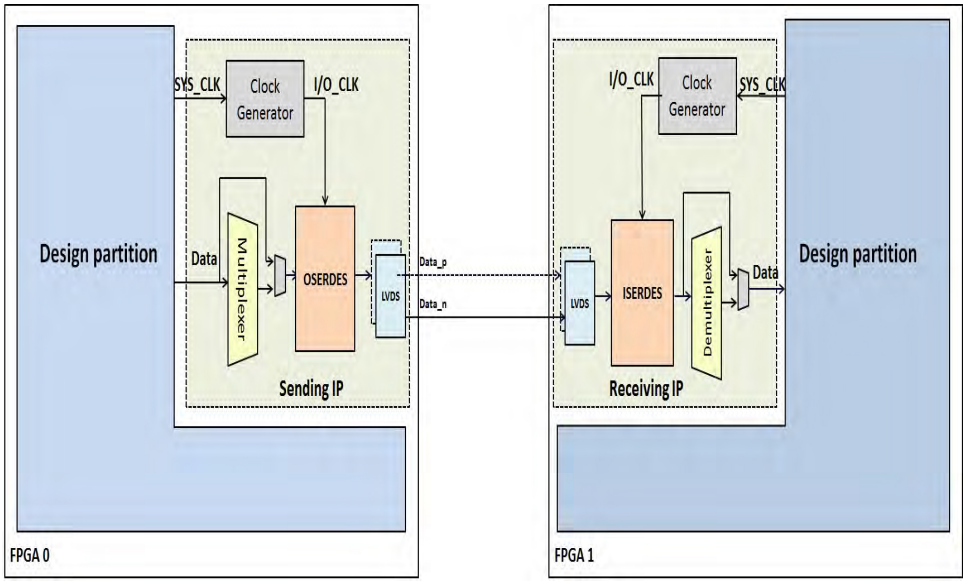


FIGURE 7.1 – Environnement des IPs de multiplexage

7.2.1 Architecture de l'IP émettrice

La figure 7.2 présente l'architecture de l'IP émettrice. Cet IP est considéré comme moyen d'interfaçage entre les liens physiques inter-FPGA et le design. En effet, pour chaque

ensemble de signaux, de taille mux_ratio, transmis à travers le même lien physique, une IP émettrice est instanciée dans le FPGA source pour assurer le multiplexage et la sérialisation de ces signaux fournis en entrée. Ces deux dernières fonctions s'effectuent à une fréquence plus élevée que celle du système prototypé afin de masquer le temps de traitement et de transmission des données.

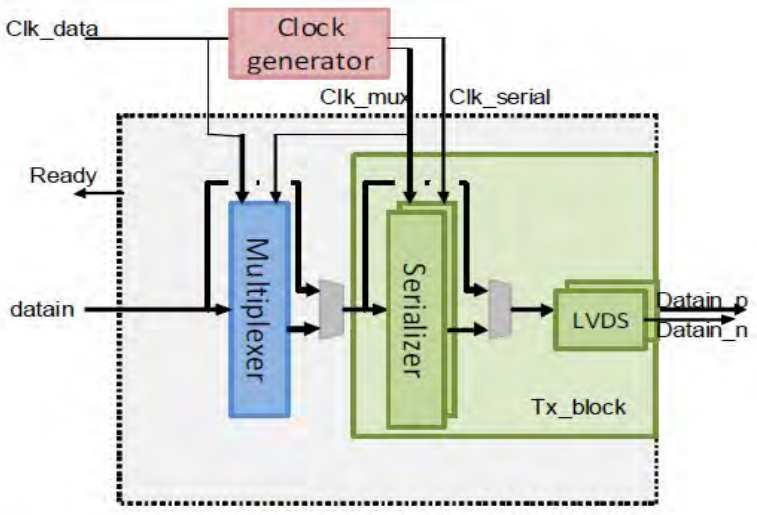


FIGURE 7.2 – Architecture de l'IP émettrice

Cette IP est composée de :

- Un bloc tx_block permettant la mise en œuvre de liens LVDS avec ou sans sérialisation.
- un multiplexeur.

7.2.1.1 Description du bloc de sérialisation

Comme son nom l'indique, un bloc de sérialisation permet de sérialiser des données, reçues en parallèle, de taille > 1 sur les liens LVDS. Ces composants sont instanciés directement depuis la bibliothèque de Xilinx et ils sont paramétrables en taille de données en entrée, facteur de sérialisation et fréquence de fonctionnement. par contre, la taille de données en entrée est relativement limitée puisqu'elle varie entre 2 et 14 bits. D'autre part, pour synchroniser l'envoie et la réception des données entre ces composants, il a été convenu de modéliser le mécanisme sous forme d'une machine d'état que nous présentons dans la figure 7.3. Durant la première phase, une séquence d'initialisation est lancée pour initialiser les liens LVDS durant quelques cycles d'horloge. Par la suite, le bloc de sérialisation envoie une séquence de synchronisation connue par le récepteur. Quand ce dernier reçoit la donnée correcte, un signal "ready" est envoyé à l'utilisateur pour annoncer le début de l'envoie des données du système.

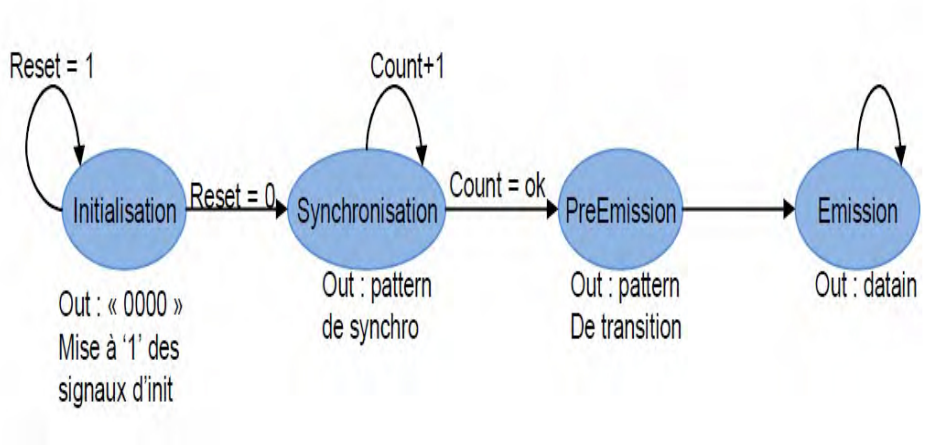


FIGURE 7.3 – Machine d'état du mécanisme d'envoi de données

7.2.1.2 Description du bloc multiplexeur

Dans les grands designs, généralement le taux de multiplexage est plus grand que 14 (taille maximale de sérialisation d'un SERDES). Pour cette raison, la solution consiste à insérer des bloc multiplexeur à l'entrée des composants SERDES. Le rôle de ce multiplexer est de répartir les données entrantes sur plusieurs cycles plus rapides, c'est-à-dire transformer des données de taille N bits arrivant à la fréquence F, en des données de taille $\frac{N}{C}$ à la fréquence F*C, avec C le coefficient de multiplexage. Les données de taille $\frac{N}{C}$ seront les entrées du bloc de sérialisation avec $\frac{N}{C} \leq 14$
La figure 7.4 montre la machine à états simplifiée du bloc multiplexer :

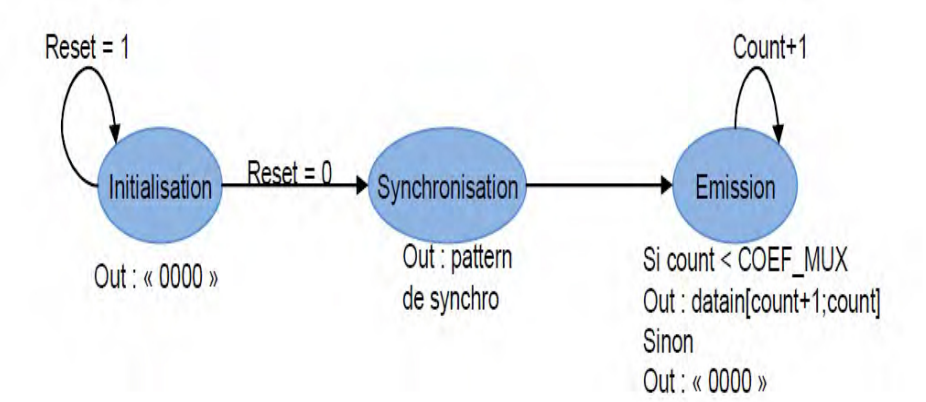


FIGURE 7.4 – Machine d'état du bloc multiplexeur

7.2.2 Architecture de l'IP réceptrice

L'IP réceptrice fait l'interfaçage entre le design et les moyens de communication inter-FPGA. Elle permet de faire le démultiplexage et la désérialisation des données reçues afin

de restituer les données envoyées. La figure 7.5 représente l'architecture de l'IP réceptrice insérée dans le FPGA destination. Les signaux reçus, via les liens LVDS, peuvent être

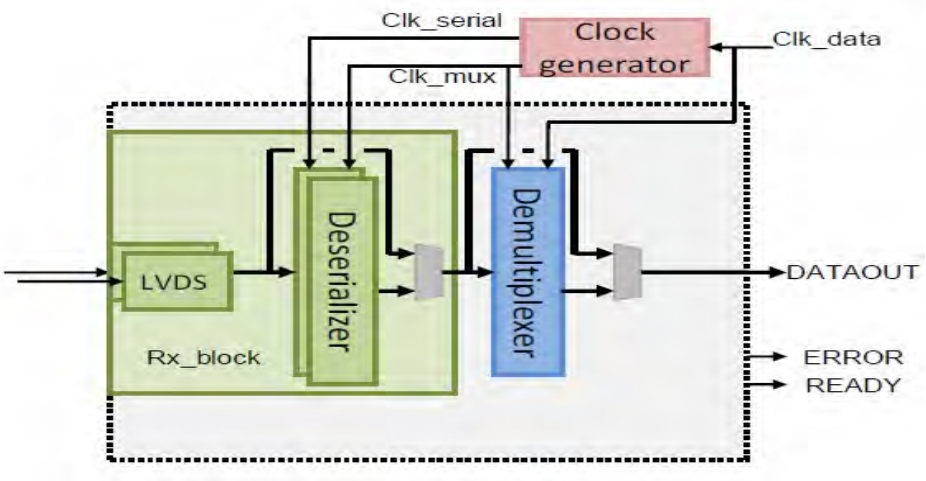


FIGURE 7.5 – Architecture de l'IP réceptrice

sérialisés et/ou multiplexés. Ces signaux sont donc désérialisés et démultiplexés avant d'être restitué. Le démultiplexage et la désérialisation se font à une fréquence plus élevée que celle des données pour masquer le temps de propagation et de traitement des données.

7.2.2.1 Description du bloc déserialisateur

Ce bloc reçoit des données sérialisées sur les pins du FPGA via des liens LVDS et les désérialisent pour retourner des données parallèles au FPGA. Il est paramétrable en taille de données d'entrée, facteur de sérialisation et en fréquence de fonctionnement. La figure 7.6 décrit la machine à états simplifiée du bloc RX. Une fois le signal de reset à 0, le bloc

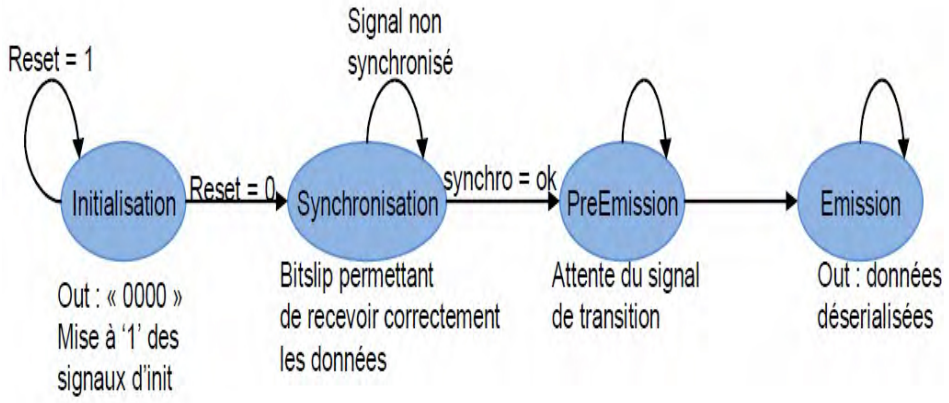


FIGURE 7.6 – Machine d'états du désérialisateur

lance une phase d'initialisation du récepteur. Une fois cette phase est terminée, le bloc attend de recevoir la séquence de synchronisation de la part de l'émetteur. La séquence

de synchronisation arrive décalée. Par la suite, le bloc de désérialisation effectue des suites de décalage afin de trouver la donnée correcte envoyée par l'émetteur. En effet, pour une donnée de taille 4, 16 opérations de décalage peuvent être effectuées avant d'avoir en sortie un pattern sans décalage. A chacune de ces opérations de décalage, le bloc de désérialisation compare la donnée décalée par celle qu'il a comme paramètre. Une fois les deux séquences sont identiques, un bit "bitslip" est mis à 1 afin de garder le même ordre de transmission des bits pour les prochaines données.
la difficulté réside dans le fait qu'aucun contrôle n'est permit entre le bloc TX et le bloc RX. Autrement dit, à part les données transmises, il n'existe pas de signal échangé entre les deux pour signaler l'état "prêt" de réception ou de transmission de données. Donc, au moment où les données de l'utilisateur commence à être envoyées par le bloc TX, le bloc RX doit être déjà synchronisé, sinon un signal d'erreur sera mis à 1 par le bloc RX.

7.2.2.2 Description du bloc démultiplexeur

Le démultiplexeur reçoit les données via le bloc RX qui les a désérialisé. Il les démultiplexe suivant le coefficient de démultiplexage donnée en paramètre. La figure 7.7 décrit la machine à états simplifiée du démultiplexeur.

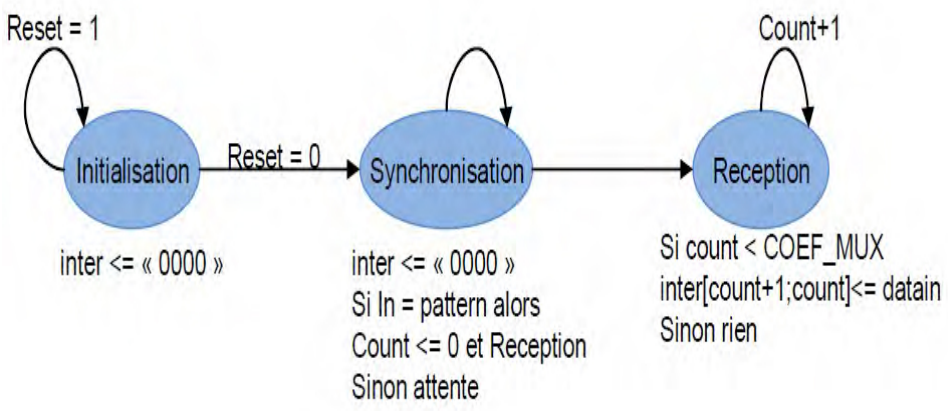


FIGURE 7.7 – Machine d'états du bloc démultiplexeur

7.2.3 Générateur d'horloge

Ce bloc instancie un bloc Mixed-Mode Clock Manager de la technologie cible. Le code utilisé est celui généré par le fabricant modifié pour prendre des paramètres afin de ne pas avoir à passer par le CoreGenerator ou le MegaWizard. Il génère une ou deux horloges en fonction de l'utilisation ou non de la sérialisation et du multiplexage. Si seulement l'un des 2 est utilisé, le bloc Clock generator ne génère qu'une horloge. Si les 2 sont utilisés, le bloc génère 2 horloges, une pour le multiplexer/demultiplexer et une pour le seriali-

ser/deserialiser. Le bloc Clock Generator est instancié hors de l'IP car ce bloc peut générer les horloges pour plusieurs instanciations d'IP.

7.2.4 Environnement de vérification

Pour valider les IP conçus, nous avons créé un environnement de test basé une fonction LFSR (Linear feedback shift register). Cette fonction est instanciée dans le bloc TX tout comme le bloc RX. Les données envoyées par le bloc émetteur, sont générées en en local par le bloc récepteur et puis comparées par les données reçues. Autrement dit, le test consiste à envoyer des signaux pseudo aléatoires à la partie émettrice de l'IP qui va les transmettre à la partie réceptrice de l'IP, et puis, contrôler que les données que retourne la partie réceptrice de l'IP sont bien ceux fourni en début de chaine.

7.2.5 Variation de taille de l'IP de multiplexage

La taille d'un IP de multiplexage dépend du taux de multiplexage (mux_ratio). La figure 7.8 présente la variation de la taille d'un IP par rapport à la valeur de mux_ratio. La courbe montre que le nombre de luts augmente en augmentant le taux de multiplexage.

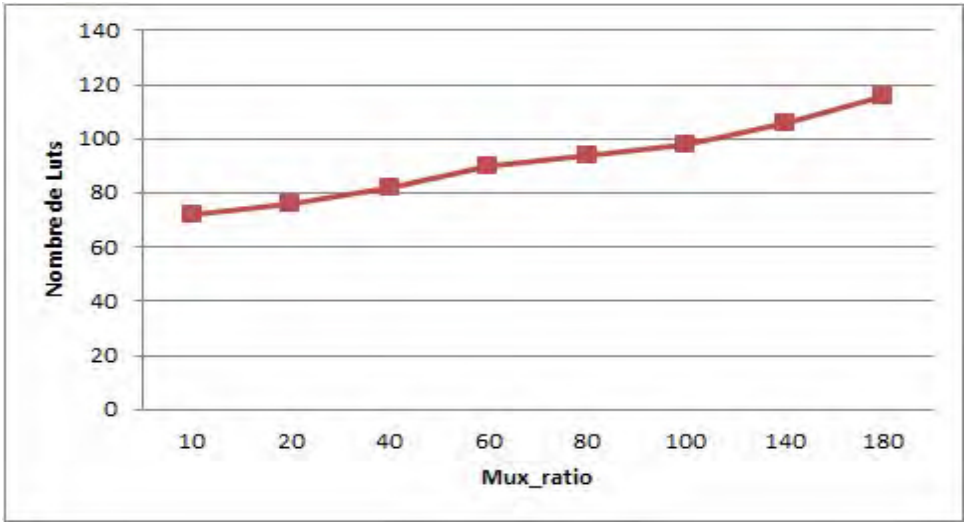


FIGURE 7.8 – Variation de la taille d'un IP par rapport au taux de multiplexage

Pour les gros circuits ayant un taux de multiplexage important, les ressources logiques utilisées après le routage est considérablement importante ce qui conduit dans certains cas à un problème de placement et/ou routage intra-FPGA. En effet, durant le partitionnement, l'utilisateur estime la surface des IPs de multiplexage qui vont être insérés après le routage inter-FPGA (après le partitionnement également). La taille nécessaire n'est pas connue à l'avance, et c'est à l'utilisateur d'estimer sa valeur. Ainsi, le taux d'occupation d'un FPGA doit être inférieur à 100% pour laisser de la place aux IPs insérés plus tard. Dans

certains cas, le taux de multiplexage résultant est inattendu. Par conséquent, deux cas se présentent :

- Le nombre de luts des IPs insérés est supérieur au nombre disponible. Dans ce cas, le partitionnement est refait.
- Le nombre de ressources logiques disponibles est suffisant pour implémenter les IPs de multiplexage, mais le design devient congestionné à cause des ressources supplémentaires ajoutées. Dans ce cas, le routage intra-FPGA peut ne pas être possible. Pour remédier à ce problème, une nouvelle technique de placement intra-FPGA est développée. Cette technique est basée sur l'estimation de congestion et qui a pour but de bien répartir les blocs logiques sur la surface du FPGA afin de faciliter l'étape de routage.

7.3 Technique de placement basée sur l'estimation de congestion

Après le routage inter-FPGA, des IP de multiplexage sont insérés dans les FPGAs source et destination. Même si la surface logique restante dans le FPGA est suffisante pour intégrer tous les IP de multiplexage nécessaires, la solution de routage peut être inexistante à cause de la grande congestion dans des zones à l'intérieur du FPGA. Pour cela, nous avons proposé une technique pour mieux répartir les blocs logiques à l'intérieur du FPGA afin d'équilibrer l'utilisation des ressources de routage, et par la suite, obtenir la solution de routage la plus optimisée.
L'étude que nous avons faite dans ce chapitre avait pour cible une architecture particulière de FPGA. En effet, le code des outils de placement des vendeurs traditionnels de FPGA, à citer Xilinx et Altera, ne sont pas accessibles pour modifications. Par conséquent, nous avons choisi d'évaluer la technique de placement, détaillée par la suite, avec un outil de placement développé dans le laboratoire LIP6.

7.3.1 Architecture du FPGA cible

Le circuit FPGA cible a une architecture matricielle contenant un réseau d'interconnexion local. Ce FPGA a été fabriqué en utilisant la technologie ST 65nm avec le processus silicium sur isolant ou (SOI : Silicon On Insulator).
L'architecture matricielle est composée de blocs logiques configurables placé régulièrement en deux dimensions et formant ainsi une structure matricielle comme l'illustre la figure 7.9. Les lignes d'interconnexion, entourant les blocs logiques, sont organisées en lignes et en colonnes, formant ainsi des canaux de routage horizontaux et verticaux. Comme le montre la figure 7.9, il existe deux types de réseau d'interconnexion :

- Un réseau global qui est formé par 4 * W sous-réseaux disjoints. Avec W représente la largeur du canal, c'est à dire, chaque canal contient un nombre "W" de segments

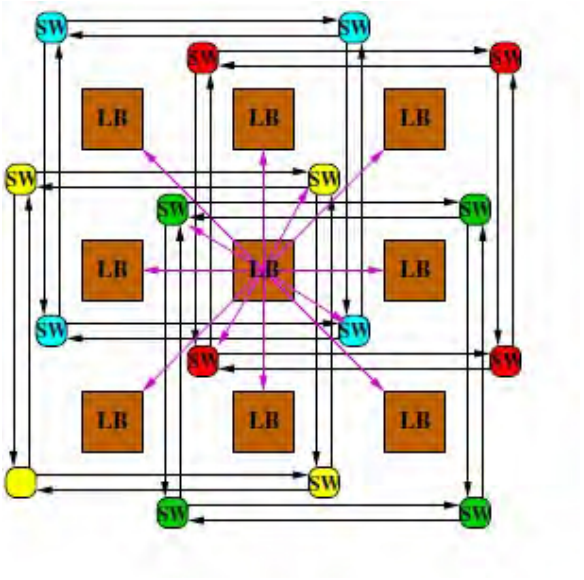


FIGURE 7.9 – Architecture FPGA matricielle

de routage. Tous les segments sont unidirectionnels et chacun d'eux s'étend sur deux blocs logiques.

- Un réseau d'interconnexion local. Grâce à ce réseau, chaque bloc logique est directement connecté à ses 8 voisins ce qui rend flexible le routage des blocs qui sont proches les uns aux autres. En plus, ce réseau permet d'alléger la congestion sur le réseau global.

7.4 Placement avec estimation de congestion

Le placement consiste à attribuer des éléments spécifiques du FPGAs aux cellules logiques du circuit. La manière avec laquelle le circuit est placé, influe considérablement le résultat de routage.
Plusieurs techniques de placement ont été utilisées, mais nous nous intéressons à la technique du recuit simulé qui a été détaillée dans la partie état de l'art.

7.4.1 Placement recuit simulé

Le terme "recuit simulé" a été inspiré du processus de refroidissement de métaux. Cet algorithme a été proposé par Kirkpatrick en 1983 [65]. Des années plus tard, ce processus a été adapté au problème de placement sur FPGA [64], [85]. Le pseudo code est représenté par l'algorithme 5.

Le principe est de commencer par une solution de placement aléatoire, et puis, améliorer itérativement ce placement en déplaçant les blocs logiques tout en se basant sur une fonction coût qui reflète la qualité du placement. La fonction coût la plus utilisée est donnée

algorithme 5 Pseudo code de l'algorithme de placement basé sur le recuit simulé [2]

```
S = PlacementAléatoire() ;
T = TempératureInitiale() ;
R_limit = R_limitInitial ;
Tantque (Exit() == Faux) faire
  Tantque (Loop() == Faux) faire
    S_new = Générer(S, R_limit) ;
    δC = Cout(S_new) − Cout(S) ;
    r = random(0, 1) ;
    si r < e^(−δC/T) alors
      S = S_new ;
    finsi
  Fin Tantque
  T = MettreAjourTemp() ;
  R_limit = MettreAjourR_limit() ;
Fin Tantque
```

par l'équation 7.1 [85] et elle est définie comme étant la somme des demi périmètres des rectangles englobants de tous les signaux du circuit.

$$cot = \sum_{i=1}^{N_{nets}} (q(i) \cdot (bbx(i) + bby(i))) \tag{7.1}$$

7.4.2 Estimation de congestion

Pour tenir compte de la congestion lors du placement, un coefficient de congestion a été ajouté à la fonction coût dans [86] comme le montre l'équation 7.2 :

$$cot = CC^K * \sum_{i=1}^{N_{nets}} (q(i) \cdot (bbx(i) + bby(i))) \tag{7.2}$$

Ce coefficient présente une estimation de congestion qui sera prise en compte lors du placement. k est le coefficient d'atténuation que nous allons détailler plus tard. L'expression du coefficient dans [86] est représentée par l'équation 7.3

$$CC = \left(\frac{\sum_{(i,j)} U_{i,j}^2}{nx * ny} \right) / \left(\frac{\sum_{i,j} U_{i,j}}{nx * ny} \right)^2 \tag{7.3}$$

Avec nx et ny représentent les dimensions du circuit FPGA (en nombre de blocs logiques). $U_{i,j}$ c'est le nombre de bounding box auxquels appartient le bloc logique dans la position (i,j). Autrement dit, à chaque bloque logique est assignée une valeur qui correspond au nombre de rectangles englobant auxquels appartient ce bloc logique. La figure 7.10 représente la valeur $U_{i,j}$ assignée à chaque bloc logique de la matrice. Par la suite, le coefficient de congestion est calculé à partir des valeurs assignées à tous les blocs. La valeur de conges-

tion de chaque bloc logique donne une information sur le nombre de signaux qui vont être routés autour de chaque bloc. Une valeur très grande implique que plusieurs signaux vont être routés en utilisant les ressources de routage autour de ce bloc logique, et par la suite, cette zone sera fortement congestionnée. Cependant, cette approche néglige le fait que pour

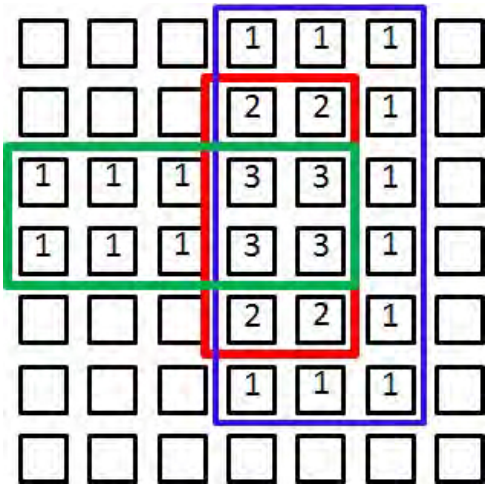


FIGURE 7.10 – Exemple de répartition de congestion suivant le nombre de rectangles englobant

les grands bounding box, la probabilité qu'un signal produit un effet sur un point spécifique est nettement réduite. En effet, cette technique donne la même importance à tous les rectangles englobant indépendamment du nombre de terminaux de chaque signal. Or, un bounding box plus grand qu'un autre, peut avoir moins de terminaux que le deuxième, et par la suite consomme moins de ressources. Ces deux rectangles ne doivent pas donner la même information de congestion. Les auteurs dans [70] ont développé un outil visuel permettant de visualiser la congestion estimée après le placement. A chaque bloc logique est assignée une valeur qui correspond à la quantité de fil utilisée par zone. En effet, pour chaque signal, l'outil estime la quantité de fil qui peut être utilisée pour router tous les terminaux de ce signal. Cette quantité est divisée par la surface du bounding box de ce signal pour déterminer finalement la valeur de congestion par surface de ce signal. La dernière étape consiste à assigner cette valeur à tous les blocs logiques qui appartiennent à ce bounding box. Finalement, la valeur de congestion de chaque bloc logique est la somme de toutes les valeurs de congestion des signaux dont leur bounding box couvre le bloc logique considéré.

Nous avons utilisé cette information pour l'implémenter dans l'outil de placement afin de tenir compte du congestion lors du placement. Cette information est modifiée afin de l'adapter à l'architecture cible et plus précisément au réseau de routage de cette architecture [87].

7.4.2.1 Coefficient de congestion

Pour tenir compte de la congestion lors du placement, un coefficient de congestion est introduit dans la fonction coût tout comme dans [69]. Dans notre étude, nous proposons l'expression du coefficient représentée par l'équation 7.4. Plusieurs tests ont été fait pour trouver les valeurs convenables de chaque paramètres.

$$CC = \left(\frac{\sum_{(i,j)} U_{i,j}^{N}}{nx * ny} \right) / \left(\frac{\sum_{i,j} U_{i,j}}{nx * ny} \right)^{N} \tag{7.4}$$

Avec $U_{i,j} = \sum_{CLB_{i,j} \in BB_{Net}} W_{net}$.$U_{i,j}$ est le coefficient de congestion locale de chaque bloc logique et qui correspond à la somme des valeurs de congestion de tous les signaux dont leurs rectangles englobants incluent ce bloc. Le paramètre K permet de contrôler le poids du coefficient de congestion. Si la valeur de K augmente, la congestion est de plus en plus considérée lors du placement. Le paramètre N contrôle le déséquilibre de congestion entre les blocs logiques.
La valeur de congestion de chaque signal est notée W_{net}. Son expression est donnée par l'équation 7.5

$$W_{net} = \frac{L}{A} \tag{7.5}$$

avec A c'est la surface du rectangle englobant du signal net. L c'est la longueur de fils estimée pour router tous les terminaux du signal net. Dans [86], L est définie par l'équation 7.6

$$L = \frac{1}{2} * BB + \beta * q \tag{7.6}$$

Avec BB c'est le périmètre du rectangle englobant, β c'est un paramètre de réglage et q c'est un facteur de correction relatif au nombre de terminaux.
Le demi périmètre du rectangle englobant ne reflète pas d'une manière précise la quantité de fil qui peut être utilisée pour router tous les terminaux. Cette expression est adaptée à l'architecture cible utilisée dans cette thèse pour tenir compte de la position de la source d'un signal par rapport aux destinations.
Comme nous l'avons indiqué dans la section 7.3.1, l'architecture du FPGA cible contient deux réseaux d'interconnexion : un réseau local, et un autre global. Les ressources locales sont utilisées pour les connexions directes entre les blocs voisins. Par conséquent, une source et une destination qui sont voisines sont routées en utilisant les ressources locales sans ajouter plus de congestion au réseau global. Pour tenir compte de ce point, nous avons utilisé l'algorithme 6 pour estimer la longueur de fils utilisés pour le routage d'un signal donné. Donc, l'idée est de calculer la distance entre la source du signal et ses destinations à l'exception de celles qui lui ont voisines. En effet, le réseau local permet une connexion directe entre une source et une destination voisines sans pénaliser la congestion dans le réseau global. Dans l'équation 7.4, nous avons utilisé deux paramètres, K et N, pour plus d'efficacité pour le coefficient de congestion. Pour explorer l'effet de ces deux paramètres,

algorithme 6 Pseudo code de l'algorithme d'estimation de la longueur de fils utilisés pour router un signal donné [87]

```
pour tout signal "N" faire
  src = source(N) ;
  x_s = src->x() ;
  y_s = src->y() ;
  pour tout destination dest de "N" faire
    x_d = dest->x() ;
    y_d = dest->y() ;
    si (dest et src ne sont pas voisines) alors
      L += abs(x_s - x_d) + abs(y_s - y_d) ;
    finsi
  fin pour
fin pour
```

nous avons implémenté les benchmarks de test MCNC [31] sur l'architecture cible utilisée. Plusieurs tests ont été faits afin de déterminer l'influence de la variation des paramètres K et N.

7.4.2.2 Adaptation du paramètre K

nous avons analysé le comportement du coefficient de congestion CC en variant la valeur de K. D'après la figure 7.11, le coefficient CC diminue lorsque le paramètre K augmente. En effet, le fait d'augmenter la valeur de K, ceci met en évidence l'effet de la congestion

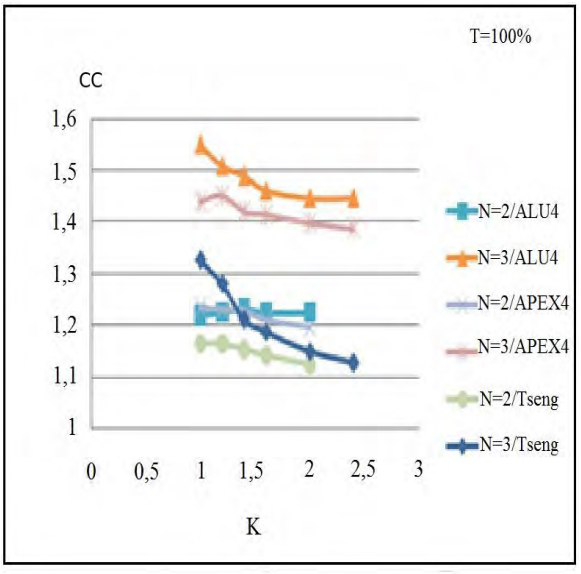


FIGURE 7.11 – L'effet de variation du paramètre K sur le coefficient de congestion

lors de l'opération de placement et le coefficient de congestion a plus d'influence dans la fonction objective.

D'autre part, la figure 7.12 montre que la somme les demi périmètres des rectangles en-

globant croit également lorsque la valeur de K augmente. En effet, en essayant d'alléger la congestion dans les zones congestionnées, les blocs sont répartis sur toute la surface du FPGA, et ils sont de plus en plus loin les uns aux autres, ce qui cause une augmentation au niveau de la somme des demi périmètres des rectangles englobants.

Pour cette raison, il a fallu trouver un compromis entre la manière avec laquelle le para-

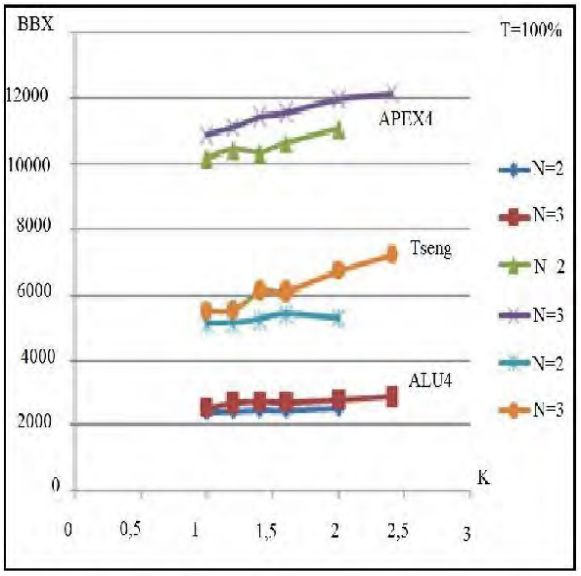


FIGURE 7.12 – L'effet de variation du paramètre K sur la somme des demi périmètres des rectangles englobants

mètre K augmente et son effet sur les deux termes de la fonction objective (coefficient de congestion et somme des demi-périmètres).

Dans un premier temps nous avons proposé l'expression de K représentée par l'équation 7.7

$$L = 1 + SuccRatio \tag{7.7}$$

Avec SuccRatio c'est le rapport entre le nombre de déplacements acceptés et celui effectués pendant une itération de placement(Loop). Au début du placement, presque tous les déplacements sont acceptés puisque la valeur de la température est grande. Donc la valeur de SuccRatio $\simeq 1$ et K $\simeq 2$. Tout en continuant le placement, la valeur de la température diminue, et le taux des déplacements acceptés diminue également. Par conséquent la valeur de K est à son tour diminuée. A la dernière itération, SuccRatio $\simeq 1$.

Cette nouvelle expression du paramètre K garde le même gain en termes de congestion. Par contre, la longueur des fils utilisés est réduite de 7.3% pour le circuit ALU4 et de 12.3% pour le circuit TSENG.

Autre déduction que nous pouvons tirer, c'est que le coefficient de congestion dépend du taux d'occupation dans le circuit FPGA. Le taux d'occupation représente le rapport entre le nombre d'instances dans le circuit et le nombre de blocs logiques dans l'architecture FPGA. Dans la figure 7.13, nous notons que le coefficient de congestion diminue lorsque le

taux d'occupation T augmente.
Dire que le taux d'occupation augmente revient à dire que le nombre de blocs logiques

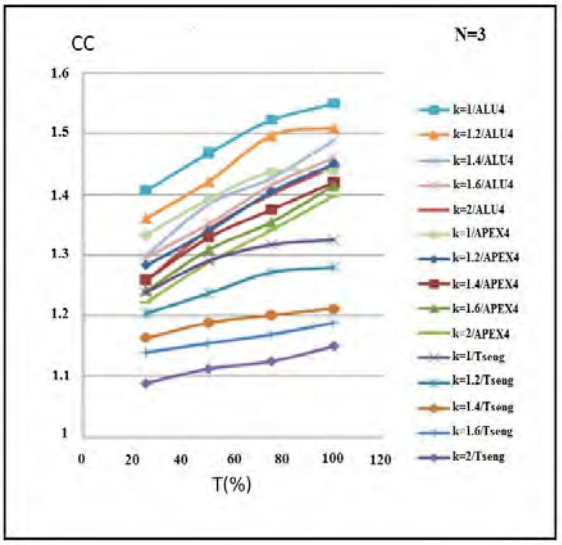


FIGURE 7.13 – L'effet de variation de T sur le coefficient de congestion

disponibles dans l'architecture FPGA est de plus en plus grand en le comparant avec le nombre des instances logiques du circuit. Dans ce cas, les instances sont mieux placées dans toute la surface du FPGA et la congestion est remarquablement réduite.
Pour mieux considérer les régions vides dans le FPGA, nous avons intégré le paramètre du taux d'occupation dans l'expression de K comme le montre l'équation 7.8 :

$$L = \frac{1 + SuccRatio}{T} \tag{7.8}$$

Avec $T = \frac{NB_{instances}}{NB_{BL}}$.
Lorsque le nombre de blocs logiques dans un circuit FPGA augmente, le paramètre K augmente. Ainsi, nous mettons en évidence le coefficient de congestion dans l'expression de la fonction objective.

7.4.2.3 Adaptation du paramètre N

Le paramètre N agit sur le déséquilibre entre les valeurs de congestion des blocs logiques. Il permet d'accentuer l'effet des blocs fortement congestionnés. Nous avons fait un certain nombre d'expérimentations pour sélectionner la valeur convenable du paramètre N. Nous avons testé les valeurs de N=1, 2, 3, 4 et 5.
Expérimentalement, nous avons trouvé que la valeur de N=3 donne les résultats les plus performants.

Ainsi, l'expression finale du coefficient de congestion est donnée par l'équation 7.9

$$CC = \left(\frac{\sum_{(i,j)} U_{i,j}^3}{nx * ny} \right) / \left(\frac{\sum_{i,j} U_{i,j}}{nx * ny} \right)^3 \tag{7.9}$$

7.4.3 Qualité de l'estimation de congestion

Pour générer une comparaison visuelle de la congestion dans les différentes zones, nous présentons les cartes de congestion d'un circuit avant et après le routage (congestion réelle) en utilisant une dégradation de couleur (Jaune=faible congestion, Rouge=forte congestion). Les figures 7.14(a) et 7.14(c) représentent respectivement la congestion estimée du circuit placé sans et avec estimation de congestion. Les figures 7.14(b) et 7.14(d) représentent respectivement la congestion réelle du circuit routé sans et avec estimation de congestion lors du placement. La congestion réelle est évaluée après le routage en se basant sur le nombre de segments utilisé dans chaque canal de routage.

D'après la figure 7.14, nous notons que la congestion estimée relative à la figure 7.14(c)

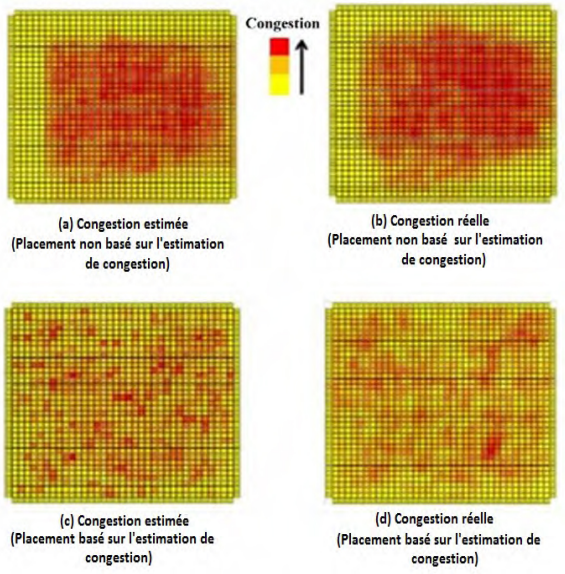


FIGURE 7.14 – La carte de congestion du circuit ALU4

est corrélée à la congestion réelle représentée dans la figure 7.14(d). Ceci montre bien que notre coefficient de congestion utilisé, donne une bonne estimation de la congestion pour une meilleure répartition des instances dans toute la surface FPGA. Dans la figure 7.14(a) et la figure 7.14(b), les instances du circuits sont concentrées dans le centre du FPGA. Ceci est évident du fait que la fonction coût utilisée vise à ramener les blocs qui communiquent ensemble, les uns près des autres afin de réduire la somme des demi périmètres des rectangles englobants. Dans les figure 7.14(c) et 7.14(d), la congestion est bien répartie dans toute la surface FPGA. Cet équilibre permettra une meilleure gestion des ressources de routage.

TABLE 7.1 – L'impact du placement basé sur l'estimation de congestion sur la surface du FPGA et sur le délai du chemin critique

MCNC		**Placement BBX**		**Placement avec estimation de congestion**		**Gain**	
Circuits	**Luts**	**Surface(λ^2)** $*10^6$	**Délai(ns)**	**Surface(λ^2)** $*10^6$	**Délai(ns)**	**Surface** %	**Delai** %
alu4	606	319	8.3	290	8.4	9	-1
apex2	1919	1541	9.2	1356	8.5	12	7
apex4	1290	1092	9.6	950	9.3	13	3
bigkey	1707	1065	6.6	947	7	11	-6
clma	8383	7672	19.2	6521	17.5	15	8
des	2092	2047	8	1739	8.1	15	-1
diffeq	1599	954	5.8	849	6.2	11	-6
dsip	1796	934	6.9	934	6.1	0	11
elliptic	3848	2883	12.6	2652	12	8	4
ex1010	4589	3763	16.2	3424	15	9	7
exp5p	1135	915	7.1	786	7.5	14	-5
frisc	3691	3287	12.2	2695	11	18	9
misex3	1425	1085	7.5	987	6.8	9	9
pdc	4575	4889	18.1	4497	18.2	8	0
s298	1940	1192	13.8	1060	13	11	5
s38584	6406	4662	9.6	4102	7.2	12	25
s38584	6447	4590	10.4	3855	8.1	16	22
seq	1826	1411	10.1	1411	10	0	0
spla	3690	3448	15.3	3068	11	11	28
tseng	1220	665	5.2	665	5.3	0	-1
Moyenne	3009.2	2420.7	10.6	2139.4	9.81	10	6

7.4.4 Résultats

Nous avons testé l'impact du placement basé sur l'estimation de congestion sur le nombre des ressources de routage utilisés. Une réduction au niveau des ressources de routage implique une réduction de la surface totale du circuit utilisé. Les résultats d'implémentation des circuits MCNC sont présentés dans la table 7.1.

Pour évaluer notre technique, nous avons placé tous les circuits avec la nouvelle technique proposée, et encore avec la technique traditionnelle basée sur la réduction de la somme des demi périmètres des rectangles englobants. Par la suite, nous avons routé ces circuits par un routeur basé sur l'analyse du timing.

Pour tous les benchs que nous avons testés, nous avons utilisé des architectures avec un taux d'occupation égale à 75%. Nous remarquons que dans la plupart des circuits, nous avons obtenu une réduction de surface avec une moyenne de 10%. En plus, nous notons qu'avec la nouvelle technique proposée, le délai sur le chemin critique est nettement diminué. En effet, le plus la congestion est diminuée, le mieux le routeur routera les signaux du chemin critique.

7.5 Conclusion

Dans ce chapitre nous avons présenté l'architecture des IP de multiplexage qui assurent le transfert de signaux entre le FPGA source et destination. Ces IP sont paramétrables en taille de données en entrées, coefficient de sérialisation, coefficient de multiplexage, fréquence...Et c'est à l'utilisateur de saisir les paramètres qui permettent d'obtenir les meilleurs résultats de point de vue fréquence de fonctionnement. L'insertion "imprévue" de ces IP après la phase de routage inter-FPGA pour causer des problèmes de routage et/ou placement intra-FPGA. En effet, lors du partitionnement du design, l'utilisateur doit saisir un taux d'occupation pour chaque FPGA pour garantir un routage aisé à l'intérieur du FPGA. L'outil de partitionnement quant à lui, essaie de trouver une solution de partitionnement qui satisfait ce taux. Après l'instantiation des IP de multiplexage, le taux d'occupation n'est plus respecté, et du coup, la solution de routage n'est plus garantie à cause de la congestion "logique" à l'intérieur du FPGA. Pour cela, nous avons proposé une technique de placement basée sur l'évitement de congestion. L'idée consiste à insérer un coefficient de congestion dans la fonction coût pour rejeter tout essaie de déplacer un bloc logique qui peut congestionner certaines zone. L'implémentation a été faite dans un outil de placement développée au sein du LIP6 et qui vise une architecture particulière de FPGA.

Conclusion et Perspectives

Dans le cycle de développement des systèmes sur puce, il a été clairement démontré que le prototypage matériel réduit considérablement le cycle de conception pour faire face à la contrainte de "Time to market". En effet, le prototypage rapide sur une carte à base de circuit FPGA présente le meilleur compromis entre le temps de conception de ce circuit et le temps d'exécution d'une application sur ce circuit.

Ce travail de thèse s'inscrit dans le cadre du projet PPR qui inclut 4 partenaires. Ce projet consiste à concevoir une plateforme logicielle/matérielle pour le prototypage rapide sur FPGA. Le but est de fournir une solution complète et performante aux concepteurs de circuits complexes pour évaluer les fonctionnalités de leurs circuits et leurs logiciels dans une phase avancée du développement. Ceci leur permettra aussi d'anticiper la fabrication du circuit et la mise au point des applications logicielles qui vont tourner dessus.

Ce travail de thèse consiste à développer un générateur de benchmarks qui permet de générer des gros circuits de test. Ces circuits sont utiles pour valider les différentes techniques développées dans cette thèse ainsi que celles des autres partenaires. Puis, notre travail principal est de concevoir un outil CAO qui se charge de faire le routage des signaux inter-FPGA. Finalement, il est nécessaire de définir les modules de multiplexage qui permettent de transférer les signaux entre les différents FPGA. L'insertion de ces modules après le partitionnement engendre un dépassement de surface que nous avons essayé de résoudre par le développement d'une technique de placement intra-FPGA basée sur l'estimation de congestion.

Dans la première partie de ce manuscrit, nous avons présenté la problématique, et nous avons abordé toutes les étapes du flot logiciel d'implémentation sur une plateforme de prototypage. En plus nous nous sommes attardés sur la présentation des algorithmes de routages inter-FPGA et ceux du placement intra-FPGA les plus utilisés dans l'état de l'art. La deuxième partie de l'état de l'art consiste à présenter les plateformes de prototypage qui existent sur le marché. Ces plateformes sont classées sous forme de plusieurs familles suivant la nature de la solution proposée.

Dans la partie suivante de ce travail, nous avons présenté notre environnement de

génération de benchmarks de test. Ce générateur est basé sur l'outil DSX et il permet de générer des circuits VHDL synthétisables et prêts à être implémentés sur les FPGA ALTERA et XILINX. Ces benchmarks inclus un certain nombre de caractéristiques leur permettant d'avoir un comportement semblable à celui des circuits réels. En effet, nous avons choisi d'intégrer l'aspect multi horloges en insérant une fifo bi-synchrone entre le composant UART et le bus. D'autre part, nous nous sommes intéressés à introduire un FPGA embarqué jouant le roule d'un coprocesseur. Ce composant permettant d'ajouter des cellules reconfigurables dans notre circuit. La taille des benchmarks généré est relativement grande, et peut atteindre quelques millions de LUTs.

Par la suite, nous avons proposé un outil de routage des signaux inter-FPGA. Cet outil est basé sur une approche itérative qui permet de réduire le taux de multiplexage. Pour trouver un chemin de routage de tous les signaux, nous avons utilisé l'algorithme de routage Pathfinder. Cet algorithme, initialement utilisé pour le routage des signaux intra-FPGA, a été adapté pour cibler les signaux qui apparaissent à l'interface des différents FPGAs. L'algorithme de routage Pathfinder est basé sur la négociation de congestion. En effet, au cours d'un certain nombre d'itérations, les signaux négocient leurs besoins à utiliser une ressource. A la fin de toutes les itérations, le nombre de signaux qui partagent une ressource, doit être inférieur ou égale à la capacité de cette ressource. Pour appliquer la technique proposée, il a été convenu de modéliser la plateforme matérielle sous forme d'un graphe de routage. Deux types de graphe ont été proposés : un graphe bidirectionnel et un graphe unidirectionnel. Les signaux routés peuvent admettre deux formes : la forme multi-terminaux et la forme bi-points.

Ainsi, pour identifier la meilleure forme des signaux à router ainsi que celle du graphe de routage, nous avons proposé 4 scénarios à tester. Les résultats de cette expérimentation ont été détaillés dans le chapitre 6. Dans ce chapitre, nous avons démontré que le routage d'un groupe de branches sur un graphe bidirectionnel donne les meilleurs résultats en termes de fréquence de fonctionnement. En effet, le routage sur un graphe unidirectionnel n'est pas optimisé du fait que le critère de sélection de la direction des fils physique peut dans certains cas ne pas être en cohérence avec le comportement du routeur. En plus, pour le routage des signaux multi-terminaux sur un graphe bidirectionnel, le nombre de hops de routage est parfois important ce qui dégrade la performance du système de prototypage.
La forme de signal (bi-points) et du graphe de routage (bidirectionnel) sélectionnés ont été maintenus pour comparer notre approche itérative par rapport au algorithme constructif utilisé dans l'état de l'art. Le routeur de signaux inter-FPGA que nous avons développé apporte un gain moyen de 12.8% sur un ensemble de benchmarks que nous avons utilisés. Notre routeur essaie de trouver le meilleur compromis entre la diminution du taux de multiplexage est l'augmentation du nombre de hops de routage.
Finalement, nous avons comparé le flot Wasga par rapport au flot commercial de Synopsys : Certify. Dans le flot Wasga, nous avons utilisé l'outil de partitionnement de Flexras. Des

contraintes ont été ajoutés à cet outil pour donner une solution de partitionnement avec un nombre de signaux inter-FPGA bien équilibré entre les différentes paires ce qui en résulte un taux de multiplexage optimisé. Les signaux inter-FPGA sont routés avec l'outil de routage proposé dans cette thèse. Les résultats de comparaison ont montré que la fréquence de fonctionnement résultante du prototypage par le flot Wasga est nettement meilleure que celle de Certify. En effet, L'outil de Synopsys est destiné surtout aux cartes Haps qui peuvent se connecter entre elles ce qui crée plus de connectivité. Par conséquent, la solution de partitionnement sur toute autres cartes commerciales est loin d'être optimisée.

Dans le dernier chapitre, nous avons présenté les IPs de multiplexage adaptés au FPGA Virtex 7. Le rôle de ces IPs consiste à assurer le transfert et la sérialisation des signaux entre deux FPGAs. Par contre, l'insertion de ces IP après la phase de partitionnement peut engendrer un dépassement de surface, mais aussi une congestion dans certaines zones, ce qui en résulte une solution de routage intra-FPGA impossible. Pour cette raison, nous avons proposé une technique de placement intra-FPGA basée sur l'estimation de congestion. Le principe de cette technique est de placer les blocs de manière à éviter la congestion et de faciliter la phase du routage. Nous avons implémenté cette technique sur un outil de placement dédié aux FPGA de la CEA ayant une architecture matricielle à réseau d'interconnexion local. ce choix a été adopté puisque les outils de placement et de routage commerciaux (de Xilinx ou de Altera), ne sont pas accessibles pour modification. En appliquant la nouvelle technique, nous avons obtenu une réduction moyenne de 10% en termes de surface. La réduction reflète une bonne exploitation des ressources de routage qui s'explique par la répartition équilibrée des blocs logique sur la surface du FPGA.

Perspectives

Ce travail de thèse offre plusieurs perspectives de recherche.

1. Dans ce travail, nous avons présenté une technique de placement intra-FPGA basé sur l'estimation de congestion. Comme nous n'avons pas l'accès pour agir directement sur l'un des outils de placement des FPGA industriel, nous avons utilisé un outil de placement académique qui cible une architecture FPGA matricielle. La majorité des FPGAs industriels ont l'architecture d'une matrice, à noter le Virtex VI de Xilinx. Par conséquent, cette technique peut être appliquée pour ce FPGA. Le Virtex 7 présente une nouvelle architecture de type 2.5D. Ce FPGA est composé de 4 dies qui communiquent ensemble. Il est certainement intéressant d'adapter notre technique à cette nouvelle architecture afin de gérer la congestion inter-die. Cette adaptation permettra de joindre la performance de cette technique à celle de l'architecture innovante.
2. Les techniques de routage développées dans cette thèse sont statiques. Autrement dit, au moment de l'implémentation du design sur la plateforme, les liens de communication ainsi que les groupes de signaux transmis ensemble sont connus à l'avance.

Il peut être intéressant de cibler des protocoles de communication dynamiques, forcément plus complexes, là où le transfert des signaux varie dans le temps dépendant de l'activité de l'application qui tourne sur le circuit faisant l'objet de prototypage.

3. La technique de routage développée dans cette thèse cible les designs mono-horloge. Dans un futur travail, nous nous intéressons au routage des signaux cadencés par différentes horloges.

Liste des publications

- Frequency Optimization Objective during System Prototyping on Multi-FPGA Platform
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  International Journal of Reconfigurable Computing (IJRC), Volume 2013 (2013), Article ID 853510, 12 pages.

- Congestion driven placement for mesh-based fpga architecture with local interconnect
  Zied Marrakchi, Mariem Turki, J. Rebourg, Mohamed Abid, Habib Mehrez
  International Conference on Microelectronics (ICM), Hammamet, Tunisia, December 2011

- Towards Synthetic Benchmark Generator For CAD Tool Evaluation
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  8th Phd Research in Microelectronics and Electronics, PRIME 2012, Aachen, Germany, June 2012

- Multi-FPGA prototyping environment : Large benchmark generation and signals routing
  Mariem Turki, Zied Marrakchi, Habib Mehrez
  International Conference on Reconfigurable Computing and FPGA, (ReConfig'2012), Cancun, mexico, December 2012,

- Iterative Routing Algorithm of Inter-FPGA Signals for Multi-FPGA Prototyping Platform
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  the 9th International Symposium on Applied Reconfigurable Computing, Los Angeles, Marsh 2013.

- New Synthesis Approach of hierarchical Benchmarks for Hardware Prototyping
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  Design and Technology of Integrated Systems (DTIS'13), Abu Dhabi, Marsh 2013

- Partitioning Constraints and Signal Routing Approach For Multi-FPGA Prototyping Platform
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  International Symposium on System-on-Chip 2013, Tampere, Finland, October 22-24 2013

- Routing Environment of Inter-FPGA Signals Based on Iterative Approach
  Mariem Turki, Zied Marrakchi, Habib Mehrez, Mohamed Abid
  21st IFIP/IEEE International Conference on Very Large Scale integration(VLSI-SoC) Istanbul, Turkey, October 7-9 ,2013

- MPSoC architecture for H.264/AVC intra prediction chain on SoCLiB platform and FPGA technology
  Nidhameddine Belhadj, Mariem Turki, Zied Marrakchi, Mohamed Ali Ben Ayed, Nouri Masmoudi, Habib Mehrez
  14th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), 2013

Références Bibliographiques

[1] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A cycle-accurate, cycle reproducible multi-FPGA system for accelerating mutli-core processor simulation," *in Proc. of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pp. 153–162, 2012. xv, 1, 10, 26

[2] V.Betz, J. ROSE and A. MARQUARDT , *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, January 1999. xv, xvi, 17, 19, 46, 48, 50, 120

[3] Y. Y. C. Huang and C. Hsu, "Soc hw/sw verification and validation," *in Proc. of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 297–300, 2011. 1

[4] M. Santarini, "Asic prototyping: Make versus buy," November 2005. 1, 10

[5] , "FPGA-Based Prototyping Methodology Manual," *Synopsys*, 2011. 1

[6] I. Kuon, J. Rose, "Measuring the gap between FPGAs and ASICs," *International Symposium on Field-Programmable Gate Array*, February 2006. 1

[7] H. Krupnova, "Mapping multi-million gate socs on FPGAs: industrial methodology and experience," *in Proc. of Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, pp. 1236–1241, 2004. 1

[8] D. Group *www.dinigroup.com/new/dnv6f6pcie.php*. 1, 69

[9] *http://www.aldec.com/en*. 1

[10] *www.gidel.com*. 1

[11] Synopsys, "Certify® Partition Driven Synthesis User Guide," March 2011. 2

[12] *http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/HAPS.aspx*. 2

[13] *http : //www.s2cinc.com/magic/FPGA_ Prototyping.htm*. 2, 29

[14] *http://www.cadence.com*. 2

[15] *http://www.systematic-paris-region.org/*. 2

[16] *http://reflexces.com/*. 2

[17] Flexras *www.flexras.com*. 2, 69, 98

[18] *http://www.adacsys.com/*. 2

[19] J. Babb, R. Tessier and A. Agarwal, "Virtual wires:overcoming pin limitations in FPGA-based logic emulators," in *Proceedings of IEEE Workshop FPGAs Custom Computing Machines*, pp. 142–151, 1993. 3, 36, 93

[20] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki and A. Agarwal, "Logic Emulation with Virtual Wires," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 16, JUNE 1997. 3, 5, 37, 43, 77

[21] M. Inagi, Y. Takashima and Y. Nakamura, "Globally Optimal Time-Multiplexing Inter-FPGA Connections for Accelerating Multi-FPGA Systems," *Field Programmable Logic Conference*, 2009. 3, 5, 37, 42, 45

[22] M. Inagi, Y. Takashima, Y. Nakamura and A. Takahashi, "Optimal Time-multiplexing in inter-FPGA connections for Accelerating Multi-FPGA Prototyping Systems," *IEICE Trans. Fundamentals*, vol. E91A, pp. 3539–3547, Dec 2008. 3, 42, 77, 93

[23] H. Su and Y. Lin, "A phase Assignement method for virtual-wire-based hardware emulation," *IEEE Transactions on Computer-Aided Design of integrated Circuits and systems*, July 1997. 3

[24] Y. Kwon and C. Kyung, "Performance-driven event-based synchronization for multi-FPGA simulation accelerator with event time-multiplexing bus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Sep 2005. 3

[25] Qingshan TANG, Habib MEHREZ and Matthieu Tuna, "Routing Algorithm for Multi-FPGA Based Systems Using Multi-Point Physical Tracks," *IEEE International Symposium on Rapid System Prototyping, RSP*, October 2013. 5

[26] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array," 5

[27] J.Rose and V. Betz., "Effect of the prefabricated routing track distribution on fpga area-efficiency," 6

[28] M. Tom, D. Leong, G. Lemieux, "Un/DoPack: Re-Clustering of Large System-on-Chip Designs with Interconnect Variation for Low-Cost FPGAs," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 680–687, 2006. 6, 47

[29] "Computer aided design benchmarking laboratory," *http://www.cbl.ncsu.edu/benchmarks/*. 6, 54

[30] C. J. Alpert, "The ispd circuit benchmark suite," in *in Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pp. 85–90, 1998. 6, 54

[31] "layout synthesis benchmark set," *microelectronics center of north carolina, research triangle park, NC*, May 2006. 6, 54, 123

[32] R. Kuznar, F. Brglez, and K. Kozminski, "Cost minimization of partitions into multiple devices," in *In Proc. 30th ACM/IEEE Design Automation Conf*, pp. 315–320, June 1993. 6

[33] D. Stroobandt, P. Verplaetse, and J. van Campenhout, "Generating synthetic benchmark circuits for evaluating cad tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1011–1022, Sept. 2000. 7, 54

[34] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Trans. on Comput*, vol. C.20, pp. 1469–1479, 1971. 7

[35] C. Albrecht, "Iwls benchmarks," *www.iwls.org/iwls2005*, 2005. 7

[36] "Soclib project: Platform for modeling and simulation of integrated systems on chip," *http://www.soclib.fr/*. 7, 56

[37] L.McMurchie and C.Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," *International Workshop on Field Programmable Gate Array*, 1995. 8, 48, 49, 50, 79, 80

[38] *http : //fr.wikipedia.org/wiki/Bug_de_la_division_du_Pentium*, 1994. 11

[39] *http://www.journaldunet.com/solutions/99dec/991207pentium3.shtml*, 1999. 11

[40] *http://www.clubic.com/actualite-87786-quadri-coeurs-amd-phenom-buggue.html*, 2007. 11

[41] , "," *http://www.mentor.com/products/fv/modelsim/*. 11

[42] , "," *http://www.xilinx.com/tools/isim.htm*. 11

[43] , "," *http://www.cadence.com/products/sd/palladium_series/pages/default.aspx*. 12

[44] , "," *http://www.synopsys.com/Tools/Verification/hardware-verification/emulation/Pages/zebu-server-asic-emulator.aspx*. 13

[45] , "," *http://www.mentor.com/products/fv/emulation-systems/*. 13

[46] J.Rose and R.Francis and D.Lewis and P.Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1217–1225, 1990. 16

[47] Elias Ahmed and Jonathan Rose , "The effect of LUT and cluster size on deep-submicron FPGA performance and density," in *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, FPGA '00, (Monterey, California, United States), pp. 3–12, 2000. 16

[48] Altera, "Stratix V FPGAs," *http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp*. 17

[49] Xilinx, "Benchmark designs for the quartus university interface program (QUIP)," tech. rep., March 2005. 17

[50] Hongbing Fan, Yu-Liang Wu and Catherine L. Zhou , "Augmented disjoint switch boxes for FPGAs," in *Proceedings of the 4th international symposium on Information and communication technologies*, WISICT '05, (Cape Town, South Africa), pp. 129–134, Trinity College Dublin, 2005. 19

[51] Steven J. E. Wilton, *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory*. PhD thesis, University of Toronto, 1997. 19

[52] Yao-Wen Chang, D. F. Wong, and C. K. Wong , "Universal switch modules for FPGA design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, pp. 80–101, January 1996. 19

[53] , "," *www.auspy.com.* 27

[54] Zied Marrakchi, Christophe Alexandre and Ramsis Farhat, "Timing-Driven Hybrid RTL/Gate Partitioning for Predictable FPGA-Based Prototyping," *Design and Verification conference, DVcon*, March 2014. 32

[55] R. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," in *Proceedings of ISCAS*, pp. 29–54, 1982. 33

[56] R. Brayton, G.Hachtel and A. Sangiovanni-Vincentelli, "Multi-level logic synthesis," *Proceedings of the IEEE*, vol. 78, pp. 264–300, Feb 1990. 33

[57] K. Nelsen, "High-level design methodology overview," *Synopsys Online Documentation : Methodology Notes.* 34

[58] *Synopsys FPGA Synthesis User Guide*, 2011. 34, 69

[59] C.M. Fiduccia and R.M.Mattheyeses , "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th Design Automation Conference*, DAC '82, pp. 175–181, 1982. 35

[60] D.A.Papa and I.L.Markov, "Hypergraph Partitioning and Clustering," *Technical Report, University of Michigan, EECS Department*, 1987. 35

[61] Vijayshri Maheshwari , Joel Darnauer , John Ramirez and Wayne Wei-Ming Dai, "Design of FPGAs with area I/O for field programmable MCM," *In FPGA'95*, Feb 1995. 36

[62] S.Hauck, "The role of fpga in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, April 1998. 36

[63] M. Kudlugi and R.Tessier, "Static scheduling of multidomain circuits for fast functional verification," 39

[64] C.Sechen and A.Sangiovanni-Vincentelli , "The Timberwolf Placement and Routing Package ," *IEEE Journal of Solid-State Circuits*, vol. SC-20, no. 2, pp. 510–522, 1985. 45, 119

[65] S.Kirkpatrick, C.D.Gelatt, and M.P.Vecchi , "Optimization by Simulated Annealing ," *Science*, vol. 220, pp. 671–680, 1983. 45, 119

[66] V.Betz and J.Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA ," in *International workshop on Field-programmable logic and applications* , (London), pp. 213–222, 1997. 46, 48

[67] Joachim Pistorius , Mike Hutton , "Placement Rent Exponent Calculation Methods, Temporal Behaviour and FPGA Architecture Evaluation," *SLIP'03: Proceedings of the 2003 international workshop on System-level interconnect prediction*, pp. 31–38, 2003. 47

[68] B.Landman and R.Russo, "On Pin Versus Block Relationship for Partition of Logic Circuits," *IEEE Transactions on Computers*, vol. 20, no. 1469-1479, 1971. 47

[69] Yue Zhuo, Hao Li , Mohanty, S.P., "A Congestion Driven Placement Algorithm for FPGA Synthesis," *FPL '06. International Conference on Field Programmable Logic and Applications*, pp. 1–4, August, 2006. 47, 48, 122

[70] D. Yeager, D. Chiu, and G. Lemieux, "Congestion estimation and localization in fpgas: A visual tool for interconnect prediction," 47, 121

[71] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*. 2001. 49, 77, 81

[72] Daniel Gomez-Prado and Maciej Ciesielski , "A Tutorial on FPGA Routing," *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.9313*. 49

[73] N. Pouillon and A. Greiner *URL=https://www-asim.lip6.fr/trac/dsx/*, 2006-2008. 56

[74] M. Turki, H. Mehrez, Z. Marrakchi, and M. Abid, "Towards synthetic benchmarks generator for cad tool evaluation," *PRIME 2012 - 8th Conference on Ph.D. Research in Microelectronics & Electronics*, 2012. 64, 65

[75] M. Inc, "Menta efpga core-ii data sheet brief," *http://www.menta.fr/down/DatasheetBrief-eFPGA-core-II.pdf*, Feb. 2009. 64

[76] M. Inc, "M2000 intros largest 90nm efpga, design and reuse," *http://www.design-reuse.com/news/9614/m2000-introslargest-90nm-efpga.html*, Feb. 2005. 64

[77] "http://www-soc.lip6.fr/recherche/cian/," 64

[78] Nidhameddine Belhadj, M. Ali Ben Ayed, Nouri Masmoudi, Mariem Turki, Zied Marrakchi and Habib Mehrez, "MPSoC Architecture for H.264/AVC Intra Prediction Chain on SoCLib Platform and FPGA Technology," *14th international conference on Sciences and Techniques of Automatic control and computer engineering* , December 2013. 66

[79] M. Turki and H. Mehrez and Z. Marrakchi and M. Abid, "New Synthesis Approach of hierarchical Benchmarks for Hardware Prototyping," *DTIS*. 67

[80] "Xilinx. xst. www.xilinx.com/products/design tools/logic design/synthesis/xst.htm," 69

[81] M. Turki and H. Mehrez and Z. Marrakchi and M. Abid, "Multi-FPGA Prototyping Environment: Large Benchmark Generation and Signals Routing," , December 2012. 78

[82] Daniel Gomez-Prado and Maciej Ciesielski , "A Tutorial on FPGA Routing," *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.9313*. 81

[83] M. Turki and H. Mehrez and Z. Marrakchi and M. Abid, "Partitioning Constraints and Iterative Routing Approach for Multi-FPGA Prototyping Platform," , October 2013. 104

[84] , "Certify Partition Driven Synthesis, User Guide," p. 153, March, 2011. 108

[85] V.Betz and J.Rose, "VPR: A New Packing Placement and Routing Tool for FPGA Research," *International Workshop on FPGA*, pp. 213–22, 1997. 119, 120

[86] H. L. yue Zhue and S. Mohanty, "Congestion driven placement for fpga synthesis," *International Conference on Field Programmable Logic and Application*, pp. 1–4, 2006. 120, 122

[87] Z. Marrakchi, M. Turki, J. Rebourg, M. Abid, and H. Mehrez, "Congestion driven placement for mesh-based fpga architecture with local interconnect," *International Conference on Microelectronics (ICM)*, 2011. 121, 123

Annexe

.1 Fichiers d'entrée pour la génération d'un benchmark

Dans cet annexe, nous présentons les fichiers nécessaires pour la génération de l'architecture représentée par la figure 15.dans le fichier de metadonnées, seul l'exemple du bus Sring est présenté.

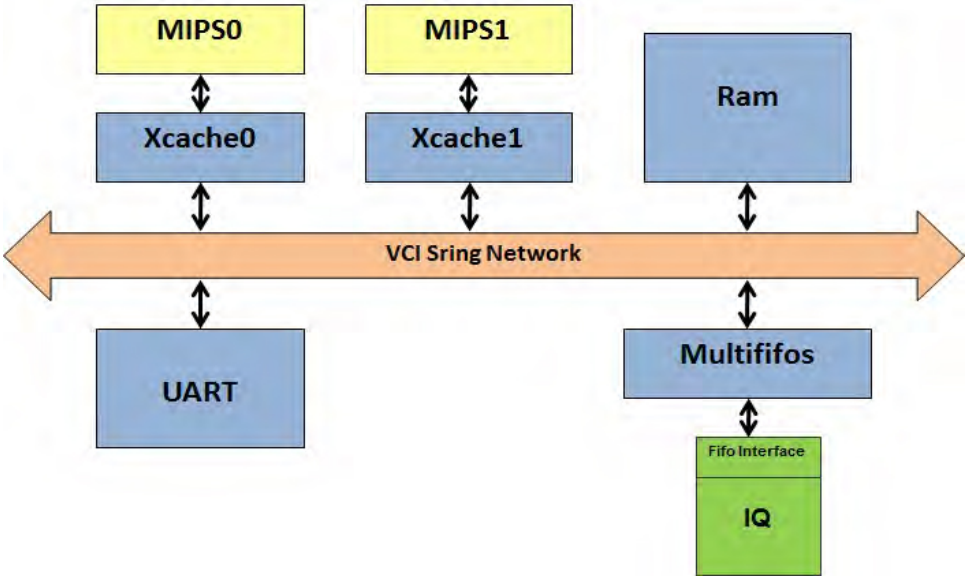


FIGURE 15 – Exemple d'architecture à base de deux processeurs MIPS

.1.1 Description de l'architecture

```
    import soclib
import vhdlib
import dsx
from dsx import *


    def gen_arch( umapping) :
if umapping['mode']=='rtl' :
```

```
pf = vhdlib.Architecture(
abstraction_level = umapping['mode'],
cellsize = 4,
plensize = 8,
addrsize = 32,
rerrorsize = 1,
clensize = 1,
rflagsize = 1,
srcidsize = 8,
pktidsize = 4,
trdidsize = 4,
wrplensize = 1
)
```

sring = pf.create('vci_simplering_network', 'sring', NB_INIT = 1, NB_TARGET = 3, MSBSIZE = 8, CMDDATASIZE = 37,RSPDATASIZE = 33, CMDDEPTH = 2, RSPDEPTH = 2)

xcache = pf.create('soclib_vci_xcache', 'xcache', ICACHE_LINES = 32, ICACHE_WORDS = 4, DCACHE_LINES = 32, DCACHE_WORDS = 4,XCACHE_ENABLE = 0)

ram = pf.create('soclib_vci_simpleram', 'ram', NBITS_RAM_SIZE = 18)

uart = pf.create('soclib_vci_uart','uart')

mips0 = pf.create('soclib_classic_mips','mips0', EXCP_HANDLER = 128, MASK_UNCACHE = 8, ID_PROC = 0)

mips1 = pf.create('soclib_classic_mips','mips1', EXCP_HANDLER = 128, MASK_UNCACHE = 8, ID_PROC = 1)

coproc = pf.create('iq', 'iq0')

multi_fifos = pf.create('soclib_vci_target_multi_fifos','multi_fifos',FIFO_READ_SIZE = 2, FIFO_WRITE_SIZE = 2, N_FIFO_READ = 2, N_FIFO_WRITE = 1, N_CONFIG_REG = 2, N_STATUS_REG = 1, THRESHOLD_READ = 1, THRESHOLD_WRITE = 1,)

sring.to_target[2] //multi_fifos.vci

coproc.iq2zz // multi_fifos.fifo_to_target_w[0]

coproc.demux2iq // multi_fifos.fifo_to_target_r[0]

coproc.vld2iq // multi_fifos.fifo_to_target_r[1]

coproc.reset // multi_fifos.softresetn

sring.clk // multi_fifos.clk // coproc.CK

sring.resetn // multi_fifos.resetn

coproc.start.setConstant("'1'")

coproc.abort.setConstant("multi_fifos_fifo_to_target_cfg_0_config(0)")

```
    mips0.it_0 // mips1.it_0 // uart.irq
mips0.it_5 // mips0.it_4 // mips0.it_3 // mips0.it_2 // mips0.it_1 //mips1.it_1
//mips1.it_2 //mips1.it_3 //mips1.it_4 //mips1.it_5
mips0.it_5.setConstant("'0'")

    xcache.cfg_cmd.setConstant('B"00"')
xcache.cfg_data_in.setConstant("'0'")
xcache.cfg_data_out.setConstant("'0'")
mips0.cfg_cmd.setConstant('B"00"')
mips0.cfg_data_in.setConstant("'0'")
mips0.cfg_data_out.setConstant("'0'")

    mips1.cfg_cmd.setConstant('B"00"')
mips1.cfg_data_in.setConstant("'0'")
mips1.cfg_data_out.setConstant("'0'")

    mips0.icache // mips1.icache // xcache.icache
mips0.dcache // mips1.dcache // xcache.dcache

    sring.to_initiator[0] // xcache.vci
sring.to_target[0] // ram.vci
sring.to_target[1] // uart.vci

    uart.rx.setExternal('rs232_rx')
uart.tx.setExternal('rs232_tx')
uart.rx8.setExternal('s')

    sring.clk // ram.clk // xcache.clk // uart.clk // mips0.clk // mips1.clk
sring.resetn // ram.resetn // xcache.resetn // uart.resetn // mips0.resetn // mips1.resetn

    sring.clk.setExternal('clk')
sring.resetn.setExternal('resetn')

    return pf

    else :
print('unsupported mode :
```

.1.2 Description des metadonnées

```
    Module('vci_simplering_network',
tmpl_parameters = [parameter.Module('vci_params', typename = 'rtl :vci_params',
default = 'rtl :vci_params')],
instance_parameters = [
parameter.Int('NB_INIT'),
parameter.Int('NB_TARGET'),
parameter.Int('MSBSIZE'),
parameter.Int('CMDDATASIZE'),
parameter.Int('RSPDATASIZE'),
parameter.Int('CMDDEPTH'),
parameter.Int('RSPDEPTH'),
],


    implementation_files = ['../source/vhdl/vci_vgmn.vhd',],
ports = [
Port('rtl :simplering_initiator', 'p_to_target', parameter.Reference('NB_TARGET')),
Port('rtl :simplering_target', 'p_to_initiator', parameter.Reference('NB_INIT')),
Port('rtl :bit_in', 'resetn'),
Port('rtl :bit_in', 'clk'),
],
)


    Signal('rtl :bit',
classname = 'std_logic',
accepts =
'rtl :bit_in' : True,
'rtl :bit_out' : True,
)


    Signal('rtl :vector',
classname = 'std_logic_vector',
tmpl_parameters = [parameter.Int('W'),],
accepts =
'rtl :in_vector' : True,
'rtl :out_vector' : True,
'rtl :fifo_to_target_status' : True,
'rtl :fifo_to_target_config' : True,
)
```

```
MetaSignal('rtl :vci',
classname = 'vci_meta_signal',
tmpl_parameters = [parameter.Module('vci_params',typename = 'rtl :vci_params',
default = 'rtl :vci_params')],
accepts =
'rtl :vci_initiator' : True,
'rtl :vci_target' : True,
'rtl :simplering_initiator' : True,
'rtl :simplering_target' : True,
,
sub_signals =
'cmdval' :SubSignal('rtl :bit'),
'rspack' :SubSignal('rtl :bit'),
'address' :SubSignal('rtl :vector',W = parameter.Reference('addrsize')),
'be' :SubSignal('rtl :vector', W = parameter.Reference('cellsize')),
'plen' :SubSignal('rtl :vector', W = parameter.Reference('plensize')),
'cmd' :SubSignal('rtl :vector', W = 2),
'wdata' :SubSignal('rtl :vector', W = 8*parameter.Reference('cellsize')),
'eop' :SubSignal('rtl :bit'),
'contig' :SubSignal('rtl :bit'),
'cons' :SubSignal('rtl :bit'),
'srcid' :SubSignal('rtl :vector', W = parameter.Reference('srcidsize')),
'trdid' :SubSignal('rtl :vector', W = parameter.Reference('trdidsize')),
'pktid' :SubSignal('rtl :vector', W = parameter.Reference('pktidsize')),
'cmdack' :SubSignal('rtl :bit'),
'rspval' :SubSignal('rtl :bit'),
'rdata' :SubSignal('rtl :vector', W = 8*parameter.Reference('cellsize')),
'reop' :SubSignal('rtl :bit'),
'rerror' :SubSignal('rtl :bit'),
'rsrcid' :SubSignal('rtl :vector', W = parameter.Reference('srcidsize')),
'rpktid' :SubSignal('rtl :vector', W = parameter.Reference('pktidsize')),
'rtrdid' :SubSignal('rtl :vector', W = parameter.Reference('trdidsize')),
,
can_metaconnect = True,
)


    PortDecl('rtl :bit_in',
classname = 'in std_logic',
signal = 'rtl :bit',
)
```

```
    PortDecl('rtl :bit_out',
classname = 'out std_logic',
signal = 'rtl :bit',
)

    PortDecl('rtl :in_vector',
classname = 'in std_logic_vector',
tmpl_parameters = [parameter.Int('W'),],
signal = 'rtl :vector',
)

    PortDecl('rtl :out_vector',
classname = 'out std_logic_vector',
tmpl_parameters = [parameter.Int('W'),],
signal = 'rtl :vector',
)

    PortDecl('rtl :simplering_initiator',
tmpl_parameters = [parameter.Module('vci_params',typename = 'rtl :vci_params',
default = 'rtl :vci_params')],
instance_parameters = [],
signal = 'rtl :vci',
ports = [
Port('rtl :bit_out', 'cmdval'),
Port('rtl :bit_out', 'rspack'),
Port('rtl :out_vector', 'address', W = parameter.Reference('addrsize')),
Port('rtl :out_vector', 'be', W = parameter.Reference('cellsize')),
Port('rtl :out_vector', 'plen', W = parameter.Reference('plensize')),
Port('rtl :out_vector', 'cmd', W = 2),
Port('rtl :out_vector', 'wdata', W = 8*parameter.Reference('cellsize')),
Port('rtl :bit_out', 'eop'),
Port('rtl :bit_out', 'contig'),
Port('rtl :bit_out', 'cons'),
Port('rtl :out_vector', 'srcid', W = parameter.Reference('srcidsize')),
Port('rtl :out_vector', 'trdid', W = parameter.Reference('trdidsize')),
Port('rtl :out_vector', 'pktid', W = parameter.Reference('pktidsize')),
Port('rtl :bit_in', 'cmdack'),
Port('rtl :bit_in', 'rspval'),
Port('rtl :in_vector', 'rdata', W = 8*parameter.Reference('cellsize')),
```

```
Port('rtl :bit_in', 'reop'),
Port('rtl :bit_in', 'rerror'),
Port('rtl :in_vector', 'rsrcid', W = parameter.Reference('srcidsize')),
Port('rtl :in_vector', 'rpktid', W = parameter.Reference('pktidsize')),
Port('rtl :in_vector', 'rtrdid', W = parameter.Reference('trdidsize')),
],
)
```

```
PortDecl('rtl :simplering_target', tmpl_parameters = [parameter.Module('vci_params',typename = 'rtl :vci_params', default = 'rtl :vci_params')],
instance_parameters = [],
signal = 'rtl :vci',
ports = [
Port('rtl :bit_in', 'cmdval'),
Port('rtl :bit_in', 'rspack'),
Port('rtl :in_vector', 'address', W = parameter.Reference('addrsize')),
Port('rtl :in_vector', 'be', W = parameter.Reference('cellsize')),
Port('rtl :in_vector', 'plen', W = parameter.Reference('plensize')),
Port('rtl :in_vector', 'cmd', W = 2),
Port('rtl :in_vector', 'wdata', W = 8*parameter.Reference('cellsize')),
Port('rtl :bit_in', 'eop'),
Port('rtl :bit_in', 'contig'),
Port('rtl :bit_in', 'cons'),
Port('rtl :in_vector', 'srcid', W = parameter.Reference('srcidsize')),
Port('rtl :in_vector', 'trdid', W = parameter.Reference('trdidsize')),
Port('rtl :in_vector', 'pktid', W = parameter.Reference('pktidsize')),
Port('rtl :bit_out', 'cmdack'),
Port('rtl :bit_out', 'rspval'),
Port('rtl :out_vector', 'rdata', W = 8*parameter.Reference('cellsize')),
Port('rtl :bit_out', 'reop'),
Port('rtl :bit_out', 'rerror'),
Port('rtl :out_vector', 'rsrcid', W = parameter.Reference('srcidsize')),
Port('rtl :out_vector', 'rpktid', W = parameter.Reference('pktidsize')),
Port('rtl :out_vector', 'rtrdid', W = parameter.Reference('trdidsize')),
],
)
```

.2 Exemple de script pour une synthèse logique rapide

Dans cette section, nous présentons un exemple de script permettant la synthèse logique rapide du bus "sring" ayant l'architecture représentée dans la figure 16.

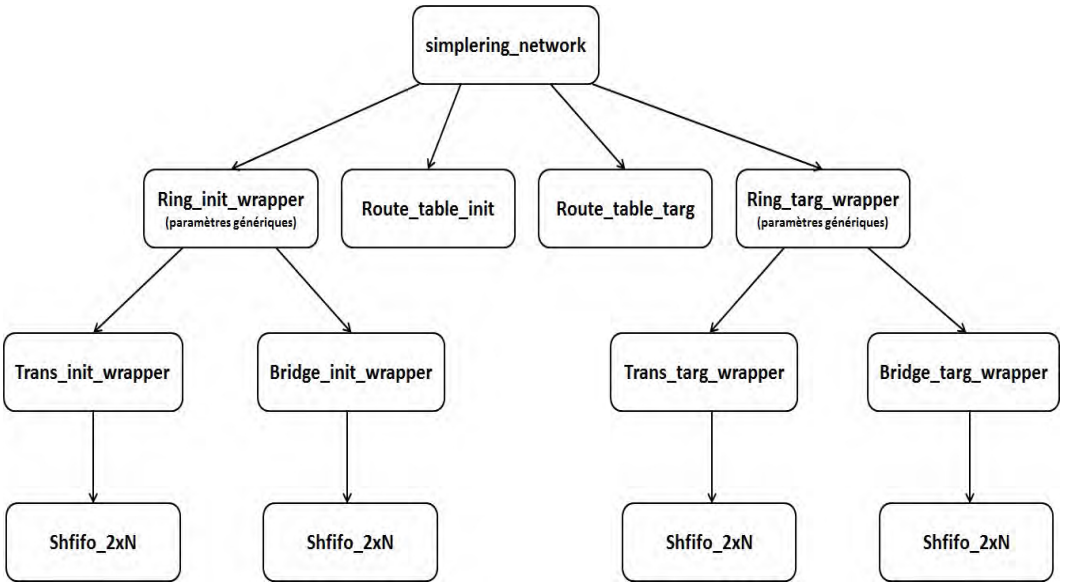


FIGURE 16 – Niveaux hiérarchique du bus local "sring"

script.tcl

project -new
impl -name Sring

set vfileList [glob -directory VHDL/ *.vhd]
foreach vfile $vfileList
add_file -vhdl $vfile

project -run compile

set_option -technology virtex5
set_option -part XC5VLX110
set_option -package FF1760
set_option -top_module simplering_network
set_option -fast_synthesis 1
project -save top_level.prj

set all {bridge_init_wrapper bridge_targ_wrapper route_table_init route_table_targ ring_init_wrapper ring_targ_wrapper simplering_network trans_init_wrapper trans_targ_wrapper shfifo_2xN}

set simplering_network {route_table_init route_table_targ trans_init_wrapper bridge_init_wrapper trans_targ_wrapper bridge_targ_wrapper}
set trans_init_wrapper {shfifo_2xN}
set bridge_init_wrapper {shfifo_2xN}
set trans_targ_wrapper {shfifo_2xN}
set bridge_targ_wrapper {shfifo_2xN}
set route_table_init {}
set route_table_targ {}
set shfifo_2xN {}

```
foreach vfile $all
set_option -top_module $vfile
project -log_file log/$vfile.srr
set_option -write_verilog 1
project -result_file $vfile
# no buffer generation
set_option -disable_io_insertion 1
# remove module children from project
# to be considered as black boxes
set father $vfile
foreach child [expr $$father]
project_file -remove VHDL/$child.vhd

# synthesize the module
project -run synthesis
# add back module children to project
foreach child [expr $$father]
add_file -vhdl VHDL/$child.vhd
```