Placement des tâches matérielles de tailles variables sur des architectures reconfigurables dynamiquement et partiellement

Marwa Hannachi

Placement des tâches matérielles de tailles variables sur des architectures reconfigurables dynamiquement et partiellement

THÈSE

présentée et soutenue publiquement le 20 décembre 2017

pour l'obtention du

Doctorat de l'Université de Lorraine et de l'Université de Monastir
(mention: Système èlectronique) (mention: Génie Electrique)

par

Marwa HANNACHI

Composition du jury

Président :	M. El-bay BOURENNANE, Professeur, Le2i, Université de Bourgogne
Rapporteurs :	Mme. Fan YANG, Professeur, Le2i, Université de Bourgogne
	Mme. Dorra SELLAMI MASMOUDI, Professeur, ENIS, Université de Sfax
Examinateurs :	M. Noureddine LIOUANE, Professeur, ENIM, Université de Monastir
Directeurs :	M. Hassan RABAH, Professeur, IJL, Université Lorraine
	M. Abdellatif MTIBAA, Professeur, ENIM, Université de Monastir
Invité :	M. Slavisa JOVANOVIC, Maître de conférences, IJL, Université Lorraine

Laboratoire Institut Jean Lamour (UMR7198)
Equipe MAE 406, France
Laboratoire Electronique et micro-electronique
Monastir, Tunisie

Remerciements

Je tiens à remercier tout particulièrement mes directeurs de thèse Monsieur Hassan Rabah professeur à l'université de Lorraine et Monsieur Abdellatif Mtibaa professeur à l'université de Monastir, pour la confiance qu'ils m'ont témoignée tout au long de ces années de travail, pour leurs participations et pour leurs précieux conseils au niveau scientifique.

Je remercie tout particulièrement Monsieur Abdesslam Ben Abdelali qui a participé à l'encadrement de cette thèse et su manifester son grand intérêt pour ce travail avec beaucoup d'efforts. Ses qualités pédagogiques remarquables m'ont permis de profiter de ses connaissances et ont contribué à l'avancement de mon travail en ne négligeant ni ses conseils avisés et ni ses critiques constructives.

Je tiens également à remercier Monsieur El-Bay Bourennane professeur à l'université de Bourgogne pour avoir accepté de présider le jury de soutenance. J'adresse aussi mes sincères remerciements à Madame Fan Yang, professeur à l'université de Bourgogne ainsi qu'à Madame Dorra Sellami Masmoudi, Professeur à l'ENIS, université de Sfax pour avoir accepté d'en être les rapporteurs. Je tiens également à remercier Monsieur Noureddine Liouane Professer à l'ENIM, université de Monastir, pour avoir accepté d'examiner ce travail et à Monsieur Slavisa Jovanovic, Maitre de conférence à l'université de Lorraine, pour avoir accepté de participer au jury.

Je remercie tous les collègues et amis à l'équipe CSR-ENIM, leurs amabilités et surtout la convivialité. Je remercie également tous les membres et collègues de l'équipe MAE 406 pour leur accueil et les meilleures conditions de travail.

Enfin, un très grand MERCI à mes parents, mon époux et mes sœurs qui m'ont gratifié de leurs amours et qui m'ont soutenu durant cette aventure. Je leur adresse toute ma gratitude du fond du cœur.

Je dédie cette thèse à mon mari Atef et mon bébé Adam

Sommaire

Table des figures

Liste des tableaux

Introduction générale

1 Contexte général

Sous la pression des innovations des applications et des systèmes électroniques, les technologies de l'information occupent de plus en plus de place préférentielle dans notre vie quotidienne. Cette révolution technologique a amené les concepteurs à faire appel pour réaliser un système complet sur puce SoC (System on Chip). Les SoC présentent un support pour des applications de plus en plus complexes. Cette évolution dans le marché pose de nouvelles contraintes sévères lors de la conception d'un SoC tel que la consommation, la performance, l'adaptabilité et la flexibilité à part les autres contraintes du coût et du temps de mise sur le marché. Ces contraintes deviennent de plus en plus difficiles à satisfaire avec la capacité d'intégration sur silicium qui augmente exponentiellement dans le temps. Selon la loi de Moore, la densité d'intégration des transistors sur une seule puce double environ tous les deux ans. Pour cela, les circuits logiques reconfigurables sont apparus en réponse à ces exigences. Le circuit intégré spécifique à une application (ASIC : Application Specific Integrated Circuit) représente une solution optimale pour faire face aux taux élevés d'intégration. Ces derniers permettent d'atteindre des très hautes performances pour une faible consommation d'énergie comparativement à des solutions logicielles. Cependant, ces circuits nécessitent un temps et un coût de développement très élevés, aussi ils manquent de flexibilité. En effet, cette solution technologique n'est viable que pour un nombre limité des applications. Pour cela, les FPGA (Field Programmable Gate Array) sont une solution qui dispose d'une grande flexibilité avec de bonnes performances et un temps de mise en œuvre réduit par rapport aux ASIC. La flexibilité des FPGA n'est pas le seul avantage le plus envisageable. La technique de la reconfiguration dynamique partielle (RDP) dans les FPGA permet la modification de certains blocs du FPGA au moment de l'exécution. La RDP peut améliorer l'efficacité des FPGA en permettant à différentes fonctions de traitement de partager les mêmes ressources matérielles et par conséquent, réduire l'utilisation globale des ressources. De plus, l'implémentation des nouvelles fonctions de traitement sur des régions reconfigurables permet d'augmenter l'adaptabilité du système. La RDP peut également améliorer les performances du système en multiplexant le temps de plus grandes fonctions qui partagent les mêmes ressources matérielles. En fait, la RDP offre des opportunités intéressantes pour implémenter de nouvelles architectures qui exploitent efficacement les ressources du FPGA.

La notion de la reconfiguration dynamique est apparue depuis 1960 par Estrin dans

[Est60]. L'idée de base était de reconfigurer une partie variable dans le système pour traiter une nouvelle tâche. Cette notion est devenue l'une des principales tendances technologiques qui permet de fusionner les avantages du logiciel et du matériel en terme de flexibilité et de performance. Actuellement, le FPGA est considéré comme la solution la plus adaptée aux systèmes reconfigurables. La capacité de reconfiguration des FPGA modernes permet d'implémenter des applications dynamiques très complexes capable d'adapter leurs fonctionnements pour répondre aux besoins en temps réel. En fait, les FPGA modernes ont beaucoup profité par les avancements technologiques pour devenir des périphérique contenant de nombreux composants spécialisés. La nouveauté réside dans le fait d'associer au processeur classique un FPGA pour effectuer des tâches de pré-traitement de données qui rendent leur exécution par le processeur plus rapide.

Malgré tous les avantages, la reconfiguration dynamique sur FPGA n'est pourtant pas encore dans une position dominante sur le marché puisque les outils de conception n'évoluent pas au même rythme que la technologie matérielle. L'absence des méthodologies de conception bien adaptées et la limitation des moyens et des outils mis à la disposition des concepteurs sont la cause principale d'une grande complexité de conception. Sa mise en œuvre nécessite une participation du concepteur et d'un effort supplémentaire. Les travaux de cette thèse se situent dans ce contexte.

2 Motivations

Bien que la conception des FPGA basée sur la reconfiguration dynamique partielle est dotée d'une grande flexibilité et de performance à la mise en œuvre des SOC adaptatifs en théorie, le manque des méthodologies de conception et des moyens polyvalents excluent de nombreux systèmes. Dans les flots RDP du fournisseur, le concepteur doit fournir plusieurs entrées et prendre plusieurs décisions manuellement où l'efficacité du système dépend fortement de celles-ci à savoir : identifier les portions à reconfigurer, gérer les ressources du système (zone reconfigurable principalement), accroitre la réutilisation des modules matériels et vérifier le changement de contexte du système. L'introduction de ces paramètres nécessite une connaissance approfondie par le concepteur de le FPGA cible. De plus, pour avoir une conception optimale, le concepteur doit connaître les opérations de bas niveau nécessaires pour la mise en œuvre de la RDP. Un tel processus de conception manuelle nécessite beaucoup de temps et conduit généralement à des résultats non optimisés.

Afin de contourner la limitation des systèmes reconfigurables ciblés par les technologies FPGA, un nombre de facteurs clés est requis pour améliorer la méthodologie de conception qui sont :

— Relocation
— Gestion des zones reconfigurables
— Floorlanning

Relocation En terme de flexibilité, l'ajout croissant des accélérateurs matériels dans les architectures FPGA hétérogènes présente des avantages marqués, mais aussi des faiblesses importantes. En effet, les accélérateurs dans les FPGA améliorent considérablement l'efficacité des conceptions matérielles par rapport aux implémentations logicielles, mais limitent la flexibilité de l'architecture en cas de changement de contexte. Nous souhaitons pouvoir appliquer la technique de la relocation afin d'assurer une adaptation du changement de contexte plus flexible aux FPGA. La relocation des tâches offre une flexibilité maximale d'une architecture reconfigurable en optimisant au moment de l'exécution les performances spatiale et temporelle d'une application. D'autre part, la relocation permet également la réutilisation d'une tâche matérielle sur plusieurs endroits compatibles, de la même manière qu'une tâche logicielle. Néanmoins, la relocation des tâches matérielles en temps réel a été devenu presque irréalisable avec la complexité des nouvelles technologies des FPGA. Cette complexité est due à l'hétérogénéité de l'architecture matérielle FPGA et au manque des supports logiciels dans la plupart des outils de conception proposés par les fournisseurs.

Floorplanning La RDP peut augmenter l'efficacité d'utilisation de la surface nécessaire à l'exécution d'une application, mais cette capacité doit être explorée efficacement. Le choix des tailles et des emplacements des zones reconfigurables sont un problème complexe. La RDP nécessite un découpage du FPGA en zone reconfigurable dynamiquement PRR (Partial Reconfigurable Région). Le processus de découpage dans le flot de conception est nommé floorplanning. Cette étape est considérée essentielle dans la méthode de conception RDP. Cependant, un mauvais floorplanning impliquera une augmentation de la quantité des ressources matérielles inutilisées et donc une diminution des performances et d'efficacité du système. Le floorplanning des zones doit être réalisé selon la taille de la PRR qui est un arrondi supérieur de chaque type de ressources aux besoins des tâches. Plusieurs approches ont été proposées dans la littérature pour traiter le problème du floorplanning dans le FPGA, mais la majorité de ces approches ne tiennent pas compte des évolutions technologiques des architectures hétérogènes des FPGA. De plus, les travaux proposés ne prennent pas en considération l'aspect de relocation dans les architectures reconfigurables. Par conséquent, les contraintes imposées sur les PRR ne sont pas satisfaites ce qui peut entrainer une relocation des tâches non réalisables et par la suite un circuit endommagé.

Gestion des ressources reconfigurables Dans les architectures reconfigurables, plusieurs tâches peuvent être allouées dans une même zone reconfigurable. Toutes les tâches matérielles n'ont pas les mêmes tailles et par la suite, elles n'ont pas les mêmes besoins en ressources. De ce fait, les petites tâches peuvent être allouées sur de grandes PRR ce qui augmente de plus en plus la quantité de ressources non utilisées. Cette contrainte impose plus des limitations dans le cas de relocation, où les zones doivent être aussi de la même taille. Dans ce cas, la taille de la zone la plus grande est appliquée pour toutes les zones qui vont recevoir une même tâche. Pour augmenter l'efficacité d'utilisation dans une zone

reconfigurable, deux solutions ont été proposées dans la littérature, soit adapter la taille de la tâche à la taille de la zone en partitionnant l'IP en sous-tâches, ou adapter la taille de la région aux besoins des tâches matérielles. La première solution est une technique similaire au partitionnement d'un graphe de données. Dans [VF12], les auteurs proposent un nouvel outil de partitionnement. Le taux d'utilisation de ressources est de 62%. Cet outil nécessite un partitionnement manuel des tâches. Cette solution peut montrer une efficacité élevée dans le cas de graphe de données simples, mais reste complexe et inefficace pour les cas les plus complexes. La deuxième solution consiste à partitionner la région reconfigurable en plusieurs partitions. Une nouvelle méthode de partitionnement a été proposée dans [Mar12]. Cette méthode est basée sur le partitionnement de la zone reconfigurable en plusieurs partitions en insérant une couche intermédiaire entre le FPGA et l'application. Cette solution assure un placement efficace des tâches matérielles de tailles différentes sur des partitions reconfigurables fixes. Néanmoins elle ne tient pas compte des contraintes de relocation.

3 Contributions

L'objectif de cette thèse est d'apporter des solutions innovantes aux problématiques de conception des architectures reconfigurables dynamiques. Plus précisément, les principales contributions de ce travail consistent à développer des techniques et des algorithmes mettant en œuvre les concepts de flexibilité et d'adaptabilité en se basant sur la technologie reconfigurable des FPGA. Ces contributions visent à répondre aux questions suivantes.

Premièrement, nous allons proposer une méthodologie de conception permettant la relocation efficace des tâches matérielles dans les architectures reconfigurables. Les objectifs de cette partie sont résumés par les points suivants :

- Proposer un flot de conception adapté aux nouvelles générations des FPGA
- Automatiser la manipulation des bitstreams partiels
- Développer des interfaces de communications génériques pour toutes partitions
- Validation de la méthode sur différentes plateformes reconfigurables.
- Évaluer les performances de la méthode (temps de reconfiguration, ressources matérielles optimisées)

Ensuite, pour traiter les problèmes d'utilisation des ressources matérielles dans les FPGA lors de la relocation de tâches matérielles de tailles différentes sur des zones reconfigurables fixes, une méthodologie améliorée sera proposée. Les différents points à traiter seront les suivants :

- Améliorer le flot de conception de relocation proposé dans la première partie.
- Chercher un meilleur partitionnement du FPGA
- Générer des interfaces de communications des partitions tout en permettant le placement des tâches de différentes tailles.

Deuxièmement, nous allons proposer un algorithme de floorplanning efficace en prenant

en compte à la fois l'architecture FPGA cible et les besoins de PRR d'une application. L'algorithme respecte toutes les contraintes imposées par les architectures reconfigurables et par les d'outils du fournisseur. Par la suite, une extensibilité de cet algorithme est développée afin de garantir la flexibilité offerte par la technique de relocation ainsi, d'assurer une utilisation efficace des ressources matérielles dans le cas de placement ou de relocation des tâches de tailles variables sur des zones reconfigurables fixes.

Enfin, nous allons évaluer et valider l'ensemble de ces points au niveau expérimental sur des différents exemples.

4 Plan du manuscrit

Ce document de thèse est articulé en cinq chapitres. Le premier chapitre introduit les notions nécessaires à la compréhension de nos travaux tandis que les quatre autres chapitres présentent nos contributions et résultats.

Le premier chapitre sera entièrement dédié à présenter l'état de l'art. Dans un premier temps, nous présentons les principales caractéristiques et travaux sur les architectures reconfigurables. Ensuite, les principes de base de la relocation des bitstreams partiels, ainsi que les différents travaux existants seront introduits. Finalement, une étude sur les différents travaux traitant le problème de floorplanning sera détaillée.

Le deuxième chapitre sera dédié à l'étude d'une méthodologie de relocation. Après une présentation des différentes contraintes, une méthodologie de conception de système en tenant compte de la gestion des tâches de tailles variables en cas de relocation sera proposée .

Le troisième chapitre traite le problème d'automatisation de floorplanning dans les FPGAs reconfigurables. Une description de l'algorithme sera détaillée, prenant en compte à la fois de l'architecture du FPGA cible et des contraintes PR à respecter.

Dans le quatrième chapitre nous allons détailler les extensions apportées à l'algorithme de floorplannig pour répondre aux contraintes de relocation des tâches matérielles de tailles différentes sur des partitions reconfigurables.

Le cinquième chapitre présentera des études de cas pour valider les différentes méthodes proposées pour la relocation et le floorplanning.

Le dernier chapitre tiendra lieu de conclusion finale. Nous discuterons et dresserons le bilan du travail de thèse en présentant les apports proposés et en dégageant les perspectives ouvertes par ces travaux.

Chapitre 1

État de l'art

Sommaire

1 Introduction

Le concept de la reconfiguration dynamique partielle (RDP) est apparu pour mieux répondre à toutes les exigences et les besoins des SOC. Les architectures reconfigurables présentent des architectures plus riches en combinant les avantages des deux méthodes matérielles et logicielles. En intégrant des nouveaux blocs de traitement tels que les processeurs intégrés, les accélérateurs matériels dédiés, les mémoires distribuées, ces architectures assurent la mise en œuvre des applications de plus en plus complexes. Elles offrent, des compromis entre flexibilité, coût et performances inaccessibles aux ASIC et aux processeurs.

Ce chapitre aborde trois grandes parties. La première partie traite un état de l'art sur les architectures reconfigurables. Nous décrierons dans cette partie, la notion et le principe de la RDP, ainsi l'évolution des méthodologies de conception de la RDP. Différents travaux traitant les problèmes de gestion de la RDP, ainsi que différents outils dédiés à la RDP seront présentés. A la fin de cette partie nous présenterons quelques domaines exploitant les architectures reconfigurables dans la littérature. Dans la deuxième partie nous traitons un état de l'art sur la relocation des biststreams partiels. Nous commencerons par décrire brièvement le principe de cette technique. Après, nous présenterons les différents travaux existants dans la littérature qui traitent le problème de la relocation dynamique. Des travaux exploitant la relocation pour la tolérance aux fautes seront aussi présentés. La troisième partie de ce chapitre se focalise sur la présentation des algorithmes de floorplanning dans des différents travaux effectués dans la littérature.

2 Architectures reconfigurables

Au cours des dernières années, de nombreuses applications exploitent la nature dynamique des dispositifs reconfigurables. Grâce à leur technologie SRAM, les dispositifs reconfigurables peuvent être reprogrammés à tout moment. Cette reconfigurabilité sert à personnaliser traitements dans l'application pendant le fonctionnement afin d'améliorer sa performances en termes de différents aspects, tels que la vitesse, la puissance et les ressources logiques.

2.1 Reconfiguration dynamique et partielle

La reconfiguration dynamique et partielle (RDP) désigne la modification d'une ou plusieurs parties du FPGA sans impacter les autres parties qui peuvent continuer son fonctionnement durant toute l'exécution de l'application. Grâce à la RDP, certaines tâches matérielles du FPGA sont supprimées et remplacées de nouvelles tâches. L'implémentation de ces nouvelles tâches matérielles sur des régions reconfigurables permet d'augmenter l'adaptabilité et la flexibilité du système. Elle permet de gérer efficacement la surface de

FPGA en modifiant qu'une petite région en fonction des besoins de l'application. Ceci a pour but d'offrir une meilleure gestion des ressources internes du FPGA.

La RDP illustrée dans la FIGURE 1.1, désigne la modification de certaines régions du FPGA durant l'exécution, sans impacter les autres régions qui peuvent continuer leur exécution. Ceci permet le partage temporel des ressources physiques de cette région reconfigurable afin de supporter plusieurs états mutuellement exclusifs, chacun permettant d'exécuter une ou plusieurs tâches. Nous faisons appel à la RDP dans plusieurs cas : lorsque le système doit s'adapter au changement de l'environnement, ou bien pour répondre aux exigences de qualité de service, pour réduire les ressources physiques utilisées qui sont limitées, ou encore pour respecter certaines contraintes de consommation d'énergie, etc. Par conséquent, la RDP offre une meilleure gestion des ressources du FPGA et permet d'augmenter l'adaptabilité et la flexibilité du système.

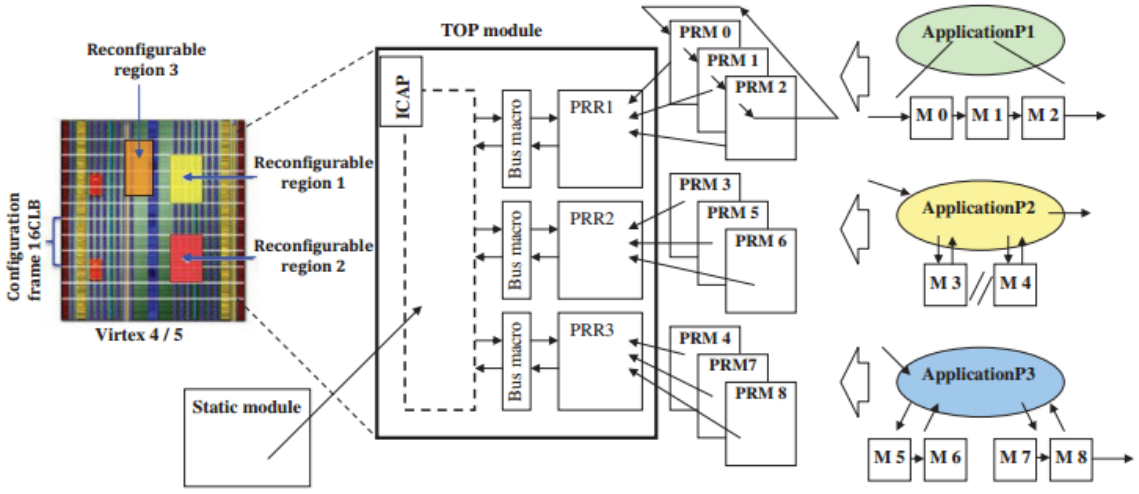

FIGURE *1.1 – Architecture basique d'un système reconfigurable dynamiquement et partiellement [BAHK+17]*

Actuellement, la plupart des FPGA récents introduits par Xilinx et Altera supportent la RDP, en rendant la technologie largement disponible.

2.1.1 DPR Xilinx

Xilinx a une longue avance historique, ils ont soutenu avec succès la reconfiguration partielle dans les FPGA depuis des années (depuis la série XC6200). Le flot DPR de Xilinx sépare la conception en deux parties (FIGURE 1.1) : une partie statique et une partie partiellement reconfigurables. Les fonctions implémentées dans la partie statique (modules statiques (SM)) ne changent pas et continuent à s'exécuter alors que la partie dynamique peut être changée en cours d'exécution.

La mise en œuvre d'une architecture reconfigurable dynamiquement et partiellement introduit plusieurs étapes dans le processus de conception. La FIGURE 1.2 présente le flot de conception global de la méthode basée sur les partitions reconfigurables(partition based).

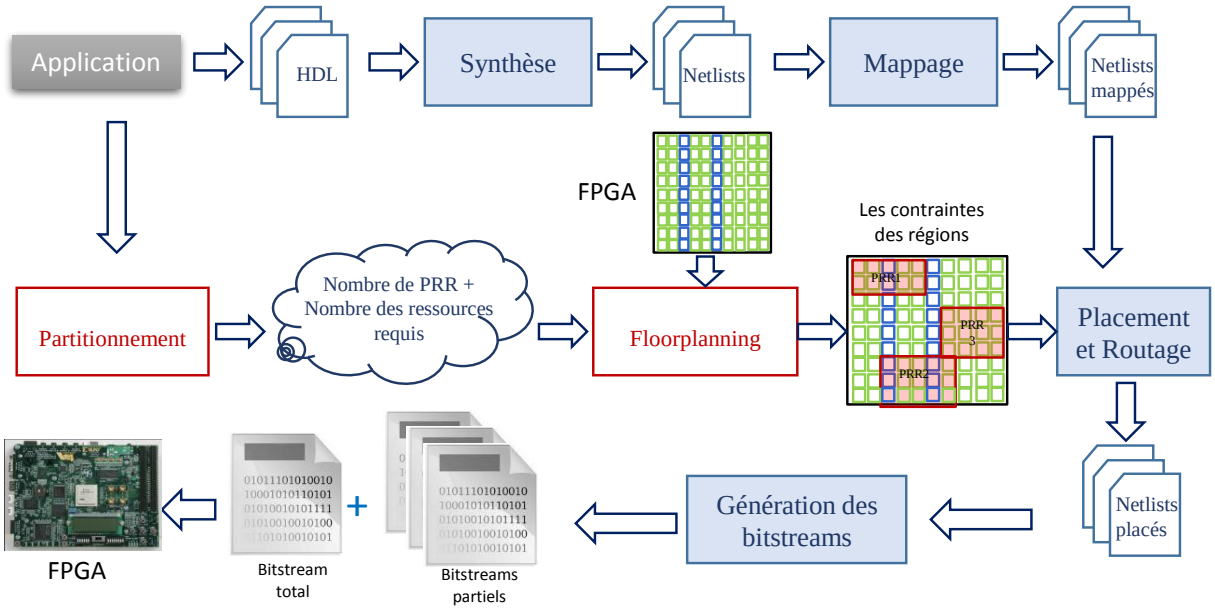

FIGURE 1.2 – *Flot de conception globale d'une architecture reconfigurable dynamiquement et partiellement*

Comme première étape, le concepteur doit fournir une description fonctionnelle de tous les éléments de l'application (les modules reconfigurables et la partie statique) en HDL. Les descriptions HDL sont ensuite synthétisées pour produire un ensemble de netlists. La synthèse traduit la description fonctionnelle du système en une structure matérielle composée d'un ensemble de portes logiques interconnectées qui permet au concepteur d'estimer la taille et le nombre des ressources de chaque tâche. Exploitant ces dernières informations, le concepteur décide comment partitionner les tâches dans des régions reconfigurables, de sorte que finalement chaque région reconfigurable est caractérisée par les besoins en ressources des tâches qui lui sont assignées.

Les contraintes de la zone reconfigurable dépendent de l'étape de floorplanning lors de processus de conception qui est effectué aussi manuellement par le concepteur. Les régions reconfigurables doivent couvrir suffisamment de ressources pour répondre aux besoins des tâches reconfigurées au cours du temps. Puisque la forme et la taille d'une zone reconfigurable ne peuvent pas être modifiées pendant l'exécution du système, la région doit être choisie suffisamment grande pour accueillir le nombre maximum de ressources utilisées par les tâches assignées.

Pour le placement et le routage, les contraintes des zones reconfigurables et les netlists des tâches reconfigurables sont les entrées de cette phase. Ici, chaque netlist est placé sur le FPGA et les signaux entre les différents composants sont acheminés. L'étape finale du flot de conception est la génération des bitstreams partiels associés aux régions reconfigurables et un bitstream complet pour la configuration de la partie statique et des tâches initialement présentes dans les régions reconfigurables. La communication entre les régions reconfigurables et la partie statique est garantie par l'insertion d'une logique proxy (pin

partition) qui est générée automatiquement par l'outil.

Une fois les bitstreams sont obtenus, ils peuvent être chargés sur le FPGA au moyen du port JTAG (Joint Test Action Group), ou éventuellement en utilisant ICAP (Internal Configuration Access Port) ou PCAP (Processor Configuration Access Port) pour les récentes architectures reconfigurables.

2.1.2 DPR Altera

En plus de Xilinx, Altera (comme l'un des principaux fournisseurs de FPGA) a annoncé le support de la reconfiguration partielle dans la plupart de ses nouveaux périphériques FPGA et outils de conception. Selon [Bou11], le flot de conception DPR d'Altera est fondamentalement similaire au flolt de conception DPR de Xilinx. Le flot DPR d'Altera commence par une conception de base contenant une région statique et au moins une région reconfigurable. Comme pour Xilinx, un certain nombre de modules reconfigurables (appelés «personas») peuvent être affectés à chaque région reconfigurable. En ayant des révisions différentes de la conception de base, chacune contenant un ensemble différent de personas, l'outil logiciel Quartus peut générer les bitstreams partiels de la conception [Bou11].

La première étape menée par l'outil Quartus pour générer les bitstreams partiels dans une conception DPR consiste à compiler toutes les révisions de la conception pour générer les fichiers masqués d'objet SRAM (MSF) et les fichiers d'objets SRAM (SOF) (voir FIGURE 1.3). Dans chaque révision, un fichier MSF et un fichier SOF sont créés pour chaque persona. Ces deux fichiers sont utilisés par l'outil Quartus pour générer un fichier partiellement masqué d'objet SRAM (PMSF) pour chaque module avant de générer les fichiers bitstreams partiels.

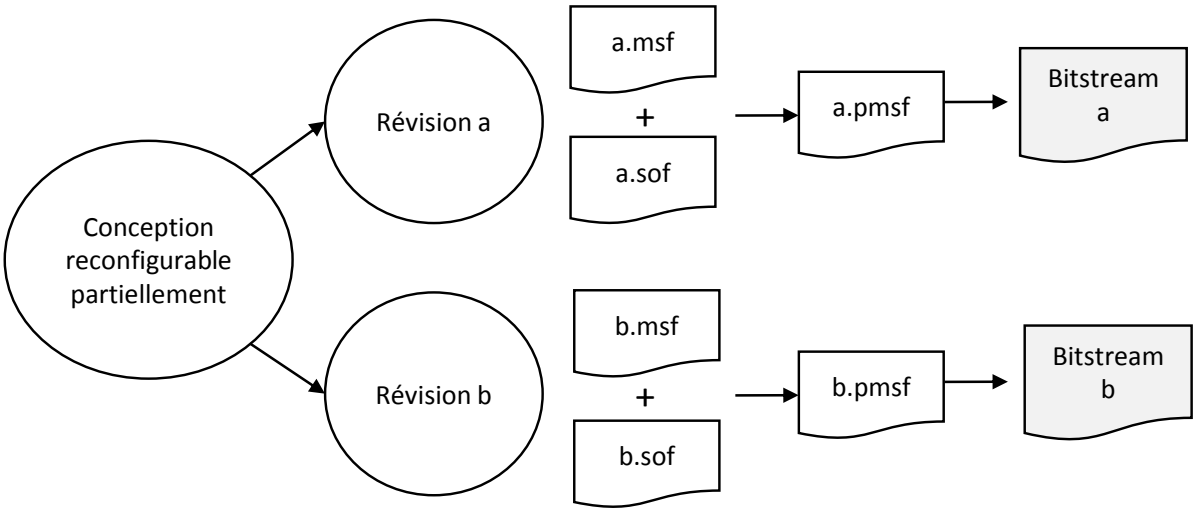

FIGURE *1.3 – Flot de conception DPR simplifié d'ALTERA*

2.2 Travaux traitants les problèmes de gestion de la RDP

Bien que, la RDP fournit plusieurs avantages aux systèmes tels que l'amélioration de flexibilité, la diminution des ressources utilisées et la réduction de la consommation de puissance, l'introduction des systèmes embarqués basés sur la technique RDP a posé plusieurs problèmes au niveau de la conception qui reste complexe et éreintant. Dans l'absence des outils qui supportent des méthodologies de conceptions adéquates, un grand nombre de travaux proposent l'ajout de quelques étapes au processus de conception selon différents aspects pour assurer une utilisation efficace de la RDP. Quelques exemples d'étapes ajoutées dans la conception de la RDP et traitées dans la littérature sont présentées dans cette section.

2.2.1 Algorithmes de Placement

L'objectif des algorithmes de placement proposés dans la littérature est de chercher un meilleur emplacement des tâches d'une application sur un matériel reconfigurable afin d'optimiser les ressources utilisées et minimiser la fragmentation de la surface libre sur le FPGA. Les principaux algorithmes de placement traités peuvent être divisés en deux catégories : placement en ligne et placement hors ligne.

Les algorithmes de placement en ligne : Le système doit chercher en cours de fonctionnement, les emplacements disponibles pour les nouvelles tâches. Le plus ancien algorithme de placement en ligne était proposé par Bazargan et *al* [BKS+00]. Cet algorithme est basé sur la notion de fusionner des rectangles vides pour placer une nouvelle tâche arrivée. Bazargan et *al* ont proposé deux techniques de placement. KAMER(Keeping All Maximal Empty Rectangles) est une technique basée sur la recherche des maximums des rectangles vides et qui ne sont pas nécessairement disjoints. La deuxième méthode est KNOER (Keeping Non-overlapping Empty Rectangles). Cette méthode consiste à garder les rectangles vides et qui ne se chevauchent pas.

L'une des méthodes les plus utilisées et les plus efficaces pour le placement en ligne est la méthode KAMER (Keeping All Maximum Empty Rectangles) [ABT04] [ABBT04]. Le principe de l'algorithme KAMER est de partitionner l'espace libre en gardant un ensemble de rectangles vides appelé MER (Maximal Empty Rectangle). La nouvelle tâche arrivée est placée sur un rectangle libre au coin gauche de la surface matérielle qui peut la contenir. Cette méthode consiste à assurer une gestion de la surface occupée au lieu de la surface libre.

Un nouvel algorithme est proposé par Marconi et *al* [MLBG08], nommé Intelligent Merging (IM). Cet algorithme est basé sur une amélioration de l'algorithme de Barzargan [BKS+00]. IM combine dynamiquement trois techniques pour gérer des ressources libres, selon la taille des tâches. La technique MON (Merging Only if Needed) consiste à combiner

les rectangles vides uniquement quand il n'y a pas de places possibles pour la tâche nouvellement arrivée. La technique PM (Partial Merging) permet de fusionner uniquement un sous-ensemble de rectangles libres s'il n'y a pas d'espace libre pour placer la nouvelle tâche. La technique DC (Direct Combine) permet de fusionner tous les rectangles vides quand la nouvelle tâche est assez grande et il n'y a pas de places possibles pour la contenir.

Le principe des algorithmes en ligne est que les emplacements des tâches ne sont pas connues à l'avance. En effet, tous ces algorithmes ne tiennent pas compte des positions prédéfinies des tâches. D'où, la reconfiguration continue des tâches dans des différents emplacements engendrent un grand nombre de bistreams partiels et une fragmentation importante de la surface reconfigurable.

Les algorithmes de placement hors ligne : Dans ce cas, l'emplacement des tâches est défini avant l'exécution de l'application. Dans [BKS+00], les auteurs ont proposé aussi un algorithme de placement hors ligne. Il est basé sur deux méthodes de recherche : l'approche de recherche tabou et le recuit simulé pour une meilleure gestion de l'espace libre. De plus, des méthodes heuristiques ont été utilisées pour proposer des algorithmes de placement hors ligne [TSMM06] [DS05]. Ces algorithmes visent à minimiser la fragmentation de la surface, le temps d'exécution et la surface utilisée. Lodi et *al* [LMM02] considèrent le problème du placement comme un problème bidimensionnel 2d. Ils proposent de nombreuses stratégies heuristiques off-line telles que Next-Fit, First-Fit et Best-Fit. Cependant, tous ces algorithmes ne sont applicables que sur des surfaces homogènes. En plus, ils ont besoin de beaucoup du temps pour la gestion de l'espace libre. Pour surmonter cette limitation, Belaid et *al.* [BMB10] ont proposé une approche qui tient compte de l'hétérogénéité des FPGA. Cette approche est basée sur la méthode Branch et Bound en permettant une gestion des ressources matérielles utilisées.

2.2.2 Partitionnement

La phase de partitionnement consiste à déterminer un nombre de régions reconfigurables PRR pour les utiliser dans la conception et à y attribuer des tâches reconfigurables spécifiques.

Les outils actuels des fournisseurs nécessitent que toutes les régions soient fixées et déterminées manuellement par le concepteur avant la génération de n'importe quel bitstream partiel. En outre chaque combinaison région-module est également de la responsabilité du concepteur. Dans ce cas, le concepteur doit tenir compte de la dépendance entre les tâches et leurs priorités et de la communication entre les différentes tâches pour s'adapter aux différentes configurations permises par l'application.

Dans la littérature, plusieurs travaux proposent un partitionnement automatisé qui tente d'ordonner un graphe de tâches dépendantes sur un nombre fixe de régions afin de réduire le temps d'exécution. Ils supposent que les régions reconfigurables dans FPGA sont utilisées

d'une manière similaire qu'un système multiprocesseur, c'est-à-dire chaque tâche alloue une région indépendante.

Dans [GV00], les auteurs décrivent un système reconfigurable seulement avec deux régions reconfigurables pour gérer l'exécution et le pipeline des reconfigurations. Ils partent d'une spécification en C/VHDL séquentielle traduite ensuite sous forme d'un graphe de tâches qui est partitionné de manière que la reconfiguration et l'exécution peuvent être réalisées simultanément. L'objectif de la méthode proposée dans [RMA+09] était de minimiser le temps de reconfiguration en basant sur une analyse graphique des communications entre les tâches. L'algorithme consiste à regrouper les modules qui nécessitent une reconfiguration simultanée dans la même région reconfigurable. Cependant, le nombre de régions reconfigurables doit être déterminé par le concepteur. Par ailleurs, dans [JBGR09], les auteurs supposent que le nombre et la taille des PRR sont fixés par le concepteur. Ils ont proposé un algorithme basé sur le recuit simulé. Cet algorithme assure l'affectation des tâches reconfigurables aux PRRs tout en minimisant le temps de reconfiguration. Le nombre des modules reconfigurables requis est supposée égale au nombre des PRRS et si une région est inoccupée, un module vide lui y est affecté. Dans cette approche, les ressources sont supposées homogènes. Cependant, les FPGA modernes ont une architecture hétérogène avec des DSP distribués et des BlockRAM, ce qui contrevient l'hypothèse de ressource homogène.

La plupart des travaux existants ne traitent pas le problème de partitionnement d'une manière qui prend en compte les temps d'exécution de PR et ne tient pas compte de l'hétérogénéité des nouvelles FPGAs . Ils supposent généralement un graphe de tâche comme entrée où chaque tâche s'exécute indépendamment dans une région.

En 2013, Vipin et *al.* [VF13] ont proposé un algorithme basé sur la technique ILP (Integer Linear Programming) pour automatiser le partitionnement dans une architecture hétérogène des FPGA modernes. Cette technique est une approche mathématique consistant à trouver une solution optimale qui satisfait des contraintes imposées. Le séquencement de reconfiguration est supposé inconnue à l'avance, la seule entrée que le concepteur doit fournir est la description de la conception et le type de FPGA. L'approche proposée cherche un partitionnement optimal tout en assurant un temps de reconfiguration minimal.

2.2.3 Communication entre les modules reconfigurables

La communication est un aspect très important pour les performances globales d'une architecture reconfigurable. Cette communication assure le bon acheminement des données entre les différents modules reconfigurables d'un système. Le besoin de disposer de mécanismes d'interconnexions efficaces permettant une adaptabilité et une flexibilité de la structure matérielle constituant un système reconfigurable dynamiquement. Plusieurs types de communication sont employés dans les architectures reconfigurables à savoir : le bus , le crossbar et les réseaux-sur-puce.

Communication par Bus Un bus partagé est composé de lignes de données, de contrôle et d'un module d'arbitrage. Dans ce type d'interconnexion, un même bus est partagé par plusieurs modules. Il est caractérisé par une mise en œuvre simple permettant de répondre aux besoins de conception de systèmes sur puce. Néanmoins, cette structure de communication présente une forte limitation de performances : taux de transfert diminue rapidement lorsque le nombre de modules inter-connectés augmente. D'autre part, elle limite la flexibilité et l'extensibilité de la structure matérielle du SOPC puisqu'elle introduit des contraintes d'intégration et de placement des modules du système sur la même puce.

Dans [WP04], Herbert Walder et *al.* ont proposé un système de communication basé sur un Bus (FIGURE 1.4). Ce bus est composé de fils de connexion, de bus macros, de deux arbitres de bus (BARL, BARR) situés respectivement à droite et à gauche de l'architecture et un bus de contrôle d'accès (BAC). Il est divisé en un bus gauche, un bus droit et un bus de contrôle. Le bus de gauche sert des accès de communication pour la lecture et l'écriture aux éléments du système d'exploitation situés dans la partie gauche et le bus droit accède aux éléments d'OS dans le partie droite. Le bus de gauche et de droit sont arbitrés respectivement par le BARL et le BARR. Le bus de contrôle gère les signaux de demande/délivrance en connectant chaque BAC aux deux arbitres.

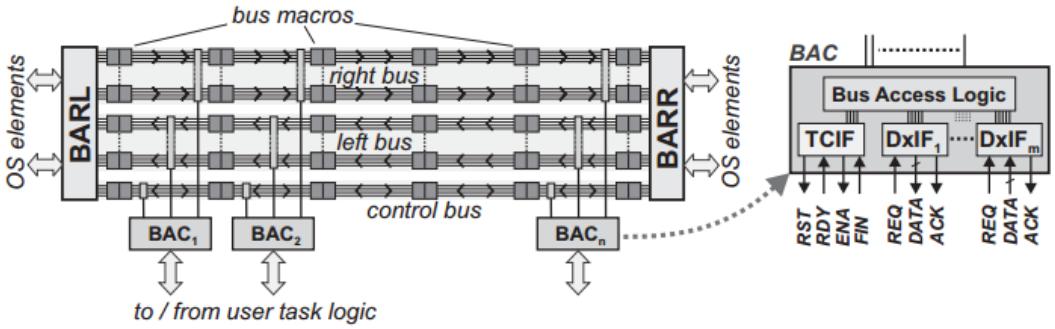

FIGURE *1.4 – Structure et éléments de bus proposé dans [WP04]*

Dirk Koch et *al.* ont proposé un bus reconfigurable nommé ReCoBus [KHT08]. Le ReCoBus est un bus sur puce qui convient à l'intégration dynamique de modules matériels reconfigurables dans un FPGA. Il assure aux modules une connexion à différentes interfaces avec un nombre variable d'emplacements. Il se compose d'un Hard macro avec une structure régulière permettant une reconfiguration dynamique partielle au moment de l'exécution. La taille du bus peut être augmentée en fonction du nombre de slots que le module reconfigurable utilise. L'architecture basée sur ReCoBus doit obligatoirement être réalisée avec l'outil ReCoBus-Builder [KBT08].

Réseau sur puce Les réseaux sur puce constituent une approche globale d'interconnexion pour les SOC. Une haute performance de ces interconnexions structurées par rapport aux architectures de bus traditionnels ont été d'un grand intérêt pour la communauté de

recherche. Dans [BMK+04, JwB09, JTBW09, PAKM08], les auteurs proposent un réseau sur puce (NoC) comme un moyen de communication entre les différents noyaux (IP) constituant un système. Il est spécialement conçu pour répondre aux besoins des architectures reconfigurables. Les NOC introduisent la notion de transfert parallèle, d'où une plus importante bande passante. Même si un NOC peut être favorisé pour des raisons de flexibilité et de scalabilité par rapport aux structures assez simples comme les bus partagés, les frais généraux qui sont introduits dans sa mise en œuvre à la fois pour la zone de la logique et de retard de traitement reste l'inconvénient majeur de cette approche.

Dans [PAKM08], les auteurs proposent un réseau sur puce reconfigurable nommé CoNoChi, spécialement conçu pour être utilisé dans les systèmes sur puce reconfigurables. Ce réseau est basé sur une topologie adaptative de types NoC. La structure de base de réseau reconfigurable proposé est présentée dans la FIGURE 1.5. Lors de l'exécution, CoNoChi prend en charge l'adaptation de la topologie du réseau en fournissant des mécanismes pour ajouter ou supprimer des commutateurs du réseau pendant l'exécution sans arrêter ou bloquer le réseau. Selon le besoin de système, quatre types de liens de communication peuvent être mappés sur les partitions reconfigurables : des liens de communication horizontale (Type H) et communication verticale (Type V), des commutateurs (Type S), et des liens spécifiques à l'application (Type 0). Les liaisons de communication sont basées sur des bus macros.

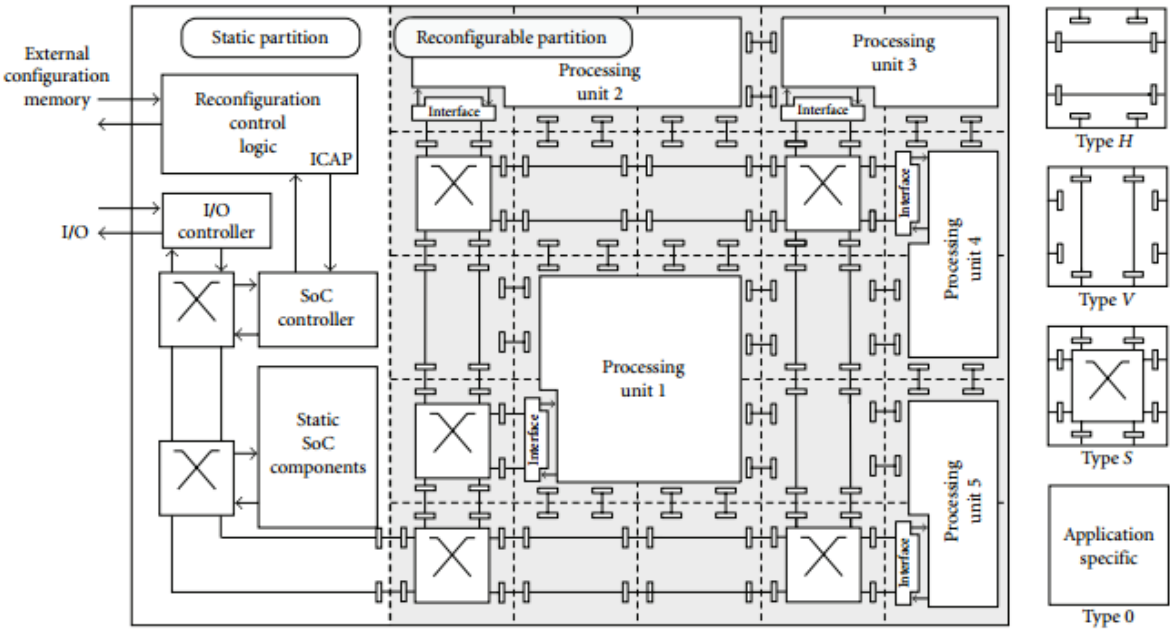

FIGURE 1.5 – – *Structure de base d'une architecture reconfigurable à base de CoNoChi[PAKM08]*

2.3 Adaptation des régions reconfigurables aux tâches matérielles de tailles variables

L'adaptation d'un système reconfigurable est la capacité de maintenir sa fonction principale tout en modifiant les régions reconfigurables par différentes tâches reconfigurables. Dans la majorité des systèmes reconfigurables, la taille d'une région reconfigurable est fixe. Dans le cas où les différentes tâches matérielles implémentées ont des tailles différentes, la région reconfigurable devra avoir une taille suffisamment grande pour être capable d'accueillir la plus grande des tâches. L'utilisation des régions de taille fixe implique une perte des ressources non utilisées et une dégradation en termes de l'efficacité en surface.

Dans [Mar12], les auteurs ont proposé une nouvelle méthodologie d'adaptation des régions PR aux tâches matérielles de tailles variables. Cette méthodologie consiste à partitionner les régions reconfigurables en certain nombre des partitions reconfigurables. La taille des partitions est définie pour allouer des petites tâches matérielles. Pour le placement de chaque module reconfigurable, un ensemble de partitions sont fusionnées et définies comme une nouvelle partition.

Le flot de conception proposé par l'équipe MAE est nommé « Variable Partition Based Partial Reconfiguration (VPBPR) ». Il est basé sur une modification du flot de conception proposé par Xilinx pour rendre possible le placement des tâches matérielles de tailles variables sur une architecture reconfigurable. Il introduit une couche intermédiaire pour le partitionnement de la région reconfigurable du FPGA [Mar12]. Une architecture est composée d'une conception de base qui contient tous les éléments statiques du système pour assurer son fonctionnement et de plusieurs autres sous conceptions qui définissent la partie reconfigurable. Pour chaque nouvelle conception, un nouveau partitionnement souhaité avec des partitions PR de tailles différentes est défini.

Dans cette thèse, nous allons exploiter cette nouvelle méthodologie d'adaptation et étendre son utilisation dans le cas de relocation. Par conséquent, des nouvelles contraintes, précautions et étapes seront rajoutées afin d'augmenter l'efficacité en surface dans le cas où les tâches matérielles sont de tailles différentes et doivent être allouées et ré-allouées dans différents emplacements.

3 Relocation des bitstreams partiels

Grâce à la reconfiguration dynamique partielle, nous pouvons supprimer certaines tâches matérielles du FPGA et charger de nouvelles tâches à leurs places. L'implémentation de ces nouvelles tâches matérielles sur des régions reconfigurables permet d'augmenter l'adaptabilité et la flexibilité du système. Néanmoins, les flots de conception des architectures reconfigurables fournis par les fournisseurs, oblige le concepteur à associer chaque PRM à une PRR cible. Ainsi, pour exécuter 'M' module sur 'N' PRRs $\{PRR1, PRR2, ...PRRN\}$

, il est nécessaire de générer un ensemble de 'N * M' bitstreams différents, même si une même tâche est mappée dans plusieurs régions reconfigurables. Le stockage de plusieurs bitstreams partiels n'est pas une option envisageable, car les besoins de stockage de circuit augmentent linéairement avec le nombre de PRR et le nombre des modules partiels reconfigurables PRM. Pour résoudre ce problème, plusieurs travaux dans la littérature ont été réalisés en utilisant la technique de la relocation des bitstreams partiels. Le mécanisme de relocation vise à offrir la possibilité d'exécuter sur plusieurs PRR un module synthétisé pour une seule PRR.

Par ailleur, plusieurs domaines d'applications ont été développés dans la littérature qui exploitent la capacité de la relocation. En fait, la tolérance des défaillances dans un système reconfigurable opérationnel entraine à une perte de ressources. Pour optimiser le placement des tâches, plusieurs travaux dans la littérature ont proposé d'exploiter la relocation des bitstreams partiels [MBW11, MBWM07]. Les travaux présentés dans [Még12] proposent une approche pour la relocation dynamique des tâches lors de défaillances matérielles sur une architecture reconfigurable. Cette approche a pour but d'apporter une efficace tolérance aux défaillances en permettant une flexibilité en ligne lors de l'exécution du système, ainsi de trouver à chaque défaillance un emplacement le plus optimisé.

Pour les applications spatiales, les FPGA basés sur le SRAM tels que les FPGA Xilinx et Altera sont très vulnérables aux SEU (Single Event Upset), ce qui peut entraîner une corruption dans la mémoire de configuration et des dommages graves au système. La relocation a été proposée comme une solution pour atténuer les effets SEU sur les FPGA basés sur SRAM [BQS07, HSWK09, SBC16], puisqu'elle fournit un chemin auxiliaire à la mémoire de configuration. Dans [BQS07], les auteurs partitionnent le FPGA en un certain nombre de régions afin d'isoler les erreurs SEU, puis d'appliquer la duplication avec une comparaison entre les régions pour assurer un calcul correct. Une fois qu'une erreur est détectée, cette région est reconfigurée. Une autre méthode simple pour surmonter les erreurs SEU en utilisant la reconfiguration dynamique partielle est proposée par [HSWK09]. Ici, les données de configuration sont stockées dans une mémoire et un contrôleur de configuration configure des portions du FPGA en utilisant les données stockées dans cette mémoire. Dans une méthode plus avancée, le contrôleur de configuration lit les données du FPGA et détecte la présence d'une erreur. Ce contrôleur écrit des données de configuration uniquement si une erreur est présente.

3.1 Principe de la relocation

Grâce à la relocation, les tâches matérielles peuvent être allouées et ré-allouées en fonction de la demande et de l'exigence du système. Pour offrir une vraie flexibilité à l'usage, il faut évidemment que les bitstreams ne soient pas associés à une seule zone reconfigurable. C'est pourquoi, le mécanisme de relocation a pour objectif de configurer plusieurs zones reconfigurables par un seul bitstream généré sur une seule zone.

La FIGURE 1.6 illustre un exemple d'application qui nécessite la configuration de plusieurs PRM dans différents PRR. Cette application contient 8 tâches différentes qui sont allouées dans 5 régions reconfigurables. D'où la nécessité de générer un bitstream pour chaque PRM associé à chaque PRR. Le nombre de bitstreams générés dans cet exemple est égale à 17. Grâce à la technique de relocation, nous pouvons générer 8 bitstreams dans cet exemple ce qui permet d'optimiser la mémoire de stockage.

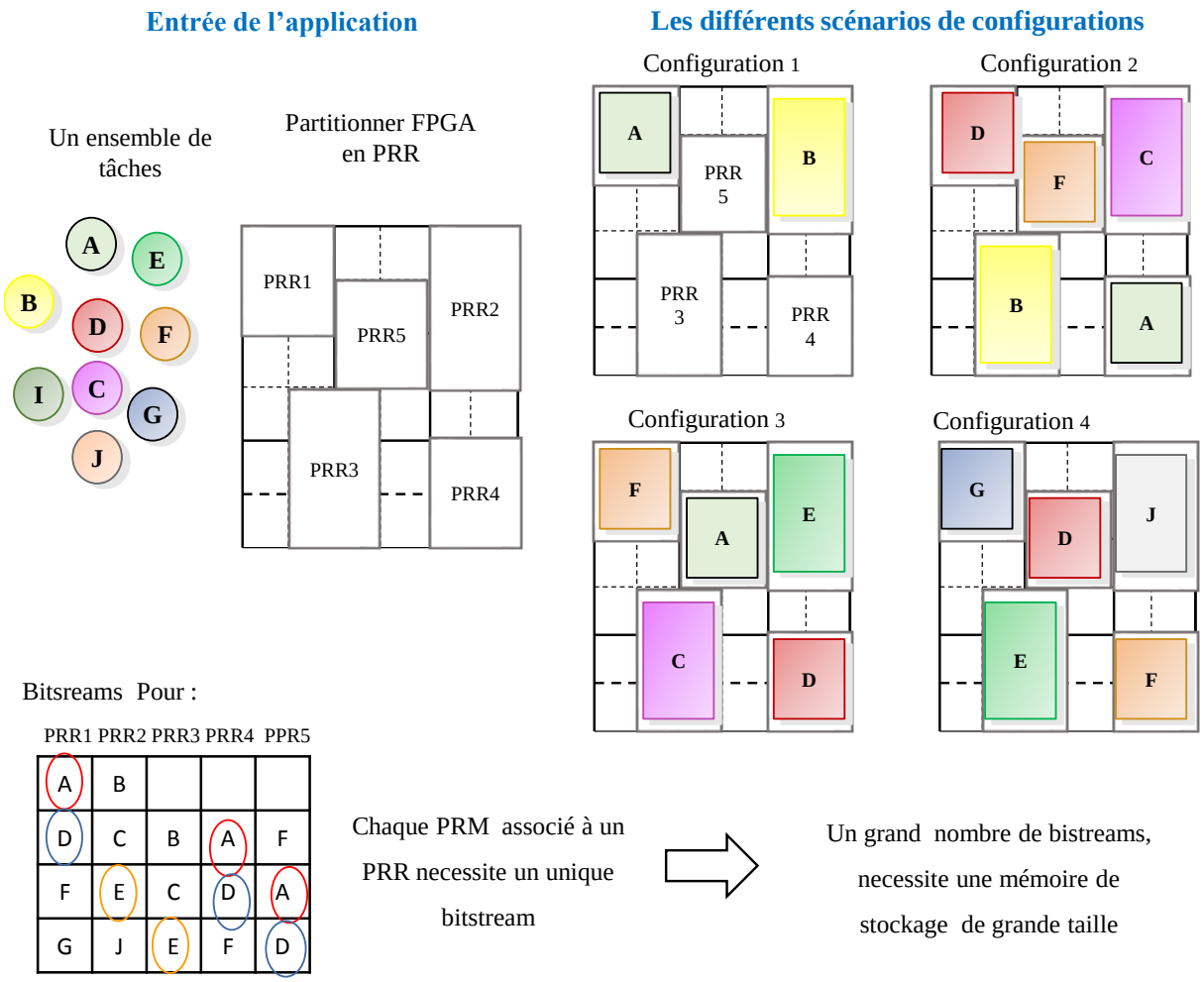

FIGURE *1.6 – Présentation de la problématique de la relocation*

La technique de la relocation aborde le problème d'allouer dynamiquement plusieurs PRM $\{PRM1, PRM2, ..., PRMN\}$ dans plusieurs PRR $\{PRR1, PRR2, ..., PRRN\}$. La relocation des bitsreams partiels, est une technique qui permet d'effectuer la relecture du bitstream d'une tâche matérielle dans une région reconfigurable afin d'en effectuer sa relocation sur une autre région reconfigurable [SKD09, FGRG09]. Elle offre la possibilité d'exécuter sur plusieurs régions reconfigurables un seul bitstream généré pour une seule région. Cette technique est généralement conçue pour réduire la taille de la mémoire nécessaire pour stocker les bitstreams partiels lorsqu'un même module reconfigurable est instancié dans

plusieurs emplacements.

Un bitstream est dit relogeable, s'il contient à la fois les informations sur l'emplacement de la zone reconfigurable qui peuvent être modifiées et aussi les informations nécessaires à la configuration d'une tâche. Ces informations permettent au processeur de déployer la tâche sur différentes autres régions reconfigurables.

La technique de relocation offre plusieurs avantages aux architectures reconfigurables tels que :

— Réduire la quantité de mémoire utilisée pour stocker les bitstreams partiels
— Réaliser une exécution préemptive
— Affecter le placement des bitstreams en cours d'exécution

3.2 Travaux existants

Le manque de capacités des outils et des flots de Xilinx pour supporter une telle caractéristique a conduit au développement de plusieurs outils avancés et des techniques de configuration dans la littérature pour contourner les limites du flot DPR de Xilinx et réaliser la relocation des bitstreams partiels.

En 2001, Horta et *al.* ont proposé l'un des premiers outils développés nommé PARBIT (PARtial BItfile Transformer) [HL01]. Cet outil permet de manipuler un bitstream automatiquement via un processeur externe. Il assure la relocation des bitstreams partiels sur les plus anciennes FPGAs Virtex et Virtex-E. Parbit génère un bitstream partiel à partir d'un bitstream total d'un FPGA reconfigurable dynamiquement par colonne. Pour générer le fichier bitstream partiel, PARBIT lit les frames de configuration à partir du bitstream totale d'origine et copie uniquement les frames de configuration liées à la zone définie par l'utilisateur au bitstream partiel. Il permet aussi d'extraire les données de configuration à partir d'un bitstream partial et les copies dans un nouveau flux binaire partial relatif à un nouveau PRR. Cependant, le processus de la manipulation et de la génération des bitstreams partiels se fait via un processeur externe ce qui prend beaucoup de temps lors de ses mises en œuvre. Le temps de reconfiguration est l'un des principales préoccupations lors de l'utilisation de la reconfiguration dynamique FPGA qui peut avoir un impact négatif sur les performances du système.

Afin de réduire le temps de génération d'un bitstream, Kalte et *al.* ont développé un filtre matériel nommé REPLICA [KLPR05] exploité pour la relocation. Il est implémenté matériellement sur FPGA. Ce filtre réalise la manipulation nécessaire de bitstream afin de modifier son emplacement lors du processus de téléchargement à partir d'une mémoire hors puce. REPLICA supporte également la relocation de bitstream dans un FPGA reconfigurable. REPLICA analyse la structure des deux bitsstreams partiels, et il détermine les frames de configuration différentes entre eux. Après, seuls les frames déterminées, ainsi l'instruction de vérification CRC (Cyclic Redundancy Check) et MJA (major column address Word) seront changées. Donc, REPLICA est capable d'effectuer la relocation dynamique

en analysant chaque mot du bitstream qui est chargé au port de configuration. Si le mot MJA est détecté, REPLICA le change par le nouveau mot d'adressage qui est calculé par un module dédié à l'intérieur du filtre REPLICA. De plus, un autre module dédié est utilisé pour mettre à jour le mot du CRC. Ces modules dédiés sont présents de telle sorte que si un mot doit être changé, le changement est effectué par un multiplexage des différentes sorties du système, afin de ne pas introduire une surcharge au processus. .

PARBIT et REPLICA supportent la relocation des bitstreams partiels dans un FPGA reconfigurable par colonne. En 2004, H. Kaltd et *al.* [KLPR04] ont étudié l'effet de relocation 1D sur FPGA. Cette étude montre une augmentation allant jusqu'à 96 % du temps de configuration et du consommation d'énergie pour les petites tâches matérielles.

Pour surmonter ces limitations, en 2006, BiRF (Bitstream Reconfiguration Filter) a été proposé par Ferrandi et *al.* dans [FNM+06]. Il est considéré comme une ré-implémentation et une extension de REPLICA où il est également basé sur une approche de filtre. Cependant, BIRF supporte la relocation par région et offre des meilleures performances en terme des ressources matérielles utilisées, temps de configuration et consommation d'énergie. L'approche proposée a également été validée dans [CMN+09] sur plusieurs FPGA : Virtex II Pro, Virtex-4 et Virtex-5.

Dans [CFM+07], les auteurs comparent BIRF avec une autre solution logicielle BAnMat. Cette dernière s'exécute dans le processeur interne du FPGA. Les résultats expérimentaux montrent que le débit obtenu par BanMaT est supérieure par 22 % à celui obtenu à l'aide de BIRF. De plus, en utilisant l'approche de BanMaT, la relocation est obtenue sans aucun coût en termes de ressources matérielles de FPGA. Cependant, BIRF peut être utilisé dans une architecture reconfigurable sans exiger la présence d'un processeur dans le système.

En 2009, Arvind et al ont proposé un nouvel algorithme de relocation nommé PRR-PRR [SKD09]. Cet algorithme est mis en œuvre en logiciel et en matériel. Une architecture matérielle, appelée ARC(Accelerated Relocation Circuit), est conçue pour la relocation des modules reconfigurables. L'idée de base de cet algorithme est de modifier le bitstream frame par frame. Autrement dit, chaque fois qu'il calcule le nouveau FAR (Frame Adress Register) et le modifie directement dans le bitstream. L'ARC se compose de trois composants principaux (1) Générateur de l'adresse de frame (FA) (2) Relocator et (3) ICAP Wrapper. Les emplacements des PRR source et de destination sont représentées à l'aide de deux mots de taille 16 bits (SrcPRR et Destrrr).

— Générateur de l'adresse de frame (FA) : Il utilise les informations de SrcPRR et DestrRR pour générer la séquence complète des adresses de frame pour les PRRs source et destination.
— Relocator : En se basant sur les valeurs de FASt et FADest, le module Relocator lit une seule frame à partir de la source PRR et écrit la frame dans la PRR de destination.
— ICAP Wrapper : Il agit comme une interface simple entre le module Relocator et les

ports ICAP (données et contrôle). Il décode les informations envoyées par Relocator (ICAP MODE) pour générer les signaux de contrôle pour l'ICAP.

En 2013, Touiza et al. [TORB+13] ont proposé une méthode similaire aux filtres (BIRF et REPLICA) destinée à reloger les bitsreams partiels. La différence réside dans le fait que leur outil, OORBIT effectue certaines étapes requises pour assurer la manipulation des bitstreams partiels en deux modes : hors et en ligne. L'objectif du mode hors ligne est d'extraire les informations sur l'emplacement d'origine dans le bitstream et calculer les nouvelles FAR et CRC du nouvel emplacement souhaité. D'autre part, l'objectif de la partie du logiciel (mode en ligne) est de modifier les anciennes informations de l'emplacement dans le flux de bitstream par le nouveau calculé.

Comme la plupart des travaux dans la littérature, les calculs qui dépend un long temps sont effectués en mode hors ligne, la seule tâche requise en ligne avant la relocation est de remplacer certains champs dans le bitstream. Par conséquent, l'implémentation matérielle d'OORBIT annonce une accélération allant jusqu'à 8000 fois par rapport à BiRF.

En 2016, Rettkowski et al [RFG16], ont proposé une nouvelle approche pour reloger des bitstreams partiels dans d'autres partitions dans le FPGA à l'aide de l'outil Vivado Design Suite de Xilinx. Par ailleurs, cette approche est automatisée par un nouvel outil nommé RePaBit (Relocatable Partial Bitstreams). RePaBit peut déplacer un bitstream dans les deux sens vertical et horizontal. Afin d'éviter le chevauchement des chemins de routage, RePaBit utilise le flot de reconfiguration partielle et le flot de conception d'isolement de Xilinx [Hal]. Il est évalué sur le ZedBoard avec deux designs contenant une logique telle qu'un MicroBlaze.

Toutes les approches de relocation proposées précédemment ont pour point commun de reloger un bitstream sur des régions reconfigurables identiques. Les FPGA actuels intègrent des blocs optimisés pour la mémorisation, les BRAM et des blocs optimisés pour le calcul, les DSP ainsi que des processeurs. Par exemple, certains types des FPGA possèdent deux cœurs de processeurs au centre de la zone reconfigurable. Ces différents blocs sont répartis sur l'ensemble de la zone reconfigurable afin que l'ensemble des CLB puisse utiliser les blocs relativement proches. L'introduction de ressources du type différent à la zone reconfigurable lui fournit un caractère hétérogène. Il s'avère donc nécessaire que la technique de la relocation soit en mesure de supporter cette hétérogénéité. Quelque travaux dans la littérature aborde le problème de reloger un bitstream partiel sur des régions reconfigurables hétérogènes.

En 2007, Becker et al. dans [BLC07], proposent une méthode qui assure la relocation sur des partitions non entièrement identiques. L'hétérogénéité dans ce travail réside sur le type des fonctions fixe (BRAM, DSP) situé entre les CLB.

Dans [HST14], Huriaux et al ont proposé une nouvelle approche sur les architectures de FPGA supportant la relocation des bitstreams partiels sur des zones hétérogènes. Cette approche est nommée Virtual BitStream. L'approche proposée dans [HST14] a la capacité

de glisser une tâche donnée horizontalement comme illustré sur la FIGURE 1.7. Seule la position horizontale des zones reconfigurables sont hétérogènes. Le flot de conception de la méthode Virtual Bitstream a été effectué par l'outil Verilog-To-Routing (VTR). Ce flot est un open-source, il réalise les différents étapes d'un flot de conception classique de FPGA, de la description matérielle en Verilog comme entrée, jusqu'au le placement-routage de l'application sur une architecture virtuelle de FPGA.

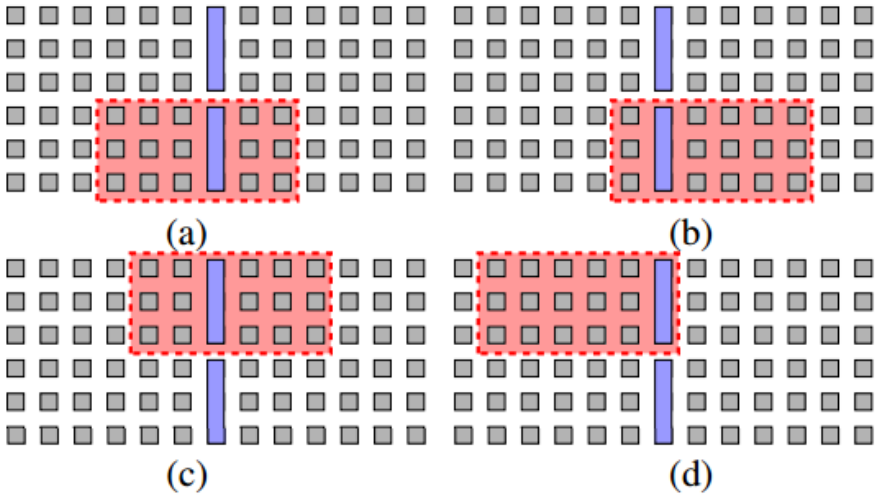

FIGURE *1.7 – Multiple possibilités de relocation pour le même bitstream partiel [HST14]*

Bien que plusieurs points ont été traité dans les travaux de relocation présentés dans la littérature tels que : la manipulation automatique des bitstreams par différents processeurs pour minimiser le temps de relocation et la répartition hétérogène des ressources matérielles dans les PRR, la relocation des PRM de tailles variables sur des zones de tailles fixes n'est plus abordé. Durant la phase de conception la taille de toutes les PRR est fixée pendant et ne peut pas être modifiée lors de l'exécution, ce qui peut entraîner une occupation inefficace des zones et un temps de reconfiguration plus élevé dans le cas des PRM de petites tailles.

3.3 Discussion

Le TABLEAU 1.1 donne une étude comparative entre les différents travaux présenté dans la littérature sur la relocation des bitstreams partiels.

Les solutions proposées dans la littérature pour répondre au problème de relocation ont été présentées dans la section précédente. Ces solutions assurent une optimisation en ressources de mémoire utilisé en fonction de plusieurs critères personnalisés. Néanmoins, la méthodologie de conception de cette technique est une opération complexe et nécessite fortement des connaissances approfondies sur le fonctionnement interne du FPGA. De plus le fabricant ne donne que très peu de détails sur la technologie de ces produits en raisons de propriété intellectuelle. Donc, il s'avère très délicat de développer ce genre de techniques.

La plupart des approches présentées précédemment abordent la technique de relocation avec moins de détails techniques de conception et en particulier le problème d'interface

TABLEAU *1.1 – Comparaison entre les différentes approches de relocation*

Référence	Interface	1D/2D	Tool	FPGA	PRRs
Parbit [HL01]	bus-macro	1D	ISE	Virtex Virtex-E	homogène
REPLICA [KLPR05]	bus-macro	1D	ISE	Virtex II/PRo	homogène
BIRF [CFM+07]	bus-macro	2D	ISE	Virtex-4 Virtex-5	homogène
PRR-PRR [SKD09]	bus-macro	2D	ISE	Virtex-4 Virtex-5	homogène
OORBIT [TORB+13]	bus-macro	2D	ISE	Virtex-6	homogène
RePaBit [RFG16]	bus-macro	2D	vivado design suite	ZYNQ	homogène
[BLC07]	bus-macro	2D	ISE	Virtex-4	hétérogène
Virtual Bit-Stream [HST14]	macro-cells	2D	Verilog-To-Routing (VTR)	virtuelle	hétérogène

de communication. De plus, tous ces techniques ont pour point commun de déplacer un bitstream sur des zones reconfigurables de tailles fixes et qui ne peuvent pas être changé pendant l'exécution du système. Cette limitation peut entrainer une occupation inefficace dans la zone reconfigurable lors d'allocation des tâches de tailles variables.

La relocation présente donc une solution pour les systèmes reconfigurables adaptatifs, mais cette solution doit être améliorée pour faciliter sa mise en œuvre et augmenter sa flexibilité. Il est donc nécessaire de proposer une méthodologie de conception détaillée qui fait face à la complexité technologique du FPGA, ainsi de résoudre le problème de communication. Ensuite, il est necessaire d'exploiter de nouvelles techniques pour contourner le problème de la relocation de tâches de tailles variables afin d'améliorer l'efficacité d'utilisation en surface du système.

Bien que la gestion d'utilisation de la zone reconfigurable est l'un des principaux facteurs pour améliorer l'efficacité du système, le choix de subdivision et de répartition des zones reconfigurables sur FPGA d'une façon optimale est aussi un facteur majeur qui conduit à une optimisation du temps de reconfiguration et augmentation des performances du système. Dans ce contexte, la section suivante va présenter les différents travaux traités dans la littérature qui aborde le problème du floorplanning.

4 Algorithmes de floorplanning

Pour les architectures reconfigurables, l'application est partitionnée en deux parties. Une première partie statique fixe est maintenue inchangée tout au long du processus de reconfiguration. Une seconde partie reconfigurable dont le contenu peut être changé dynamiquement au cours du fonctionnement. Cette deuxième partie est composée de plusieurs tâches matérielles qui pourront être implémentées sur différentes régions reconfigurables. Dans le processus de subdivision du FPGA en zones reconfigurables qui s'appelle floorplanning.

Dans la littérature, plusieurs travaux se portaient sur l'étude de l'optimisation des ressources matérielles dans le FPGA. Différents algorithmes de floorplanning ont été développés dans le but d'avoir une subdivision optimale du FPGA.

4.1 Travaux existants

Dans les premières travaux de recherches, les auteurs traitent dans leurs algorithmes de floorplanning l'aspect du placement statique dans le FPGA sans tenir compte de sa capacité de reconfiguration [CW06], [FM06]. Ces algorithmes sont basés sur l'approche du recuit simulé (SA :simulated-annealing) afin de trouver des solutions optimales de floorplanning. Leurs formulations peuvent être appliquées à des FPGA hétérogènes, mais le floorplan résultant peut contenir des formes irrégulières, qui ne sont pas conformes aux exigences des régions reconfigurables.

Dans [YYC07], les auteurs ont introduit une des plus importantes contributions dans le domaine de la reconfiguration dynamique. L'algorithme est basé sur une extension de la représentation TCG (Transitive Closure subGraph) [LC05] pour faire face en dimensionnement temporel. Il prend en compte le processus de reconfiguration partielle et la contrainte du domaine d'horloge (régions d'horloges). Cependant, les types de ressources considérés par cette approche sont limités aux blocs logiques (CLB) alors que les autres ressources sont ignorées.

Pour surmonter ces limitations, d'autres travaux tels que [BSSK11], [BMS11] ont pris en compte de l'hétérogénéité des types de ressources (CLB, BRAM et DSP) mais seulement avec une distribution homogène dans le FPGA. Dans ces propositions, les auteurs ont supposé que le FPGA est un ensemble des tuiles (ayant la même taille et contenant le même nombre de ressources pour chaque type de ressource : DSP, BRAM et CLB) répétées d'une façon uniforme. Bien que cette hypothèse soit valable pour les anciennes générations des FPGA, les plus récents FPGAs tels que la famille Virtex-5 de Xilinx, n'ont pas une telle architecture uniforme de tuiles.

Récemment, les recherches se sont essentiellement focalisées sur l'étude des algorithmes de floorplanning pour les architectures reconfigurables hétérogènes. Vipin et Fahmy [VF12] ont introduit un floorplanner nommé : Columnar kernel tessellation. Leur approche carac-

térise le FPGA en terme de tuile (tile). Chaque tuile contient un type et un nombre bien défini de ressources logiques et se compose par un ensemble des frames configurables. Par conséquent, les exigences des régions reconfigurables sont traduites en termes de besoins en tuiles. Les régions reconfigurables sont triées par ordre de priorité en fonction du type et du nombre des tuiles requis. En outre, elles sont placées d'une façon séquentielle à partir de celles qui utilisent des tuiles les plus rares telles que les DSP et les BRAM. Le floorplanning est effectué en fusionnant des tuiles adjacentes sur la même ligne pour former des zones nommées kernel. Ces zones regroupent un certain nombre de tuiles de différents types de ressources. Initialement, la zone est étendue verticalement par une seule frame. Ensuite, si la zone nécessite des tuiles de plus parmi celles les plus rares, la région est encore étendue verticalement. Le processus est répété plusieurs fois afin de satisfaire les exigences des régions reconfigurables.

Une procédure de vérification en colonne (coulnmar direction) est effectuée à la fin de chaque itération en déplaçant la région verticalement dans le but d'améliorer le wirenlength (coût de communication) total. Et le meilleur résultat obtenu par rapport à une fonction objective est alors considéré comme solution. Bien que l'approche proposée par Vipin et Fahmy [VF12] ait prouvé qu'elle a fourni des bons résultats par rapport à d'autres travaux [MSSM10] en termes de wire length et taux d'occupation de la zone reconfigurable, Rabozzi et al. [RLS14] ont montré que la qualité des solutions obtenues peut être encore améliorée par des méthodes analytiques.

L'algorithme de floorplanning le plus récent est proposé dans [RLS14]. Il est basé sur une formulation mathématique par une approche nommée programmation linéaire mixtes en nombre d'entiers (PLMNE). Le modèle analytique est formulé pour représenter les caractéristiques de l'ensemble des PRRs en ce qui concerne leurs besoins en ressources matérielles et la connectivité entre elles. Étant donné que l'algorithme basé sur PLMNE a montré qu'il est capable de tenir compte de l'espace global dans ses recherches pour trouver des solutions optimales, néanmoins l'algorithme prend beaucoup de temps pour résoudre des problèmes plus complexes qui nécessitent plusieurs régions reconfigurables et par conséquent le temps de recherche dans l'espace augmente exponentiellement avec le nombre de régions. Pour réduire le temps de recherche, ils ont proposé une intervention manuelle par le concepteur pour donner une première solution réalisable afin d'obtenir de bonnes solutions finales.

Pour affronter ces limitations, Rabozzi et *al.* ont proposé une amélioration de leur algorithme dans leurs travaux ultérieurs, [RDM+17], [RMS15], tout en donnant au concepteur la possibilité d'accorder à l'algorithme de floorplanning un ensemble des emplacements réalisables pour chaque région reconfigurable de sorte qu'il ne soit pas possible de considérer que ceux qui sont pertinents pour la conception. Tous les emplacements réalisables des régions sont représentés dans une structure appelée graphe de conflit (conflict graph) dans laquelle les nœuds du graphe représentent les emplacements réalisables et les arcs du graphe

représentent l'existence d'un chevauchement entre deux emplacements (FIGURE 1.8).

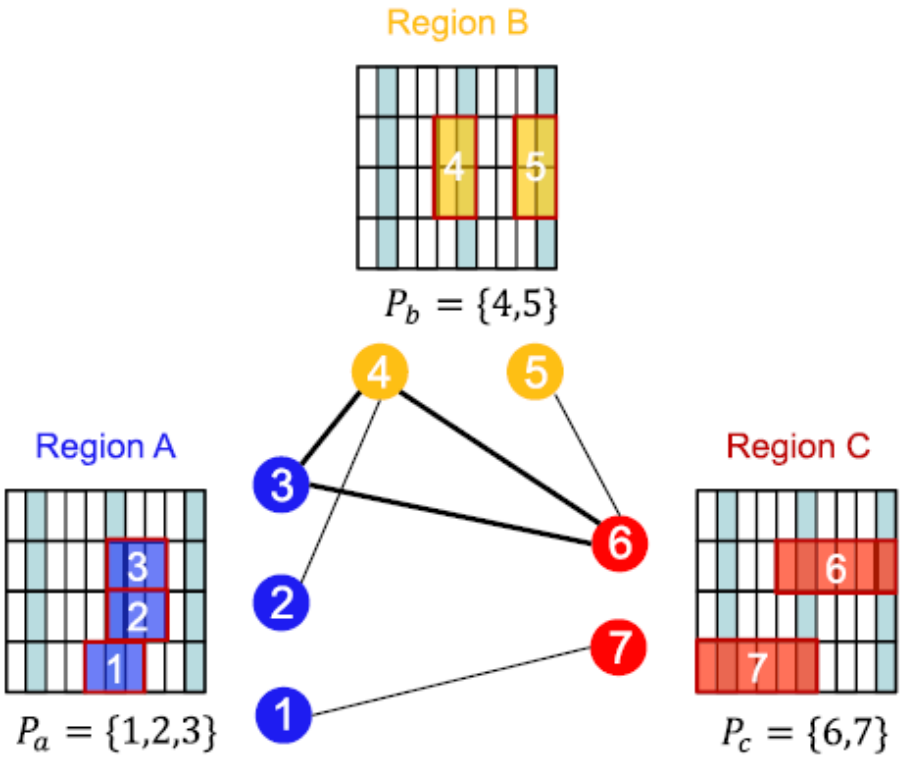

FIGURE *1.8 – Exemple d'un graphe de conflit [RDM+17]*

L'objectif commun des algorithmes de floorplanning présentés ci-dessus est de chercher une solution optimale à base d'une fonction objective et tout en respectant l'ensemble des contraintes imposées par l'architecture cible. Cependant, ces approches ne prennent pas en considération l'aspect de relocation des bitstreams partiels et ces contraintes (exemple : des zones identiques) dans leurs algorithmes de recherche.

Dans [IAI+12], Chinomiya et *al.* ont présenté une technique de conception pour la relocation des bitstreams partiels. L'identification des régions homogènes dans ce travail est faite manuellement. Une première région reconfigurable est définie en fonction des besoins en ressources de plus gros module. Après, cette région est utilisée comme modèle pour trouver d'autres régions compatibles avec elle. Cependant, avec l'augmentation du nombre des IP intégrées et l'hétérogénéité des ressources au sein du FPGA, une identification manuelle du modèle de ressource idéal devient de plus en plus difficile.

Dans [BHW+14], Backasch et *al.* ont proposé une approche automatisée adaptée à la relocation pour identifier des PRR homogènes qui répondent aux exigences de la taille et de la composition des ressources. L'objectif de l'algorithme est d'identifier le modèle de ressource optimal qui convient le plus souvent avec le périphérique FPGA pour une exigence de ressources donnée. Le processus de recherche de régions homogènes proposé est divisé en quatre étapes différentes. Tout d'abord, le concepteur doit définir les besoins en ressources

pour les PRR. Ces données sont utilisées pour l'analyse d'architecture et pour la génération de modèles. Selon les besoins en ressources, l'analyse de l'architecture génère un modèle qui répond aux exigences de ressources spécifiées. Ensuite, l'algorithme de floorplanning proposé explore le FPGA pour trouver toutes les PRR qui contiennent le modèle généré. Enfin, les emplacements résultants peuvent être exportés pour une utilisation ultérieure. La FIGURE 1.9 présente le flot de conception proposé.

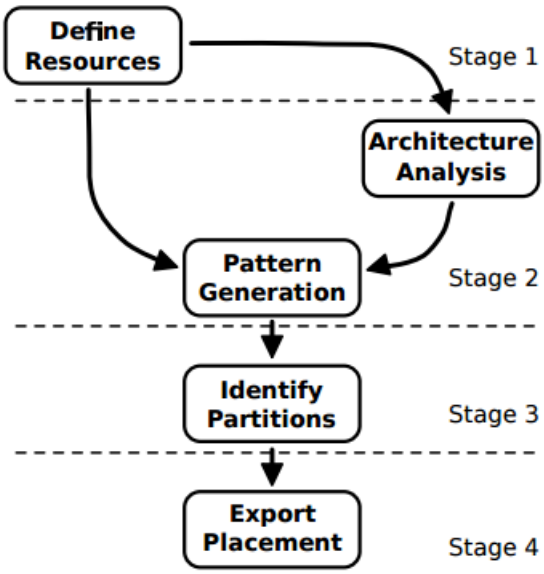

FIGURE *1.9 – flot de conception de floorplanning automatique [BHW+14]*

Bien que [BHW+14] ait automatisé le processus de floorplanning dans le cas de relocation et donne de bons résultats par rapport aux approches précédentes, l'algorithme ne donne pas des informations sur la qualité de la solution trouvée par rapport à la solution optimale.

4.2 Discussion

Le TABLEAU 1.2 montre une comparaison entre les différents algorithmes de floorplanning proposés dans la littérature.

La plupart des travaux proposés dans la littérature se concentrent sur les propriétés statiques d'un système pour un emplacement particulier. D'autres travaux traitent le problème du floorplanning mais dans des architectures FPGA homogènes.

Le floorplanning sur les FPGA modernes partiellement reconfigurables nécessite que le floorplanner tienne compte des contraintes PR et des répartitions de ressources non homogènes. Parmi les travaux dans ce domaine, les seuls qui tiennent compte de ces deux aspects sont [VF12] et [RLS14]. Bien que ces travaux portent sur le floorplanning automatique des zones reconfigurables en optimisant certains critères, mais ils ne prennent pas en considération les contraintes posées par la relocation ainsi les tailles variables des tâches

TABLEAU *1.2 – Comparaison entre les différents algorithmes de floorplanning*

	[CW06]	[FM06]	[YYC07]	[BSSK11]	[VF12]	[RLS14]	[BHW+14]
Hétérogénéités de ressources		x			x	x	x
Supporte la RDP			x	x	x	x	x
Fonction objective					x	x	
Optimisation d'interconnexion	x	x	x		x	x	
Supporte la relocation							x
Adaptation de PRR							

reconfigurables. Ces critères conduisent à une inefficacité d'utilisation des ressources matérielles et à un temps de reconfiguration élevé.

L'un de nos objectifs principaux est d'apporter une solution pour résoudre des problèmes énoncés ci-dessus. En effet, nous proposons une nouvelle approche basée sur une formulation analytique adaptée à la relocation qui surmonte ces problèmes et donne de meilleurs résultats en termes de fonction objective. Nous pouvons distinguer dans notre travail deux niveaux différents dans la complexité de l'opération de floorplanning, à savoir :

- Reloger un bitstream sur des zones totalement homogènes offrant les mêmes tailles, types, nombres et dispositions des ressources
- Reloger un bitstream sur des zones des tailles variables correspondant aux tâches de tailles variables

Nos algorithmes permettent au concepteur de décider les termes à optimiser, en donnant également la possibilité de choisir un modèle d'optimisation pour prendre en compte de l'objectif requis. Les floorplannings générés par nos méthodologies sont conformes au flot de conception des architectures reconfigurables et peuvent être utilisés pour des FPGA ayant des distributions de ressources non homogènes.

5 Conclusion

Dans ce chapitre, nous avons élaboré un état de l'art sur les architectures reconfigurables, la technique de relocation et le floorplanning. Dans un premier lieu, une étude sur la reconfiguration dynamique partielle a été présentée. Différents travaux traitant les problèmes de gestion de la RDP, dans la littérature sont présentés. Dans un second lieu, nous nous sommes intéressés à la technique de la relocation. Finalement, nous avons présenté

la plupart des travaux dans la littérature qui traitent le problème de floorplanning dans différents aspects.

Comme nous l'avons vu tout au long de ce chapitre la relocation et le floorplanning sont des méthodes très importantes dans la conception d'une architecture reconfigurable. Cependant, les travaux de la littérature traitent chaque technique toute seule sans tenir compte de l'autre ce qui engendre à une réduction des performances et de flexibilité du système. Avec les flots de conception actuels, il n'est pas possible d'adapter totalement les contraintes et les critères imposés sur la région reconfigurable, on peut donc conclure qu'elle n'est pas totalement flexible. Par conséquent, il est nécessaire de proposer de nouvelles méthodes et algorithmes qui permettront d'augmenter la flexibilité et les performances des FPGA.

Dans le chapitre suivant, nous allons proposer une méthodologie de conception détaillée pour la relocation des bitstreams partiels.

Chapitre 2

Méthodologie pour la Relocation des tâches de tailles variables

Sommaire

1 Introduction

La relocation est une technique qui permet au concepteur d'utiliser un seul bitstream pour configurer une tâche reconfigurable dans plusieurs régions reconfigurables. Ce mécanisme n'est pas réalisable avec les flots de conception RDP standard. Des étapes supplémentaires sont requises pour créer un design qui permet de générer un bitstream relogeable. Les flots décrits par Xilinx [XIL12] limitent l'emplacement des modules reconfigurables (PRM) qu'aux régions PR prédéfinies, dans lesquelles les PRM vont être ensuite alloués. Les limites du flot de conception RDP sont multiples, mais les majeurs problèmes découlent du fait que les PRM sont restreints aux PRR spécifiques. Le problème de communication est considéré comme un défi et les PRR doivent être soigneusement définies à l'avance, ce qui entraîne des systèmes avec une flexibilité limitée. En conséquence, la technique de relocation est généralement trop complexe à utiliser, malgré ses nombreux avantages.

Le deuxième défi dans la mise en œuvre de la relocation qui n'est pas encore abordé dans la littérature et qui est sûrement le plus complexe et le plus intéressant à résoudre est l'optimisation des ressources matérielles pour augmenter l'efficacité du système. Généralement, la taille de la région reconfigurable est fixée en phase de conception selon le plus gros module à accueillir et ne peut pas être changée en cours de l'exécution. Pour cette raison, la tâche la plus grande occupe la totalité des ressources dans la zone définie et les autres tâches les plus petites qui vont être configurées dans la même PRR, utilisent qu'une petite partie des ressources disponibles ce qui entraine une perte de ressources et une inefficacité d'utilisation.

Ce chapitre est divisé en deux parties. Dans la première partie, nous présenterons un nouveau flot de conception de relocation. L'objectif de ce flot est de surmonter d'une part les limitations du flot de conception RDP standard et d'augmenter d'autre part la souplesse et la flexibilité du système. Nous proposerons, en deuxième partie, une nouvelle méthode de conception pour la relocation des tâches de tailles variables en adaptant la taille de la région PR pour résoudre le deuxième défi décrit ci-dessus.

2 Relocation des modules reconfigurables

Dans cette section, nous proposons de mettre en œuvre une méthodologie permettant une flexibilité en ligne dans l'allocation et la relocation des tâches matérielles. Ces changements doivent être effectués en temps de reconfiguration réduit afin de garantir une fiabilité globale du système. Nous commençons par détailler les différentes contraintes pour garantir un bitstream relogeable. Ensuite, nous proposons une structure de communication adaptée afin d'assurer la communication entre les différents modules dans différents emplacements. Finalement, nous allons détailler dans le flot de conception, comment ces différentes contraintes ont été prises en compte.

2.1 Contraintes de la relocation

Généralement, une relocation réussie d'un bitstream partiel nécessite des exigences principales, qui sont résumées dans les points suivants :

— **Compatibilités des zones** : Un flux binaire partiel configure une hauteur et un largeur fixes des colonnes de ressources. Pour reloger un module reconfigurable, l'emplacement cible de la configuration doit avoir des ressources identiques à celle de l'emplacement d'origine du module reconfigurable. Cela implique que tous les emplacements qui vont accueillir un même bitstream doivent avoir les mêmes dimensions, les mêmes types de ressources et la même disposition des colonnes. La FIGURE 2.1 montre un exemple de deux zones identiques et deux autres non identiques.

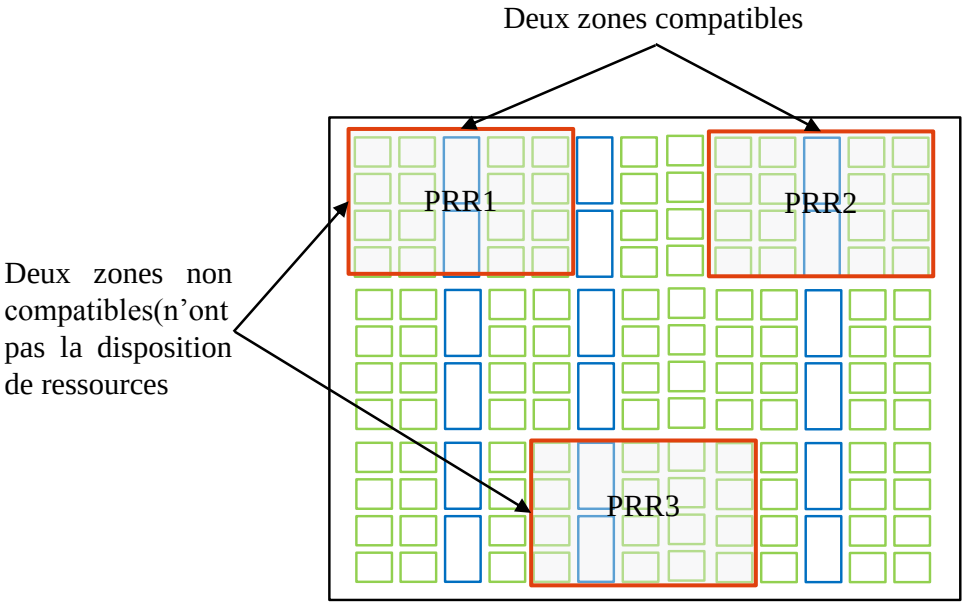

FIGURE *2.1 – Compatibilité des zones reconfigurables*

— **Placement identique des Pin Partition** (1) : Durant le processus de conception, les pins partitions sont automatiquement placés dans la région reconfigurable. Ils sont mis en œuvre sur des LUT(Look Up Tables). Cette interface de communication est caractérisée par un type (entrée ou sortie), par un placement (coordonnées XY du slice contenant la LUT), par un élément logique basique (BEL : basic element logic) et finalement par un point de connexion (A1, A2, A3, A4, A5, ou A6).
Pour déplacer la configuration d'un bitstream d'une zone à une autre, il faut que toutes les informations des pins partitions dans la nouvelle zone soient identiques à celles de l'emplacement d'origine. La FIGURE 2.2 (a) illustre un exemple de placement des pins partitions dans une LUT pour l'emplacement d'origine. La FIGURE 2.2 (b) et la figure FIGURE 2.2 (c) montrent un emplacement d'une pin partition non

(1). une nouvelle technologie de connexion physique entre la logique statique et la logique reconfigurable

identique à celle de l'origine. Elles sont différentes respectivement par le type de BEL (B6) et par le point de connexion d'entrée (D3). Le pin Partition présenté dans la FIGURE 2.2 (d) est placée sur le même type LUT-6D et elle est connectée avec la même entrée A6 de la LUT. Cette connexion est compatible avec l'emplacement d'origine.

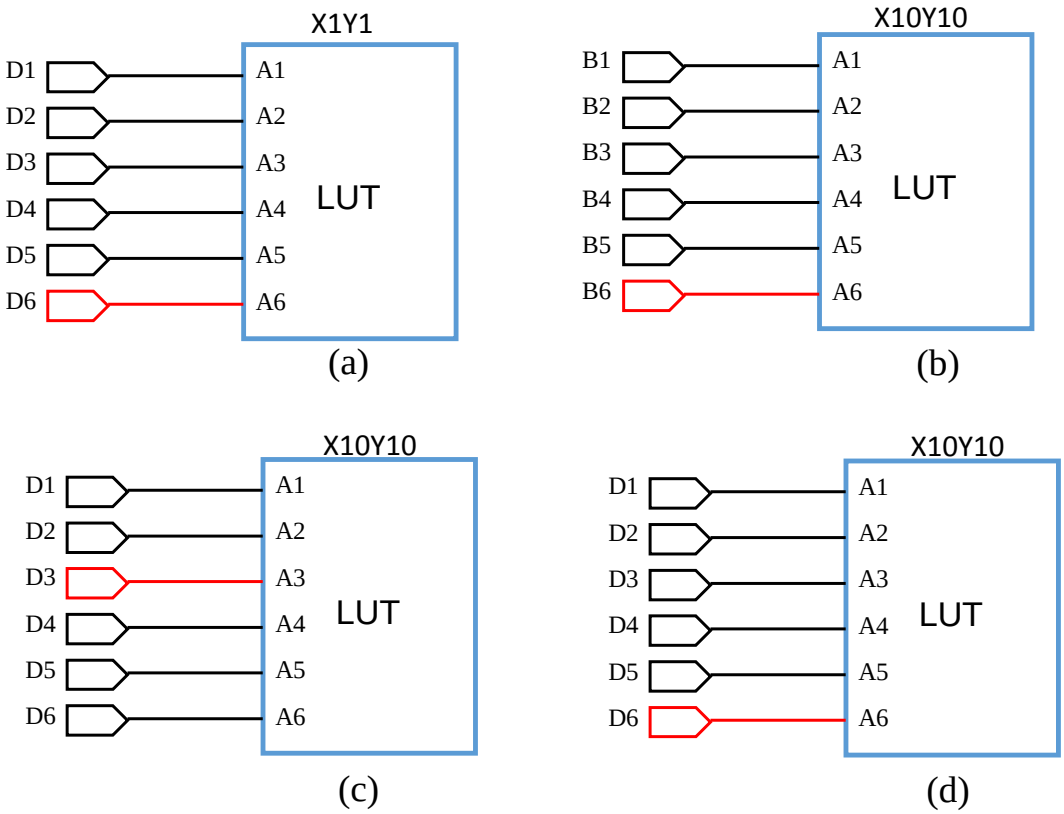

FIGURE *2.2 – Placement identique des Pins partition (a) Les informations du pin sur l'emplacement d'origine (b) et (c) Exemple de placement des pins partition non identique (d) placement identique des pins partition de l'emplacement de l'origine*

— **Chemins de routages identiques** : Afin de reloger un module reconfigurable de son emplacement de configuration d'origine dans un FPGA, les chemins de routage qui connectent ce module avec les autres parties du système doivent être identiques avec ceux du nouvel emplacement. La FIGURE 2.3 montre un exemple des chemins de routages identiques pour trois zones reconfigurables.

2.2 Gestion de communication Inter-modules

Dans toute architecture reconfigurable dynamiquement, les tâches matérielles nécessitent une interface d'interconnexion efficace et flexible permettant la communication entre elles. Les ressources de communication entre les modules reconfigurables sont amenées à être modifiées au cours des différentes phases de configuration (FIGURE 2.4). De plus, l'éven-

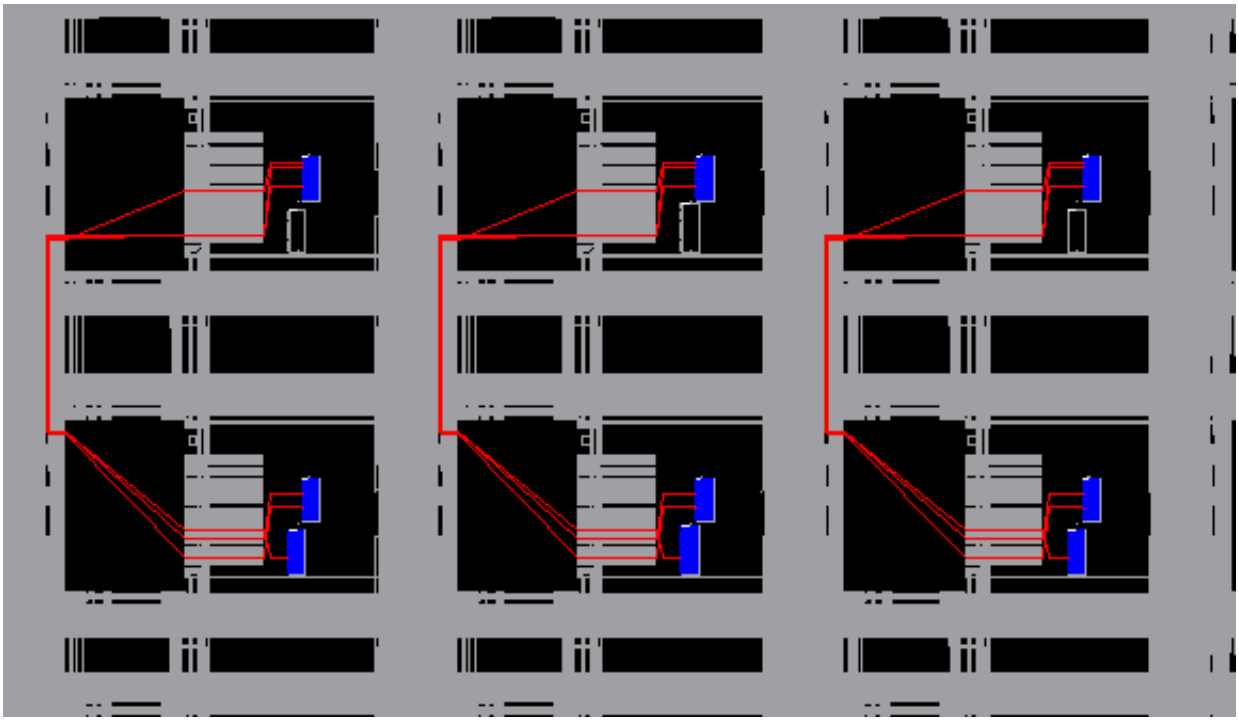

FIGURE 2.3 – *Exemple des chemins de routages identiques pour trois zones reconfigurables*

tuelle variété des tâches matérielles qui viendront occuper plusieurs régions reconfigurables dans le cas de la relocation, implique l'obligation de prévoir une interface de communication commune avec toutes les régions reconfigurables qui doivent l'occuper. En effet, la question de la communication est l'un des défis dans la mise en place des tâches matérielles dans plusieurs régions reconfigurables.

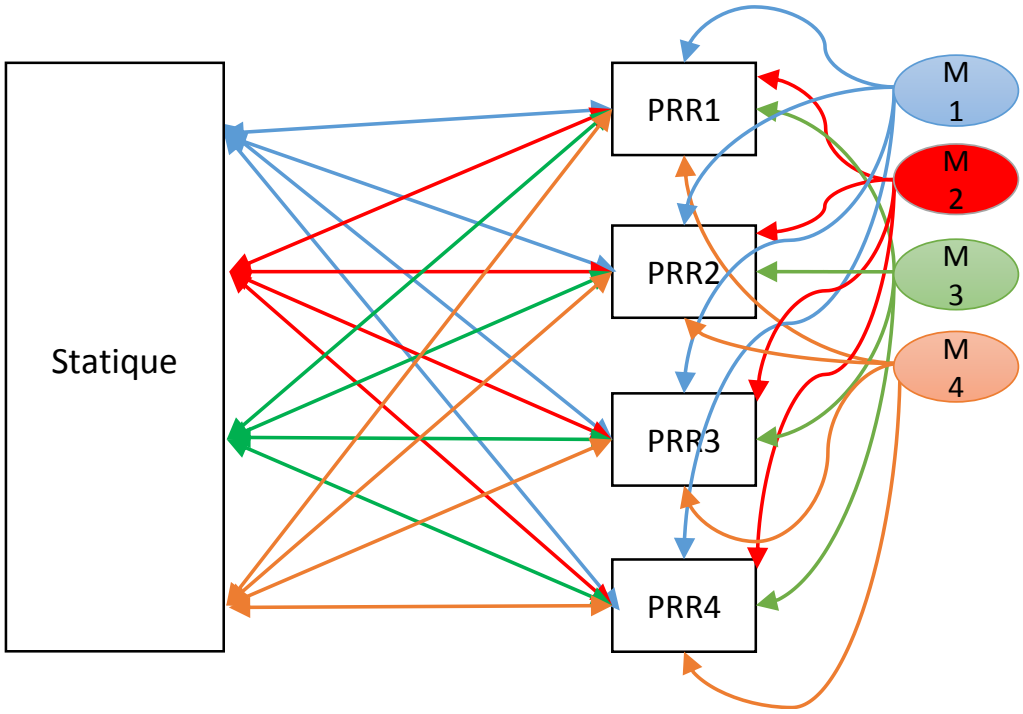

FIGURE 2.4 – *Problème de communication inter-modules*

L'objective de cette partie est de proposer une nouvelle structure de communication sur puce adaptée à la conception d'un système adaptatif qui rend interopérables les modules reconfigurables sur les différentes PRR. Le routeur d'interconnexion proposé est une unité d'interconnexion basée dans l'idée d'une architecture multi-tâches et multi-destinations (régions reconfigurables) décrites par la FIGURE 2.5. Il permet de relier chaque module reconfigurable alloué dans plusieurs régions reconfigurables à leur destination envisageable. L'avantage d'un tel routeur réside dans sa capacité d'une part, à partager de façon efficace sur chacune des régions reconfigurables les interfaces de communication (I/O) provenant des différents modules reconfigurables et, d'autre part, à répartir ces interfaces de communication vers leur destination envisageable. Pour cela, des composants appelés crossbars ou commutateur assurent la jonction entre les différents ports d'E/S et les modules reconfigurable.

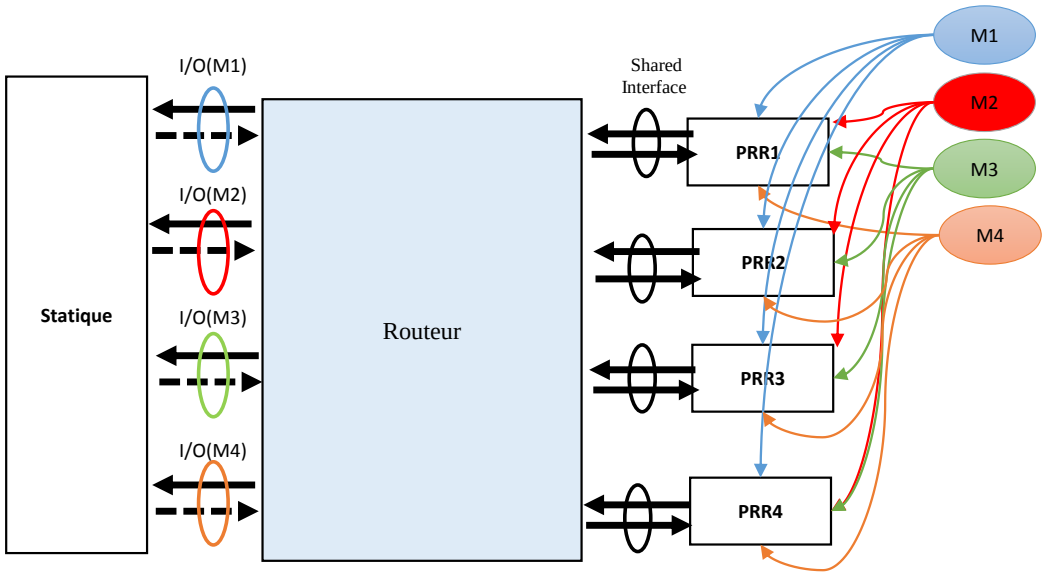

FIGURE *2.5 – Routeur d'interconnexion*

Le routeur proposé se compose de deux crossbars. Le premier communique avec les régions reconfigurables et le deuxième communique avec la partie statique. Ces deux crossbars communiquent entre eux aussi comme le montre la FIGURE 2.5. La description de ce routeur est effectuée par l'intermédiaire de deux types de paramètres :

1. Les paramètres d'identification : IDM et IDR
2. les paramètres de configuration :
   — Port de communication : Entrées/ Sorties : nombre et taille,
   — Paramètre de crossbar : Nombre de modules reconfigurables et nombre de régions reconfigurables

Les paramètres d'identification Au sein de l'architecture, l'identification (ID) au sein du routeur est nécessaire afin d'autoriser l'instanciation de plusieurs modules reconfigurables dans une même région reconfigurable et aussi d'allouer un module reconfigurable dans plusieurs régions reconfigurables. IDM est un identificateur de PRM. Il permet de reconnaître le module reconfigurable alloué dans la PRP et de lui choisir un chemin d'interconnexion qui lui assure la connexion avec sa destination dans la partie statique. IDR est un identificateur de PRR qui permet aux informations (I/O) arrivant de la partie statique vers la partie reconfigurable de connaitre la PRR allouée par le PRM destiné.

Paramètres de configuration Les nombres de ports d'E/S qui composeront le routeur sont des paramètres génériques à inclure dans la spécification VHDL. Il existe deux types de port dont la distinction est utile aux outils de placement-routage : les ports de communication reconfigurable qui relient les PRR avec le routeur et les ports de communication statique qui sont dédiés aux communications depuis/vers le routeur et vers/depuis la partie statique. La taille des ports de communication est spécifiée par le nom $\text{data}_{\text{widh}}$.

Un autre paramètre qui compose la description du routeur est la taille des crossbars utilisés. Le premier crossbar est de taille $\text{Nm} * \text{Nr}$ et le deuxième est de taille $\text{Nr} * \text{Nm}$.

où Nm présente le nombre de modules reconfigurables et Nr présente le nombre de régions reconfigurables Comme montre la FIGURE 2.6.

Description du fonctionnement de routeur L'exemple décrit dans cette partie reprend la description effectuée dans la FIGURE 2.6. Le routeur communique avec deux parties en parallèle. Il spécifie une unité d'interconnexions qui sera directement connectée aux régions reconfigurables et une autre qui communique avec la partie statique.

Les paramètres du premier sens (des modules reconfigurables vers la partie statique) qui consiste à la communication avec les régions reconfigurables sont les suivants :

- M(i) va allouer une région R(j).
- Les ports d'entrée / sortie de la région R(j) se connectent directement avec $\text{input}_{\text{link(j)}}$ de routeur (qui est réservé pour se connecter uniquement avec la région (j)).
- le $\text{input}_{\text{link(j)}}$ de routeur lit l'identifiant du module (IDM).
- Selon IDM, le dé-multiplexeur de $\text{input}_{\text{link(j)}}$ envoie les données vers le MUX(i) (qui est réservé pour recevoir uniquement les données de M(i))
- Le Mux(i) se connecte directement avec $\text{output}_{\text{link(i)}}$ (qui se connecte directement avec la destination des I/O de M(i) dans la partie statique)

Les paramètres du deuxième sens (de la partie statique vers les modules reconfigurables) qui consiste à la communication avec la partie statique sont les suivants :

- Les ports d'entrée / sortie dans la partie statique qui sont réservés pour se connecter avec le M(i) se connectent directement avec $\text{output}_{\text{link(i)}}$ de routeur (qui est réservé pour se connecter qu'avec le module M(i)).
- Le $\text{output}_{\text{link(i)}}$ envoie les données vers tous les multiplexeurs.

— Selon IDR, le mux (j) sélectionne le module qui occupe la région(j) et l'envoi vers le $\text{input}_{\text{link(j)}}$

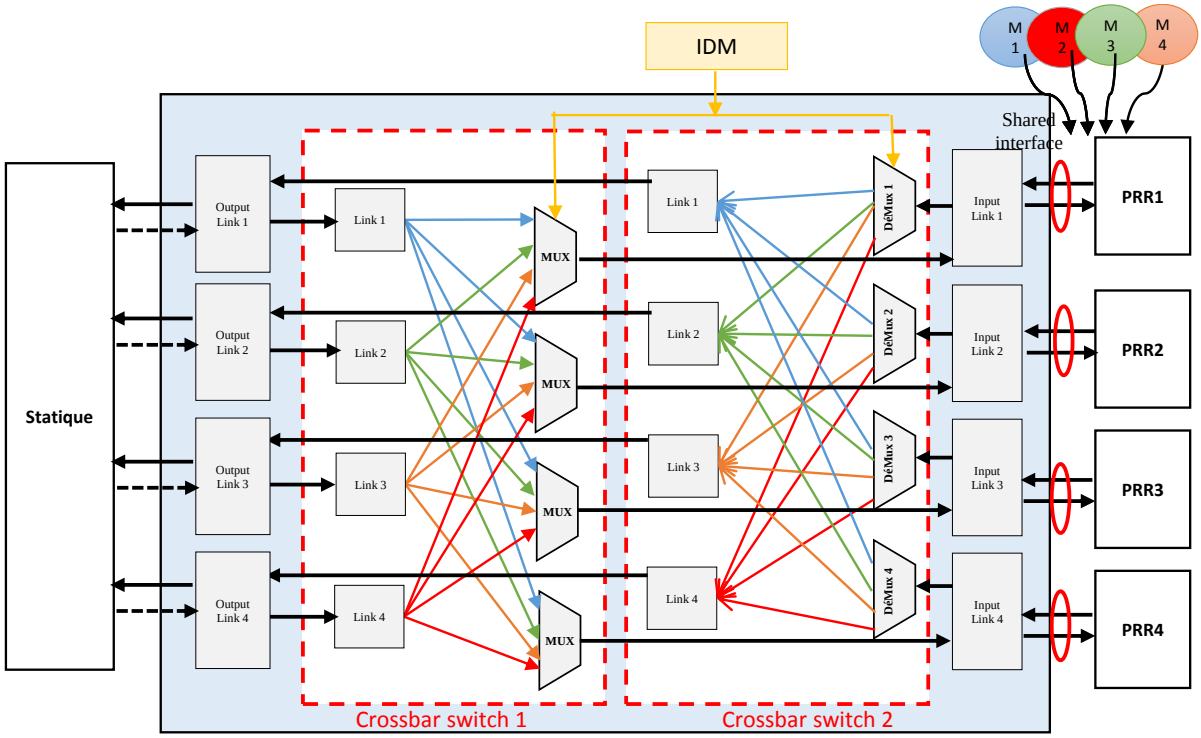

FIGURE *2.6 – Architecture interne du module d'interconnexion*

2.3 Flot de conception proposé pour la relocation

Dans cette partie, nous allons présenter une modification du flot de conception de Xilinx pour rendre possible la relocation des tâches matérielles reconfigurables sur une architecture reconfigurable. Le flot global de la conception pour la relocation des bitstreams partiels est illustré dans la figure FIGURE 2.7.

La méthodologie est basée dans un premier temps, sur la validation de l'allocation d'une tâche sur sa zone de placement initial puis dans un deuxième temps de reloger cette tâche à une autre zone inoccupée du FPGA. La validation se réalise en chargeant tout d'abord le bitstream complet de l'application qui contient des black-box associés à la zone reconfigurable, puis en effectuant le chargement du bitstream partiel qui permet de valider le bon fonctionnement de la tâche. Pour effectuer une relocation de la tâche, il faut d'abord, confirmer une certaine compatibilité sur les zones reconfigurables de placement pour s'assurer que les pins partitions (interface de communication entre la partie statique et la partie dynamique) sont positionnées dans le même emplacement et ont les mêmes caractéristiques (BEL et LOC) pour chaque module. Il faut contraindre aussi tous les chemins de routage pour qu'ils soient compatibles avec tous les modules. Par la suite une modification dans le bitstream se fait en changeant le FAR (frame address register) pour reloger la tâche d'un

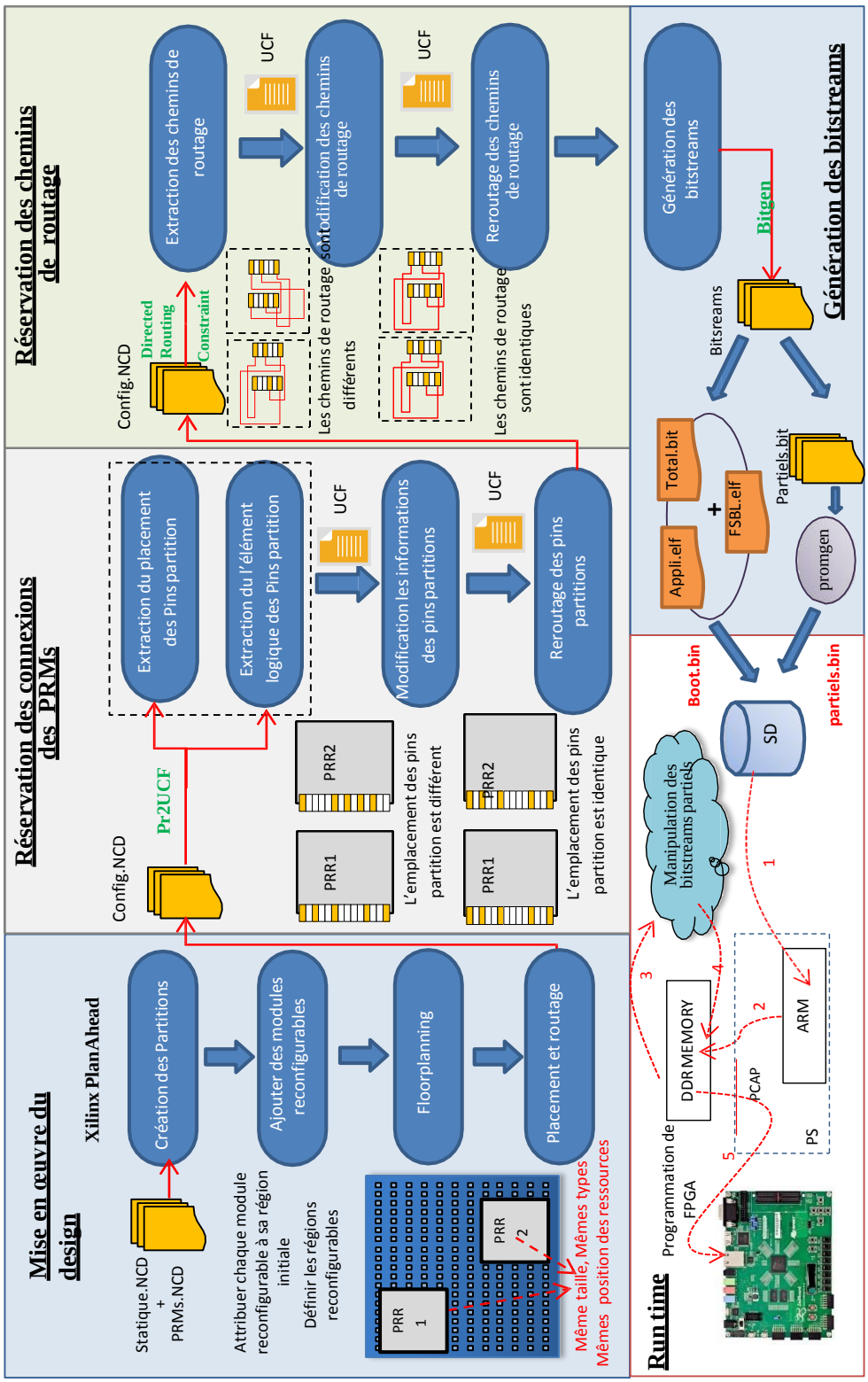

FIGURE 2.7 – *Flot de conception pour la relocation des tâches reconfigurables*

emplacement à un autre. Enfin le chargement du bitstream modifié permet de valider le bon fonctionnement de la tâche dans la nouvelle zone de placement [HRJ+14].

2.3.1 Partie design : hors ligne

Un bitstream relogeable consiste à passer par quatre étapes principales pour la partie de la conception hors ligne.

Étape 1 : Mise en œuvre de la conception initiale. Cette première partie du flot du design permet de définir et de décrire les éléments du système. Cette étape est semblable au flot PBPR (Partition Based Partial Reconfiguration) standard de Xilinx. Nous commençons par définir tous les éléments qui constitueront la région statique et les modules reconfigurables. Une fois tous les éléments construits de notre architecture du système sont complétés, les fichiers « Netlist » de la partie statique et des modules reconfigurables sont importés dans PlanAhead. Ensuite, nous définissons les régions reconfigurables (floorplanning) tout en considérant que n'importe quel emplacement cible d'un bitstream relogeable doit avoir les mêmes dimensions, mêmes types de ressource et même disposition de ressources par colonne que l'emplacement d'origine de PRM. Nous affectons ensuite, chaque module reconfigurable à une seule région reconfigurable et nous laissons les autres régions vides.

Étape 2 : Réservation des informations des interfaces de communication. Lors de chaque relocation d'un module reconfigurable à un nouvel emplacement dans le FPGA, il est nécessaire de réserver les mêmes informations sur les interfaces de communication (Pin Partition) entre le PRM et les autres éléments de l'architecture.

En premier lieu, il est nécessaire d'extraire les informations sur les Pins Partitions (BEL et LOC) de la conception initiale dans les emplacements d'origine de chaque module reconfigurable. Ces informations sont extraites à partir d'un fichier NCD générée dans la conception initiale en utilisant la commande « pr2ucf » et ils sont sauvegardés dans un fichier UCF.

```
Pr2ucf config_1_routed.ncd  -bel -o pin_partition.ucf
```

Les informations extraites rassemblent à toutes les données nécessaires de chaque pin partition : les coordonnées du slice qu'elle utilise et son type d'élément logique.

```
PIN "Inst_RP_11.E_11" LOC = SLICE_XnYm;
PIN "Inst_RP_11.E_11" BEL = B6LUT;
```

Après l'extraction des informations sur les interfaces de communication (Pins Partion) dans l'emplacement original, il est nécessaire de modifier le LOC et le BEL de chaque pin partition dans la nouvelle région pour qu'ils soient compatibles avec la région initiale. Ces

informations doivent être calculées et modifiées de sorte que lorsque le bitstream partiel est relogé dans un autre endroit, il faut qu'il trouve des pins partitions identiques et dans le même emplacement pour qu'il puisse connecter avec le reste du système. Nous modifions le fichier UCF où nous calculons les nouvelles cordonnées (XY) des pins partions dans les nouveaux emplacements dont le but d'avoir les mêmes positions que celles de la région reconfigurable initiale comme l'exemple illustré dans la FIGURE 2.8. D'autre part il faut également modifier le type des éléments logiques BEL des pins partition avec celui qui convient avec la première partition.

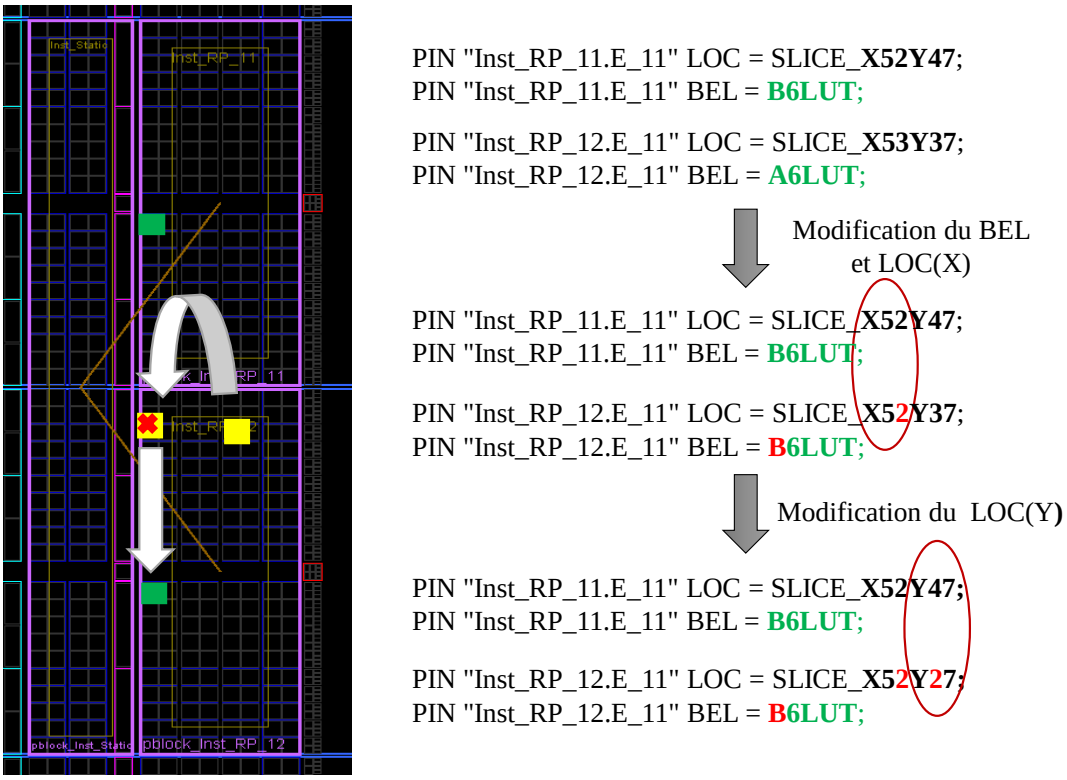

FIGURE 2.8 – *Modification des informations des pins partition*

Les modifications apportées dans le fichier UCF permettront de ré-router les pins partitions pour que leurs connexions soient communes entre toutes les régions qu'elles viennent de recevoir la même tâche.

Étape 3 : Réservation des chemins de routages. Le placement automatique des pins partition pose une certaine limitation pour effectuer la relocation des bitstreams partiels où il est difficile d'extraire et imposer les chemins de routage des entrées /sorties qui doivent être identiques dans toutes les régions reconfigurables. Afin de surmonter ce problème, nous proposons de transformer les logiques proxy en bus macro. Nous définissons une petite zone reconfigurable supplémentaire à coté de la région reconfigurable. Avec cette méthode, nous aurons des pins partition dans la partie reconfigurable ajoutée et des pins partitions dans la

partie dynamique ce qui forme des bus macro comme montre la FIGURE 2.9. Dans ce cas, les chemins de routage entre les deux parties deviennent connus et on peut les contraindre.

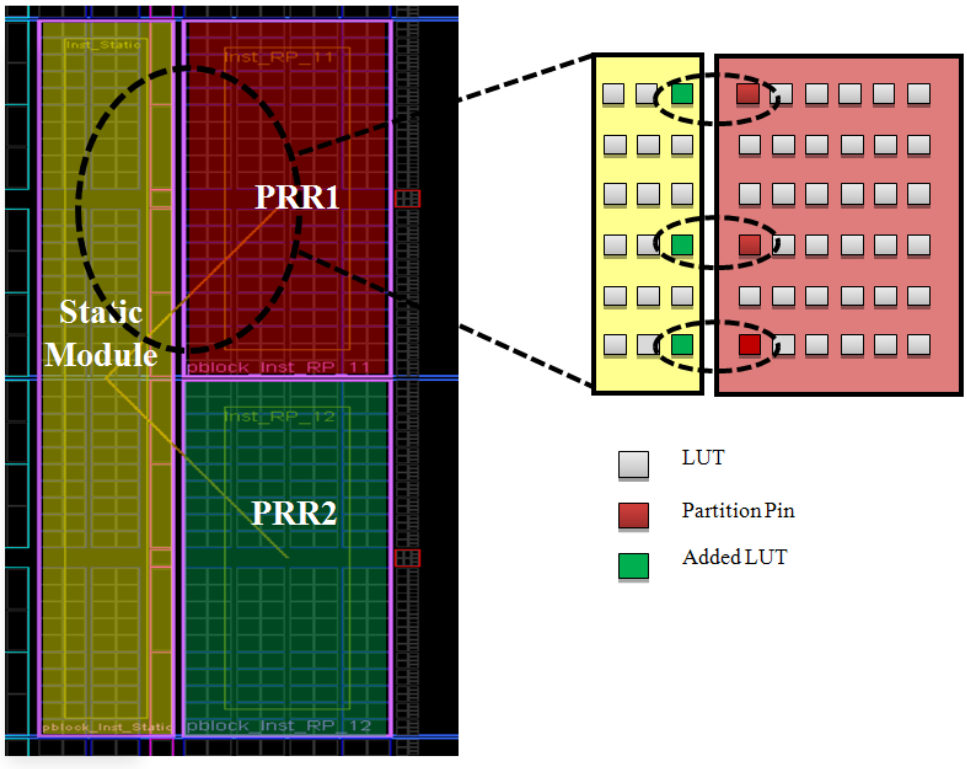

FIGURE *2.9 – Le placement de Proxy logic*

Les chemins de routages sont extraits à partir du fichier NCD généré, en utilisant la commande « Directed Routing Constarint » de l'outil FPGA Editor. Afin d'appliquer les mêmes chemins de routage dans toutes les régions reconfigurables, il est nécessaire d'extraire les chemins de routage des emplacements originaux qui relient la région reconfigurable avec la partie statique. Après, il faut copier ces chemins pour toutes les nouvelles régions PR en changeant l'adresse avec celle qui correspond la partition. L'adresse de chaque chemin de connexion est définie entre "! -1" et "S! 0" comme le montre la figure FIGURE 2.10.

Une fois tous les changements sont effectués, un nouveau ré-routage est nécessaire pour placer tous les chemins de routage à ces places souhaitées.

Étape 4 : Génération des bitstreams Dans cette étape, nous générons un bitstream total et un seul bitstream partiel pour chaque module reconfigurable. Par la suite, les bistreams partiels seront manipulés. Pour éviter la vérification de la détection d'erreurs CRC, les flux binaires sont générés en utilisant une commande de Xilinx qui désactive leurs vérifications. Cette commande est la suivante :

```
bitgen w g CRC:disable config_routed.ncd
```

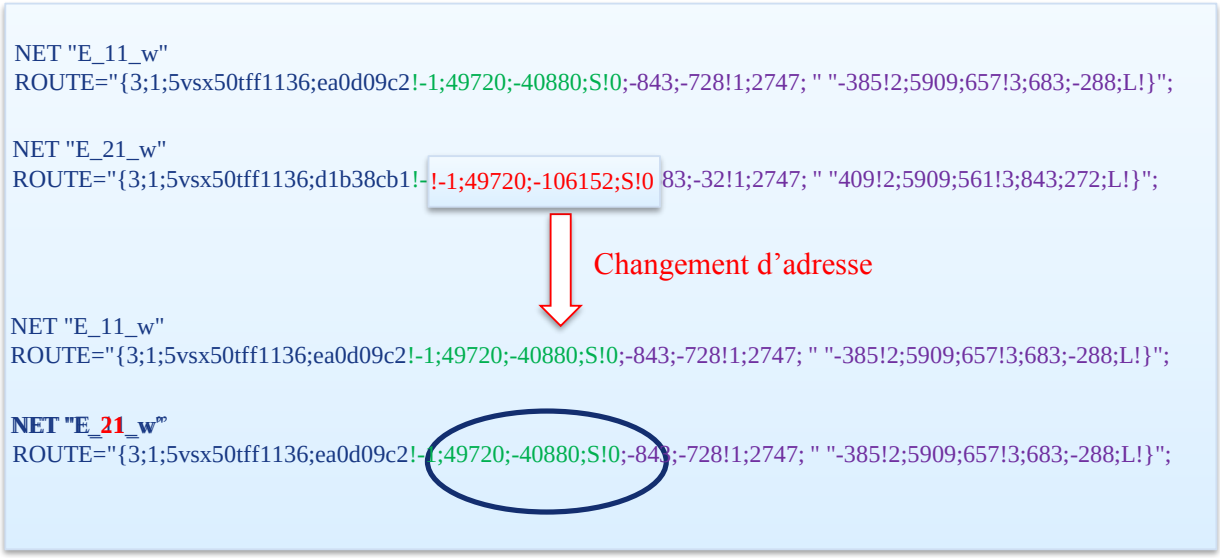

FIGURE 2.10 – *Modification des chemins de routage*

2.3.2 Partie run-time : En ligne

La manipulation des bitstream est effectuée en ligne à l'aide d'un processeur. Nous commençons par identifier les adresses FRAME de l'emplacement initial et nous les changeons par un autre qui définit l'emplacement physique de la nouvelle région. La FIGURE 2.11 illustre un exemple de la modification du FAR du bitstream qui correspond à la première PRR à partir de 00 10 14 00 par le nouveau FAR calculé pour la nouvelle PRR 00 10 94 00.

Généralement, la configuration interne nécessite plusieurs composants mis en œuvre dans le FPGA pour contrôler le chargement des bitstreams partiels de la mémoire externe à la mémoire de configuration du dispositif. Pour assurer la mise en œuvre de la reconfiguration, 3 différents modes de configuration offerts par le FPGA Zynq de Xilinx : le JTAG, l'ICAP et le PCAP. Les principales différences entre eux sont l'accessibilité et la largeur de données et les fréquences de fonctionnement (voir annexe A). Dans notre mise en œuvre, le système de reconfiguration proposé (illustré dans FIGURE 2.12) est composé par un contrôleur de reconfiguration PCAP, un contrôleur de mémoire externe DDR, un processeur ARM qui sont tous figés dans la partie PS (Processing System) de la plateforme Zynq de Xilinx.

L'emplacement de la configuration du bitstream est manipulé en modifiant le FAR (Frame Address Register). Les FAR dans un bitstream spécifient l'emplacement physique des frames de configuration sur FPGA (voir annexe B). Si un bitstream est requis pour être déplacé vers un nouvel emplacement souhaité, les FAR seront identifiés et remplacés par les nouveaux FAR du nouvel emplacement.

Le processus de manipulation de bitstream est passé par plusieurs étapes (FIGURE 2.13). Le processor ARM commence par lire les bitstreams du mémoire SD et les transmettre dans la mémoire externe DDR. Après, il cherche le FAR dans l'adresse mémoire du bitstream

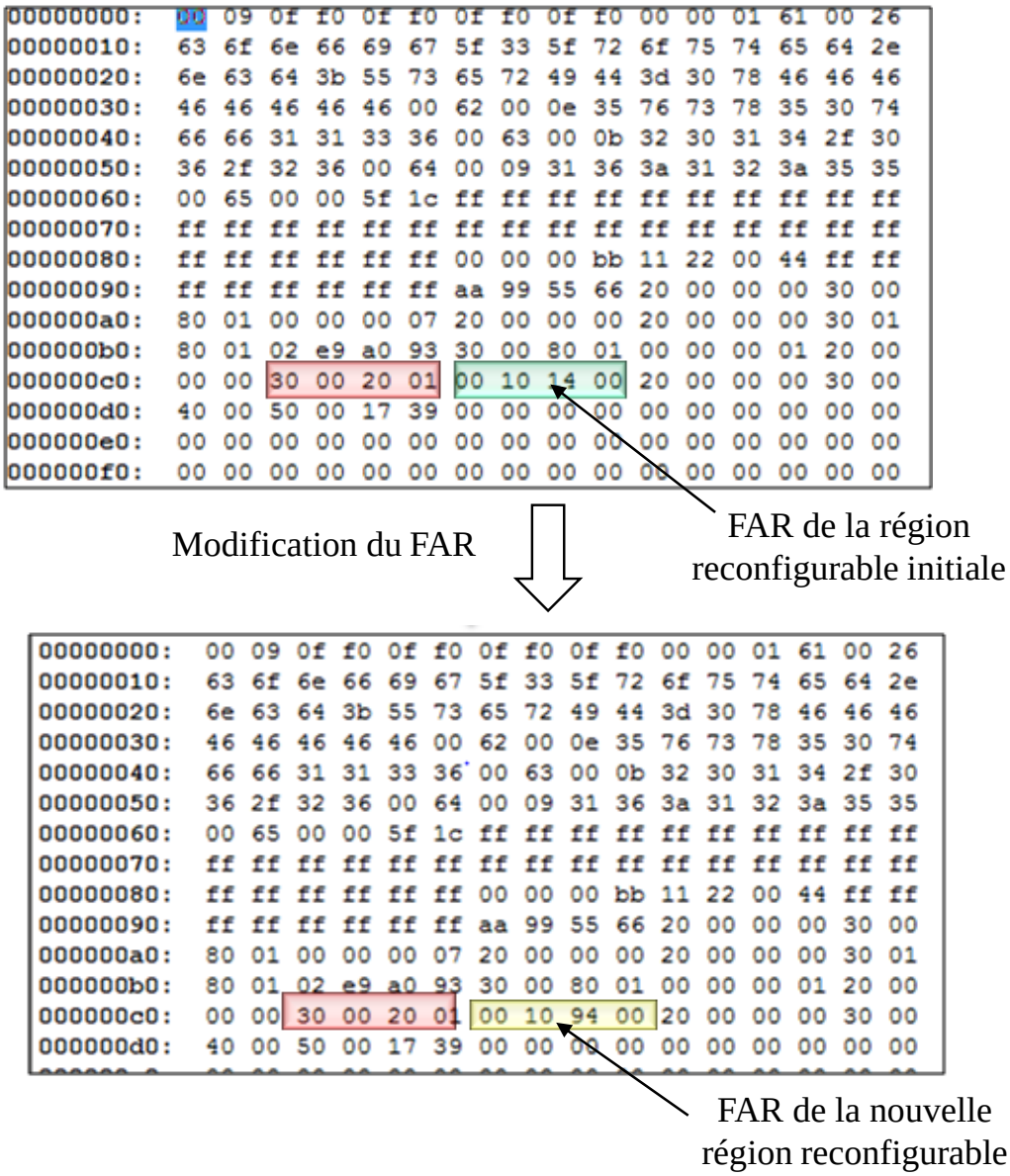

FIGURE *2.11 – Manipulation des bitstreams*

relogeable et dès qu'il l'identifie, il le change par le nouveau FAR dans la même adresse de DDR. Finalement, le PCAP configure le FPGA en chargeant le bitstream modifié.

Pour le contrôle et la gestion de configuration dans le FPGA, certains systèmes reconfigurables ne nécessitent qu'une simple machine d'état finis (FSM) pour contrôler le processus de reconfiguration. Dans d'autres systèmes nécessite un noyau OS mis en œuvre sur un FPGA, pour effectuer différents types d'opérations : ordonnancement, placement des tâches, etc. Dans ces systèmes, le contrôle de la configuration est beaucoup plus complexe que les modifications en ligne des bitstreams partiels pour permettre une tâche relogeable. Cette partie présente un contrôle interne de configuration des bitstreams partiels qui aborde l'aspect de manipulation des bitstreams relogeables. La manipulation des bitstreams dans notre système est effectuée par un processeur ARM à l'aide d'un algorithme de gestion

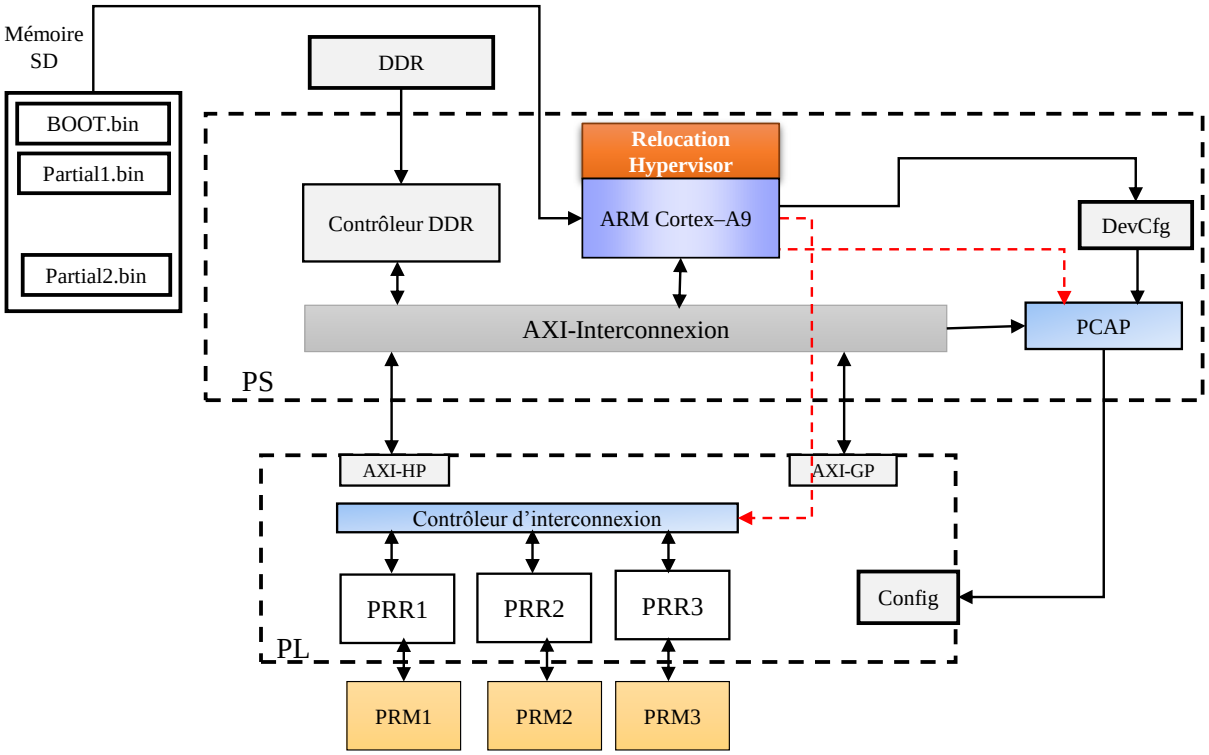

FIGURE 2.12 – *Architecture du système à base du processeur ARM*

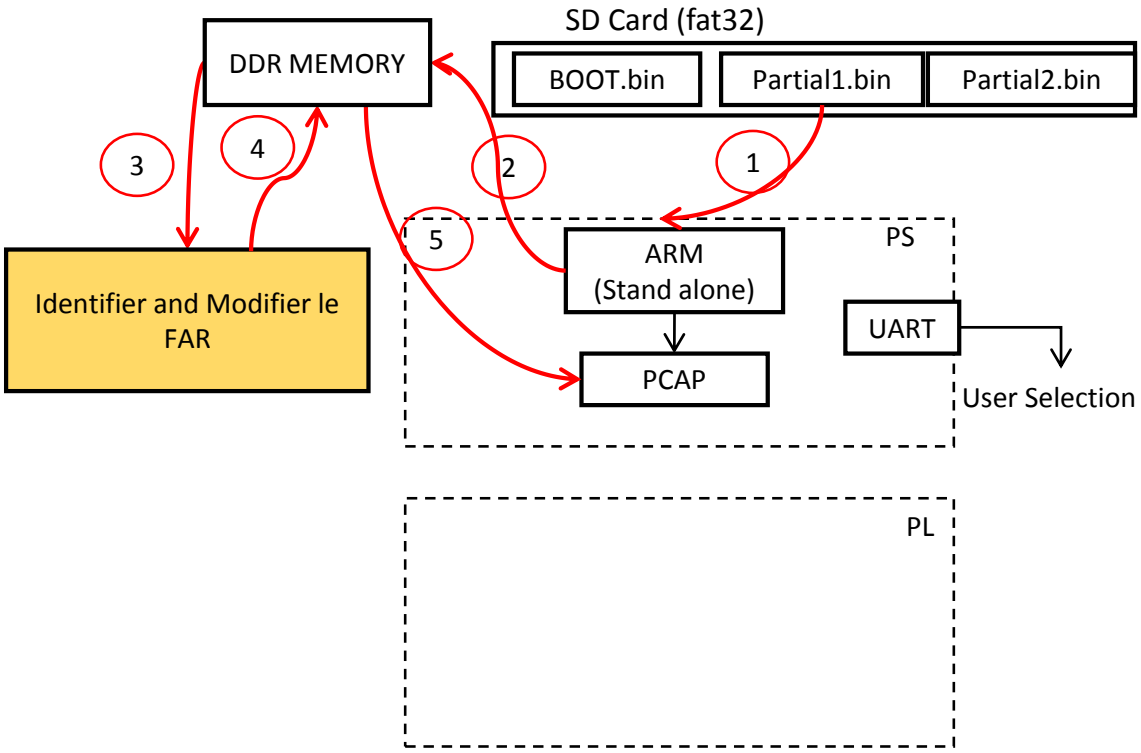

FIGURE 2.13 – *Processus de relocation en ligne proposé*

présenté dans la FIGURE 2.14.

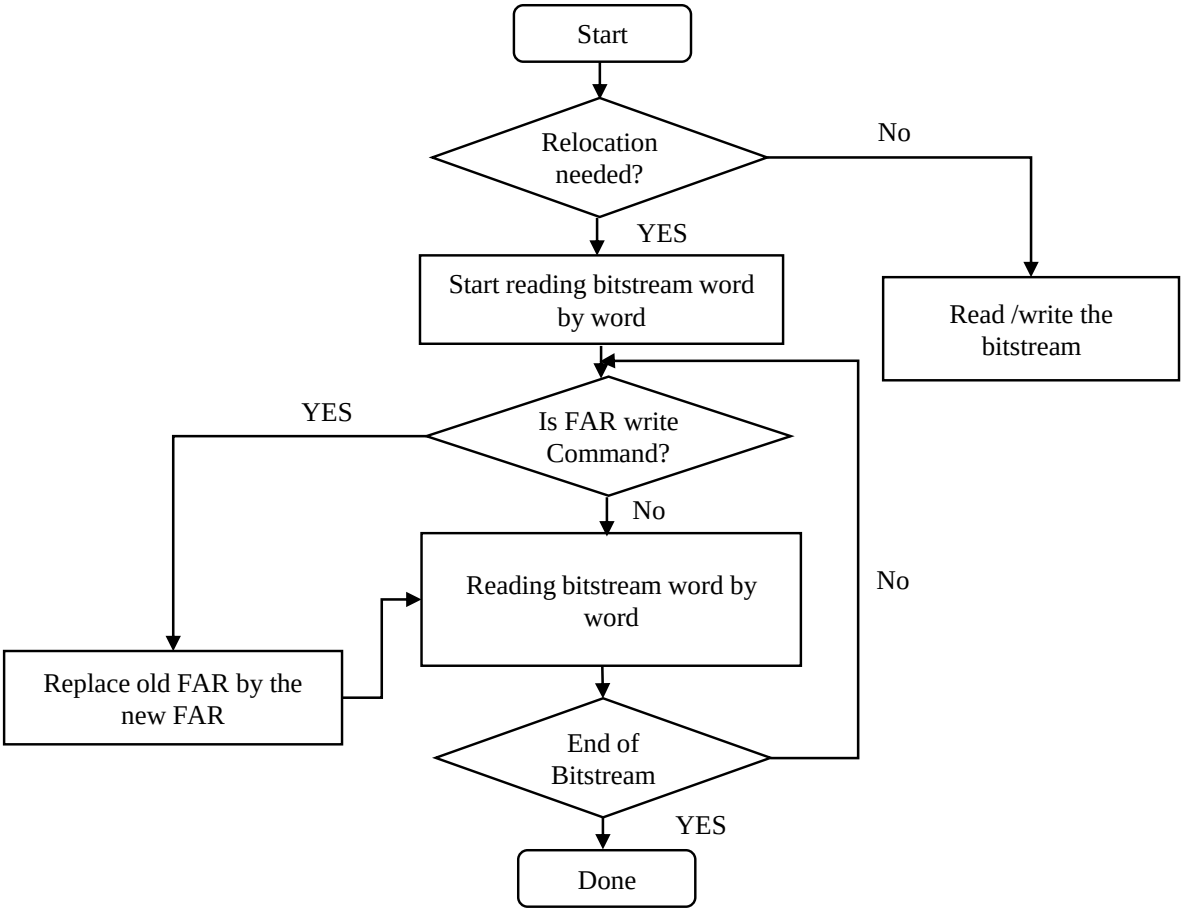

FIGURE *2.14 – Diagramme de manipulation des bitstreams*

3 Relocation des tâches de tailles variables

Dans l'état des technologies actuelles, les tailles des régions reconfigurables doivent être fixées en phase de conception et ne peuvent pas être modifiées en cours de l'exécution. L'utilisation des PRR fixes pose la problématique de l'efficacité d'utilisation en surface dans le cas où les tâches matérielles sont de tailles différentes et doivent être allouées et ré-allouées dans différents emplacements. L'inconvénient majeur de cette approche est la faible utilisation des ressources, car même un petit module PR occupe une région PR complètement. Ainsi un temps de reconfiguration élevé est atteint car la région reconfigurable doit être toute configurée même si un petit module l'occupe et n'utilise pas la totalité des ressources matérielles disponibles. Pour résoudre ce problème, nous présentons une nouvelle méthodologie d'adaptation de relocation et de placement des tâches matérielles de tailles différentes sur des régions reconfigurables.

3.1 Principe d'adaptation des régions reconfigurables

L'adaptation de la taille d'une région reconfigurable à la taille d'une tâche matérielle consiste à la partitionner en petite partition. L'objectif de cette méthode est de mieux exploiter les ressources disponibles dans le FPGA.

Le principe de la méthode proposée consiste à diviser la région reconfigurable en un certain nombre de petites régions, appelées partitions reconfigurables partielles (PRP) pour allouer des tâches matérielles de tailles différentes. La taille des partitions est définie pour la mise en place des petites tâches matérielles d'une application donnée. Pour le placement de chaque module reconfigurable, un ensemble de partitions sont fusionnées et définies comme une nouvelle partition. Cette technique permet de reconfigurer les tâches matérielles dans une partition reconfigurable plus grands que la partition initiale. La division de la région de PR en 'n' partitions permet un placement optimal des tâches matérielles de tailles variables, assure un taux d'occupation élevé des ressources matérielles dans FPGA et réduit le temps de reconfiguration.

La FIGURE 2.15 présente le principe de la méthode de partitionnement. La région reconfigurable dans cet exemple est divisée en trois partitions reconfigurables. Dans la première configuration, le module reconfigurable correspondant à la plus petite taille de module de calcul MA est placé dans la première partition, tandis que les autres partitions (PRP2 et PRP3) restent vides. Dans une seconde conception, deux partitions PR définies dans la conception initiale sont fusionnées pour créer une plus grande partition pour la mise en place du module reconfigurable MB. Et finalement dans la troisième conception nous fusionnons les trois partitions pour allouer le plus grand module MC.

3.2 Critères de partitionnement

Afin d'assurer un partitionnement optimal de la région reconfigurable et assurer un taux d'occupation élevé par les modules reconfigurables, ainsi qu'un faible temps de reconfiguration, nous définissons quelques paramètres d'évaluation.

Pour une efficacité maximale de partitionnement, la taille de partition associée à un PRM donné doit être choisie de manière à maximiser le taux d'utilisation (Ra) de la partition en termes de frames. Ra est calculé comme suit :

$$\mathrm{Ra_i} = \frac{\mathrm{W_i}}{\mathrm{N_i}}, \qquad \text{avec } \mathrm{i} \in \mathrm{CLB}, \mathrm{DSP}, \mathrm{BRAM} \tag{2.1}$$

où Wi est le nombre des frames du type i requis pour allouer un module donné et Ni est le nombre des frames de type i réservé pour la partition.

Pour une efficacité optimale d'utilisation de tout le système, il est nécessaire de maximiser le taux de partitionnement total ($\mathrm{Ra_T}$) ou bien de minimiser le taux de perte ($\mathrm{W_a}$)dans toutes les partitions. $\mathrm{Ra_T}$ et $\mathrm{W_a}$ sont calculés comme suit :

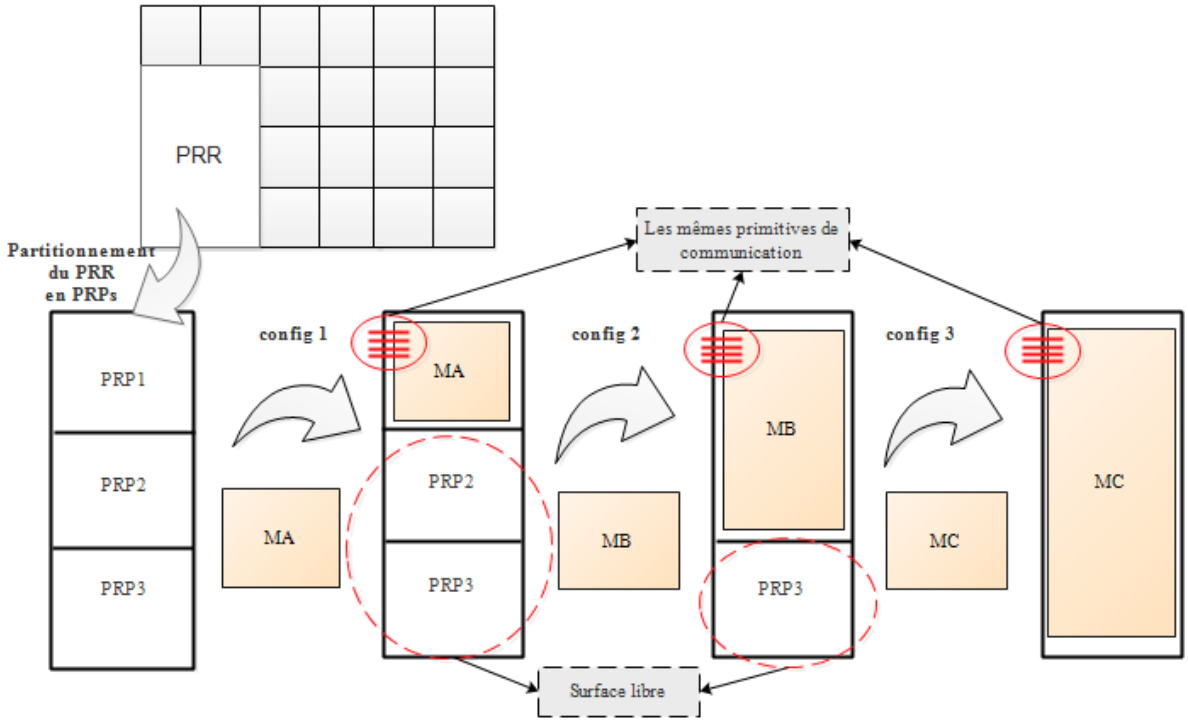

FIGURE 2.15 – *Principe de la méthode de partitionnement*

$$Ra_T = \frac{A_{req}}{A_{res}} = \frac{\sum W_i}{\sum N_i}, \quad \text{avec } i \in CLB, DSP, BRAM \tag{2.2}$$

$$W_a = \frac{A_{res} - A_{req}}{A_{res}} = 1 - Ra_T \tag{2.3}$$

Où A_{req} est la taille de la zone requise et A_{res} est la taille de la zone réservée.

3.3 Gestion de Communication

Pour réaliser une architecture reconfigurable qui permet d'allouer et de ré-allouer de tâches matérielles de tailles différentes sur des partitions PR, il est nécessaire de rendre tous les E/S des modules PR interopérables sur les partitions. Il faut donc que :

1. Tous les modules alloués dans une même région reconfigurable doivent avoir le même nombre des E/S, les E/S de même taille et de même nom.
2. Tous les partitions qui vont accueillir un même module reconfigurable, doivent avoir les Mêmes E/S et un placement des E/S identiques. En appliquant ces contraintes, nous allons avoir une consommation élevée des signaux de communication. La FIGURE 2.16 (b) montre que le nombre de signaux d'E/S dans le cas de partitionnement de la région PR est trois fois plus élevé que le nombre d'E/S dans le cas sans partitionnement.

Pour résoudre ce problème, nous avons besoin d'introduire un wrapper entre les partitions PR et le module de connexion introduit décrit précédemment. Ce wrapper a pour rôle

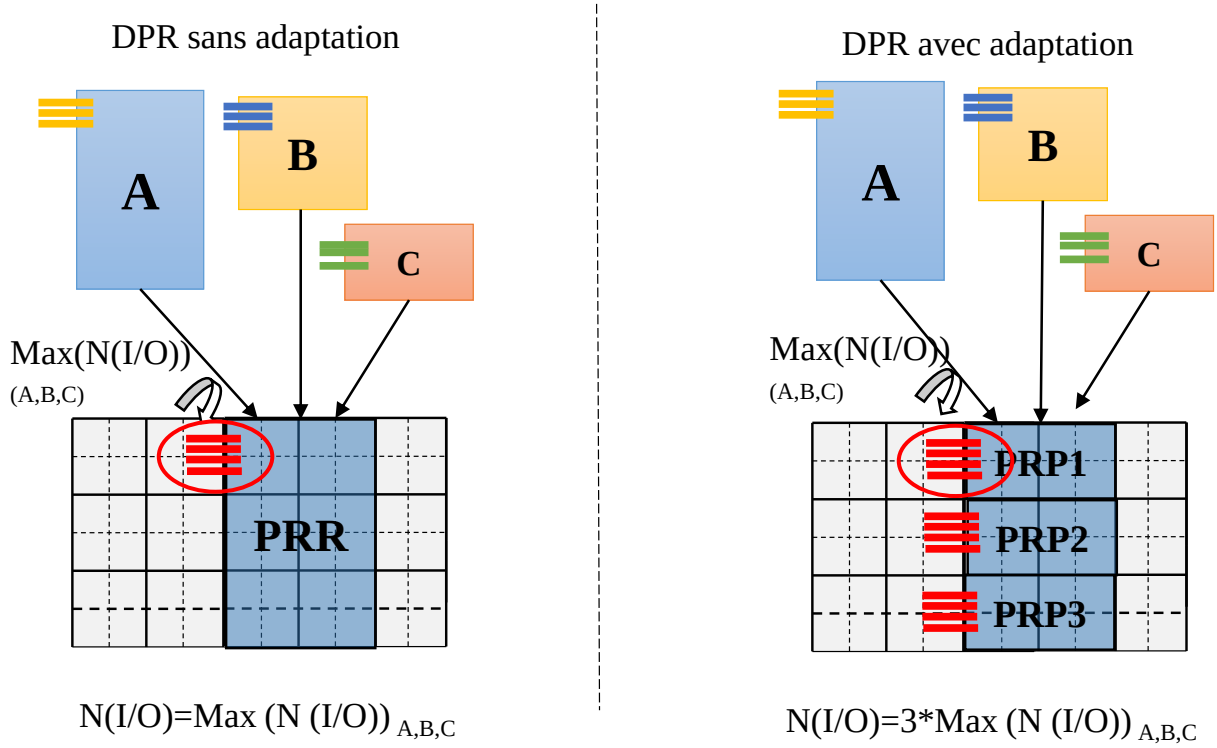

FIGURE *2.16 – Le nombre de signaux de connexion requis (a) sans partitionnement (b) avec partitionnement*

de :

— Gérer les données entre les différentes partitions
— Harmoniser les E/S entre les différents modules PR
— Connaître l'état du module

En plus de module de connexion qui a pour rôle d'adapter les signaux E/S de chaque module PR à sa destination(PRR) souhaitée, un wrapper spécifique à chaque région reconfigurable est ajouté pour mieux répartir les signaux E/S sur toutes les partitions de la région reconfigurable. Le but du wrapper est de connaitre le module PR présent dans la région PR et de laisser passer que les signaux requis afin de minimiser le nombre des signaux E/S utilisé.

La FIGURE 2.17 illustre le module complet ajouté (module de connexion + wrapper) au système pour assurer la communication entre les différentes parties du système.

Pour bien répartir et gérer les signaux entre les différentes partitions PR dans la région PR, il est nécessaire de calculer le nombre maximal des signaux E/S pour chaque partition afin que le plus petit module qui va occuper une seule partition trouve les signaux d'interconnexions requis. Le nombre d'E / S de chaque partition est définie par l'équation suivante :

$$\mathrm{NIO}_{\mathrm{partition}} = \mathrm{Max}\left(\frac{(\mathrm{NIO}(\mathrm{M_i}))}{\mathrm{NP}(\mathrm{M_i})}\right) \tag{2.4}$$

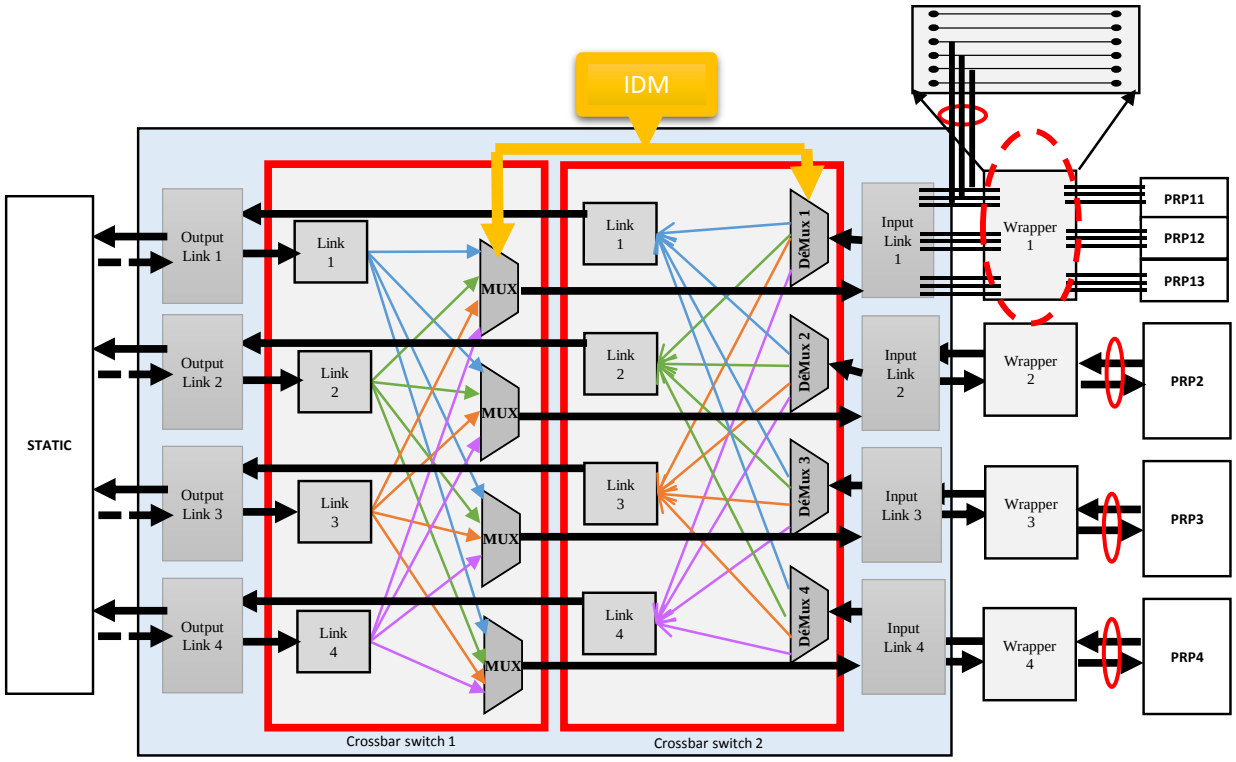

FIGURE *2.17 – Le module d'interconnexion complet*

avec NIO(M_i) est le nombre des signaux requis pour le module M_i et NP(M_i) est le nombre de partitions occupées par le module PR M_i.

Le nombre total des E/S dans chaque région reconfigurable est calculé comme suit :

$$NIO_T = NIO_{partition} * NP_T \tag{2.5}$$

avec NP_T est le nombre de partitions dans une région PR.

La FIGURE 2.18 montre un exemple de trois modules PR : M1, M2 et M3 de tailles différentes avec un nombre des signaux d'E/S différents qui sont respectivement 3, 5 et 9. Ces trois modules vont être placés dans une même région. Cette région PR est partitionnée en 3 petites partitions où M1, M2 et M3 vont occuper respectivement une seule partition, deux partitions et trois partitions. Dans cet exemple, le nombre d'E/S maximal calculé est égale 3. Dans ce cas le M1 peut être placé sur une des partitions de la région PR, le M2 qui occupe 2 partitions, il va choisir 5 signaux parmi 6 disponibles et le M3 va utiliser la totalité des signaux disponibles.

3.4 Flot de conception proposé pour la relocation des tâches de tailles variables

L'objectif est d'avoir une subdivision de la région reconfigurable en 'n' partitions PR qui pourront allouer et ré-allouer des tâches matérielles de tailles différentes. Pour cela nous

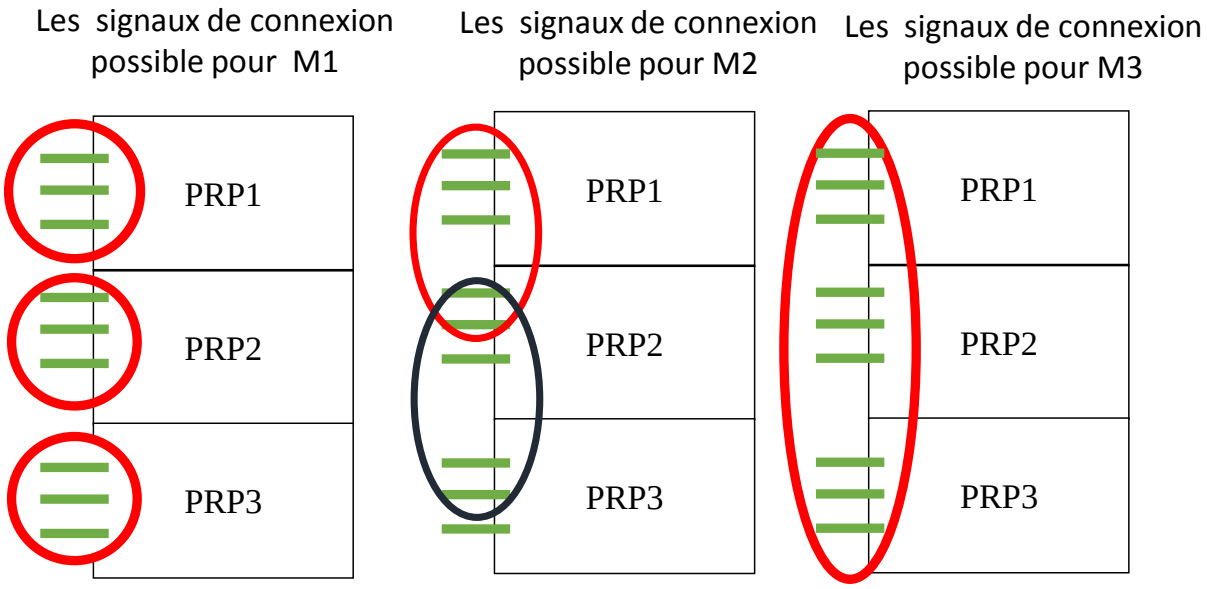

FIGURE *2.18 – Répartition des pin partition entre les différents partitions reconfigurables*

avons introduit des étapes supplémentaires dans le flot de conception des architectures reconfigurables qui supporte la technique de relocation.

Le design flot de conception correspond à la définition de la partie statique et les régions reconfigurables avec le partitionnement le plus fin. Toutes les informations sur les interfaces de communication et les chemins de routages sont extraites de la même façon et enregistrées dans un fichier UCF. Ensuite, un nouveau design est défini avec de nouveaux partitionnements afin de pouvoir implémenter des modules PR qui nécessitent plus de ressources. Le fichier UCF est apporté au nouveau pour contraindre les interfaces de communications et pour assurer le bon fonctionnement du système. Finalement, les bitstreams partiels sont générés et manipulés de la même manière.

La FIGURE 2.19 montre le flot global de conception pour la relocation des tâches de tailles variables. Ce flot est composé par cinq étapes :

Étape 1 : La mise en œuvre de la conception initiale. Cette étape est semblable au flot de conception standard PBPR. Les fichiers netlist de la partie statique et des tâches reconfigurables sont importés dans PlanAhead. Les partitions PR sont définies, selon le partitionnement le plus fin souhaité pour permettre un placement optimal des modules reconfigurables. À partir de ce partitionnement, un fichier netlist (NCD), correspond à l'implémentation de la partie statique avec les partitions reconfigurables vides, est généré.

Étape 2 : Réservation des connexions des PRM. Cette étape est similaire à la deuxième étape du flot de conception présente FIGURE 2.7. Pour assurer la relocation des tâches reconfigurables sur des partition de tailles variables, nous extrayons les informations sur les Pins Partitions (BEL et LOC) placés à chaque PRP dans un fichier UCF. Après, nous

modifions ce fichier, où nous calculons les nouvelles cordonnées (XY) des pins partitions pour les nouveaux emplacements dont le but est d'avoir les mêmes positions que celles de la région reconfigurable initiale. Aussi, nous assurons la modification de type des éléments logiques BEL de chaque pin partition avec celui qui convient dans la PRP d'origine. Finalement une nouvelle implémentation de la conception intiale est effectuée affin de placer tous les pins partition dans les positions souhaitées.

Étape 3 : Réservation des chemins de routage. Dans cette étape, les chemins de routage sont extraits et modifiés de la même façon que ceux de l'étape 3 présentés dans la FIGURE 2.7 et ils sont enregistrés dans le fichier UCF du système. Une nouvelle configuration de la conception initiale est générée pour garder les mêmes chemins de routage pour toutes les partitions PR.

Cette étape est répétée pour tous les nouveaux partitionnements souhaités.

Étape 4 : Création d'une nouvelle conception. Dans cette étape, une nouvelle conception est créée pour la mise en œuvre d'un nouveau partitionnement. Le fichier UCF crée dans la conception initiale et qui contient toutes les informations sur les pins partition et les chemins de routage, est importé pour cette conception. Avec ce fichier, nous pouvons également garder les mêmes placements des primitives de communication et réserver les mêmes chemins de routage pour les différentes conceptions.

Étape 5 : Génération des bitstreams. Pour toutes les conceptions créées avec tous les partitionnements souhaités, les bitstreams partiels (vides et des modules reconfigurables) sont générés. De plus, nous générons aussi un bitstream total contenant la partie statique avec les partitions vides.

4 Conclusion

Dans ce chapitre, nous avons proposé deux aspects de relocation des bitstreams. Un résumé des travaux présentés dans ce chapitre est illustré dans la figure FIGURE 2.20. Dans la première partie, nous avons commencé par proposer un flot de conception adapté aux nouvelles générations des FPGA qui assure la relocation des tâches reconfigurables. Dans ce flot la manipulation des fichiers binaires est effectuée automatiquement par le processeur ARM. Ainsi, l'aspect communication entre les modules est résolu par le développement des interfaces de communications génériques. Bien que le nouveau flot de conception proposé rende possible la relocation des tâches matérielles reconfigurables sur une architecture reconfigurable, le problème de relocation des tâches de tailles variables et l'exploitation efficace des ressources matérielles ne sont pas pris en considération. Pour répondre à ces exigences, nous avons proposé dans la deuxième partie une nouvelle méthode de partitionnement des zones reconfigurables. Dans ce flot de conception nous avons aussi déterminé des

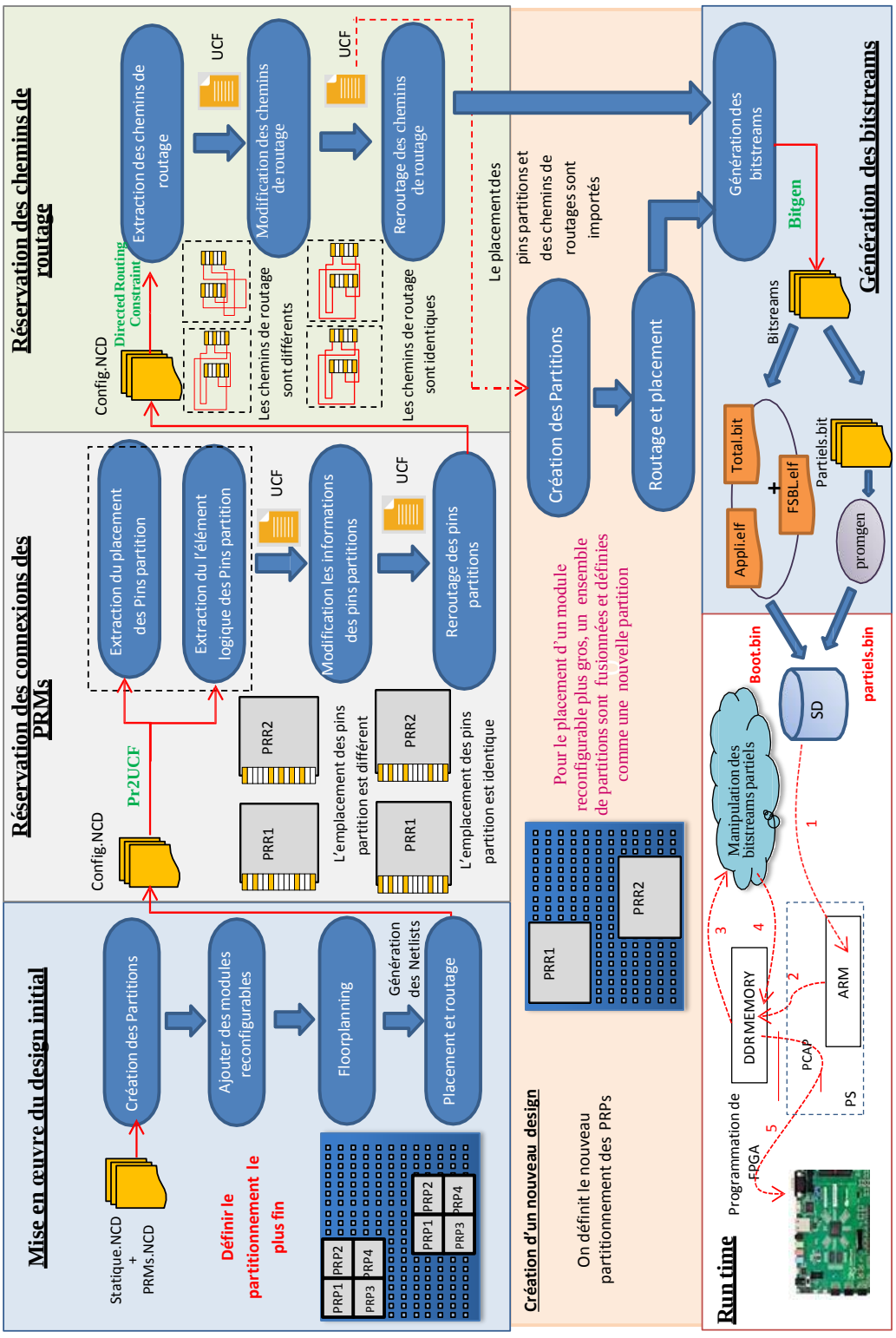

FIGURE 2.19 – *Le flot de conception pour la relocation des tâches de tailles variables*

nouvelles interfaces de communication entre les partitions qui jouent un rôle primordial dans la conception de tels systèmes. La méthode et le flot de conception proposées permettent de répondre aux contraintes de relocation des tâches matérielles de tailles différentes sur des partitions reconfigurables.

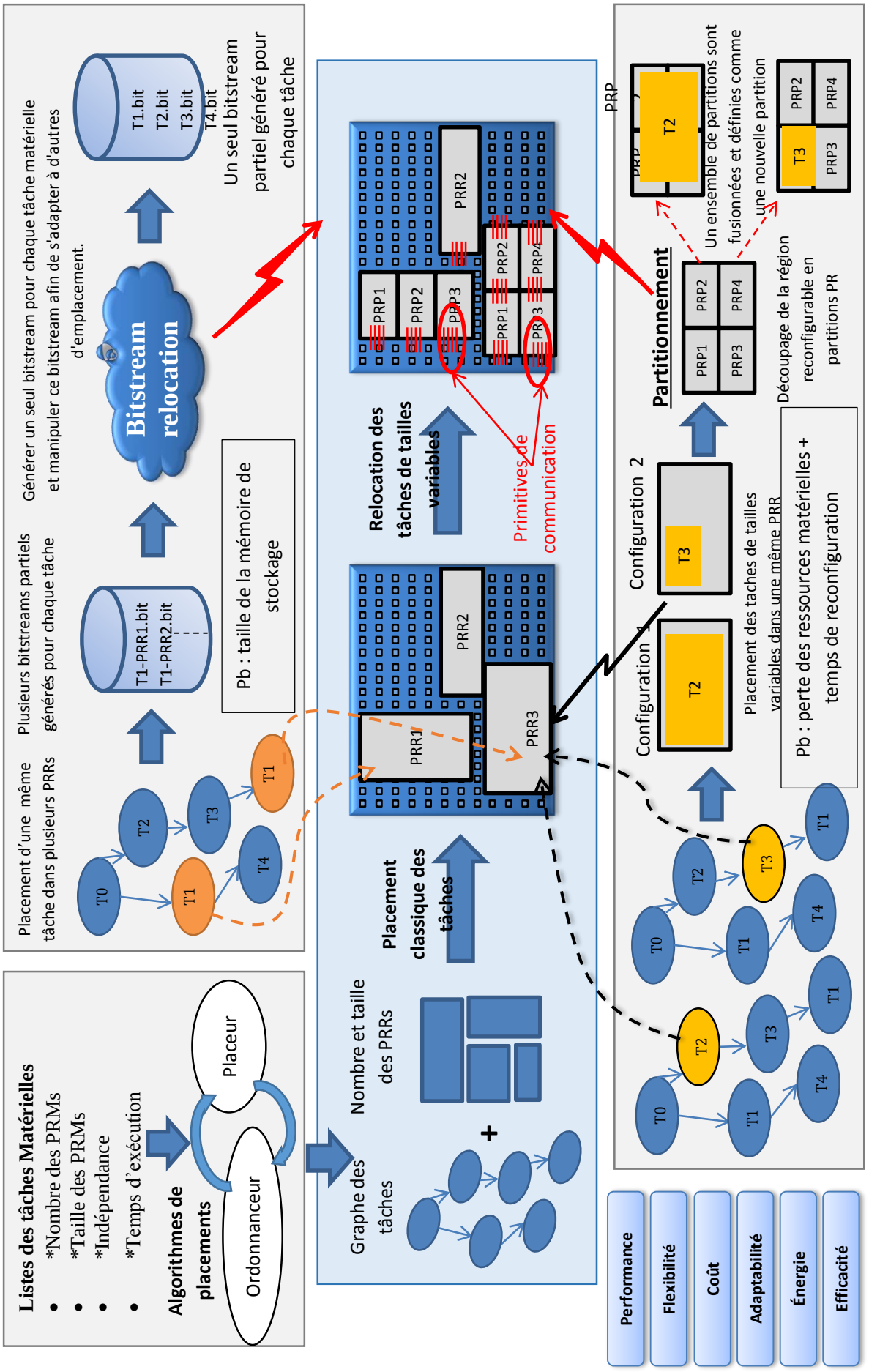

FIGURE 2.20 – *Méthodologie de relocation des tâches de tailles variables*

Chapitre 3

Formalisme et méthode de recherche automatique des zones reconfigurables

Sommaire

1 Introduction

La reconfiguration dynamique partielle permet d'apporter de la flexibilité aux architectures matérielles en modifiant dynamiquement certaines parties du système au moment de l'exécution. Cependant, les outils commerciaux existants ont évolué pour fournir des flots de reconfiguration partielle dynamique aboutis mais qui sont toutefois limités dans leurs fonctionnalités. Ils ont besoin de la participation du concepteur dans la spécification manuelle des formes et des emplacements de chaque région reconfigurable. Cette spécification manuelle nécessite non seulement au concepteur une connaissance approfondie du périphérique FPGA et de l'architecture du système, mais aussi de nombreuses tentatives d'essai et d'erreur pour trouver un découpage ou partitionnement du FPGA en zones reconfigurables (qu'on désignera à la suite du document par floorplanning) le mieux possible. Un floorplanning manuel conduit souvent à des résultats non optimaux en terme de ressources matérielles et de temps de reconfiguration et par conséquent une diminution des performances et de l'efficacité du système.

L'objectif du problème du floorplanning dans un FPGA est de placer des régions reconfigurables sur la puce de telle sorte que : chaque région affectée à un module satisfasse ses besoins en ressources, les régions pour différents modules ne se chevauchent pas les uns avec les autres, et une fonction de coût donnée est optimisée [CW06]. Floorplanning est une étape critique dans la méthodologie de conception des architectures reconfigurables puisque le concepteur doit prendre en compte toutes les contraintes, tout en essayant d'obtenir une bonne subdivision manuelle de la zone reconfigurable.

Comme le floorplanning est une étape de conception essentielle dans la méthodologie de conception DPR, nous proposons dans notre travail, une approche de floorplanning entièrement automatisée, capable de prendre en compte les contraintes de PR, tout en cherchant des solutions optimales par rapport à une fonction objective personnalisée. L'approche proposée décide les emplacements optimaux des PRR dans une surface reconfigurable hétérogène.

La suite de ce chapitre est organisée de la manière suivante. La première partie de ce chapitre présentera les problématiques posées par l'étape de floorplanning dans le flot de reconfiguration partielle dynamique. La deuxième partie sera consacrée à la formulation du problème du floorplanning ainsi qu'à la présentation des contraintes exigées par les architectures reconfigurables et les fonctions objectives requises. Le principe de l'algorithme de floorplanning proposé sera détaillé dans la troisième et dernière partie de ce chapitre.

2 Floorplanning des zones reconfigurables

Le floorplanning dans les FPGA hétérogènes et partiellement reconfigurables peut ressembler au floorplanning dans les circuits VLSI (Very-Large-Scale Intégration) [AM03].

Mais, les deux problèmes sont différents. Plusieurs aspects et considérations sont différents. Par exemple, dans les circuits VLSI, un ensemble de région est décrits en fonction de rectangles de taille fixe, tant que dans les circuits reconfigurables les modules sont décrits en terme des ressources matérielles. Dans notre travail nous sommes intéressés par résoudre le problème de floorplanning dans les FPGAs reconfigurables. Les sous-sections suivantes vont présenter plus en détail la problématique dans ce cas, ainsi les différentes précautions à prendre.

2.1 Problématique du floorplanning dans un FPGA

Pour les architectures reconfigurables, l'application est partitionnée en deux parties. Une première partie statique fixe est maintenue inchangée tout au long le processus de reconfiguration. Une seconde partie reconfigurable dont le contenu peut être changé dynamiquement au cours du fonctionnement. Cette deuxième partie est composée par plusieurs tâches matérielles qui pourront être implémentées sur différentes régions reconfigurables. Le processus de subdivision du FPGA en zones reconfigurables est appelé floorplanning.

Le rôle des algorithmes de floorplanning est de trouver une méthode qui subdivise le FPGA en un ensemble de zones reconfigurables tout en satisfaisant l'ensemble des objectifs et des contraintes.

Le problème du floorplanning peut être énoncé de la façon suivante :

— Soit PRM = $RM_1, RM_2....RM_m$ un ensemble de m PRM, ordonnés dans un graphe de tâches d'une application donnée.

— Soit $R_{RM_i} = \{N_{CLBi}, N_{BRAMi}, N_{DSPi}\}$ un vecteur 3-tuple représente respectivement le nombre de CLB, BRAM et DSP requis par le module RM_i.

— Soient W et H la largeur et la hauteur d'une architecture FPGA cible. Un système de coordonnées (0, 0, W, H) avec le coin supérieur gauche à (0, 0) et le coin inférieur droit à (W, H).

Le problème de floorplannig consiste à trouver $Z = \{Z_1, Z_2...., Zn\}$ un ensemble de **n** zones rectangulaires pour placer un ou plusieurs modules reconfigurables dans un FPGA cible. Chaque zone définie est caractérisée par les données suivantes :

— W_i : représente la largeur qui est le nombre de frames en colonne réservées par la zone Z_i

— H_i : représente la hauteur qui est le nombre de frames en ligne réservées par la zone Z_i

— $C_i(x_i, y_i)$: représente les coordonnées bas-gauche de la zone.

— $\mathcal{M}(Z_i) = \{N_{CLBz_i}, N_{BRAMz_i}, N_{DSPz_i}\}$ est un vecteur représentant respectivement le nombre de CLB, BRAM et DSP requis par la zone Z_i.

— avec $0 \leq$ i $\leq$ **n**

La figure FIGURE 3.1 représente un exemple de floorplanning d'une zone reconfigurable dans un FPGA et illustre toutes les données spécifiques de la zone définie.

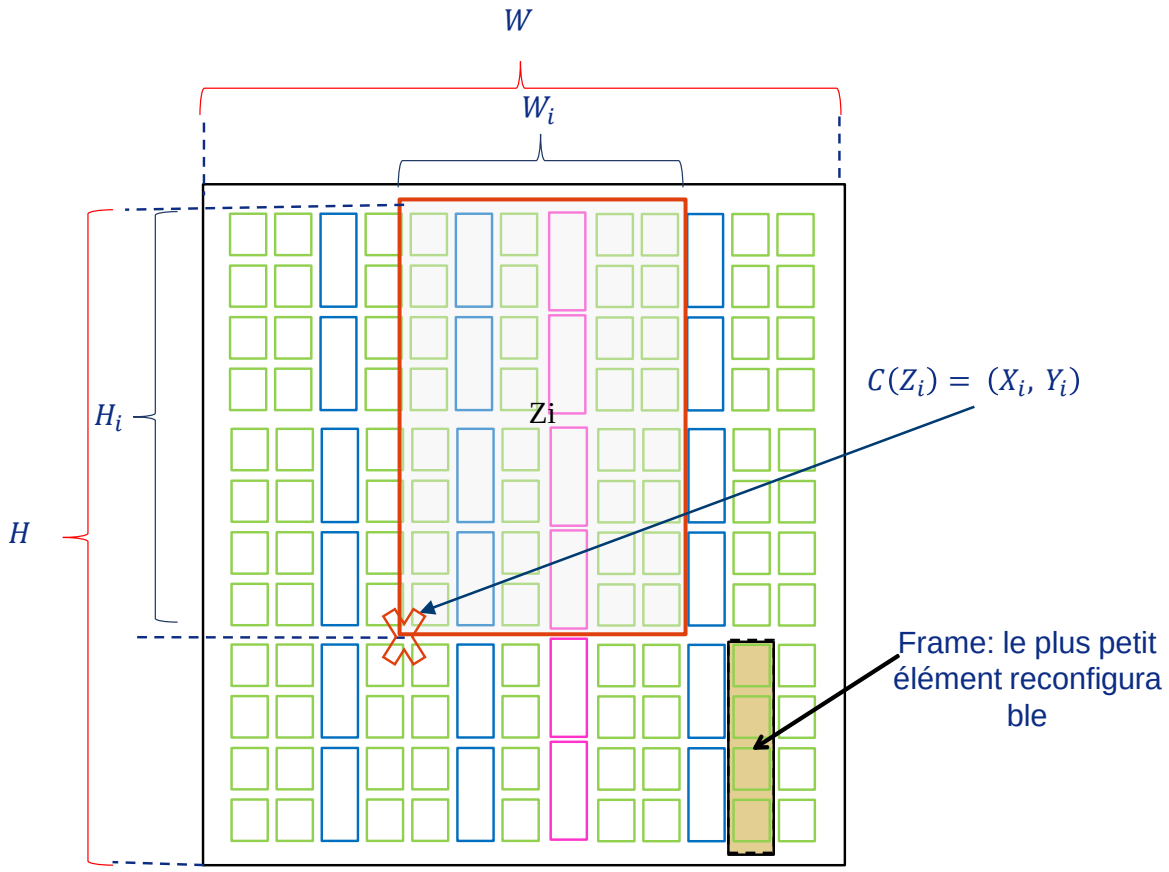

FIGURE *3.1 – Exemple de floorplanning*

2.2 Contraintes à respecter

Une région reconfigurable peu accueillir différentes tâches reconfgurables à différents moments, et sa taille devra avoir une taille suffisamment grande pour être capable d'accueillir la plus grande des tâches. Cette zone est définie en prenant en considération les besoins maximaux en ressources pour chaque type de ressource entre les différents tâches allouées. En plus, les algorithmes de floorplanning sur des architectures reconfigurables hétérogènes doivent prendre en compte la présence des différentes ressources. La présence de ressources du type différent restreint l'emplacement des zones à un sous-ensemble des positions. Par conséquent, notre problème se subdivise en trois aspects :

— une région reconfigurable même si elle est rectangulaire, elle n'est pas caractérisée que par une hauteur et une largeur fixe, mais elle nécessite aussi un nombre bien déterminé de ressources matérielles.

— les FPGA modernes disposent des ressources hétérogènes ayant des positions spécifiques dans la puce, donc une région reconfigurable peut avoir besoin de couvrir différents types de ressources en interdisant certains placements et formes de la région.
— les régions reconfigurables doivent couvrir des frames complets par rapport à la matrice du FPGA.

L'objectif de floorplanning sur des FPGA hétérogènes partiellement reconfigurables consiste à trouver un emplacement de différentes zones reconfigurables sur la puce tout en respectant :

— Chaque région PR avec une hauteur et une largeur(H_i, W_i) doit couvrir le nombre et le type de ressources nécessaires pour les PRM alloués,
— Pas de chevauchement entre les différentes régions PR,
— Les zones couvertes ne dépassent pas les frontières de la puce (H_{max}, W_{max}).

3 Approche de floorplanning proposée

Le floorplanning des zones sur une surface hétérogène reconfigurable est une problématique complexe. Pour la résoudre, nous proposons d'utiliser une méthode analytique basée sur la programmation linéaire mixte. La programmation mathématique est l'outil le plus efficace pour résoudre un problème décisionnel. La méthode proposée dans notre travail est composée en deux parties : la première consiste à modéliser le dispositif FPGA utilisé en tenant compte de l'hétérogénéité de la surface reconfigurable et la deuxième partie est consacrée à formaliser le problème de recherche qui est basée sur l'utilisation de la programmation linéaire mixte, pour un floorplann optimal des zones reconfigurables.

3.1 Principe général

Nous avons vu dans la section 3 du chapitre 1 que les méthodes utilisées pour aborder le problème de floorplanning sont nombreuses. Cependant, notre travail repose sur la technique de programmation linéaire mixte en nombres entiers (PLMNE). L'existence et la disponibilité d'outils informatiques génériques et puissants, tels que CPLEX, GLPK, GAMS et MATLAB, permettant d'appliquer aisément ces méthodes peuvent justifier notre choix. En plus, la programmation PLMNE permet une modélisation plus flexible en comparaison aux autres techniques de recherche et d'optimisation citées. L'ajout, la suppression et la modification de contraintes ainsi que la modification de l'objectif d'optimisation se font facilement sans la nécessité d'une ré-formulation particulière.

Définition : Un programme linéaire en nombres entiers (PLNE) permettant d'exprimer un problème de minimisation/de maximisation. Un problème PLNE peut s'écrire respectivement sous les formes standards suivantes [BV04] :

$$\begin{array}{lll} \text{Min} & & c^T x \\ \text{Sujet} & & Ax = b \\ & & x \geq 0 \end{array} \tag{3.1}$$

où A est une matrice constante de m lignes et n colonnes. b et c sont des vecteurs constants de dimension n. $x = (x_1, x_2 ..... x_n)$ est un vecteur de variables de décisions entières positives. On parle d'un programme linéaire mixte en nombres entiers lorsqu'il inclut à la fois des variables entières et des variables réelles. La première ligne de cette formulation représente la fonction objective à optimiser et les lignes suivantes sont les contraintes du problème.

Formalisation du problème de floorplanning L'étape du floorplanning manuel dans le flot de conception classique de Xilinx est remplacée par une exécution automatique de notre algorithme séparée en dehors de l'outil Planahead comme le montre la FIGURE 3.2.

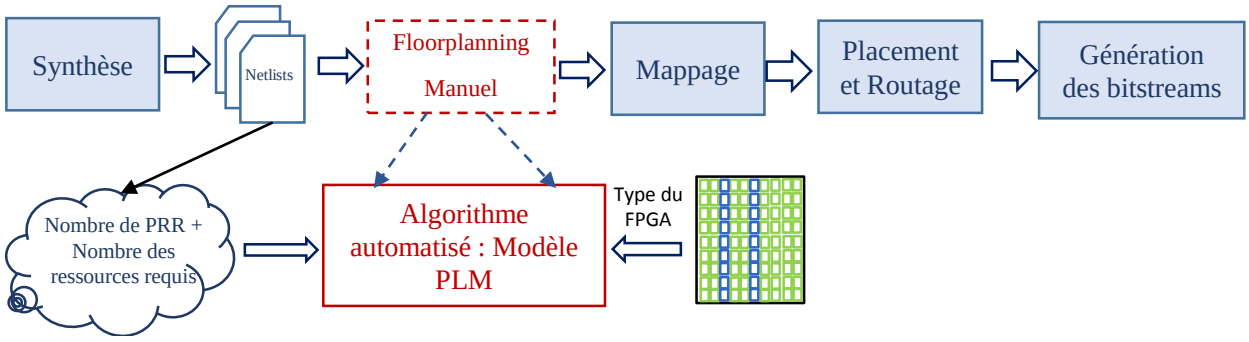

FIGURE *3.2 – Le nouveau flot de conception en intégrant le modèle PLMNE*

La résolution automatique du problème floorplanning à base d'une programmation linéaire est un problème complexe, il s'agit d'un problème NP-difficile [GJ79] du fait du grand nombre de paramètres et de contraintes à considérer. Pour formaliser le problème du floorplanning, nous avons besoin d'introduire trois fichiers. Le premier fichier est une modélisation de l'application. Il contient des informations sur le nombre des régions reconfigurables avec leurs besoins en ressources logiques. Le deuxième fichier contient la formulation du problème du floorplanning à résoudre. Il introduit les formulations mathématiques des contraintes et de la fonction objective souhaitée. Le dernier fichier contenant la modélisation de type du FPGA utilisé.

Résolution : Le solveur est un outil utilisé pour résoudre des programmes linéaires. Il est basée sur des techniques mathématiques avancées permettant de résoudre des problèmes complexes d'optimisation. Il existe des nombreux outils logiciels. Certain de ceux

sont des solveurs commerciaux de renommée mondiale tels que IBM ILOG CPLEX [IBM], FICO-XPRESS-MP [FIC], GUROBI [gur], et d'autres sont des logiciels libres tels que PCx [CMW96], lpsolve [BEN+04], glpk [M+08]. La majorité des solveurs modernes, comme CPLEX et Gurobi utilisent l'algorithme d'évaluation et de séparation (Branch-and-Bound [TCM86]) pour résoudre des problèmes de programmation linéaire en nombres entiers.

Dans nos travaux, la formulation mathématique du problème (données, variables, contraintes et objective) est défini à l'aide de langage GNU MathProg [Mak00]. Afin de compiler ce problème et de le convertir en un format standard de programmation linéaire telle que .mps ou .lp il est nécessaire d'utiliser l'outil Glpk (GNU Linear Programming Kit). Après, pour résoudre ce problème linéaire, le modèle peut être résolu directement avec glpk ou par un autre solveur. Mais, l'utilisation de glpk pour résoudre le problème n'est pas recommandée car il nécessite beaucoup de temps. Dans notre travail, le solveur Gurobi est utilisé pour générer la solution finale du problème. Le résultat du solveur Gurobi est un fichier texte contient les emplacements et les dimensions optimaux de chaque PRR. La FIGURE 3.3 illustre le processus du solveur.

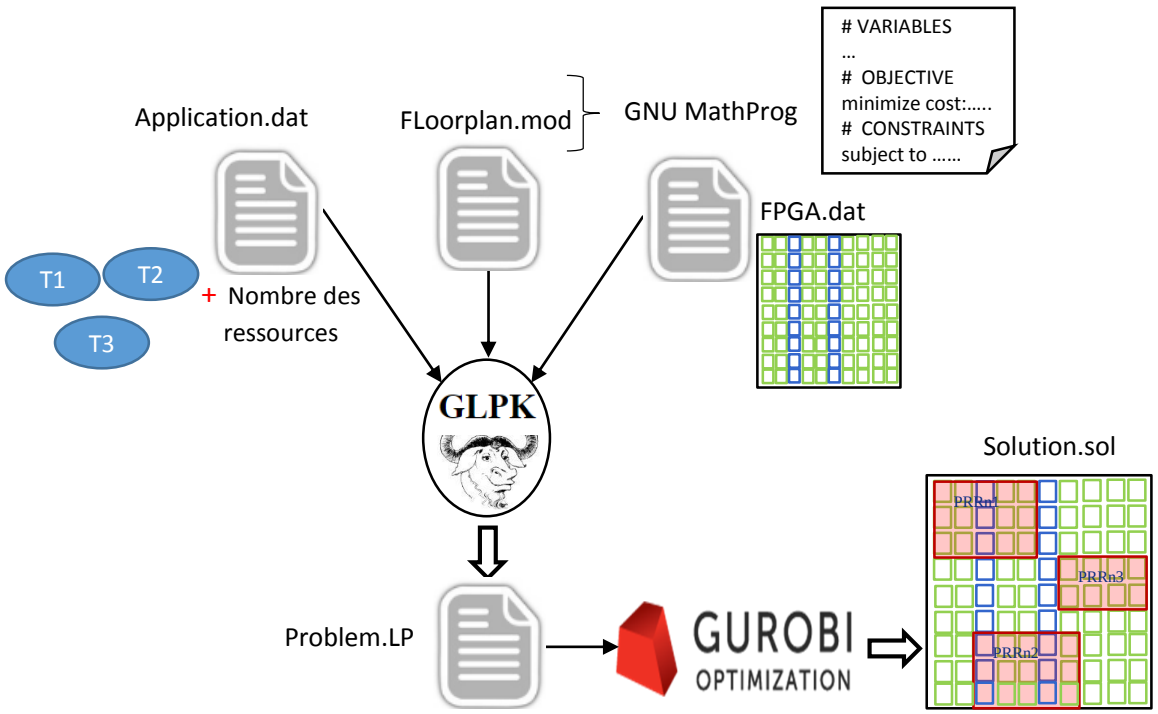

FIGURE *3.3 – Pocessus du solveur*

Dans les sous-sections suivantes, nous proposons de modéliser le FPGA, modéliser l'application et formaliser le problème en programme linéaire mixte qui formule le problème de floorplanning en un problème d'optimisation sous des contraintes. Nous décrivons les noms utilisés pour les données et les variables, puis les différentes contraintes et finalement la fonction objective.

3.2 Modélisation du FPGA

Dans cette section, l'architecture du FPGA est analysée et la représentation des ressources matérielles est simplifiée pour faciliter la formulation de l'algorithme PLMNE, en réduisant le plus possible le besoin des variables et en facilitant le processus de recherche dans l'algorithme.

3.2.1 Simplification de l'architecture du FPGA

Un FPGA est constitué essentiellement de trois types de ressources, à savoir, des CLB, des DSP ainsi que des BRAM. Il est important de définir un modèle qui tient compte de l'hétérogénéité de la zone reconfigurable. Pour simplifier le problème, les frames sont considérés comme les plus petites unités reconfigurables d'un FPGA. L'utilisation d'une matrice dont les coordonnées entières se basent sur le nombre des frames, réduit à la fois l'espace de recherche et garantit les contraintes de la reconfiguration dynamique, puisque les régions sont placées à l'aide de coordonnées entières. Donc, il nécessaire de caractériser chaque FPGA et de définir le nombre et le type de ressources disponibles pour chaque frame (FIGURE 3.4).

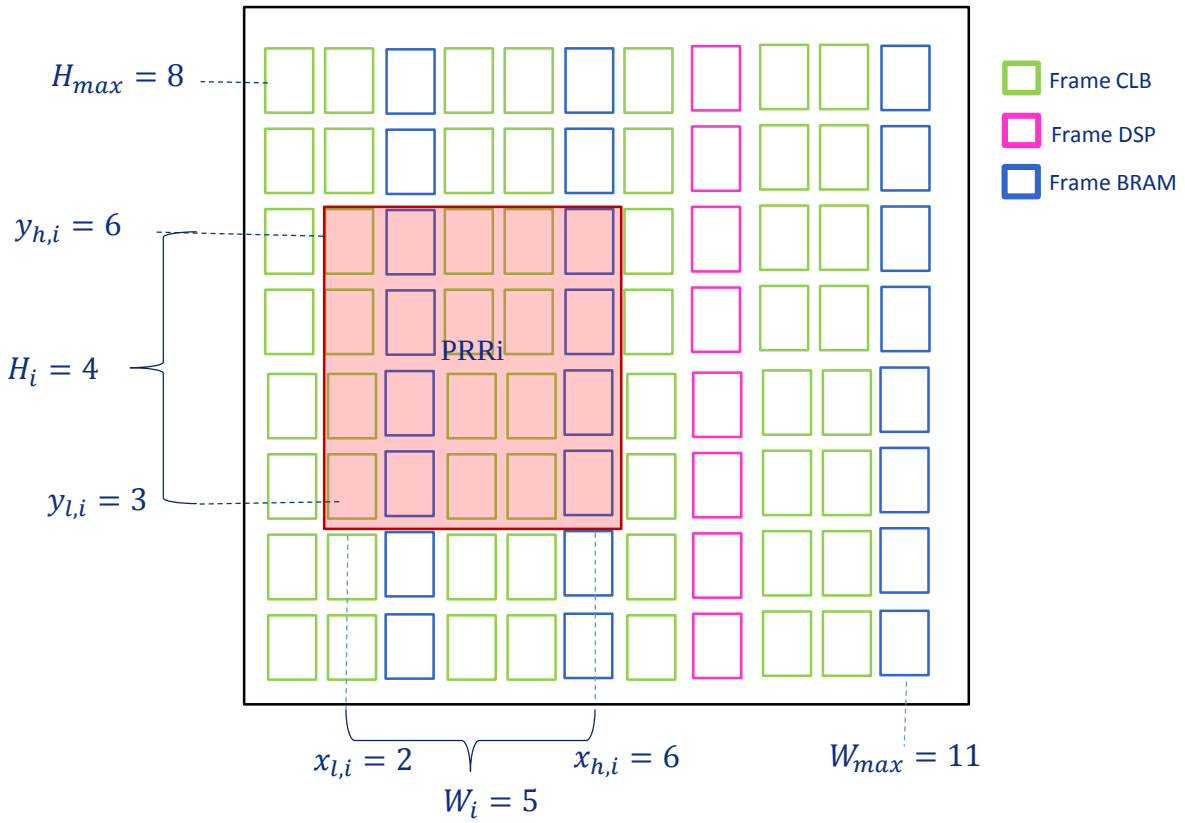

FIGURE *3.4 – Modélisation d'un FPGA*

A titre d'exemple, un FPGA peut être décrit en utilisant une matrice de carreaux avec

8 lignes et 11 colonnes. La figure FIGURE 3.4 montre comment la modélisation est réalisée : pour chaque zone reconfigurable comme celle représentée sur la figure en rouge, un ensemble de variables. Sur l'axe X (axe Y), des variables représentant les positions gauche et droite (bas-haut) ainsi la largeur (hauteur) de la zone.

3.2.2 Partitionnement du FPGA

Pour simplifier la modélisation de l'architecture de FPGA [KSCC10], [BHW+14], les frames peuvent être regroupées et n'est pas considérées séparément : même s'il y a différentes ressources disponibles dans différents emplacements dans la puce, la structure du FPGA est assez régulière et il y a de grandes zones caractérisées par le même type de frame. Par conséquent, nous divisons le FPGA en plusieurs zones rectangulaires nommées portions. Les portions sont des rectangles composées par le même type de ressources où on regroupe des frames de même type. Une technique simple qui peut être utilisée pour créer le partitionnement du FPGA est la suivante :

1. Le FPGA est scanné de haut en bas, de gauche à droite, et la première frame rencontrée qui ne fait pas partie d'aucune portion est sélectionnée, puis une nouvelle portion est créée contenant ce type de frames ;
2. La portion est étendue sur le côté droit jusqu'à ce que d'autres frames libres du même type soient rencontrées ;
3. La portion est étendue sur le côté inférieur jusqu'à ce que toutes les frames libre de la rangée soient du même type ;
4. Dans le cas, où il y a encore des frames libres, le processus est répété de l'étape 1 jusqu'à ce que toutes les frames appartiennent à une seule et même portion.

La FIGURE 3.5 montre un exemple de zone partitionnée en portion sur le FPGA. Dans ce cas 8 portions suffisent pour caractériser correctement un FPGA en termes de ressources DSP, BRAM et CLB. Nous avons associé P1= 16 frames CLB, P2=8 frames BRAM, P6= 8 frames DSP.

3.3 Modélisation de l'application

Pour déterminer correctement le nombre et la taille des PRR dans le processus de recherche du floorplanning, un fichier d'extension .dat est défini comme entrée du problème pour spécifier les besoins de l'application : le nombre des PRR ainsi leurs besoins en ressources matérielles (CLB, BRAM, DSP).

A partir des fichiers netlists, nous pouvons obtenir les informations sur les ressources matérielles de tous les PRM et les connexions entre eux. Le nombre de ressources requises pour une PRR est le maximum des ressources entres les PRM qui seront allouées à cette PRR. Ce nombre est déterminé en fonction du nombre de frame de chaque type de ressources logiques (voir annexe1).

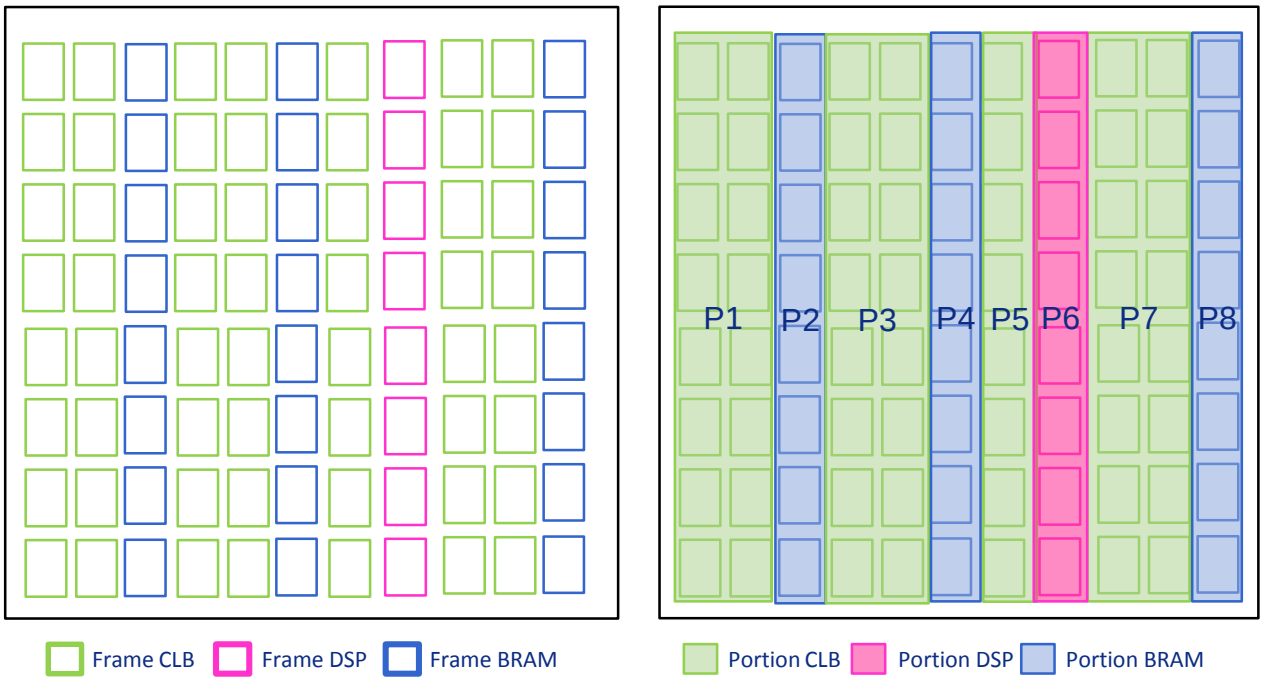

FIGURE 3.5 – *Une architecture hétérogène simplifiée*

3.4 Formulation en PLMNE

La solution du problème est une affectation de valeurs aux variables représentant les décisions qui doivent être prises. Une solution est réalisable si elle satisfait toutes les contraintes du problème exprimées en fonction des variables, représentant les conditions ou limites (physiques, logiques, ...) à prendre en compte lors de la recherche de solutions réalisables. Une solution optimale est une solution réalisable pour laquelle la fonction objective prend sa valeur maximale/minimale.

Dans cette partie, nous présentons les différents notations, variables, contraintes et la fonction objective qui formalise le probleme sous la forme standard.

3.4.1 Données

Dans notre modèle, les entrées du problème seront d'une part les informations sur le type de FPGA utilisé, où un FPGA est décrit complètement en fonction des portions comme il est expliqué dans la section 3.1. Chaque portion est décrite par sa position et son type de frame. D'autre part, les informations qui caractérisent une application bien déterminée où nous définissons le nombre de régions reconfigurables avec leurs besoins en ressources, seront aussi définies comme entrées.

Dans nos travaux, nous utilisons les ensembles et les paramètres du modèle suivants :

— $P = P_1, P_2, .., P_m$: Ensemble des portions avec m étant le nombre de portions dans un FPGA,

— $R = 1, 2, .., R_{max}$: Ensemble des lignes dans le FPGA numérotées de 1 à R_{max},

— N : Nombre de régions reconfigurables à placer,

— T : Ensemble de types de ressources (CLB, DSP, BRAM),

— W_{frame} : Nombre de ressources par frame

— $C_{n,t}$: Nombre de ressources de type T requis par la région reconfigurable n,

— $rp_{p,r}$: Nombre binaire égale à 1 si la ligne r est dans la portion p, et égale à 0 sinon,

— $d_{p,t}$: Nombre des ressources de type T disponible dans la portion p,

avec $t \in T$, $p \in P$ et $n \in N$.

3.4.2 Variables

L'objectif principal de l'algorithme floorplanning consiste à trouver les emplacements et les dimensions de chaque région reconfigurable. Pour cela nous devons définir des variables nécessaire pour formaliser les contraintes à respecter. Nous avons besoin également de définir quelques variables pour calculer la quantité d'intersection de ressources entre la portion p, la région n et la ligne r afin de déterminer correctement les contraintes d'occupation des ressources.

Les variables du modèle sont définies comme suit :

— $a_{n,r}$: variable binaire égale à 1 si et seulement si la région **n** occupe la ligne r,

— x_n : variable de type entier positif ($\geq$ 1) correspond à la position la plus à gauche de la région n,

— w_n : variable de type entier positif ($\geq$ 1) représente la largeur de la région n,

— yl_n : variable de type entier positif ($\geq$ 0) correspond à la ligne la plus bas occupée par la région n,

— yh_n : variable de type entier positif ($\geq$ 0) correspond à la ligne la plus haut occupée par la la région n,

— h_n : variable de type entier positif ($\geq$ 0) représente la hauteur de la région n,

— $l_{n,p,r}$: variable de type entier positif ($\geq$ 0) définie la quantité d'intersection de ressources entre la portion p, la région n et la ligne r

— $k_{n,p}$: variable binaire égale à 0 si la région n et la portion p ne s'intersecte pas,

La figure FIGURE 3.6 illustre toutes les variables nécessaires pour le modèle PLMNE.

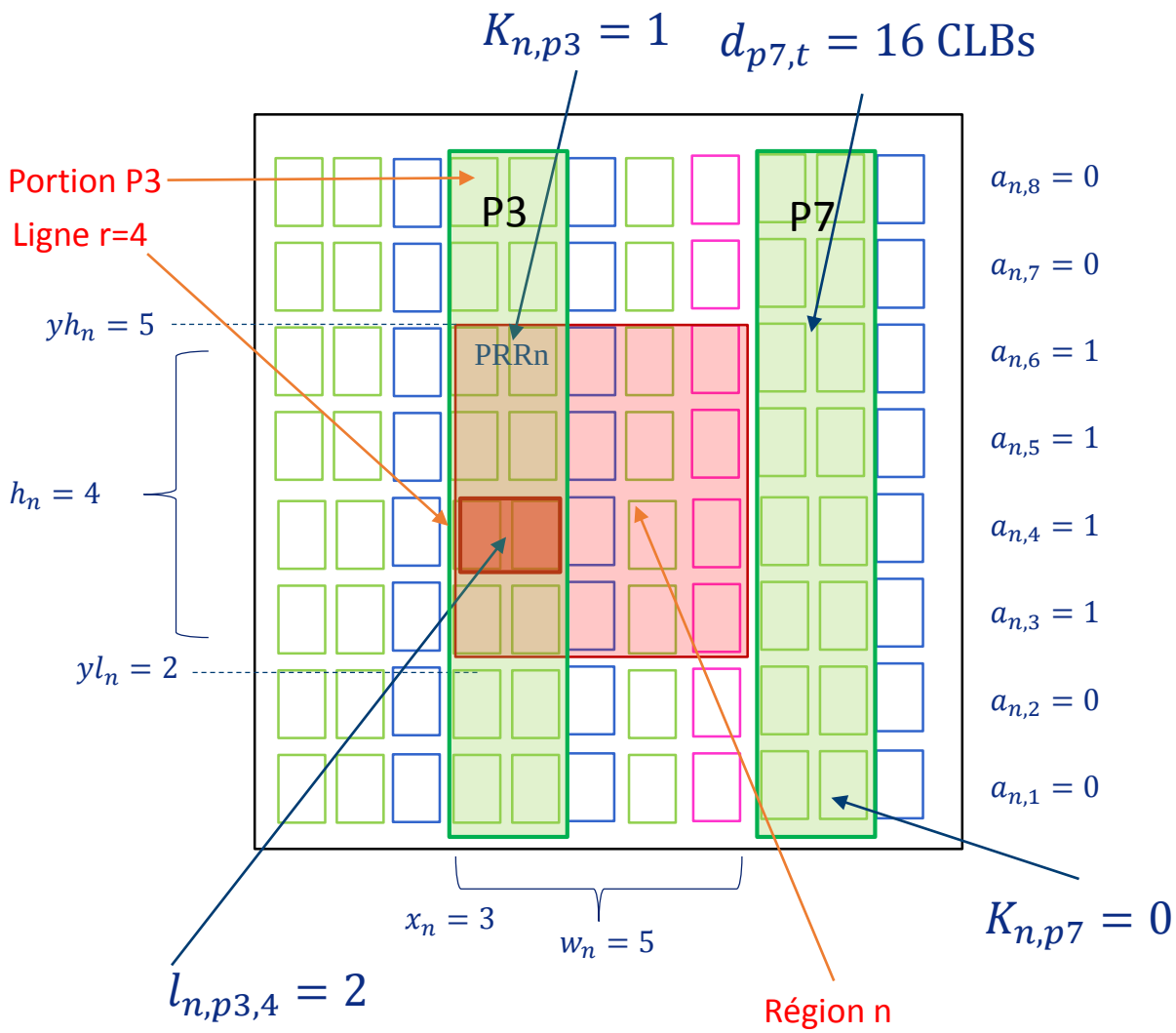

FIGURE 3.6 – *Présentation des Variables du modèle PLMNE*

3.4.3 Contraintes

Durant le processus de floorplanning d'une zone, l'algorithme doit respecter trois contraintes majeures :

- Éviter le chevauchement avec les autres zones voisines
- Éviter que la zone soit placée sur les bords du circuit
- Assurer que chaque région dispose tous ses besoins en terme de ressources

3.4.3.1 Contrainte de chevauchement

Soit PRR_{n1} et PRR_{n2} deux zones destinées à être définies sur la surface disponible de FPGA. La contrainte de non-chevauchement impose à l'algorithme de floorplanning de déterminer l'ensemble des positions que peuvent être affectées aux zones de sorte que l'intersection entre elles soit vide.

En effet, chaque paire de zones a une distance suffisante entre leurs coordonnées sur au

moins une dimension pour éviter un chevauchement. La condition de non-chevauchement revient à vérifier qu'au moins l'une des quatre conditions suivantes pour chaque paire de zones PRR_{n1} et PRR_{n2} avec $n1, n2 \in N$.

$$\begin{aligned} x_{n1} + w_{n1} &\leq x_{n2} \\ yl_{n1} + h_{n1} &\leq yln2 \\ x_{n2} + w_{n2} &\leq x_{n1} \\ yl_{n2} + h_{n2} &\leq yln1 \end{aligned} \tag{3.2}$$

La figure FIGURE 3.7 montre un exemple de chevauchement de deux zones en confirmant qu'aucune des quatre équations est vérifiée.

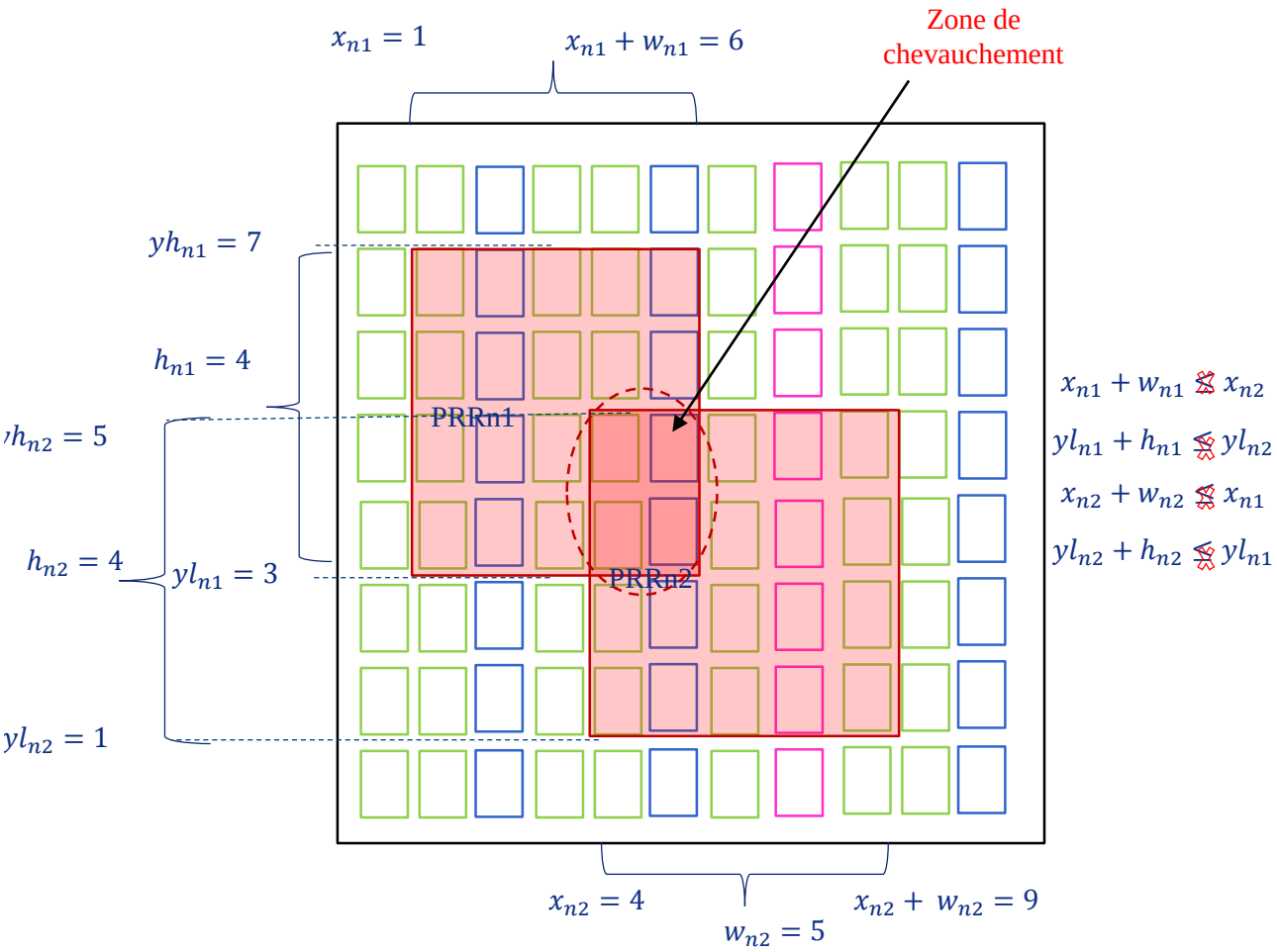

FIGURE *3.7 – Contrainte de chevauchement*

3.4.3.2 Contrainte liée aux bords

Soit PRR_n une zone reconfigurable de taille (W_n, H_n) , destinée à être positionnée dans le FPGA. Et soit un FPGA de taille (W_{max}, H_{max}). Le floorplanning est dit faisable

si et seulement si les coordonnées de la zone vérifient les conditions suivantes :

Limite du côté gauche-droit de PRR_n sur l'axe x :

$$\begin{aligned} x_n + w_n &\leq W_{max} \\ x_n &\in [\,0,\ W_{max}[\end{aligned} \tag{3.3}$$

Limite du côté bas-haut de PRR_n sur l'axe y :

$$\begin{aligned} y_n &\in [\,0,\ H_{max}[\\ yl_n + w_n &\leq H_{max} \end{aligned} \tag{3.4}$$

Définition de la hauteur par rapport aux lignes occupées :

$$\begin{aligned} &\forall n \in N \\ h_n &= \sum_{r \in R} a_{n,r} \end{aligned} \tag{3.5}$$

L'écart entre yln et yhn par rapport à la hauteur de la région.

$$\begin{aligned} &\forall n \in N \\ yh_n - yln + 1 &= h_n \end{aligned} \tag{3.6}$$

3.4.3.3 Contraintes de gestion des ressources

La reconfiguration dynamique nécessite un partitionnement du FPGA en PRR qui sont des régions reconfigurables dynamiquement. Cependant une région reconfigurable accueille différentes tâches à différents moments. En outre sa surface qui doit être suffisamment large pour tenir compte de toutes les configurations requises. La surface en frames, pour une région PRR est calculée en fonction des besoins en ressources maximales pour chaque type de ressource. Pour garantir que la région reconfigurable couvre tous les besoins en ressources pour chaque type de ressource $R_{n,t}$, la condition suivante doit être vérifiée :

$$\begin{aligned} &\forall n \in N, \forall t \in T \\ R_{n,t} &= \sum_{p \in P, r \in R} l_{n,p,r}.d_{p,t} \geq C_{n,t} \end{aligned} \tag{3.7}$$

avec $R_{n,t}$ est le nombre de ressource de type t défini par l'algorithme dans la région n, $C_{n,t}$ est le nombre des ressources de type t requis (donnée comme entrée) par la région reconfigurable n, $l_{n,p,r}$ est une variable qui définit la quantité d'intersection de ressources entre la portion p, la région n et la ligne r, $d_{p,t}$ est le nombre de ressources de type t disponible dans la portion p.

3.4.4 Fonction objective

La fonction objective dans l'algorithme de floorplanning consiste à trouver une solution optimale pour chaque région reconfigurable tout en minimisant les pertes de ressources matérielles utilisées et en réduisant les communications inter-régions (le chevauchement entre les chemins de routages).

3.4.4.1 Ressources gaspillées

L'objectif de minimiser le gaspillage des ressources conduit également à minimiser le temps de reconfiguration et à maximiser les ressources matérielles disponibles dans le FPGA. Le nombre de ressources gaspillées $WR_{n,\,t}$ par une région reconfigurable n est égale la différence entre les ressources définies $R_{n,\,t}$ et les ressources requises $C_{n,\,t}$ par cette région.

$$\begin{gathered} \forall n \in N, t \in T \\ WR_{n,\,t} = R_{n,\,t} - C_{n,\,t} \end{gathered} \tag{3.8}$$

La répartition du nombre de ressources dans le FPGA est très irrégulière. Cette répartition est basée sur le fait que les frames DSP sont les moins disponibles. Les frames BRAM sont moins rares et les frames CLB sont les plus disponibles. Donc, un poids q[t] est ajouté à chaque type de ressource, tout en considérant la rareté et l'importance du type de ressource. Le gaspillage total TWR des ressources résultant d'un floorplan est la somme des ressources gaspillées entre toutes les régions.

$$TWR = \sum_{n \in N,\ t \in T} (q[t] * WR_{n,\,t}) \tag{3.9}$$

3.4.4.2 Coûts de communication

La réduction des coûts de communication entre les régions reconfigurables est aussi un objectif à atteindre par les algorithmes de floorplanning. Les ressources de routage disponibles dans une architecture reconfigurable sont configurées pour relier les tâches reconfigurables entre elles. En effet, plus la distance entre deux régions est importante, plus les ressources de routages sont importantes pour relier les modules qui vont être alloués dans ces régions.

Le coût de communication WL prend en compte à la fois les ressources de routage qui relient deux zones et les ressources de routage qui relient les pins associées aux zones avec les E/S requises du dispositif. Le coût de communication dans notre approche est mesuré en appliquant la méthode la plus utilisée HPWL (Half Perimeter Wire Length) [SM91], comme dans [BMS11] [RLS14]. HPWL suppose que chaque broche est située au centre de la zone, donc WL est mesuré par la distance de Manhattan entre les centroïdes de deux

régions.

Un certain nombre de nouveaux paramètres et variables sont définis comme suit pour calculer la distance de Manhattan requises (FIGURE 3.8) :

— IO : ensemble d'interconnexions liées aux ports IO. Chaque élément de l'ensemble est décrit par un tuple à 4 dimensions de la forme : $(\mathrm{n}, \mathrm{iox}, \mathrm{ioy}, \mathrm{b})$ où n est la région reconfigurable, iox et ioy représentent les coordonnées du centroïde IO par rapport à la matrice de dispositif, tandis que b est la largeur d'interconnexion,

— $\mathrm{b_{n1,n2}}$: nombre réel représentant le nombre des pins d'interconnexion entre les régions n1 et n2

— $(\mathrm{cx_n}, \mathrm{cy_n})$: variables réelles représentant les cordonnées (x, y) du centre de la région n,

— $(\mathrm{dcx_{n1,n2}}, \mathrm{dcy_{n1,n2}})$: variables réelles représentent les distances entre les centres des régions n1 et n2 sur les axes x et y,

— $(\mathrm{dpx_{io}}, \mathrm{dpy_{io}})$: variables réelles représentent les distances entre le centre de la région et les ports IO sur les axes x et y,

Tout d'abord, nous commençons par calculer les centroïdes de chaque région :

$$\begin{aligned} \mathrm{cx_n} &= \mathrm{W_{frame}} * (\mathrm{x_n} + \mathrm{w_n}/2) \\ \mathrm{cy_n} &= \mathrm{yl_n} + \mathrm{h_n}/2 \end{aligned} \tag{3.10}$$

Ensuite, nous devons vérifier la distance de Manhattan entre les centroïdes de deux régions.

$$\begin{aligned} &\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{n1} \neq \mathrm{n2} \\ &\mathrm{dcx_{n1,n2}} \geq \mathrm{cx_{n1}} - \mathrm{cxn2} \\ &\mathrm{dcx_{n1,n2}} \geq \mathrm{cx_{n2}} - \mathrm{cxn1} \\ &\mathrm{dcy_{n1,n2}} \geq \mathrm{cy_{n1}} - \mathrm{cyn2} \\ &\mathrm{dcy_{n1,n2}} \geq \mathrm{cy_{n2}} - \mathrm{cyn1} \end{aligned} \tag{3.11}$$

La FIGURE 1.8 illustre la distance de Manhattan entre les centroides de deux régions.

Finalement, la distance de Manhattan entre le centroïde de la région et les ports de

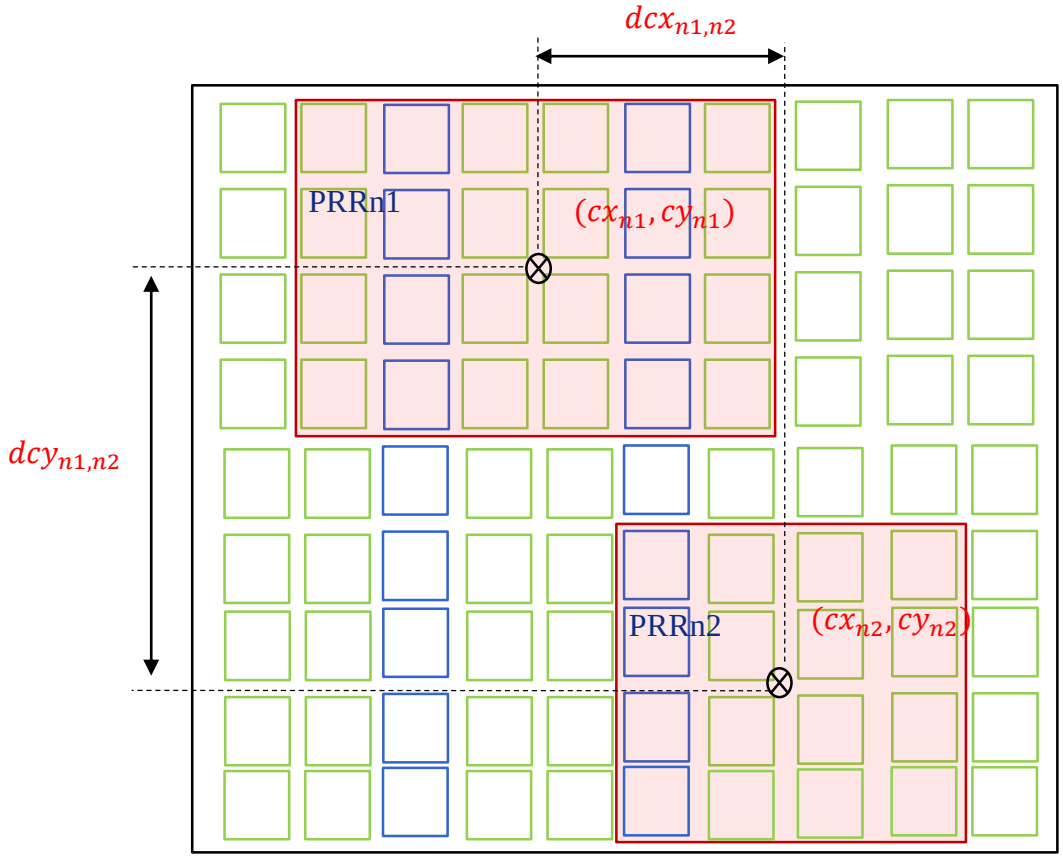

FIGURE 3.8 – *Distance de Manhattan entre les centroides de deux régions*

connexion IO est aussi vérifiée.

$$\begin{aligned} &\forall \text{io} = (\text{n}, \text{iox}, \text{ioy}, \text{b}) \in \text{IO} \\ &\text{dpx}_{\text{io}} \geq \text{cx}_{\text{n}} - \text{iox} \\ &\text{dpx}_{\text{io}} \geq \text{iox} - \text{cxn} \\ &\text{dpy}_{\text{io}} \geq \text{cy}_{\text{n}} - \text{ioy} \\ &\text{dpy}_{\text{io}} \geq \text{ioy} - \text{cyn} \end{aligned} \tag{3.12}$$

Le coût de communication WL est égale à la somme du coût de communication d'interconnexion entre les zones et le coût de communication des fils de routage entre les zones et les IOB.

$$\text{WL} = \sum_{\text{n1,n2,b}} ((\text{dcx}_{\text{n1,n2}} + \text{dcy}_{\text{n1,n2}}) * \text{b}) + \sum_{(\text{n,iox,ioy,b}) \in \text{IO}} ((\text{dpx}_{\text{io}} + \text{dpy}_{\text{io}}) * \text{b}) \tag{3.13}$$

Un poids peut être assigné à chaque fonction coût, selon les besoins du concepteur.

L'objectif final de notre modèle peut s'écrire comme suit : .

$$\text{Min}\{\alpha\text{TWR} + \beta\text{WL}\} \tag{3.14}$$

avec α et β étant les facteurs de poids.

4 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle méthode pour resoudre le problème floorplaning des PRR automatiquement. Elle est entièrement compatible avec les récents fournisseurs de circuits logiques programmables qui tiennent compte du processus de reconfiguration partielle dynamique. L'architecture hétérogène des familles de FPGA modernes, est aussi pris en considération. Le problème du floorplanning peut être traité de nombreuses façons suivant les modèles d'application et d'architecture considérés. Il ne semble pas envisageable d'obtenir une approche efficace, il est nécessaire d'identifier les besoins de l'application et le modèle d'architecture souhaité qu'il s'agit de le modéliser de manière efficace, pour obtenir une solution optimisée par rapport aux critères choisis. La solution utilisée dans nos travaux est basée sur une approche purement mathématique en particulier la programmation linéaire mixte en nombre entier (PLMNE). L'algorithme proposé tient compte des contraintes PR et de l'hétérogénéité de l'espace du FPGA. Il assure une recherche sur tous l'espace du FPGA en se basant sur une fonction objective pour trouver une solution optimale. L'algorithme PLMNE proposé détermine une cartographie optimale des régions dans le FPGA tout en optimisant le nombre de ressources et le coût de communication.

Cependant, cette approche ne tient pas compte ni de la technique de relocation ni de placement des tâches de taille variable. Dans la mise œuvre du processus de floorplanning dans le cas de relocation, les régions reconfigurables qui vont accueillir une même tâche, devront être identiques et avoir une taille suffisamment grande pour être capable d'allouer la plus grande des tâches. Ces contraintes ne sont pas prises en considération dans l'approche PLMNE proposée dans ce chapitre. De plus, toutes les régions ont une même taille et ils sont définis en fonction de besoin des ressources de la plus grande tâche, même s'il y a des régions qui ne vont pas accueillir cette tâche. Ce type de floorplanning implique une perte de ressource. De cette contrainte découle une problématique générale qui est : comment optimiser le processus de floorplanning des zones reconfigurables qui allouent des tâches matérielles de tailles différentes en tenant compte de contrainte de tailles variables.

Dans le chapitre suivant, nous allons abordé le problème de floorplanning dans le cas de relocation des modules reconfigurables en tenant compte de ces tailles variables.

Chapitre 4

Formulations étendues pour la recherche des zones reconfigurables avec contrainte de relocation

Sommaire

1 Introduction

Dans le chapitre précédent, nous avons proposé un algorithme de floorplanning des PRR. Dans ce chapitre nous allons proposer deux formulations étendues pour le floorplanning dans le cas de relocation. Plus précisément, dans la première partie, des nouvelles variables et contraintes sont ajoutées dans le modèle initial du PLMNE. Ces nouveaux paramètres assurent la recherche automatiques des zones homogènes afin de supporter la technique de relocation des bitstreams partiels. Dans la seconde partie, une amélioration de cette formulation pour optimiser les ressources matérielles utilisées et prendre en considération les tâches de tailles variables. Nous proposons d'automatiser l'adaptation de la région reconfigurable au besoin en terme de ressources matérielles de la tâche reconfigurable.

2 Floorplanning pour le cas de relocation

La technique de relocation exige au concepteur de définir des zones reconfigurables compatibles dans laquelle les données de configuration peuvent être ré-allouées. Dans ce contexte, une solution n'est réalisable que si l'algorithme de floorplanning trouve des emplacements pour toutes les régions reconfigurables et qui tient compte de cette contrainte. Pour automatiser le processus d'identification des zones identiques, de nouveaux paramètres, variables et contraintes sont ajoutés aux modèle PLMNE décrit précédemment.

2.1 Principe

Pour reloger un bitstream d'une PRR à une autre, il est nécessaire que les deux PRR soient identiques. Deux PRR sont identiques si elles ont la même forme, nombre, type et disposition de ressources. La figure Figure 4.1 montre un exemple de floorplanning de deux PRR dans le cas de relocation. Le premier PRR1 accueille la tâche la plus grande M2 alors que PRR2 accueille des tâches plus petites. Malgré que PRR2 ne reçoit pas la tâche M2, sa taille doit être en fonction de besoin de ce module afin que les de PRR aient la même taille.

Afin de chercher automatiquement un emplacement valide pour toutes les régions relogeables, une formulation étendue de l'algorithme du floorplanning PLMNE est développée. L'algorithme proposé doit satisfaire deux critères principaux. Le première critère de l'algorithme du floorplanning proposé lors de la recherche consiste à trouver une zone qui contiendra suffisamment de ressources pour allouer chaque module reconfigurable. Les besoins de ressources de chaque PRR sont identifiés par l'utilisateur dans la phase de la modélisation de l'application. Le deuxième critère de l'algorithme consiste à chercher des zones compatibles dans le cas de relocation. Ces deux critères sont formalisés mathématiquement dans le fichier du formulation en PLMNE.

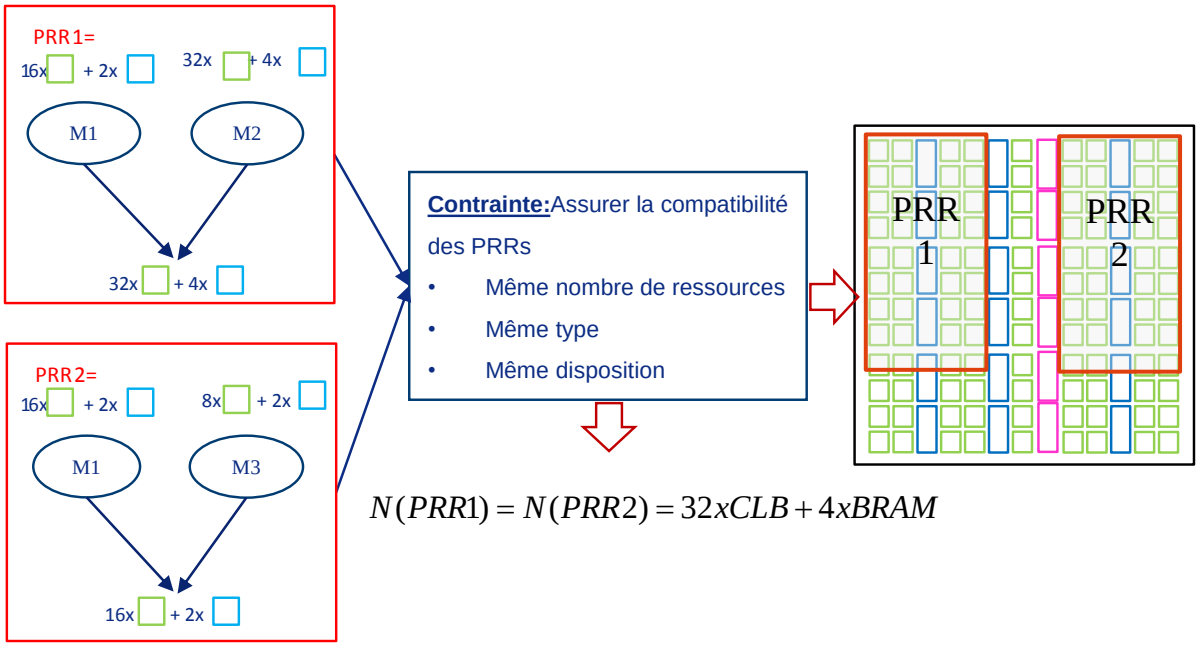

FIGURE *4.1 – Exemple de floorplanning dans le cas de relocation*

2.2 Formulation en PLMNE

Dans cette sous-section, nous montrons comment introduire la relocation en tant que contrainte dans le processus de recherche automatique des PRR en tenant compte de l'hétérogénéité des FPGA. Afin d'introduire la relocation des bitstreams partiels dans notre formulation du modèle PLMNE, nous devons définir des constantes et variables supplémentaires et des nouvelles contraintes.

2.2.1 Constantes

Le concepteur doit préciser, à titre d'information supplémentaire, quelle sont les zones compatibles qui doivent être identifiées. Un paramètre supplémentaire nécessaire est le suivant :

— $\text{relo}_{n1,n2}$: nombre binaire égale 1 si les deux zones n1 et n2 sont compatibles,

2.2.2 Variables

Dans le modèle PLM initial, $k_{n,p}$ est une variable définie pour vérifier si la région n intersecte avec la portion p ou non. Pour définir correctement les contraintes de compatibilité et notamment l'homogénéité de disposition des ressources, nous avons besoin d'un nouvel ensemble de variables (FIGURE 4.2) :

— $\text{TRes}_{n,p,t}$: variable de type entier positif ($\geq$ 0) représente le nombre de ressources dans une portion p de type t occupé par la région n,

— $u_{n,p}$: variable de décalage gauche de type entier positif ($\geq$ 0) égale à 1 lorsque p est la première portion (de gauche à droite) occupée par une région reconfigurable n, 0 sinon,

— $RRes_{n,r,t}$: variable de type entier positif ($\geq$ 0) représente le nombre des ressources dans une ligne r de type t occupé par la région n,

— $g_{n,r}$: variable de décalage bas de type entier positif ($\geq$ 0) égale à 1 lorsque r est la première ligne (de bas à haut) occupée par une région reconfigurable n, 0 sinon,

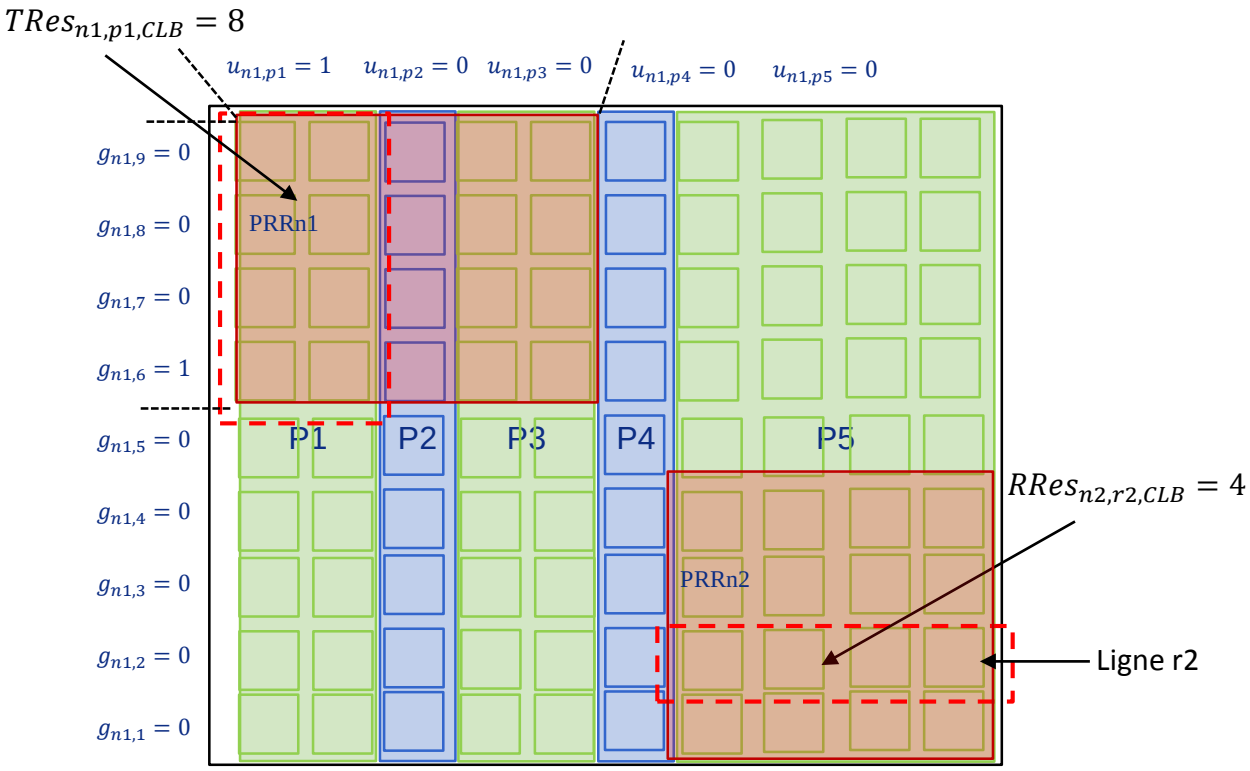

FIGURE *4.2 – Calcul des ressources couvertes :* $TRes_{n,p,t}$ *et* $RRes_{n,r,t}$

Le nombre d'occupation de ressources $TRres_{n,p,t}$ est égale à la quantité d'intersection en terme de frame entre une portion p de type t et la région n. $TRres_{n,p,t}$ est calculé comme suit :

$$\forall n \in N, \forall p \in P, \forall t \in T$$
$$TRres_{n,p,t} = \sum_{r \in R} l_{n,p,r} * d_{p,t} \tag{4.1}$$

avec $l_{n,p,r}$ est variable qui définit la quantité d'intersection de ressources entre la portion p, la région n et la ligne r, et $d_{p,t}$ est le type de ressource t disponible dans la portion

p. P est un ensemble de portions et N est le nombre des régions reconfigurables à placer, T est un ensemble des types de ressources (CLB, DSP, BRAM) et R est ensembles des lignes dans le FPGA.

Le nombre d'occupation de ressources $RRres_{n,r,t}$ est égale la quantité d'intersection en terme de frame entre une ligne r de type t et la région n. $RRres_{n,r,t}$ est calculé comme suit :

$$\begin{gathered} \forall n \in N, \forall r \in R, \forall t \in T \\ RRres_{n,r,t} = \sum_{p \in P} l_{n,p,r} * d_{p,t} \end{gathered} \tag{4.2}$$

La vérification de la compatibilité de la distribution de ressources entre deux régions nécessite l'identification du départ. Dans notre modèle, nous commençons par vérifier les deux régions de la gauche vers la droite et du bas vers le haut. Deux variables de décalages sont définies dans ce cas : $u_{n,p}$ variable de décalage gauche et $g_{n,r}$ variable de décalage bas.

L'identification de la première portion à gauche p occupée par une région n est la variable de décalage $u_{n,p}$. Un ensemble de contraintes est nécessaire pour déterminer la variable de décalage gauche $u_{n,p}$ correctement. Ces contraintes sont formulées comme suit :

Pour vérifier l'unicité :

$$\forall n \in N, \qquad \sum_{p \in P} u_{n,p} = 1 \tag{4.3}$$

Pour déterminer la première portion occupée :

$$\begin{aligned} \forall n \in N, p = 1 \qquad & u_{n,1} = k_{n,1} \\ \forall n \in N, \forall p \in P \qquad & u_{n,p} \geq (-k_{n,p-1} + k_{n,p}) \end{aligned} \tag{4.4}$$

La variable de décalage bas $g_{n,r}$ permet d'identifier le premier ligne r occupée en bas par une région n. Il est déterminé par les formulations suivantes :

Pour vérifier l'unicité :

$$\forall n \in N, \qquad \sum_{r \in R} g_{n,r} = 1 \tag{4.5}$$

Pour déterminer la première colonne occupée :

$$\begin{aligned} \forall n \in N, r = 1 \qquad & g_{n,1} = a_{n,1} \\ \forall n \in N, \forall r \in R \qquad & g_{n,r} \geq (-a_{n,r-1} + a_{n,r}) \end{aligned} \tag{4.6}$$

2.2.3 Contraintes

Dans ce paragraphe, nous introduisons toutes les contraintes qui assurent la compatibilité entre les zones reconfigurables. Dans notre approche, nous avons besoin de répondre à cinq contraintes différentes pour assurer la compatibilité entre deux régions reconfigurables. Les contraintes sont formulées respectivement par les équations suivantes :

— La même hauteur ;

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 : \\ \mathrm{h_{n1}} = \mathrm{h_{n2}}\end{gathered} \tag{4.7}$$

— La même largeur ;

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 : \\ \mathrm{w_{n1}} = \mathrm{w_{n2}}\end{gathered} \tag{4.8}$$

— Le même nombre de portions et lignes occupées

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{p} \in \mathrm{P}, \mathrm{r} \in \mathrm{R},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 : \\ (\sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n1,p}} = \sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n2,p}}) \wedge (\sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n1,r}} = \sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n2,r}})\end{gathered} \tag{4.9}$$

avec $\mathrm{k_{n,p}}$ est variable binaire égale à 0 si la région n et la portion p ne s'intersectent pas et $\mathrm{a_{n,r}}$ est variable binaire égale à 1 si et seulement si la région **n** occupe la ligne r.

— La somme du nombre de ressources totales des types t occupées par les différentes portions p dans la région n1 est égale à celle de la région n2, et la somme du nombre de ressources totales des types t occupées par les différentes lignes r dans la région n1 est égale à celle de la région n2.

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{p1}, \mathrm{p2} \in \mathrm{P}, \mathrm{r} \in \mathrm{R}, \mathrm{t} \in \mathrm{T},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 : \\ (\sum_{\mathrm{p1} \in \mathrm{P}} \mathrm{TRes_{n1,p1,t}} = \sum_{\mathrm{p2} \in \mathrm{P}} \mathrm{TRes_{n2,p2,t}}) \wedge (\sum_{\mathrm{r1} \in \mathrm{R}} \mathrm{RRes_{n1,r1,t}} = \sum_{\mathrm{r2} \in \mathrm{R}} \mathrm{RRes_{n2,r2,t}})\end{gathered} \tag{4.10}$$

— La même distribution de ressources

Pour la contrainte distribution de ressources, nous exploitons la variable de décalage $\mathrm{u_{n,p}}$ avec la variable $\mathrm{TRes_{n,p,t}}$ pour ordonner les portions de gauche à droite numérotées avec le même nombre et le même type de ressources. Les deux dernières équations d'inégalités sont les suivantes :

$$\begin{gathered}\forall n1, n2 \in N, t \in T, p1, p2 \in P, i \in [0, |P| - 1] \\ \text{Si } relo_{n1,n2} = 1, 1 \leq p2 + i, p2 + i \leq |P|, 1 \leq p1 + i, p1 + i \leq |P| : \\ TRes_{n2,p2+i,t} \leq TRes_{n1,p1+i,t} + |R| * P_{max} * W_{frame} * (3 - k_{n1,p1+i} - u_{n1,p1} - u_{n2,p2}) \\ TRes_{n2,p2+i,t} \geq TRes_{n1,p1+i,t} - |R| * P_{max} * W_{frame} * (3 - k_{n1,p1+i} - u_{n1,p1} - u_{n2,p2})\end{gathered} \tag{4.11}$$

avec $|R|$, P_{max} et W_{frame} sont respectivement le nombre maximal de lignes, le nombre maximal de portions et le nombre maximal de frames par portion dans le FPGA.

Nous considérons dans l'équation 4.11 que les contraintes ne sont actives que lorsque p1 et p2 représentent les premières portions occupées et que i est incrémenté uniquement sur une partie couverte. Ainsi, les constantes à droite dans chaque couple d'inégalités ($3 - k_{n1,p1+i} - u_{n1,p1} - u_{n2,p2}$) sont annulées et les termes restants coïncident ($TRes_{n2,p2+i,t} = TRes_{n1,p1+i,t}$) .

La nouvelle formulation de l'algorithme PLMNE commence par chercher l'emplacement des différents PRR et satisfaire les contraintes PR définies dans le chapitre précédent. Par la suite elle va chercher à satisfaire les nouvelles contraintes définies dans cette partie pour assurer la compatibilité des zones reconfigurables dans le cas de relocation.

La figure FIGURE 4.3 illustre un exemple des zones reconfigurables placées dans un FPGA qui est modélisé en portions. Les valeurs des variables $u_{n,p}$, $k_{n,p}$ $g_{n,r}$ et $a_{n,r}$ mentionnées dans la figure représentent les endroits d'intersection des zones avec les portions et les lignes du FPGA. Le rôle des variables $u_{n,p}$ et $g_{n,r}$ est de fixer respectivement la première ligne et la première colonne occupée par la région. Avec ces variables l'algorithme de recherche fixe son départ et assure qu'il ne va pas changer tout au long de sa vérification. Par la suite, il va s'incrémenter soit par une ligne soit par une portion et vérifier le nombre et le type de ressources dans une portion $TRes_{n,p,t}$ ou dans une ligne $RRes_{n,r,t}$. Avec les variables $k_{n,p+i}$ et $a_{n,r+i}$ nous assurons que la région 'n' occupe encore la nouvelle portion p+i et la nouvelle ligne r+i sinon l'algorithme va terminer sa vérification. La figure montre que les zones n1 et n2 sont identiques en vérifiant toutes les contraintes. Les zones n1 et n3 ne sont pas identiques, elles ne vérifient qu'une seule contrainte représentée par l'équation 4.11.

3 Floorplanning pour le cas de relocation avec adaptation

Sur la base de la technologie actuelle, le floorplanning consiste à définir des régions reconfigurables de tailles fixes capables d'accueillir des tâches matérielles de tailles variables. Généralement pour pouvoir ré-allouer une tâche sur deux régions reconfigurables, il faut que la taille de deux régions soient identiques et égale à la plus grande des tâches qui va être

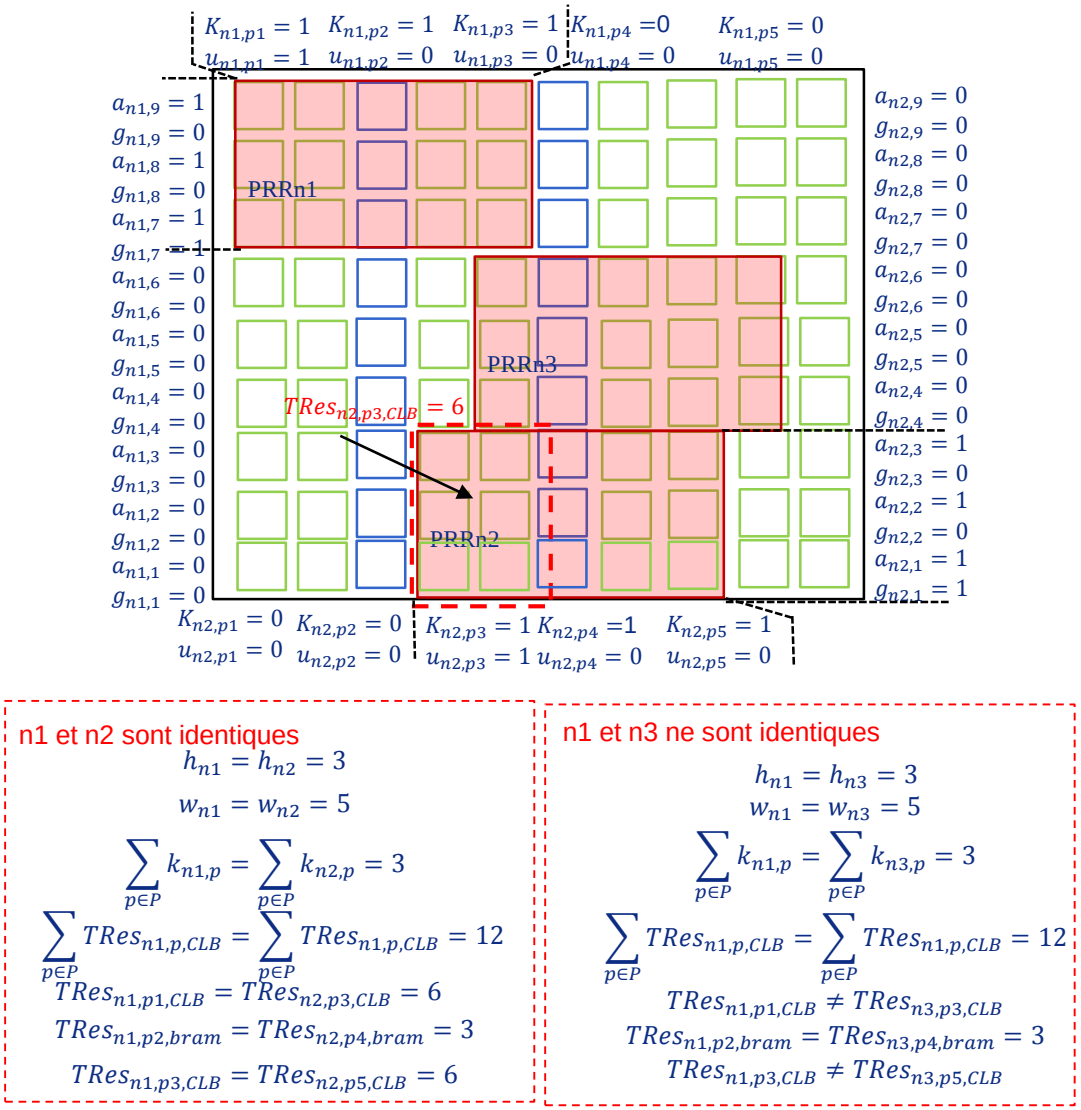

FIGURE 4.3 – *Exemple de vérification des contraintes de compatibilité des PRR*

accueillie seulement par une entre elles. Considérons deux régions reconfigurables PRR1 plus gros que PPR2. Généralement, pour pouvoir ré-allouer un même module dans deux régions, il faut que la taille des deux région soient totalement identique et égale à la plus grande région (dans ce cas PRR1). Par conséquent, quand les modules seront placés sur la région PRR2, ils n'utiliseront qu'une partie des ressources, les autres ressources seront perdues et l'efficacité d'utilisation en surface seras dégradée.

3.1 Principe

Grâce à la méthode d'adaptation, nous pouvons définir seulement une partie de la plus grande région reconfigurable compatible avec la plus petite. Ce principe est avantageux par rapport à la méthode classique, car il permet d'ajuster la taille de la plus petite région seulement au besoin en ressources de la tâche matérielle partagée entre les régions.

Nous proposons dans cette partie une modification de formulation étendue de PLMNE

pour relocation permettant de chercher une adaptation optimale de la région reconfigurable en fonction de besoin de chaque tâche. Reposant sur un partitionnement de la région, il vise différents objectifs : minimisation des ressources logiques et minimisation de temps de reconfiguration. Nous proposons donc un modèle mathématique qui nous sert à déterminer le choix du partitionnement optimal.

3.2 Formulation en PLMNE

Toutes les contraintes qui assurent la compatibilité entre les zones reconfigurables présentées dans la partie précédente seront modifiées. En tenant compte du partitionnement les deux régions reconfigurables ne sont pas compatibles que par la partie qui se partagent entre elles. Ainsi les deux régions n'ont pas forcement les mêmes : hauteur, longueur, nombre de portions et nombre de ressources.

Dans nos travaux, nous adoptons trois cas d'études en changeant les paramètres w_n et h_n (FIGURE 4.4).

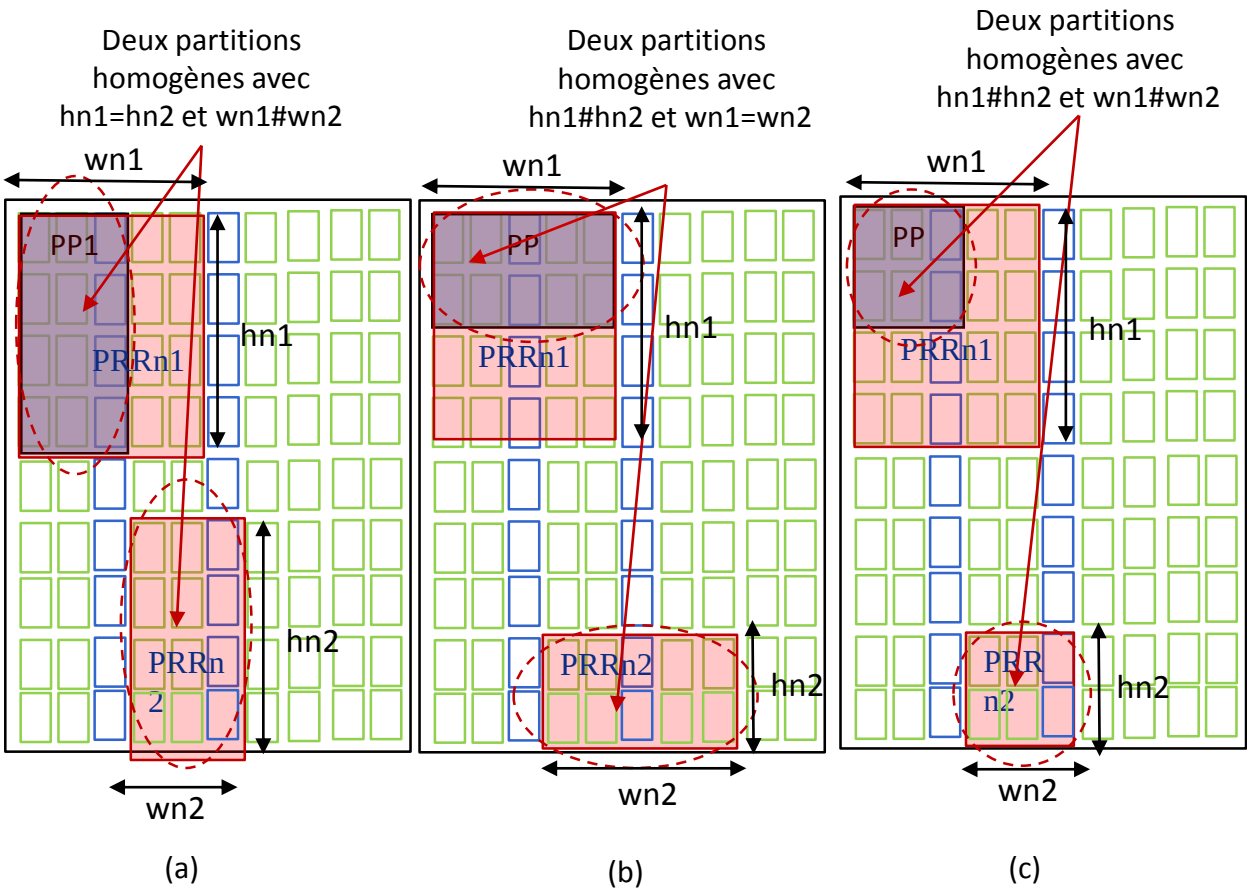

FIGURE 4.4 – *Types d'adaptation de la zone reconfigurable : (a) adaptation horizontale, (b) adaptation verticale (c) adaptation 2-D*

Dans les trois cas, les contraintes qui justifient l'égalité totale du nombre de ressources

est ré-formulée par l'équation suivante :

$$\begin{aligned}&\forall n1, n2 \in N, p1, p2 \in P, r1 \in R, r2 \in R, t \in T, \text{ Si } relo_{n1,n2} = 1 \text{ et } C_{n1,t} \leq C_{n2,t} : \\ &(\sum_{p1 \in P} TRes_{n1,p1,t} \leq \sum_{p2 \in P} TRes_{n2,p2,t}) \wedge (\sum_{r1 \in R} RRes_{n1,r1,t} \leq \sum_{r2 \in R} RRes_{n2,r2,t})\end{aligned} \tag{4.12}$$

L'ensemble de contraintes qui sont représentées par les équations 4.7, 4.8, 4.9 et 4.11 seront ré-formulées selon le type d'adaptation souhaitée.

3.2.1 Adaptation horizontale : $h_{n1} = h_{n1}$ et $w_{n1} \neq w_{n1}$

Dans le cas d'adaptation horizontale, différentes contraintes doivent être définies dans la formulation étendue du PLMNE. Dans le cas ou le nombre de ressource $C_{n1,t}$ de la région n1 est inférieure au nombre de ressources $C_{n2,t}$ de la région n2, ces contraintes sont définies comme suit :

— la même hauteur :

$$\begin{aligned}&\forall n1, n2 \in N, \text{ Si } relo_{n1,n2} = 1 \text{ et } C_{n1,t} \leq C_{n2,t} : \\ &h_{n1} = h_{n2}\end{aligned} \tag{4.13}$$

— La largeur de région n1 est inférieur du largeur de région n2

$$\begin{aligned}&\forall n1, n2 \in N, \text{ Si } relo_{n1,n2} = 1 \text{ textet} C_{n1,t} \leq C_{n2,t} : \\ &w_{n1} \leq w_{n2}\end{aligned} \tag{4.14}$$

— Le nombre de portions de la région n1 est inférieur à celles de la région n2, le nombre des lignes des deux régions sont égaux.

$$\begin{aligned}&\forall n1, n2 \in N, p \in P, \text{ Si } relo_{n1,n2} = 1 \text{ et } C_{n1,t} \leq C_{n2,t} : \\ &(\sum_{p \in P} k_{n1,p} \leq \sum_{p \in P} k_{n2,p}) \wedge (\sum_{r \in R} a_{n1,r} = \sum_{r \in R} a_{n2,r})\end{aligned} \tag{4.15}$$

Dans ce cas nous gardons la même équation 4.11, pour réserver la même distribution de ressources sur la partie commune entre les deux zones.

3.2.2 Adaptation verticale : $h_{n1} \neq h_{n1}$, $w_{n1} = w_{n1}$

Dans le cas d'adaptation verticale, les hauteurs des deux régions ne sont pas égales et les largeurs sont identiques. Dans le cas où le nombre de ressources $C_{n1,t}$ de la région n1 est inférieure au nombre de ressources $C_{n2,t}$ de la région n2 ; ces contraintes sont définies comme suit :

— L'hauteur de la région n1 est inférieur à celle de la région n2

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \text{ et } \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ \mathrm{h_{n1}} \leq \mathrm{h_{n2}}\end{gathered} \tag{4.16}$$

— la même largeur :

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \ \mathrm{textet C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ \mathrm{w_{n1}} = \mathrm{w_{n2}}\end{gathered} \tag{4.17}$$

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{p} \in \mathrm{P},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \text{ et } \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ (\sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n1,p}} = \sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n2,p}}) \wedge (\sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n1,r}} \leq \sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n2,r}})\end{gathered} \tag{4.18}$$

— Le nombre des portions sont identiques et le nombre des lignes de la région n1 est inférieur à ceux de la région n2

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{t} \in \mathrm{T}, \mathrm{r1}, \mathrm{r2} \in \mathrm{R}, \mathrm{j} \in [0, |\mathrm{R}| - 1] \\ \text{Si } \mathrm{relo_{n1,n2}} = 1, 1 \leq \mathrm{r2} + \mathrm{j}, \mathrm{r2} + \mathrm{j} \leq |\mathrm{R}|, 1 \leq \mathrm{r1} + \mathrm{j}, \mathrm{r1} + \mathrm{j} \leq |\mathrm{R}| : \\ \mathrm{RRes_{n2,r2+j,t}} \leq \mathrm{RRes_{n1,r1+j,t}} + \mathrm{P_{max}} * |\mathrm{P}| * \mathrm{W_{frame}} * (3 - \mathrm{a_{n1,r1+j}} - \mathrm{g_{n1,r1}} - \mathrm{g_{n2,r2}}) \\ \mathrm{RRes_{n2,r2+j,t}} \geq \mathrm{RRes_{n1,r1+j,t}} - \mathrm{P_{max}} * |\mathrm{P}| * \mathrm{W_{frame}} * (3 - \mathrm{a_{n1,r1+j}} - \mathrm{g_{n1,r1}} - \mathrm{g_{n2,r2}})\end{gathered} \tag{4.19}$$

3.2.3 Adaptation 2-D : $\mathrm{h_{n1}} \neq \mathrm{h_{n1}}$, $\mathrm{w_{n1}} \neq \mathrm{w_{n1}}$

Dans le cas d'adaptation 2-D, tous les paramètres : hauteur, largeur, nombre de portion et nombre de ligne ne sont pas égaux. Toutes ces contraintes sont définies comme suit :

— Hauteur

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \text{ et } \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ \mathrm{h_{n1}} \leq \mathrm{h_{n2}}\end{gathered} \tag{4.20}$$

— Largeur

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \text{ et } \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ \mathrm{w_{n1}} \leq \mathrm{w_{n2}}\end{gathered} \tag{4.21}$$

— Nombre de portions et nombre de lignes

$$\begin{gathered}\forall \mathrm{n1}, \mathrm{n2} \in \mathrm{N}, \mathrm{p} \in \mathrm{P},\ \text{Si } \mathrm{relo_{n1,n2}} = 1 \text{ et } \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}} : \\ (\sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n1,p}} \leq \sum_{\mathrm{p} \in \mathrm{P}} \mathrm{k_{n2,p}}) \wedge (\sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n1,r}} \leq \sum_{\mathrm{r} \in \mathrm{R}} \mathrm{a_{n2,r}})\end{gathered} \tag{4.22}$$

— Nombre de ressources occupées

$$\begin{gathered}\forall \mathrm{n1,n2} \in \mathrm{N}, \mathrm{t} \in \mathrm{T}, \mathrm{p1,p2} \in \mathrm{P}, \mathrm{i} \in [0, |\mathrm{P}| - 1], \mathrm{r1,r2} \in \mathrm{R} \\ \text{Si } \mathrm{relo_{n1,n2}} = 1, \mathrm{C_{n1,t}} \leq \mathrm{C_{n2,t}}, 1 \leq \mathrm{p2} + \mathrm{i}, \mathrm{p2} + \mathrm{i} \leq |\mathrm{P}|, 1 \leq \mathrm{p1} + \mathrm{i}, \mathrm{p1} + \mathrm{i} \leq |\mathrm{P}| : \\ \mathrm{l_{n2,p2+i,r2}} * \mathrm{d_{p2,t}} \leq \mathrm{l_{n1,p1+i,r1}} * \mathrm{d_{p1,t}} + \mathrm{P_{max}} * \mathrm{W_{frame}} * (5 - \mathrm{k_{n1,p1+i}} - \mathrm{u_{n1,p1}} - \mathrm{u_{n2,p2}} \\ - \mathrm{g_{n1,r1}} - \mathrm{g_{n2,r2}}) \\ \mathrm{l_{n2,p2+i,r2}} * \mathrm{d_{p2,t}} \geq \mathrm{l_{n1,p1+i,r1}} * \mathrm{d_{p1,t}} - \mathrm{P_{max}} * \mathrm{W_{frame}} * (5 - \mathrm{k_{n1,p1+i}} - \mathrm{u_{n1,p1}} - \mathrm{u_{n2,p2}} \\ - \mathrm{g_{n1,r1}} - \mathrm{g_{n2,r2}})\end{gathered} \tag{4.23}$$

4 Discussion

La détermination des emplacements, des formes et des dimensions des régions PR n'est pas toujours banale, mais ce choix peut avoir une incidence sur l'utilisation des ressources FPGA, le temps de reconfiguration et l'exigence de stockage des bitstreams. Plusieurs solutions académiques ont été présentées dans la littérature pour automatiser le floorplanning [CW06, SB06, BMS11, MSSM10, VF12, BSSK11, RLS14, YYC07]. Cependant, seules quelques-unes [VF12, RLS14, BMS11] tiennent compte des contraintes PR et de la distribution hétérogène des ressources dans le FPGA, en même temps. En effet, la plupart des algorithmes ne considèrent qu'un seul aspect des deux, c'est-à-dire soit les contraintes PR [MSSM10, YYC07], ou soit la distribution hétérogène des ressources [CW06, SB06, FM06]. Parmi les travaux existants dans ce domaine, les seuls qui tiennent compte de ces deux aspects sont [VF12, RLS14]. Bien que les approches proposées dans [VF12] et [RLS14] donnent de meilleurs résultats par rapport aux approches antérieures, elles ne considèrent pas le principe de l'adaptation des partitions PR aux tâches matérielles de taille différente dans le cas de relocation. Ces approches obligent le concepteur à définir et fixer la taille de la zone reconfigurable dans ces algorithmes de recherche.

Les principaux avantages de notre formulation PLMNE de recherche sont qu'elle est entièrement compatible avec les nouvelles générations des FPGA qui possèdent une distribution hétérogène des ressources matérielles et qu'elle assure automatiquement la satisfaction de toutes les contraintes PR ainsi les contraintes de relocation. Avec cet algorithme de recherche, le concepteur n'est pas obligé de fixer la taille de la zone reconfigurable selon la taille du plus gros module dans le cas de relocation. L'optimisation des ressources qui permet d'augmenter l'efficacité est un aspect principal défini dans notre approche de recherche en utilisant la technique d'adaptation de la région reconfigurable au besoin du module alloué. Cette technique est formalisée dans l'algorithme PLMNE proposé.

L'algorithme PLMNE proposé avec ses deux extensions garantit une recherche optimale et automatique des zones reconfigurables en augmentant l'efficacité d'utilisation des ressources ainsi la flexibilité des FPGA qui utilisent la reconfiguration dynamique.

5 Conclusion

La contribution principale de ce chapitre est d'apporter au concepteur une méthode automatique à utiliser pour identifier les emplacements optimales des PRR dans une surface reconfigurable hétérogène dans le cas de relocation des bitstream partiels et aussi pour adapter la taille des partitions PR aux modules PR de tailles différentes. Cette méthode est basée sur une étude exploratoire de l'approche PLM. Des nouvelles contraintes et des fonctions de coût de floorplanning sur mesure pour PRR sont introduits. Ce floorplanning automatique permet une utilisation de la surface logique du FPGA beaucoup plus efficace et donc une réduction des besoins en surface logique et minimisation de temps de reconfiguration. En effet, notre approche fournit les caractéristiques nécessaires à l'identification d'une zone reconfigurable (taille de la zone et ses cordonnées) pour un emplacement optimal.

Chapitre 5

Étude de cas

Sommaire

1 Introduction

Dans ce chapitre, nous allons aborder deux grandes parties de nos travaux. Dans un premier temps, nous commençons par valider et évaluer la technique de relocation sur les architectures reconfigurables. Nous présenterons deux cas d'exemples d'applications pour le placement et la relocation des tâches matérielles de tailles différentes sur des partitions PR fixes. Dans cette partie, nous allons évaluer l'impact du partitionnement sur l'efficacité d'utilisation des ressources matérielles.

Dans la deuxième partie, nous présenterons les résultats expérimentaux de l'algorithme de floorplanning PLMNE et leurs extensions.

2 Technique de Relocation : Validation et évaluations

Dans cette partie, l'évaluation a été faite sur deux plateformes FPGA de type Xilinx Virtex 5 et ZYNQ. Nous avons validé notre méthode de relocation par la mise en œuvre dynamique et adaptative d'une application de détection de coupure de vidéo basée sur le descripteur de couleur mpeg7 (CSD). Dans la mise en oeuvre de la méthodologie, différents scénarios ont été testés sur la carte ML506, FPGA Virtex-5. Selon l'application et les exigences du système, le module CSD peut être chargé à tout moment et alloué dans différentes régions reconfigurables possibles. Cette méthodologie permet de gérer efficacement la zone reconfigurable et la communication entre les différentes régions reconfigurables. En effet, les résultats de la mise en œuvre montrent une optimisation importante en termes de ressources matérielles, temps de reconfiguration et de la mémoire de stockage des bitstreams. En outre, nous avons proposé de mettre en oeuvre une architecture reconfigurable pour implémenter la transformée en cosinus discrète (DCT) de tailles variables. La reconfiguration dynamique partielle est utilisée pour intégrer à la demande différentes versions du module de calcul DCT. Pour ce faire, nous avons partitionné la région reconfigurable en petites partitions que nous avons ensuite adaptées aux différentes tailles des tâches matérielles liées au module DCT. L'architecture proposée a été validé sur le SoC Xilinx Zynq-7000. Les résultats obtenus ont montré que le partitionnement proposé permet d'optimiser les ressources matérielles, gérer efficacement la zone reconfigurable et réduire le temps de reconfiguration.

2.1 Application 1 : Détection de coupure de vidéo basée sur CSD

Les systèmes multimédias modernes se caractérisent par un ensemble de services dont l'utilisateur peut facilement choisir et basculer entre eux. Il peut également ajouter de nouveaux services ou remplacer ceux qui existent par des autres. Pour supporter les exigences élevées en termes de capacité, de portabilité et de consommation des multiples traitements, l'utilisation de systèmes matériels adéquats avec un haut niveau d'adaptabilité devient nécessaire. Dans ce contexte, l'utilisation de notre approche proposée peut être une solution

très intéressante.

2.1.1 Présentation de l'application

Le scénario d'application considéré dans ce travail est illustré à la FIGURE 5.1. Étant donné un ensemble de traitements multimédia, un FPGA divisé en régions PR, différents modules de traitement seront chargés dans le FPGA selon le choix de l'utilisateur. Ces modules peuvent être répartis dans différentes régions PR en fonction de la configuration du système choisie. Ils possèdent des services et des tailles matérielles différents. Les différents scénarios de la mise en œuvre du module CSD évoquent de nombreux problèmes qui doivent être traités. Ces derniers peuvent être résumés dans les points suivants : Relocation, communication et la gestion des ressources matérielles [HARM17].

Dans cette partie, nous proposons une implémentation adaptative et dynamique d'un système de détection de coupure vidéo basé sur le CSD MPEG-7 en utilisant le flot de conception proposé pour la relocation des tâches de tailles variables. Comme l'illustre la FIGURE 5.1, le CSD existe avec différents niveaux de quantification, et peut être attribué dans différentes régions reconfigurables avec une taille de module variable. En fait, comme démontré dans [AHTM14], le CSD peut être mise en œuvre pour différents niveaux de quantification tels que : 16, 32, 64, etc. L'utilisation d'un faible nombre de niveaux de quantification a démontré une réduction significative de la complexité de l'algorithme et des ressources matérielles occupées tout en préservant un niveau de précision satisfaisant en termes de taux de détection de coupure. Ceci est utile lorsque le système est limité par le nombre des ressources matérielles occupées. Dans ce cas, une version de la CSD moins complexe peut être chargée pour répondre aux exigences du système.

2.1.2 Partitionnement

La méthode de relocation proposée a été appliquée pour l'application de détection de coupure. L'application a été implémentée sur la plateforme XUP-ML506 à base d'un FPGA Virtex-5 . Dans notre conception, nous avons initialement défini trois PRR et le reste de circuit a été également défini reconfigurable pour la partie statique mais avec une seule configuration. Chaque PRR est partitionnée en plusieurs PRP pour permettre la mise en œuvre des modules PR de tailles variables (détection de coupure basés sur CSD avec différents niveaux de quantification). Ces différents modules peuvent être placés dans plusieurs emplacements possibles. La FIGURE 5.2 illustre les différentes PRR et les différents partitions définies par rapport à chaque module. Le module matériel de détection de coupure basé sur le CSD avec un niveau de quantification égale 32 (CSD_{32}) peut être placé en deux positions, celui basé sur le CSD avec un niveau de quantification égale 16 (CSD_16) peut être situé dans 5 emplacements différents et celui basé sur le CSD avec un niveau de quantification égale à 8 (CSD_8) peut être situé dans 8 partitions reconfigurables différentes.

Le TABLEAU 5.1 illustre le nombre et la taille des bitstreams (Nb et Tb) générés pour

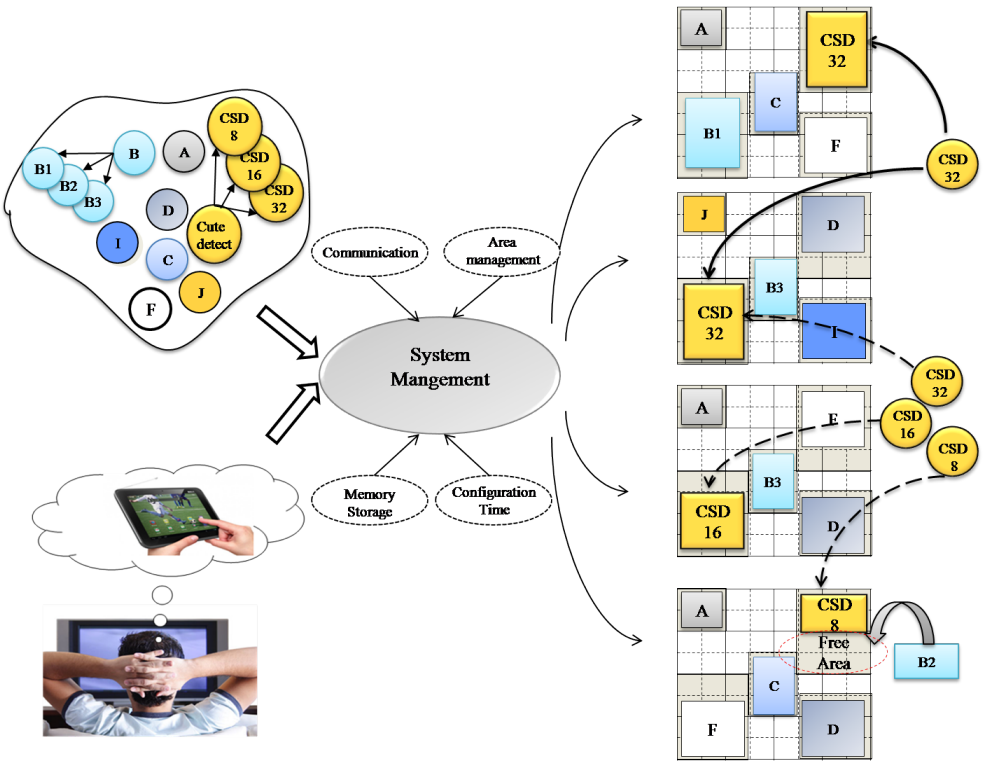

FIGURE 5.1 – *Application proposée*

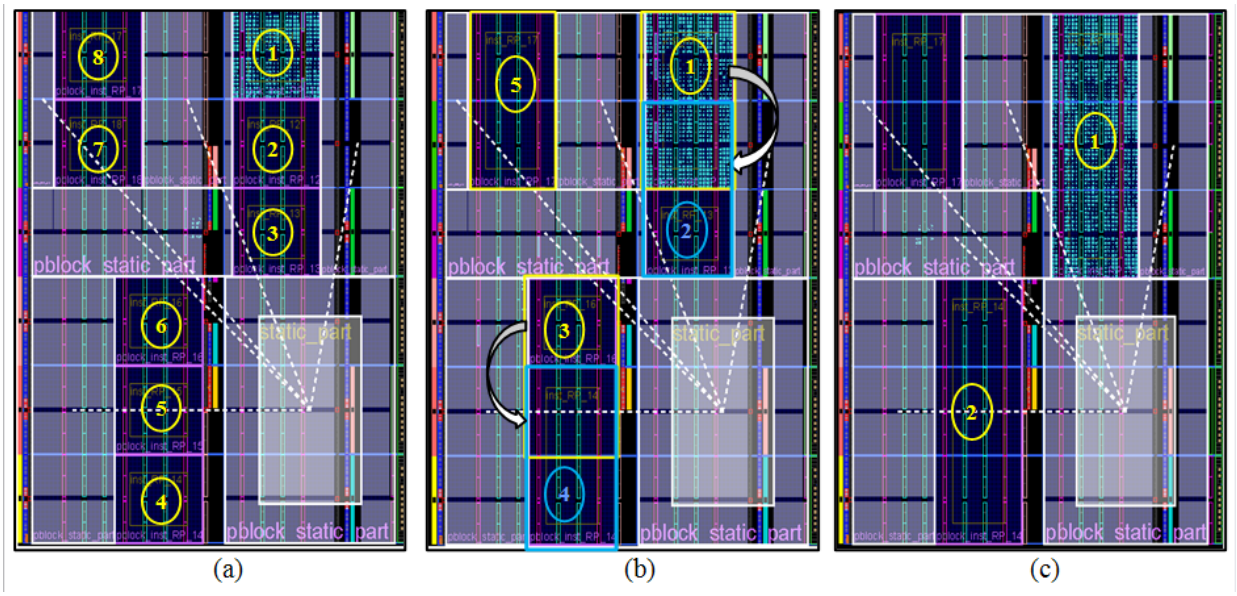

FIGURE 5.2 – *Différents emplacements possibles du CSD : (a) détecteur de coupe basé sur* CSD_8*, (b) détecteur de coupe basé sur le* CSD_{16}*, (c) détecteur de coupe basé sur le* CSD_{32}

les différents niveaux de quantification (8, 16 et 32) pour la conception avec et sans relocalisation. En appliquant la technique de conception basée sur le flot classique de conception de Xilinx, il est nécessaire de générer plusieurs bitstreams pour le même PRM pour chaque emplacement tels que : 8 bitstreams pour le module de CSD_8, 5 bitstreams pour CSD_{16} et 2 bitstreams pour CSD_{32}. En utilisant la technique de relocation, un seul bitstream est généré pour l'emplacement initial de chaque PRM. Les résultats obtenus montrent que la

taille de la mémoire utilisée pour stocker les bitstreams partiels diminue jusqu'à 87,5%, 80% et 50% respectivement pour les versions CSD_8, CSD_{16} et CSD_{32}.

TABLEAU *5.1 – Résultats d'optimisation de la mémoire*

CSD version	**Sans relocation**		**Avec relocation**		**Optimisation mémoire**
	Nb	Tb (Kbyte)	Nb	Tb (Kbyte)	
CSD_8	8	8*112= 896	1	112	87,5%
CSD_{16}	5	5*224=1120	1	224	80%
CSD_{32}	2	2*336=672	1	336	50 %

Dans un FPGA Virtex-5, un frame CLB équivaut à 160 LUT et 40 slices, un frame BRAM se compose de 4 blocs mémoire BRAM36 et un frame DSP se compose de 8 blocs DSP48E disposés verticalement. Le TABLEAU 5.2 fournit des informations sur le nombre équivalent de trames requises pour implémenter les différentes versions du module CSD sur un FPGA Virtex-5.

TABLEAU *5.2 – Occupation des frames pour les différentes versions de CSD*

CSD IP	**Nombre de Frames**	
	CLB	BRAM
CSD_8	9	1
CSD_{16}	16	1
CSD_{32}	27	1

2.1.3 Résultats

L'objectif de notre méthode proposée est d'optimiser le nombre de ressources utilisées et les performances du système en appliquant l'adaptation de chaque PRR à la taille de chaque tâche matérielle.

La FIGURE 5.3 illustre deux modes d'implémentation du module du descripteur structure de couleurs (CSD) à différents niveaux de quantification pour chaque PRR donnée qui sont avec et sans adaptation. La première implémentation des architectures CSD est basée sur la méthode de conception classique, qui ne permet pas l'adaptation de la taille des modules matériels à la taille des régions reconfigurables (TABLEAU 5.3 sans adaptation). Dans ce cas, tout les PRR sont réservés pour implémenter les trois versions de CSD (CSD_8, CSD_{16} et CSD_{32})dans les régions 1 et 2 ou les deux versions (CSD_8, CSD_{16}) dans la région 3. Ce mode d'implémentation conduit à une utilisation inefficace des ressources matérielles. Les ressources matérielles non utilisées de la région reconfigurable sont perdues et ne peuvent pas être utilisées pour implémenter d'autres supports matériels IP.

Dans la seconde implémentation, nous avons évalué la méthode de relocation des tâches de taille variables en utilisant l'exemple de différentes versions de CSD en modifiant le niveau de quantification. La méthode de conception proposée basée sur l'adaptation de la région reconfigurable à la taille des cœurs matériels IP (TABLEAU 5.3 avec adaptation). Dans ce mode de mise en œuvre, les PRR sont partitionnées en différentes partitions chacune adaptée à la taille des versions du module CSD qui vont être allouées. Les partitionnements de trois PRR (PRR1, PRR2 et PRR3) sont illustrés dans la FIGURE 5.3 et sont définis comme suit :

— PRR1 et PRR2 ont la même taille : 33 frames (27 frames CLB + 6 frames BRAM)
— La taille de PRR3 est égale à 22 frames (18 frames CLB + 4 frames BRAM)
— PRR1 et PRR2 sont divisées en trois partitions égales
— PRR3 est partitionnée en deux partitions égales

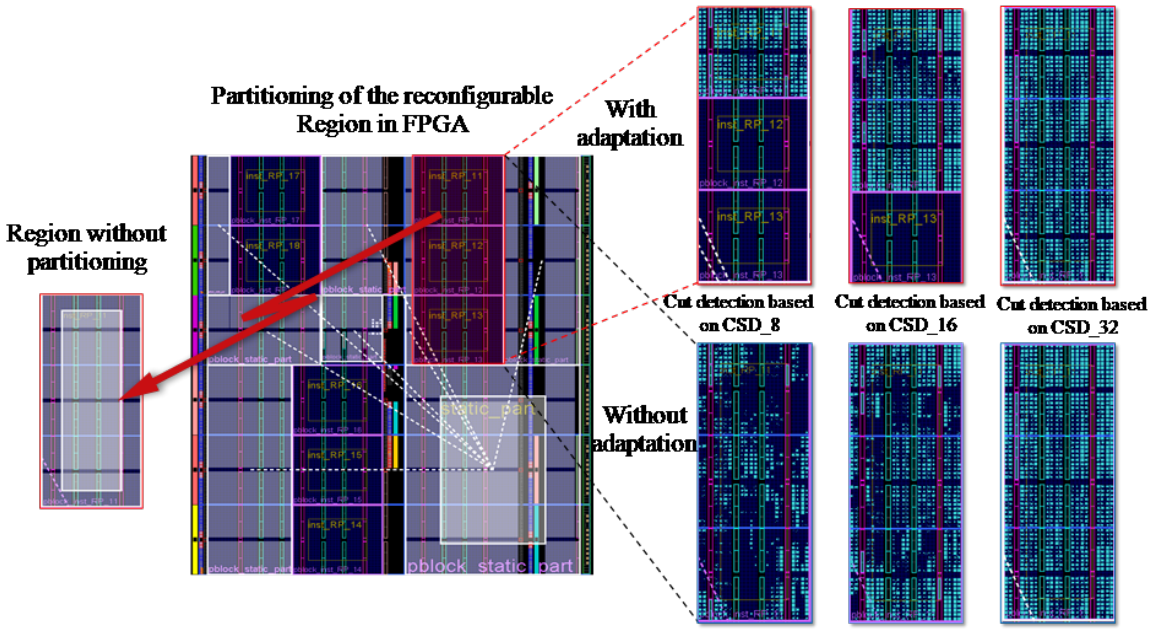

FIGURE *5.3 – Relocation avec adaptation*

Le TABLEAU 5.3 illustre les métriques d'évaluation de notre approche qui sont les taux d'utilisation des ressources matérielles par rapport à chaque type de ressource reconfigurable (Ra_i, avec i = CLB ou DSP ou BRAM) et le taux d'utilisation global (Ra_T) pour les différents PRR (1, 2 et 3). Ces métriques sont définies précédemment par les équations (2.1, 2.2 et 2.3). Dans cet exemple, elles sont calculées pour chaque version de CSD.

Le TABLEAU 5.3 montre l'efficacité des ressources matérielles utilisées dans les partitions PR qui peut atteindre des taux d'utilisation élevés par rapport à ceux obtenus pour le cas d'implémentation sans adaptation. Par exemple, des taux atteignent 90%, 77% et 84% pour la région PRR1.

Grâce à notre nouvelle technique exploitée (partitionnement des PRR) dans le cas de relocation, nous avons pu adapter la taille de la région reconfigurable à la taille de la tâche matérielle. Par conséquent, un autre avantage est l'optimisation du temps de re-

Tableau 5.3 – *Taux d'occupation des ressources matérielles*

CSD IP	**Sans adaptation**			**Avec adaptation**			
	RaCLB	RaBRAM	RaT	RaCLB	RaBRAM	RaT	Partitions libres
PRR1/2							
CSD_8	33%	17%	30%	82%	50%	90%	2
CSD_{16}	59%	17%	51%	89%	25%	77%	1
CSD_{32}	100%	17%	84%	100%	17%	84%	0
PRR3							
CSD_8	50%	25%	45%	82%	50%	90%	1
CSD_{16}	89%	25%	77%	89%	25%	77%	0

configuration. Comme montre le Tableau 5.4, un gain important en termes de temps de reconfiguration est obtenu en appliquant l'adaptation des PRR. En fait, le temps total de reconfiguration dépend de la taille de bitstream, qui dépend en même temps de la taille PRR. Sans adaptation, le temps de reconfiguration est identique pour les différentes versions de CSD (CSD_8, CSD_{16} et CSD_{32}), car une même taille de zone reconfigurable est utilisée pour implémenter les trois modules matériels correspondants. Cependant, pour le cas d'implémentation avec adaptation, chaque module occupe un nombre choisi des partitions occupées ce qui nous permet d'optimiser la taille de la zone à configurer par le bitstream et par conséquent réduire le temps de reconfiguration.

Les résultats obtenus dans le Tableau 5.4 montrent que le temps de reconfiguration pour la plus petite tâche matérielle (CSD_8) est divisé par trois, tandis que celui correspondant au module CSD_{16} présente une diminution de 33%.

Tableau 5.4 – *Évaluation du temps de reconfiguration(ms)*

CSD IP	**temps de reconfiguration**		**Gain**
	Sans adaptation	Avec adaptation	
CSD_8	0,84	0,28	66%
CSD_{16}	0,84	0,56	33%
CSD_{32}	0,84	0,84	0%

Les temps de reconfiguration, illustrés dans le Tableau 5.4, sont calculés à l'aide de l'équation (5.1). Pour une fréquence de fonctionnement de 100 MHz du port d'accès à la configuration (ICAP) et une largeur de bus de données de 32 bits, un débit maximal de 400KB / ms est atteint .

$$\text{reconfiguration}_{\text{time}} = \frac{\text{bitstream}_{\text{size}}}{\text{configuration}_{\text{frequency}} * \text{databus}_{\text{widh}}} \tag{5.1}$$

2.2 Application 2 : DCT de taille variable

La contribution de cette partie est double. D'une part, nous proposons une architecture de calcul temps réel d'un DCT adaptatif à base de la reconfiguration dynamique partielle. D'autre part, nous proposons une gestion efficace des régions reconfigurables. Cette gestion est effectuée en ajustant la région reconfigurable aux ressources matérielles nécessaires aux différentes tailles de la DCT. Avec cette méthode, la taille des régions reconfigurables peut être modifiée en cours d'exécution. Elle permet d'optimiser l'utilisation des ressources, gérer efficacement les zones reconfigurables et réduire le temps de la reconfiguration. Par conséquent, nous visons à implémenter le module de calcul DCT basé sur l'arithmétique distribuée sur un système reconfigurable afin d'assurer les performances et réduire le temps de reconfiguration [HRAM16]

La DCT est une technique permettant la conversion d'un signal temporel en composantes fréquentielles. Cette technique est largement utilisée dans la compression des données et d'images. Il y a huit variantes de la transformée discrète en cosinus, mais la plus utilisée est la DCT type II. L'expression de la transformation directe de DCT est présentée ci-dessous.

$$\mathrm{Y_k} = \frac{2}{\sqrt{\mathrm{N}}} \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{X_n} \frac{\cos(2\mathrm{n}+1)\mathrm{K}\boldsymbol{\pi}}{2\mathrm{N}} \tag{5.2}$$

2.2.1 Architecture reconfigurable proposée

L'architecture Zynq All Programmable SoC est composée de deux parties : d'un côté le Processing System (PS) et de l'autre côté le Programmable Logic (PL). L'interface AXI permet au PS de communiquer avec les IP de la partie PL. Cette plateforme intègre un processeur ARM Dual Core implanté en dur. L'architecture est composée d'une partie qui contient tous les éléments statiques du système et d'une partie reconfigurable. La reconfiguration est effectuée par le processeur interne (PS) sur la logique programmable (PL) via le port ICAP. Le module DCT est alloué dynamiquement dans la région reconfigurable (PRR) avec des tailles différentes (16, 8 et 4) correspondant aux exigences du système. La PRR est configurée en chargeant un nouveau bitstream partiel par l'ICAP. Le contrôler AXI HWICAP assure le transfert du bitstream jusqu'à ICAP. La Figure 5.4 représente la conception matérielle nécessaire pour mettre en œuvre un module de calcul DCT adaptatif.

Nous mettons en œuvre l'ensemble de conception matérielle proposés sur la plateforme Zedboard de Xilinx. Les résultats de synthèses obtenues à l'aide de l'outil Xilinx EDK, sont présentés dans la Tableau 5.5. Ce tableau compare l'utilisation des ressources matérielles avant et après l'intégration de la DCT avec différentes tailles. L'utilisation totale des slices du système est 2055, 2605 et 4262 respectivement pour N=4, 8 et 16.

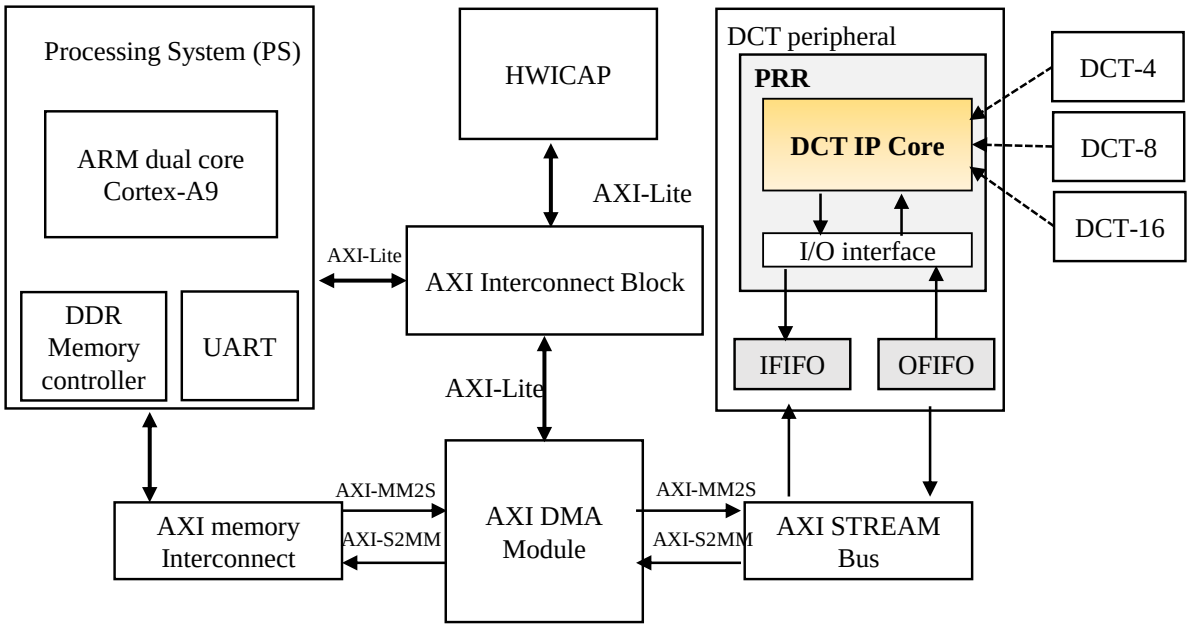

FIGURE *5.4 – Conception matérielle proposée*

TABLEAU *5.5 – Résultats de Synthèse*

Ressources Matérielles	**Avant l'intégration du DCT**	**Après l'intégration du DCT**		
		N=4	N=8	N=16
Registre	3891	4545	6322	13314
LUT	2855	3585	6486	13270
SLICES	1769	2055	2605	4262
BRAM36	4	4	4	4

2.2.2 Résultats

Grâce à la méthode de partitionnement, nous pouvons adapter la taille de la région reconfigurable aux tâches matérielles. Pour cela nous avons créé trois partitions PR que nous pouvons fusionner en fonction des besoins de chaque tâche matérielle. Pour la DCT de taille 16 les ressources logiques utilisées sont identiques à une architecture sans adaption, car la fonction utilise les trois partitions PR. Par contre avec la nouvelle méthode, pour la plus petite taille du DCT (N=4) nous avons une grande diminution au niveau des ressources qu'avec une architecture sans adaptation, car la fonction utilise une seule PR partition. Nous obtenons ainsi au niveau des ressources logiques une optimisation par rapport à la conception statique.

Le TABLEAU 5.6 montre le taux d'utilisation des ressources matérielles des tâches matérielles dct_4, dct_8 et dct_{16}. Avec une architecture reconfigurable basique, le taux d'utilisation dans une région reconfigurable de taille fixe est de 98 % pour le dct_{16} et pour dct_4 il est de 9,3 %. L'utilisation des ressources disponibles dans la région reconfigurable est faible pour

ce type d'architecture, ceci est dû à l'obligation de sélectionner suffisamment de ressources pour que la région reconfigurable soit compatible avec les différentes tailles des tâches matérielles. Pour une architecture adaptative, l'utilisation des ressources disponibles dans une région reconfigurable de module dct_4 augmente à 28 %. De plus nous avons deux partitions PR qui sont entièrement libres. Le partitionnement de la région reconfigurable nous a permis de réaliser une meilleure sélection ressources mémoire par rapport à une architecture reconfigurable sans partitionnement.

TABLEAU *5.6 – Utilisation des ressources matérielles*

N	**Sans adaptation**		**Avec adaptation**		
	RaCLB	RaBRAM	RaCLB	RaBRAM	Partitions libres
4	9,3%	9,2%	28%	28%	2
8	26%	26%	39%	38%	1
16	98%	97%	98%	97%	0

Comme montre le TABLEAU 5.7 , un gain important en termes de temps de reconfiguration est obtenu lorsque la région d'adaptation reconfigurable est appliquée. Sans adaptation, le temps de reconfiguration est le même pour les différentes versions du module DCT, parce que la même zone reconfigurable est utilisée pour mettre en œuvre les trois modules matériels correspondants. Toutefois, la taille de la région reconfigurable est adaptée à la tâche matérielle, ce module possède son propre temps de reconfiguration, qui dépend de la surface reconfigurable occupée. Les résultats obtenus montrent que le temps de reconfiguration de la tâche matérielle la plus petite (N = 4) est divisé par trois, tandis que celui correspondant à l'adresse DCT pour N égal à 8 présente une diminution de 33%.

TABLEAU *5.7 – Temps de reconfiguration(ms)*

N	**temps de reconfiguration**		**Gain**
	Sans adaptation	Avec adaptation	
4	1,84	0,28	66 %
8	1,84	0,56	33%
16	1,84	1,84	0%

3 Floorplanning : Analyse et évaluation des résultats

La comparaison directe de notre approche complète proposée avec les travaux présentés dans la littérature n'est pas possible. Le problème que notre approche traite plus de points que ne sont pas pris en charge par les travaux existants (relocation des tâches de tailles variables).

Dans cette section, nous considérons une application qui est présentée en fonction d'un graphe de tâches pour tester notre approche. Le nombre de régions reconfigurables et l'assignation des tâches à elles, sont connues à l'avance et sont présentées dans le graphe de tâches. Tout d'abord, nous déterminons les ressources de chaque région reconfigurable en fonction de besoin de plus gros module qui va l'accueillir. Ensuite, il faut mettre en évidence les contraintes de relocation. Enfin, nous servons de l'approche proposée pour déterminer les emplacements et la taille des régions en fonction de l'algorithme généré.

L'application est composée par six tâches de tailles différentes (Tableau 5.8) dont quatre sont exécutées en parallèle d'où la nécessité de quatre régions reconfigurables. Deux tâches vont être ré-allouées dans deux autres régions. La Figure 5.5 illustre la graphe de tâches de l'application.

Tableau *5.8 – Besoins en termes de ressources des tâches*

Tâches	**SLICE**	**BRAM**	**DSP**	**Frame CLB**	**Frame BRAM**	**Frame DSP**
(T1) FFT	572	3	28	15	1	4
(T2) DCT	352	0	0	9	0	0
(T3) FIR	308	0	16	8	0	2
(T4) CSD	1445	4	0	37	1	0
(T5) CSC	139	0	1	4	0	1
(T6) DFT	78	0	0	2	0	0

Dans le cas de reconfiguration dynamique partielle classique, la définition de ressources requises par les zones correspond au plus gros module reconfigurable y affecté. Néanmoins dans le cas de relocation, les régions qui allouent le même module reconfigurable doivent être identiques en terme de nombre, type et disposition de ressource. La taille de ces régions correspond au plus gros module occupé par l'un d'entre eux. En effet, une mauvaise gestion de ressources matérielles est prouvée lors du processus de relocation. Le Tableau 5.9 présente les différents ressources requis pour chaque région dans les cas sans et avec relocation.

Comme nous considérons des modules de taille variable à placer dans la même région reconfigurable et les ré-allouer à des autres régions, l'approche d'adaptation de zone est appliquée pour une meilleure gestion de la zone reconfigurable. Dans notre travail, nous proposons quatre algorithmes de recherche des zones homogènes. Bien que, le première algorithme est conçu pour une recherche automatique des zones totalement identiques, les trois autres algorithmes conduisent à une recherche optimisée des zones reconfigurables. Ces

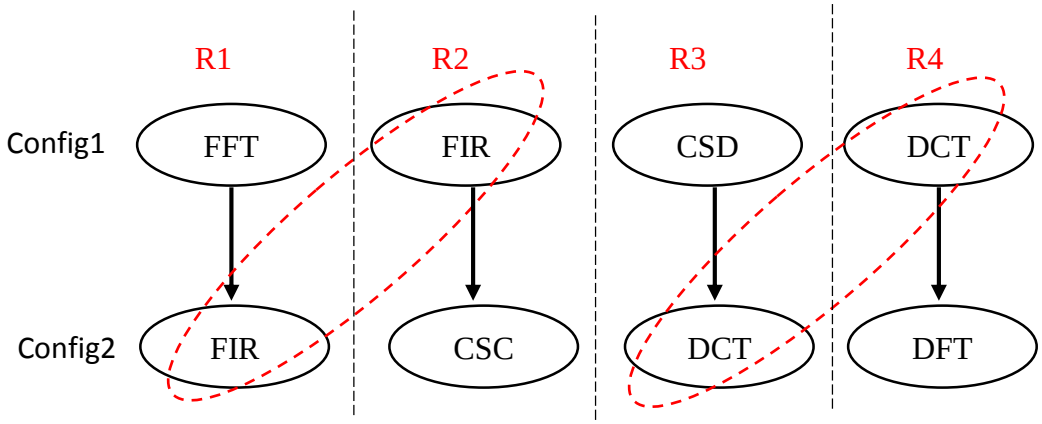

FIGURE 5.5 – *Graphe de tâches*

TABLEAU 5.9 – *Estimation des ressources requis pour chaque PRR*

régions	**SLICE**		**BRAM**		**DSP**	
	Sans relocation	Avec relocation	Sans relocation	Avec relocation	Sans relocation	Avec relocation
PRR1	572	572	3	3	28	28
PRR2	308	572	0	3	16	28
PRR3	1445	1445	4	4	0	0
PRR4	352	1445	0	4	0	0

trois algorithmes cherchent une zone adaptée à la petite zone en fonction de ses nombres de ressources. Ils effectuent une recherche locale dans la grande région pour trouver une partie totalement compatible avec la petite région en fonction de nombre, disposition et type de ressources. L'adaptation de la zone se fait soit verticalement, horizontalement ou en 2D.

La FIGURE 5.5 représente les résultats d'implémentation de la première configuration du graphe des tâches, où différents cas de floorplanning ont été générés en tenant compte de la relocation : homogénéité totale sans adaptation des zones reconfigurables, adaptation verticale, adaptation horizontale et adaptation 2D. Nous avons exploité la carte FPGA Virtex-5(XUPlx100t) en tenant compte de l'hétérogénéité et la répartition des ressources sur sa zone reconfigurable.

L'objectif de notre approche est d'optimiser les ressources matérielles utilisées. Nous évaluons la qualité de floorplanning obtenu par l'approche proposée en calculant le gain des ressources gaspillées par l'équation 5.3. Comme montre le TABLEAU 5.10, le gaspillage des ressources matérielles dans la région peut atteindre des gains très faibles comparés à ceux obtenus avec le floorplanning sans adaptation. Dans le cas d'une implémentation classique, l'ensemble des PRR est réservé pour réaliser toutes les tâches correspondantes

d'où la nécessité d'être identiques. Ce mode d'implémentation conduit à une utilisation inefficace des ressources matérielles.

$$G_{WR} = WR_{n,\,t}/R_{n,\,t} \tag{5.3}$$

TABLEAU *5.10 – Tableau récapitulatif des différents résultats obtenus*

Régions	**Totalement**			**Verticalement**			**Horizontalement**			**2-D**		
	slice	BRA	DSP	slice	BRA	DSP	slice	BRA	DSP	slice	BRA	DSP
PRR1												
$R_{n,t}$	640	16	32	640	16	32	640	16	32	640	16	32
G_{WR_t}	10,6 %	81%	12,5%	10,6 %	81%	12,5%	10,6 %	81%	12,5%	10,6 %	81%	12,5%
PRR2												
$R_{n,t}$	640	16	32	320	8	16	480	16	32	320	8	16
G_{WR_t}	52 %	100%	50 %	3,7 %	100%	0%	36 %	100%	50%	3,7 %	100%	0%
PRR3												
$R_{n,t}$	1520	8	0	1520	8	0	1520	8	0	1520	8	0
G_{WR_t}	5 %	50%	0	5 %	50%	0	5 %	50%	0	5 %	50%	0
PRR4												
$R_{n,t}$	1520	8	0	760	4	0	400	0	0	360	0	0
G_{WR_t}	77 %	100 %	0 %	27 %	50%	0 %	3,15 %	0%	0%	0,5 %	0%	0%

Grâce à l'exploitation de l'approche du partitionnement pour le floorplanning dans le cas de relocation, l'algorithme proposé a pu trouver des partitions reconfigurbles adaptées aux besoins de chaque tâche matérielle. Pour cela, une recherche locale est effectué dans la grande zone pour identifier la petite zone qui correspond au besoin de la tâche relogable entre elles. Le gain obtenu avec ces algorithmes dépendent de la tâche matérielle implémentée sur la région reconfigurable. Plus précisément, cela dépend du type d'adaptation souhaité pour identifier région qu'occupe cette tâche. Pour la première et la troisième région (PRR1 et PPR3) reconfigurable, les gains de ressources gaspillées sont identiques pour tous les algorithmes avec ou sans adaptation, car ils vont accueillir les plus gros modules. Pour PRR2 et PRR4, nous avons remarqué qu'il y une diminution des ressources gaspillées ainsi une optimisation de ressources utilisées avec les nouveaux algorithmes d'adaptation. Par exemple, pour PRR4 , le gain de gaspillage passe de 77 % à 0,5% de ressources utilisées

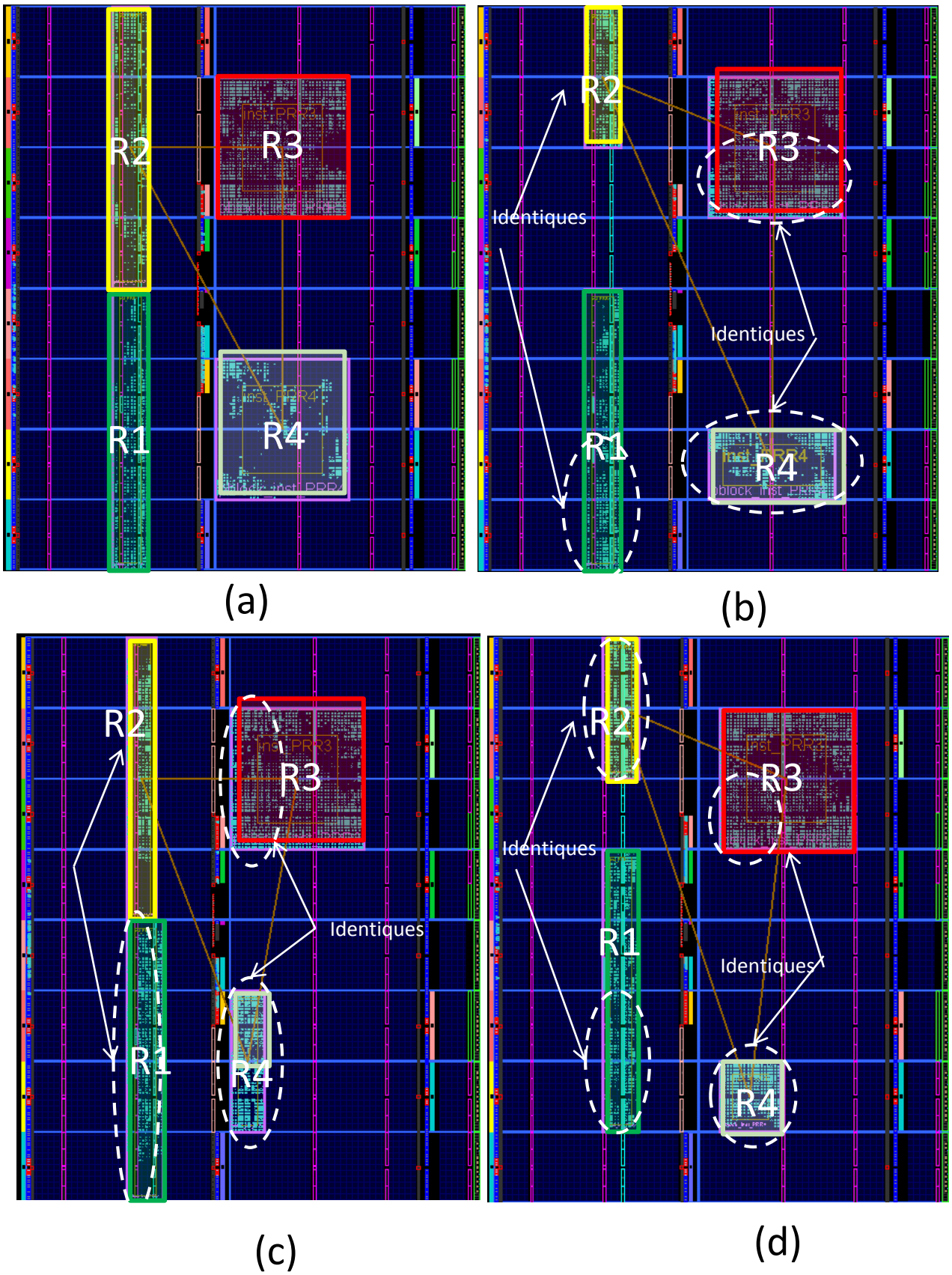

FIGURE 5.6 – *Resultat de Floorplanning (a) Totalement identique (b) adaptation verticale (c) adaptation horizontale (d) adaptation 2D*

avec une adaptation 2-D.

L'adaptation de la région reconfigurable nous a permis de réduire le gain de gaspillage G_{WR_t} et réaliser une meilleure sélection de ressources par rapport à une architecture reconfigurable sans adaptation. Néanmoins, dans cette approche, nous n'avons pas tenu compte des ressources gaspillées lors de configuration des tâches reconfigurables de tailles variables dans une même région reconfigurable. Cependant, une autre amélioration est apportée à l'approche proposée en exploitant la technique du partitionnement. Cette technique consiste à subdiviser les régions reconfigurables en plusieurs partitions reconfigurables (PPR) qui permettent de recevoir n'importe quel module reconfigurable. Chaque module fusionne une ou plusieurs partitions reconfigurables en fonction de ses besoins en ressources. La Figure 5.7 montre un exemple de partitionnement du PRR3.

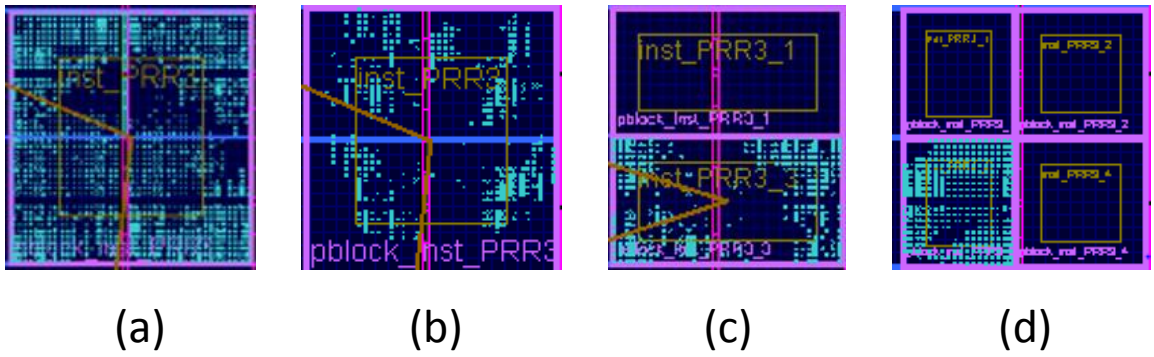

Figure *5.7 – Resultat de partitionnement (a) Totalement identique (b) adaptation verticale (c) adaptation horizontale (d) adaptation 2D*

L'implémentation des tâches CSD et DCT sur la région PPR3 sans partitionnement est efficace pour la tâche CSD en terme d'utilisation des ressources (Figure 3.15.(a) et (b)). Par contre, pour la deuxième configuration du graphe de tâche, le module matériel DCT n'utilise qu'une partie des ressources de la région reconfigurable et les ressources restantes sont perdues. Dans la Figure 3.15.(c) et .(d), la région reconfigurable est divisée respectivement en deux et en quatre partitions. Une seule partition PR accueillir la tâche DCT en libérant les ressources inutilisées. Si nous considérons l'efficacité de la surface comme le rapport entre le nombre de ressources requises par le module reconfigurable et le nombre de ressources réservées par partition. Sans partitionnement l'efficacité de la zone est respectivement de 88% et 40,8% pour les modules CSD et DCT. Avec adaptation, l'efficacité de la zone est respectivement de 88% et 93% pour les modules CSD et DCT. De plus, dans cet exemple dans la figure 3.15.(d), trois partitions PR reste libres et par conséquent elle peut être allouée à un autre module, montrant que la méthode de partitionnement proposée fournit un gain significatif dans l'utilisation des ressources matérielles par rapport aux approches existantes.

4 Conclusion

Dans ce chapitre, différentes études de cas ont été présentées pour valider et évaluer les différentes contributions proposées dans ce manuscrit.

Dans la première application, nous avons proposé, une implémentation dynamique d'un système de détection de coupure vidéo basé sur le CSD MPEG-7. La conception proposée a été implémentée pour les différents niveaux de quantification 32, 16 et 8. Les modules obtenus peuvent être chargés en fonction des exigences de l'application et situés dans différents PRR possibles. Notre objectif était également de générer un seul bitstream pour chaque PRM en utilisant la technique de relocation et l'adapter à l'emplacement souhaité. L'efficacité de la méthode de relocation et la capacité d'allouer des modules de tailles variables ont été les points clés à prendre en considération. Un Virtex 5 FPGA a été utilisé pour implémenter le système. Les résultats expérimentaux montrent une diminution de la quantité de mémoire utilisée pour stocker les bitstreams partiels, un gain significatif en termes d'efficacité d'utilisation des ressources matérielles et une réduction de temps de reconfiguration par rapport au flot de conception standard Xilinx DPR.

Notre étude, dans la seconde application, a été concrétisée une implémentation de la transformé DCT sur la plateforme Zedboard. Il s'agit d'une application composée de trois différentes tailles de DCT : 4, 8 et 16 qui ont été implémentés de façon alternée dans des régions reconfigurables de façon dynamique et partielle. Nous avons présenté la conception d'une architecture adaptative appliquée à la transformé DCT et basée sur les concepts de partitionnement d'un FPGA reconfigurable. Nous avons explicité à travers cet exemple d'application, la stratégie de mise en œuvre de cette méthode de partitionnement et nous avons validé expérimentalement la solution d'implantation matérielle du partitionnement. En effet, nous avons montré que le partitionnent apportait une réduction des ressources matérielles utilisées et un gain sur le temps de reconfiguration.

Dans la dernière partie, nous avons présenté une application en fonction d'un graphe de tâches. Cette application a pour rôle de tester et évaluer notre algorithme PLM qui assure un floorplanning automatique dans le cas de relocation en exploitation la technique du partitionnement. Cette étude a montré une efficacité de recherche de cet algorithme en trouvant et définissant des partitions reconfigurables adaptées aux besoins de chaque tâche matérielle. Une amélioration significative dans l'efficacité d'utilisation des ressources matérielles ainsi qu'une réduction du temps de reconfiguration et du besoin de la mémoire ont été montré.

Conclusion générale et perspectives

Les architectures reconfigurables sont considérées comme une solution primordiale pour réaliser des systèmes sur puce efficaces et flexibles afin de répondre aux nouveaux besoins et aux nouvelles exigences des applications technologiques d'aujourd'hui. L'objectif des architectures reconfigurables est de combiner la capacité de modification du comportement du système en temps réel avec les exigences élevées des performances et de fiabilité. Cependant, l'exploitation de RDP dans les applications industrielles est loin d'être complètement réalisable, malgré toutes ces évolutions. Cela s'explique principalement par les difficultés rencontrées dans la mise en œuvre pratique des conceptions des applications RDP.

Bien que les flots de conception RDP traditionnels exploitent le RDP comme une méthode de changement de contexte pour les opérations basiques de certains modules sur un espace physique bien défini et de taille fixe sur un FPGA, l'absence des méthodologies de conception bien adaptées et la limitation des moyens et des outils mis à la disposition des concepteurs sont la cause principale d'une grande complexité de conception. Cette thèse a visé à exploiter mieux les ressources matérielles du FPGA en gérant efficacement l'emplacement et l'exécution des tâches reconfigurables de tailles variables. Dans cette thèse, nous avons abordé certain problèmes de conception et les défis dans la mise en œuvre d'un système reconfigurable en mettant en œuvre les concepts efficacité d'utilisation des ressources matérielles, flexibilité et adaptabilité. Il s'agit de proposer des nouvelles approches et méthodologies de conception pour répondre de façon efficace à quelques problèmes rencontrés pendant le processus de conception pour les architectures reconfigurables tels que les problèmes de la relocation et du floorplanning.

Pour la relocation, nous avons traité deux défis. Dans un premier lieu, nous avons proposé un nouveau flot de conception qui assure la relocation des tâches reconfigurables sur des zones reconfigurables fixes. Les objectifs de ce flot est de surmonter les limitations du flot de conception DPR standard et d'augmenter la souplesse et la flexibilité du système. Nous avons aussi géré les interfaces de communication inter-modules qui jouent un rôle primordial dans la conception. Le deuxième problème traité dans la mise en œuvre de la relocation, qui n'est pas encore abordé dans la littérature et qui est sûrement le plus complexe et le plus intéressant à résoudre est l'efficacité d'utilisation des ressources matérielles pour augmenter l'efficacité du système dans le cas des tâches de tailles variables. Afin de répondre à ces contraintes, nous avons proposé une nouvelle méthodologie de conception pour la relocation des tâches de taille variable en adaptant la taille de la région PR à la taille de la tâche allouée.

La deuxième partie de ces travaux de cette thèse a été consacrée à présenter une nouvelle

approche pour automatiser le floorplanning dans les FPGA reconfigurables. L'algorithme est basé sur un modèle mathématique PLMNE. Cet algorithme assure une exploitation efficace de l'espace disponible dans le FPGA en utilisant des solveurs avancées permettant de résoudre des problèmes complexes d'optimisation. Avec cette approche, nous avons garanti la satisfaction des contraintes des architectures partiellement reconfigurables et nous avons proposé une technique simple pour modéliser et caractériser efficacement un FPGA. Le modèle présenté pour le floorplanning permet au concepteur d'effectuer une recherche dans tout l'espace de FPGA pour obtenir des meilleurs résultats. Le processus de recherche est déterminé par une fonction objective personnalisée capable de définir différentes métriques, de sorte qu'un grand contrôle est donné sur le type de solutions souhaitées.

De plus, deux extensions de l'algorithme de floorplanning ont été présentées. La première consiste à supporter la technique de relocation des bitstreams partiels. Ce modèle offre au concepteur la possibilité de rechercher des zones compatibles en ajoutant des nouvelles contraintes et formulations dans le modèle initial de floorplanning PLM. Dans la seconde extension, une nouvelle amélioration du modèle PLM a été proposée pour optimiser les ressources matérielles utilisées et prendre en considération les tâches de tailles variables. Nous avons proposé d'automatiser l'adaptation de la région reconfigurable au besoin en terme de ressources matérielles de la tâche reconfigurable.

La dernière partie a été consacrée à l'étude expérimentale pour la validation de l'ensemble des points présentés dans ces travaux de thèse. En effet, nous avons validé et évalué la technique de relocation des tâches matérielles de tailles différentes sur deux exemples d'applications. Nous avons montré à travers ces applications l'efficacité d'utilisation des ressources matérielles en exploitant le partitionnement de la région reconfigurable dans le cas de relocation. Ensuite, nous avons considéré une étude de cas présenté sous forme d'un graphe de tâches pour évaluer l'approche PLM proposée et ses extensions. Cette étude montre l'impact de partitionnement sur les solutions finales du floorplanning dans le cas de relocation des tâches de tailles variables.

Le travail effectué dans cette thèse peut être suivi selon plusieurs directions. Nous citons ici quelques-unes :

Architectures overlay pour les FPGA

Un overlay est une architecture de traitement reconfigurable [KSGL12, BL12], qui peut être mis en œuvre sur un FPGA pour implémenter des circuits applicatifs. Les overlays présentent actuellement un grand intérêt car ces architectures offrent plusieurs avantages : rapidité de configuration, facilité de programmation. L'accent est mis sur l'exploitation de la technique de relocation des tâches matérielles pour assurer une mise en œuvre efficace et flexible de la technologie reconfigurable dans les overlay .

Position indépendante du bitstream : Dans le chapitre 2, nous avons proposé une

méthode de conception du système reconfigurable basée sur la technique de relocation. Les différentes zones reconfigurables qui vont allouer un même module sont prévues et fixées avant la phase de conception. Une nouvelle méthodologie qui assure la relocation d'un bitstream privé de toute information relative aux détails de la zone reconfigurable ciblée peut être de grand intérêt. Pour traiter cette solution, toute la surface logique du FPGA sera partitionnée en zones reconfigurables identiques. Une partie de ces zones sera dédiée aux réseaux d'interconnexion et le reste sera alloué par les modules reconfigurables et la partie statique. Dans ce cas, selon la disponibilité des zones reconfigurables dans toute le FPGA, le module peut être alloué à n'importe quelle position sans aucune information prévue et en garantissant la compatibilité entre eux. En revanche, pour pallier le problème des modules de tailles différentes, un ensemble de zones sera fusionné en gardant les contraintes d'homogénéité des zones reconfigurables.

Annexe A

Contraintes de mise en oeuvre de la reconfiguration

1 Unité de reconfiguration

La plus petite unité reconfigurable adressable est appelée trame (frame). Une trame est un ensemble de ressources programmables (CLB, DSP, BRAM). La région reconfigurable (PRR) doit donc être un multiple de trames et contient principalement des CLBs, DSPs et des BRAMs. Le nombre de ressources logiques par trame ainsi le nombre de trames dépendent de la famille du FPGA. Le Tableau A.1 résume le nombre de ressources logique par chaque type de trame pour différentes familles du FPGAs.

Tableau *A.1 – Nombre de ressources matérielles*

Type de l'FPGA	**Nombre des CLBs par frame**	**Nombre des BRAM 18k par frame**	**Nombre des DSP par frame**
Virtex4	16	4	4
Virtex5	20	4 8	
Virtex6	40	8	8
Serie-7 (ZYNQ)	50	10	10

La taille des trames correspond aussi aux différents domaines d'horloge du FPGA. Dans chaque domaine d'horloge, nous pouvons définir une zone reconfigurable avec leur propre horloge de fonctionnement. La Figure A.1 illustre l'architecture du FPGA Zynq et la distribution de son arbre d'horloge.

2 Mémoires de stockage des bitstreams

Les performances des architectures reconfigurables sont fortement limitées à cause de la capacité des unités mémoires. La technique de relocation est particulièrement visé à optimiser l'utilisation des mémoires pour stocker les bitstreams partiels. Plusieurs approches dans la littérature ont été effectuées pour exploiter la technique de la relocation. Chaque

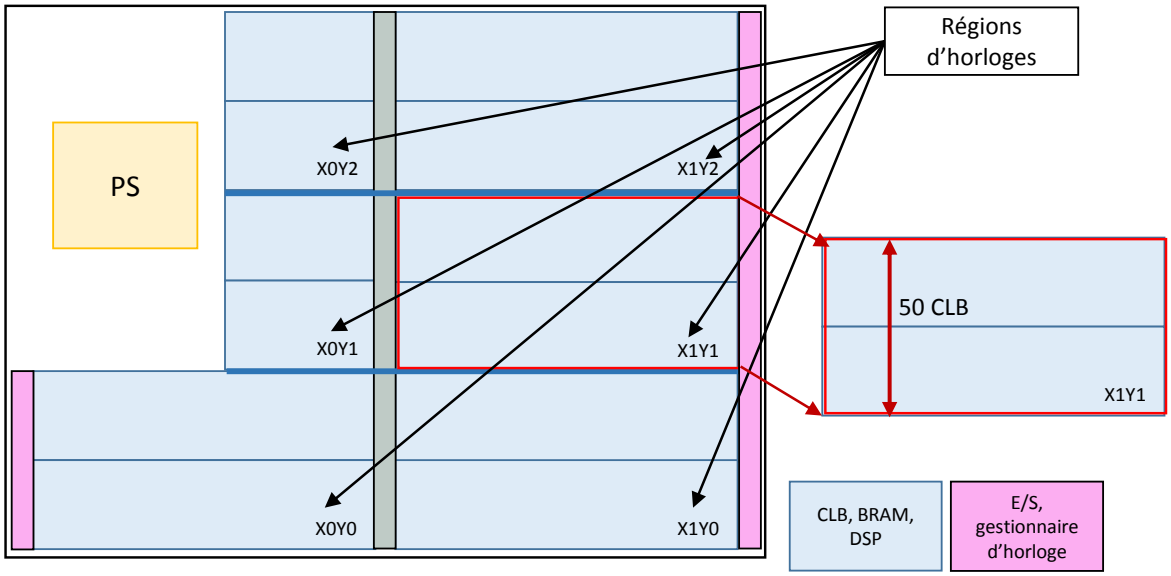

FIGURE *A.1 – Unité de reconfiguration dans le FPGA Zynq*

approche vise à des différents scénarios pour répondre aux performances des systèmes en termes des besoins de mémoire.

Le choix du type de la mémoire utilisée dépend d'un ensemble des paramètres (nombre de mémoires disponibles, tailles, nombre de ports d'entrées/sorties et temps d'accès). Plusieurs systèmes utilisent les mémoires caches (BRAM) pour le stockage des bitstreams grâce à son temps d'accès rapide. Néanmoins, cette mémoire ne supporte pas le stockage d'un grand nombre de bitstreams dans le cas des applications plus complexes, d'où la nécessité d'ajouter sur la plateforme du SoC une mémoire externe telle que DDR, compact flash, SD.

3 Éléments de communication

Les interfaces de communications proposées par Xilinx pour la reconfiguration dynamique partielle sont des moyens physiques de communication qui relient un module statique à un module dynamique. Ces interfaces sont évoluées en parallèle avec les flots de conceptions. Dans les anciennes plateformes telles que Virtex-II et Virtex-II-Pro, Xilinx utilise la première version des bus macros qui sont basés sur les buffers à trois états par Signal (Tri-State Buffer : TBUF) [Xil04]. Cette technologie est n'est pas flexible où la communication est unidirectionnelle soit de gauche à droite ou de droite à gauche. S'il y a un signal bidirectionnel, deux bus macros des directions opposées sont nécessaires. En 2004, une nouvelle solution de communication plus flexible nommés slices bus macro a été développé par Hübner et al [HBB04]. Les slices bus macros sont basés sur des LUT et des fils d'acheminement entre eux comme montre la FIGURE A.2 . Cette technologie permettent d'améliorer les

performances de synchronisation de simplifier le flot de conception et l'implémentation des interconnexions des modules reconfigurables. L'inconvénient majeur de cette technologie est que pour chaque signal il y a deux LUTs perdus.

En 2009, une nouvelle technologie de connexion physique entre la logique statique et la logique reconfigurable nommée pin partition (ou aussi appelés Proxy Logic) a été proposée par Xilinx. Cette technologie simplifie énormément le flot de conception des architectures reconfigurables grâce au placement automatique des pins dans la région reconfigurable. Il utilise un seul LUT qui est inséré automatiquement dans partie reconfigurable. Les pins partitions sont caractérisés par leurs type (entrée ou sortie), par leurs emplacement (coordonnés XY de la SLICE où est instancié la LUT), par leur élément logique de base (BEL) et par leur point de connexion.

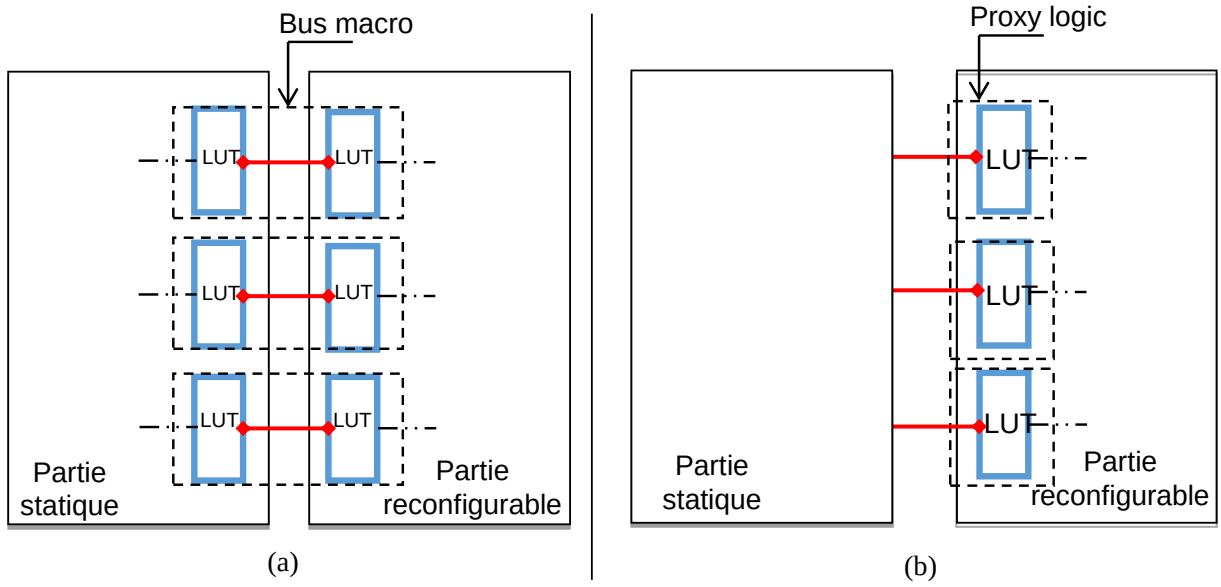

FIGURE *A.2 – Systèmes d'interconnexion (a) Architecture de slice bus macro, (b) Architecture de pin partition*

4 Ports de configuration

La configuration est l'étape où le bitstream est envoyé au FPGA. Un bitstream complet configure la totalité de la mémoire de configuration du FPGA pour un système statique où il définit l'état initial des cellules SRAM pour un système reconfigurable. Le bitstream partiel configure une partie seulement du FPGA. Pour cela il existe plusieurs interfaces de configuration permettant de charger les bitstreams complet et partiels à la carte cible où est câblée le FPGA. La configuration peut être effectuée soit de l'extérieur par un ordinateur via le port JTAG (Joint Test Action Group) [For02], soit de l'intérieur par un processeur via le port ICAP (Internal Configuration Access Port) [HKT11] ou PCAP (Processor Configu-

ration Access Port) [SGZW16]. La (FIGURE A.3) illustre l'aspect d'utilisation des différents ports de configuration mentionnés.

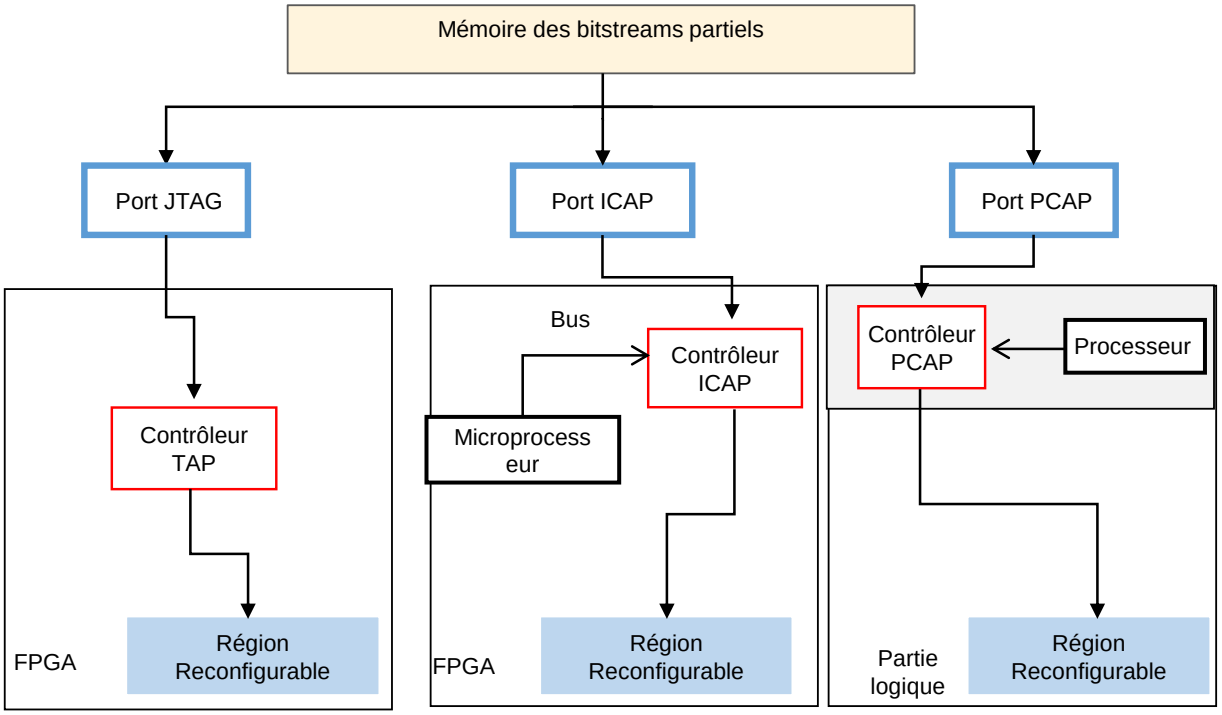

FIGURE *A.3 – Différents ports de configuration*

Annexe B

structure du bitstream

Un bitstream est un fichier binaire dans lequel les informations de configuration du FPGA sont stockées. Il contient toutes les données pour configurer les cellules SRAM, ainsi toutes les commandes pour commander le fonctionnement du FPGA.

Un bitstream peut être total ou partiel. Un Bitstream total configure entièrement la mémoire de configuration (SRAM) dans le FPGA. Il est utilisé pour les systèmes statiques ou au début de l'exécution d'un système reconfigurable pour définir l'état initial des cellules SRAM. Un bitstream partiel configure uniquement une partie du dispositif. Le bitstream partiel est créé pour chaque PRM ait lieu sur un PRR en particulier.

Un bitstream est généralement composé d'une entité, un mot de synchronisation et un ensemble de paquets de configuration de tailles différentes. Les paquets contiennent des commandes de configuration ou des données de configuration. Les différents éléments du fichier bitstream illustré à la FIGURE B.1sont les suivant :

— En-tête : contient des informations spécifiques comme l'heure et la date de création, le nom du fichier NCD, la famille et le type de FPGA utilisé.

— DUMMY et un mot de synchronisation : Le bitstream commence toujours par une séquence d'ouverture composée d'une suite de mot (0xFFFFFFFF) et un mot de synchronisations (0xAA995566). Le but de ces mots est d'aligner les limites de mots à 32 bits et pour signaler le début des commandes de configuration.

— Commandes de configuration : configurent les registres de configuration en mode de lecture ou en mode d'écriture ;

— Données de configuration : peuvent être représentés en une trame unique, ou en un ensemble de la trame, ou en toute l'espace du FPGA. Ils sont des données représentant les valeurs dans différentes ressources matérielles (CLB, IOB, BRAM, et DSP) correspondant à la région de configuration.

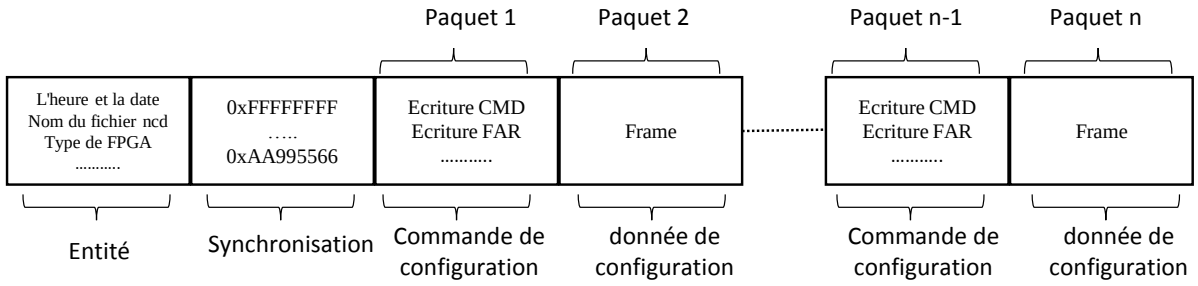

FIGURE *B.1 – Composition simplifiée d'un bitstream*

Afin de mieux comprendre le fonctionnement interne d'un flux binaire partiel, il faut comprendre les différents types de paquets de commande et les différents registres de configuration.

1 Paquets de configuration

Le flux binaire de FPGA se compose de deux types de paquets :

— Paquet du type 1 : Dans ce paquet le type d'opération de configuration (lecture ou écriture) à effectuer sur le registre est défini, ainsi l'adresse du registre et le nombre de mots de données destinés à être écrit dans ce registre. LaFIGURE B.2 illustre la composition du paquet de type 1. Il est composé de 32 bits : 3 bits pour l'entête ([31 29]), 2 bits pour le type de l'opération (opcode(FIGURE B.3)), 13 bits pour l'adresse de registre visé ([26 13]), 2 bits réservé ([12 :11]) et 11 bit pour définir le nombre de mot des données ([10 :0]). Les paquets de type 1 sont le type de paquet le plus courant dans le fichier binaire de configuration.

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxx

FIGURE *B.2 – Paquet type 1 de bitstream Virtex-5*

Opcode	Function
00	NOP
01	Read
10	Write
11	Reserved

FIGURE *B.3 – Codage Opcode*

— Paquet du type 2 : ce type de Paquet n'est utilisé que lorsque le nombre de bit pour présenter le nombre de mot des données de type 1 (11 bits) est insuffisant. Il est utilisé pour la lecture ou l'écriture d'une longue configuration. Aucune adresse n'est présentée dans ce type parce qu'il utilise l'adresse précédente du type 1. Par conséquent, un paquet de type 2 doit toujours suivre un paquet de type 1, et il ne peut être considéré que comme un champ étendu pour le paquet de type 1. Dans

ce paquet, la partie réservée à l'entête est toujours sur 3 bits, 2 bits réservés pour l'Opcode et les 27 bits restants pour compter le nombre de mots de configuration comme montre la FIGURE B.4.

Header Type	Opcode	Word Count
[31:29]	[28:27]	[26:0]
010	RR	xxxxxxxxxxxxxxxxxxxxxxxxxx

FIGURE *B.4 – Paquet du type 2*

2 Registres de configuration

L'opération d'écriture ou de lecture effectuée par les paquets type 1 et type 2 ont pour but d'écrire (ou de lire) à un registre de configuration bien spécifique. Plusieurs types des registres de configuration qui peuvent se diffèrent d'un FPGA à un autre. La FIGURE B.5 illustre un ensemble des registres de configuration pour le Virtex-5.

Reg. Name	Read/Write	Address	Description
CRC	Read/Write	00000	CRC Register
FAR	Read/Write	00001	Frame Address Register
FDRI	Write	00010	Frame Data Register, Input Register (write configuration data)
FDRO	Read	00011	Frame Data Register, Output Register (read configuration data)
CMD	Read/Write	00100	Command Register
CTL0	Read/Write	00101	Control Register 0
MASK	Read/Write	00110	Masking Register for CTL0 and CTL1
STAT	Read	00111	Status Register
LOUT	Write	01000	Legacy Output Register (DOUT for daisy chain)
COR0	Read/Write	01001	Configuration Option Register 0
MFWR	Write	01010	Multiple Frame Write Register

FIGURE *B.5 – Registres de configuration*

Les registres qui nous intéressent pour effectuer la relocation des bitstreams partiels

sont : les registres FAR (Frame Address Register) et les registres CRC (contrôle de redondance cyclique). Ces registres sont expliqués dans les paragraphes suivantes.

FAR Le FAR est l'adresse de la mémoire de configuration dans laquelle les données du premier frame seront écrites. Un frame est la plus petite région reconfigurable dans l'FPGA. Le nombre des CLBs par Frame se diffère d'un FPGA à un autre.

Dans le bitstream total la valeur de registre FAR est toujours à zéro. Le FAR est automatiquement incrémenté après chaque écriture au registre FDRI ou lecture de registre FDRO. Le FAR est composé par 5 éléments comme montre la FIGURE B.6 :

— Top/bottom : Le FPGA est divisé verticalement en deux parties qui sont appelées moitié supérieur et moitié inférieur. Les frames de la moitié supérieure de l'FPGA sont les images miroir des frames de la moitié inférieure.

— Type de bloc : Pour indiquer le type de ressource (CLB, DSP, BRAM) utilisé. Le TABLEAU B.1 montre le codage de chaque type en 3 bit.

TABLEAU *B.1 – Codage du type de configuration*

Type du bloc logique	**Code de configuration**
Interconnect and Block Configuration (CLB, DSP, IOB, GCLK)	000
Block RAM	001
Interconnect and block special frames	010
Block RAM non-configuration frame	011

— Adresse ligne : le FPGA est divisé aussi verticalement par les régions d'horloge horizontales (HCLK). Ces régions sont numérotées à partir de 0, en commençant par le milieu de la puce et en l'incrémentant vers le haut. Pour La moitié inférieure de la puce en commençant au milieu de la puce par 0 et en l'incrémentant vers le bas.

— Adresse colonne ou adresse majeure : Chaque type de bloc logique à son propre adressage en commençant de gauche vers droite. cet adressage est identifié par l'adresse majeure.

— Adresse mineure : Les frames de chaque colonne sont identifiés par l'adresse mineure.

Les 32 bits de la FAR sont divisés en cinq parties comme le montre la FIGURE B.7 : type du bloc sur 3 bits , le Top/bottom sur 1 bit , l'adresse ligne sur 5 bits , l'adresse colonne sur 8 bits et l'adresse mineure sur 7 bits.

Le TABLEAU B.2 illustre la commande du FAR et un exemple d'adressage du registre frame dans un bitstream partiel.

Le FAR est représenté dans le bitstream en hexadécimale. Donc, il est nécessaire de le convertir de format hexadécimal au format binaire. Un exemple de FAR converti est illustré dans la FIGURE B.8

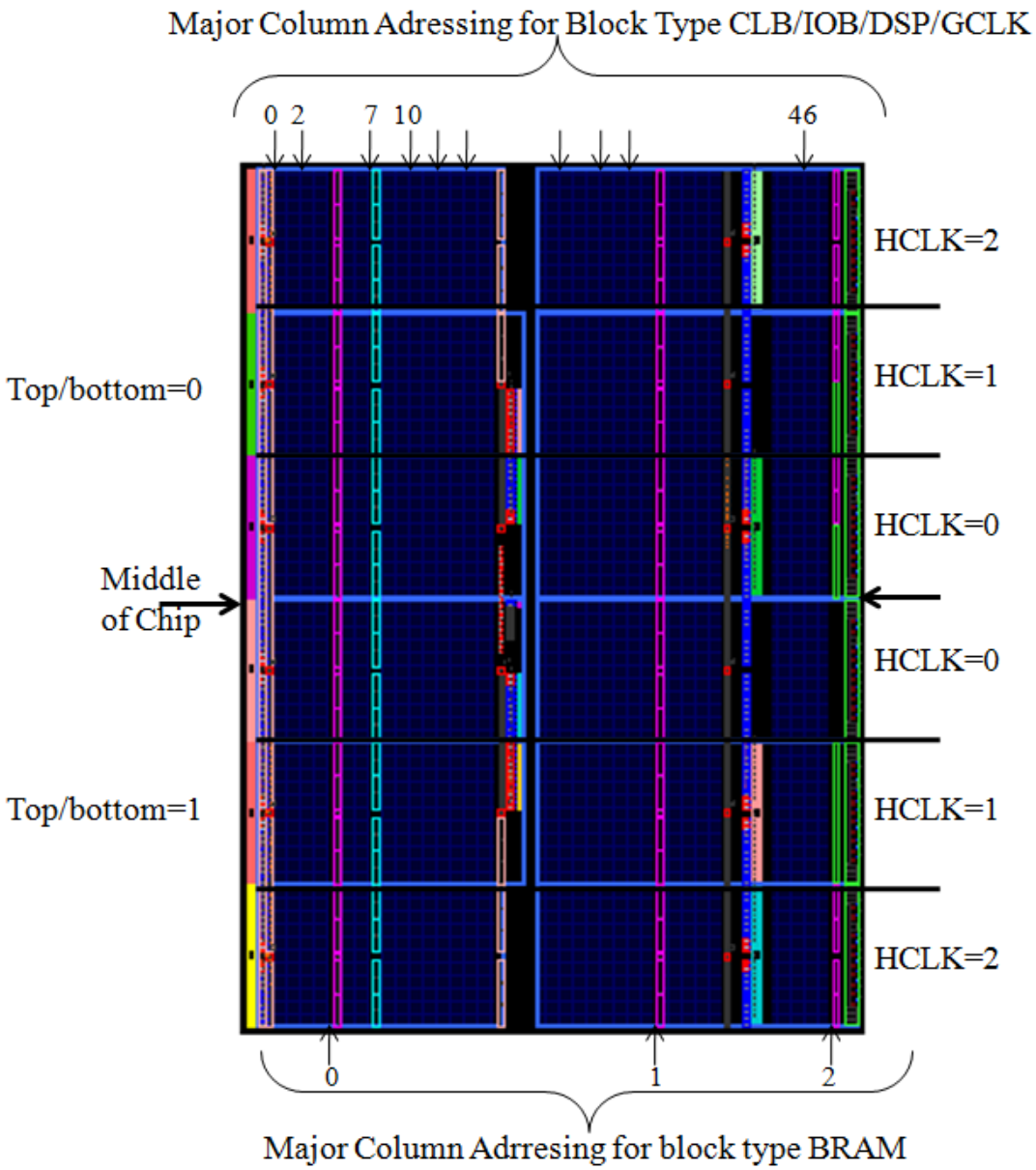

FIGURE *B.6 – Structure d'un FPGA Virtex-5*

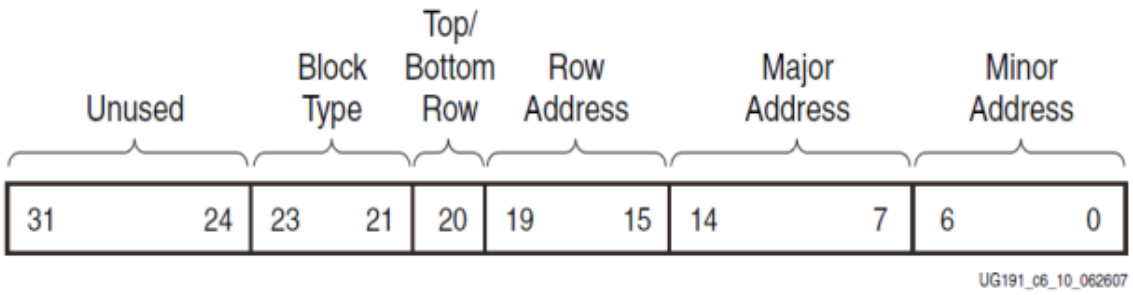

FIGURE *B.7 – Décomposition de registre FAR sur 32 bits*

TABLEAU *B.2 – Commande du FAR*

Commande pour Écrire dans le FAR	**30 00 20 01**
Donnée de configuration du FAR	00 00 80 40

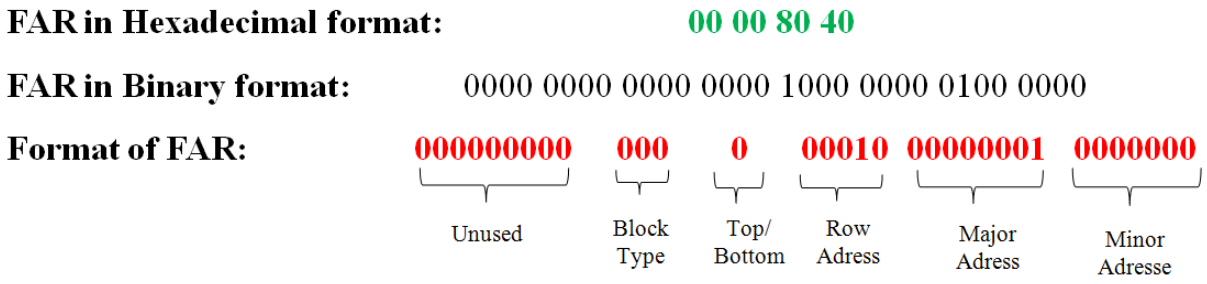

FIGURE *B.8 – Format de FAR*

CRC Le contrôle de redondance cyclique permet de détecter les erreurs de transmission ou de transfert par ajout, combinaison et comparaison de données redondantes. Pour la relocation, le CRC Permet de vérifier l'emplacement de bitstream, le type de FPGA, Synchronisation, la composition des commandes des données de configuration aux registres de configurations. Les CRC sont évalués avant et après la transmission ou le transfert, puis comparés pour s'assurer que les données sont strictement identiques. Une vérification du CRC se produit à la fin de chaque configuration de registre FDRI (frame data registrer write). Le CRC est basé sur une manipulation de polynômiale qui traite chaque configuration reçue comme un nombre binair.

$$\begin{aligned} x32 + x28 + x27 + x26 + x25 + x23 + x22 + x20 + x19 + x18 + x14 + x13 + x11 \\ + x10 + x9 + x8 + x6 + 1 \end{aligned} \tag{B.1}$$

Un exemple d'un paquet CRC dans le bitstream est montré dans le TABLEAU B.3

TABLEAU *B.3 – Commande du CRC*

Commande pour Écrire dans le Registre CRC	**30 00 80 01**
Donnée de configuration	00 0 00 07

Acronymes et abréviations

ASIC : Application Specific Integrated Circuit

BEL : Basic element logic

BiRF : Bitstream Reconfiguration Filter

BRAM : Block RAM

Bus Macro : Bus de communication entre PRR

CLB : Configurable Logic Block

CRC : Cyclic Redundancy Check

FAR : Frame adress register

FPGA : Field-Programmable Gate Array

FSM : Machine d'état finis

ICAP : Internal Configuration Access Port

IP : Intellectual Property

JTAG : Joint Test Action Group

LUT : Look-Up-Table

PCAP :Processor Configuration Access Port

PLMNE :Programmation linéaire mixtes en nombre entiers

PRM : Partially Reconfigurable Module

PRR :Partially Reconfigurable Region

PS : Processing System

RDP :Reconfiguration dynamique partielle

RTL : Register Transfer Level

SOC : System On Chip

SRAM : Static Random Access Memory

MJA : Major column address Word

NOC : Network-on-Chip

Publications personnelles

Revues internationales à comité de lecture

1) **Marwa HANNACHI**, Abdessalem BEN ABDELALI, Hassan RABAH et Abdellatif MTIBAA. « FPGA reconfiguration management technique and its application for adaptive and dynamic video cut detection system »., in Kuwait Journal of Science, Octobre 2017 , (ISSN 2307-4108 (print)) .

2) Abdessalem Ben Abdelali, **Marwa HANNACHI**, Mohamed Nidhal Krifa , Hassan RABAH et Abdellatif Mtibaa,« High-level design flow and environment for FPGA-based dynamic partial reconfiguration »,International Journal of Electronics · February 2017, doi:10.1080/00207217.2017.1293172

Conférences internationales avec actes et comité de lecture

1) **Marwa HANNACHI**, Hassan RABAH, Abdessalem BEN ABDELALI et Abdellatif MTIBAA. « Dynamic reconfigurable architecture for adaptive DCT implementation », 2016 2nd International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), Monastir, 2016, pp. 189-193. Mar 2016, doi:10.1109/ATSIP.2016.7523066

2) **Marwa HANNACHI**, Hassan RABAH, Slavisa JOVANOVIC, Abdessalem BEN ABDELALI et Abdellatif MTIBAA. « Efficient Relocation of Variable-sized Hardware Tasks for FPGA-based Adaptive Systems », conference ICM 2014 à Doha-Qatar, doi:10.1109/ICM.2014.7071847

Bibliographie

[ABBT04] Ali Ahmadinia, Christophe Bobda, Marcus Bednara, and Jürgen Teich. A new approach for on-line placement on reconfigurable devices. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 134. IEEE, 2004. 12

[ABT04] Ali Ahmadinia, Christophe Bobda, and Jürgen Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware. In *International Conference on Architecture of Computing Systems*, pages 125–139. Springer, 2004. 12

[AHTM14] Abdessalem Ben Abdelali, Marwa Hannachi, Lamjed Touil, and Abdellatif Mtibaa. Adequation and hardware implementation of the color structure descriptor for real-time temporal video segmentation. *Journal of Real-Time Image Processing*, pages 1–20, 2014. 91

[AM03] Saurabh N Adya and Igor L Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6) :1120–1135, 2003. 58

[BAHK+17] Abdessalem Ben Abdelali, Marwa Hannachi, Mohamed Nidhal Krifa, Hassan Rabah, and Abdellatif Mtibaa. High-level design flow and environment for fpga-based dynamic partial reconfiguration. *International Journal of Electronics*, pages 1–31, 2017. ix, 9

[BEN+04] Michel Berkelaar, Kjell Eikland, Peter Notebaert, et al. lpsolve : Open source (mixed-integer) linear programming system. *Eindhoven U. of Technology*, 2004. 63

[BHW+14] Rico Backasch, Gerald Hempel, Stefan Werner, Sven Groppe, and Thilo Pionteck. Identifying homogenous reconfigurable regions in heterogeneous fpgas for module relocation. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6. IEEE, 2014. ix, 27, 28, 29, 65

[BKS+00] Kiarash Bazargan, Ryan Kastner, Majid Sarrafzadeh, et al. Fast template placement for reconfigurable computing systems. *IEEE design & Test of Computers*, 17(1) :68–83, 2000. 12, 13

[BL12] Alexander Brant and Guy GF Lemieux. Zuma : An open fpga overlay architecture. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 93–96. IEEE, 2012. 106

[BLC07] Tobias Becker, Wayne Luk, and Peter YK Cheung. Enhancing relocatability of partial bitstreams for run-time reconfiguration. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 35–44. IEEE, 2007. 22, 24

[BMB10] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. New three-level resource management enhancing quality of offline hardware task placement on fpga. *International Journal of Reconfigurable Computing*, 2010 :4, 2010. 13

[BMK+04] Christophe Bobda, Mateusz Majer, Dirk Koch, Ali Ahmadinia, and Jürgen Teich. A dynamic noc approach for communication in reconfigurable devices. *Field Programmable Logic and Application*, pages 1032–1036, 2004. 16

[BMS11] Cristiana Bolchini, Antonio Miele, and Chiara Sandionigi. Automated resource-aware floorplanning of reconfigurable areas in partially-reconfigurable fpga systems. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 532–538. IEEE, 2011. 25, 71, 86

[Bou11] Mark Bourgeault. Alteras partial reconfiguration flow. *Tech. Rep.*, 2011. 11

[BQS07] Cristiana Bolchini, Davide Quarta, and Marco D Santambrogio. Seu mitigation for sram-based fpgas through dynamic partial reconfiguration. In *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pages 55–60. ACM, 2007. 18

[BSSK11] Pritha Banerjee, Megha Sangtani, and Susmita Sur-Kolay. Floorplanning for partially reconfigurable fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(1) :8–17, 2011. 25, 29, 86

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 61

[CFM+07] S. Corbetta, F. Ferrandi, M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto. Two novel approaches to online partial bitstream relocation in a dynamically reconfigurable system. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pages 457–458, March 2007. doi: 10.1109/ISVLSI.2007.99. 21, 24

[CMN+09] S. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, D. Sciuto, and P. Spoletini. Internal and external bitstream relocation for partial dynamic reconfiguration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(11) :1650–1654, Nov 2009. doi: 10.1109/TVLSI.2008.2005670. 21

[CMW96] Joseph Czyzyk, Sanjay Mehrotra, and Stephen J Wright. Pcx user guide. *Optimization Thechnology Center, Technical Report*, 96(01) :1–21, 1996. 63

[CW06] Lei Cheng and Martin DF Wong. Floorplan design for multimillion gate fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12) :2795–2805, 2006. 25, 29, 58, 86

[DS05] Klaus Danne and Sven Stühmeier. Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants. *From specification to embedded systems application*, pages 311–311, 2005. 13

[Est60] Gerald Estrin. Organization of computer systems : the fixed plus variable structure computer. In *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, pages 33–40. ACM, 1960. 2

[FGRG09] Adam Flynn, Ann Gordon-Ross, and Alan D George. Bitstream relocation with local clock domains for partially reconfigurable fpgas. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 300–303. European Design and Automation Association, 2009. 19

[FIC] Fico xpress optimization suite. URL : `http://www.fico.com/en/products/fico-xpress-optimization-suite`. 63

[FM06] Yan Feng and Dinesh P Mehta. Heterogeneous floorplanning for fpgas. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, pages 6–pp. IEEE, 2006. 25, 29, 86

[FNM+06] Fabrizio Ferrandi, Marco Novati, Massimo Morandi, Marco D Santambrogio, and Donatella Sciuto. Dynamic reconfiguration : Core relocation via partial bitstreams filtering with minimal overhead. In *System-on-Chip, 2006. International Symposium on*, pages 1–4. IEEE, 2006. 21

[For02] Tim Forcer. Xilinx fpga reconfiguration using jtag. 2002. 111

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. :1979

[gur] Gurobi optimization. URL : `http://www.gurobi.com/index`. 63

[GV00] Satish Ganesan and Ranga Vemuri. An integrated temporal partioning and partial reconfiguration technique for design latency improvement. In *Proceedings of the conference on Design, automation and test in Europe*, pages 320–325. ACM, 2000. 14

[Hal] Ed Hallett. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools)*. 22

[HARM17] Marwa Hannachi, Abdesslam B Abdelali, Hassan Rabah, and Abdellatif Mtibaa. Efficient reconfigurable regions management method for adaptive and dynamic fpga based systems. *Kuwait Journal of Science*, 44(4), 2017. 91

[HBB04] Michael Huebner, Tobias Becker, and Juergen Becker. Real-time lut-based network topologies for dynamic and partial fpga self-reconfiguration. In *Integrated Circuits and Systems Design, 2004. SBCCI 2004. 17th Symposium on*, pages 28–32. IEEE, 2004. 110

[HKT11] Simen Gimle Hansen, Dirk Koch, and Jim Torresen. High speed partial run-time reconfiguration using enhanced icap hard macro. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 174–180. IEEE, 2011. 111

[HL01] Edson L Horta and John W Lockwood. Parbit : a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas). 2001. 20, 24

[HRAM16] Marwa Hannachi, Hassan Rabah, Abdessalem Ben Abdelali, and Abdellatif Mtibaa. Dynamic reconfigurable architecture for adaptive dct implementation. In *Advanced Technologies for Signal and Image Processing (ATSIP), 2016 2nd International Conference on*, pages 189–193. IEEE, 2016. 96

[HRJ+14] Marwa Hannachi, Hassan Rabah, Slavisa Jovanovic, Abdessalem Ben Abdelali, and Abdellatif Mtibaa. Efficient relocation of variable-sized hardware tasks for fpga-based adaptive systems. In *Microelectronics (ICM), 2014 26th International Conference on*, pages 224–227. IEEE, 2014. 40

[HST14] Christophe Huriaux, Olivier Sentieys, and Russell Tessier. Fpga architecture support for heterogeneous, relocatable partial bitstreams. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–6. IEEE, 2014. ix, 22, 23, 24

[HSWK09] Jonathan Heiner, Benjamin Sellers, Michael Wirthlin, and Jeff Kalb. Fpga partial reconfiguration via configuration scrubbing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 99–104. IEEE, 2009. 18

[IAI+12] Yoshihiro Ichinomiya, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. A bitstream relocation technique to improve flexibility of partial reconfiguration. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 139–152. Springer, 2012. 27

[IBM] Ibm ilog cplex optimization studio. URL : `http://www-03.ibm.com/software/products/fr/ibmilogcpleoptistud`. 63

[JBGR09] Abelardo Jara-Berrocal and Ann Gordon-Ross. Runtime temporal partitioning assembly to reduce fpga reconfiguration time. In *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pages 374–379. IEEE, 2009. 14

[JTBW09] S Jovanović, Camel Tanougast, Christophe Bobda, and Serge Weber. Cunoc : A dynamic scalable communication structure for dynamically reconfigurable fpgas. *Microprocessors and Microsystems*, 33(1) :24–36, 2009. 16

[JwB09] Pan Jing-wei and Yan Bo. A new fabric of dynamic noc for communication in reconfigurable devices. In *Computer Technology and Development, 2009. ICCTD'09. International Conference on*, volume 1, pages 590–593. IEEE, 2009. 16

[KBT08] Dirk Koch, Christian Beckhoff, and Jurgen Teich. Recobus-buildera novel tool and technique to build statically and dynamically reconfigurable systems for fpgas. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 119–124. IEEE, 2008. 15

[KHT08] Dirk Koch, Christian Haubelt, and Jürgen Teich. Efficient reconfigurable on-chip buses for fpgas. In *Field-Programmable Custom Computing Machines, 2008. FCCM'08. 16th International Symposium on*, pages 287–290. IEEE, 2008. 15

[KLPR04] Heiko Kalte, Gareth Lee, Mario Porrmann, and U Ruckert. Study on column wise design compaction for reconfigurable systems. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 413–416. IEEE, 2004. 21

[KLPR05] Heiko Kalte, Gareth Lee, Mario Porrmann, and U Ruckert. Replica : A bitstream manipulation filter for module relocation in partial reconfigurable systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE, 2005. 20, 24

[KSCC10] Asma Kahoul, Alastair M Smith, George A Constantinides, and Peter YK Cheung. Efficient heterogeneous architecture floorplan optimization using analytical methods. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 4(1) :3, 2010. 65

[KSGL12] Robert Kirchgessner, Greg Stitt, Alan George, and Herman Lam. Virtualrc : a virtual fpga platform for applications and tools portability. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pages 205–208. ACM, 2012. 106

[LC05] Jai-Ming Lin and Yao-Wen Chang. Tcg : A transitive closure graph-based representation for general floorplans. *IEEE transactions on very large scale integration (VLSI) systems*, 13(2) :288–292, 2005. 25

[LMM02] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems : A survey. *European journal of operational research*, 141(2) :241–252, 2002. 13

[M+08] Andrew Makhorin et al. Glpk (gnu linear programming kit), 2008. 63

[Mak00] Andrew Makhorin. Modeling language gnu mathprog. *Relatório Técnico, Moscow Aviation Institute*, 2000. 63

[Mar12] Nicolas Marques. *Modologie et architecture adaptative pour le placement e cace de tes matelles de tailles variables sur des partitions recongurables.* PhD thesis, Universite de Lorraine, 2012. 4, 17

[MBW11] David P Montminy, Rusty O Baldwin, and Paul D Williams. Relocatable field programmable gate array bitstreams for fault tolerance, March 15 2011. US Patent 7,906,984. 18

[MBWM07] D. P. Montminy, R. O. Baldwin, P. D. Williams, and B. E. Mullins. Using relocatable bitstreams for fault tolerance. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 701–708, Aug 2007. doi:10.1109/AHS.2007.108. 18

[Még12] Thomas Mégel. *Placement, ordonnancement et mécanismes de migration de tâches temps-réel pour des architectures distribuées multicoeurs*. PhD thesis, 2012. 18

[MLBG08] Thomas Marconi, Yi Lu, Koen Bertels, and Georgi Gaydadjiev. Intelligent merging online task placement algorithm for partial reconfigurable systems. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 1346–1351. IEEE, 2008. 12

[MSSM10] Alessio Montone, Marco D Santambrogio, Donatella Sciuto, and Seda Ogrenci Memik. Placement and floorplanning in dynamically reconfigurable fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4) :24, 2010. 26, 86

[PAKM08] Thilo Pionteck, Carsten Albrecht, Roman Koch, and Erik Maehle. Adaptive communication architectures for runtime reconfigurable system-on-chips. *Parallel Processing Letters*, 18(02) :275–289, 2008. ix, 16

[RDM+17] Marco Rabozzi, Gianluca Carlo Durelli, Antonio Miele, John Lillis, and Marco Domenico Santambrogio. Floorplanning automation for partial-reconfigurable fpgas via feasible placements generation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1) :151–164, 2017. ix, 26, 27

[RFG16] Jens Rettkowski, Konstantin Friesen, and Diana Göhringer. Repabit : Automated generation of relocatable partial bitstreams for xilinx zynq fpgas.

In *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*, pages 1–8. IEEE, 2016. 22, 24

[RLS14] Marco Rabozzi, John Lillis, and Marco D Santambrogio. Floorplanning for partially-reconfigurable fpga systems via mixed-integer linear programming. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pages 186–193. IEEE, 2014. 26, 28, 29, 71, 86

[RMA+09] Vincenzo Rana, Srinivasan Murali, David Atienza, Marco Domenico Santambrogio, Luca Benini, and Donatella Sciuto. Minimization of the reconfiguration latency for the mapping of applications on fpga-based systems. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 325–334. ACM, 2009. 14

[RMS15] Marco Rabozzi, Antonio Miele, and Marco D Santambrogio. Floorplanning for partially-reconfigurable fpgas via feasible placements detection. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 252–255. IEEE, 2015. 26

[SB06] Love Singhal and Elaheh Bozorgzadeh. Multi-layer floorplanning on a sequence of reconfigurable designs. In *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pages 1–8. IEEE, 2006. 86

[SBC16] L. Sterpone, L. Boragno, and D. M. Codinachs. Analysis of radiation-induced seus on dynamic reconfigurable systems. In *2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–6, June 2016. doi:10.1109/ReCoSoC.2016.7533907. 18

[SGZW16] Aaron Stoddard, Ammon Gruwell, Peter Zabriskie, and Michael Wirthlin. High-speed pcap configuration scrubbing on zynq-7000 all programmable socs. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–8. IEEE, 2016. 112

[SKD09] Arvind Sudarsanam, Ramachandra Kallam, and Aravind Dasu. Prr-prr dynamic relocation. *IEEE Computer Architecture Letters*, 8(2) :44–47, 2009. 19, 21, 24

[SM91] Khushro Shahookar and Pinaki Mazumder. Vlsi cell placement techniques. *ACM Computing Surveys (CSUR)*, 23(2) :143–220, 1991. 71

[TCM86] Palle Thoft-Christensen and Yoshisada Murotsu. The branch-and-bound method. In *Application of Structural Systems Reliability Theory*, pages 215–265. Springer, 1986. 63

[TORB+13] Maamar Touiza, Gilberto Ochoa-Ruiz, El-Bay Bourennane, Abderrezak Guessoum, and Kamel Messaoudi. A novel methodology for accelerating bitstream relocation in partially reconfigurable systems. *Microprocessors and Microsystems*, 37(3) :358–372, 2013. 22, 24

[TSMM06] Jesús Tabero, Julio Septién, Hortensia Mecha, and Daniel Mozos. Task placement heuristic based on 3d-adjacency and look-ahead in reconfigurable systems. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 396–401. IEEE Press, 2006. 13

[VF12] Kizheppatt Vipin and Suhaib A Fahmy. Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In *International Symposium on Applied Reconfigurable Computing*, pages 13–25. Springer, 2012. 4, 25, 26, 28, 29, 86

[VF13] Kizheppatt Vipin and Suhaib A Fahmy. Automated partitioning for partial reconfiguration design of adaptive systems. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 172–181. IEEE, 2013. 14

[WP04] Herbert Walder and Marco Platzner. A runtime environment for reconfigurable hardware operating systems. *Field Programmable Logic and Application*, pages 831–835, 2004. ix, 15

[Xil04] I Xilinx. Xapp290 : Two flows for partial reconfiguration : Module based or difference based, 2004. 110

[XIL12] XILINX. *Partial Reconfiguration User Guide, UG702*, April 2012. 32

[YYC07] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. Temporal floorplanning using the three-dimensional transitive closure subgraph. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(4) :37, 2007. 25, 29, 86

Résumé

Les systèmes adaptatifs basés sur les architectures FPGA (Field-Programmable Gate Arrays) peuvent bénéficier grandement de la grande flexibilité offerte par la reconfiguration partielle dynamique (DPR). Grâce au DPR, les tâches matérielles composant un système adaptatif peuvent être allouées et re-allouées à la demande ou en fonction de l'environnement dynamique. Les flots de conceptions disponibles et les outils commerciaux ont évolué pour répondre aux exigences des architectures reconfigurables qui sont toutefois limitées dans leurs fonctionnalités. Ces outils ne permettent pas un placement et une relocation efficaces de tâches matérielles de tailles variables. L'objectif principal de ces travaux de thèse consiste à proposer des nouvelles méthodologies et de nouvelles approches pour faciliter au concepteur la phase de conception d'un système adaptatif reconfigurable opérationnelle, valide, optimisé et adapté aux changements dynamiques de l'environnement. La première contribution de cette thèse porte sur la problématique de la relocation des tâches matérielles de tailles différentes. Une méthodologie de conception est proposée pour répondre à un problème majeur des mécanismes de relogement : le stockage d'une unique bitstream de configuration pour réduire les besoins de la mémoire et pour accroître la réutilisable des modules matériels générés. Une technique de partitionnement de la région reconfigurable est appliquée dans la méthodologie de relogement proposée pour augmenter l'efficacité d'utilisation des ressources matérielles dans le cas des tâches reconfigurables de tailles variables. Cette méthodologie prend en compte aussi la communication entre différentes régions reconfigurables et la région statique. Pour valider la méthode, plusieurs études de cas sont implémentées. Cette validation montre une utilisation efficace des ressources matérielles ainsi une réduction importante du temps de reconfiguration. La deuxième partie de cette thèse présente et détaille une formulation mathématique afin d'automatiser le floorplanning des zones reconfigurables dans les FPGAs. Les algorithmes de recherche présentés dans cette thèse sont basés sur la technique d'optimisation PLMNE (programmation linéaire mixte en nombres entiers). Ces algorithmes permettent de définir automatiquement l'emplacement, la taille et la forme de la zone reconfigurable dynamique. Nous nous intéressons principalement dans cette recherche à la satisfaction des contraintes de placement des zones reconfigurables et celles liées à la relocation. De plus, nous considérons l'optimisation des ressources matérielles dans le FPGA en tenant compte des tâches de tailles variables. Finalement, une évaluation de l'approche proposée est présentée.

Mots-clés: reconfiguration dynamiqu partielle, FPGA, relocation, floorplanning, tâches de tailles varaibles, partitionnement, programmation linéaire mixte en nombres entiers

Abstract

Adaptive systems based on Field-Programmable Gate Arrays (FPGA) architectures can benefit greatly from the high degree of flexibility offered by dynamic partial reconfiguration (DPR). Thanks to DPR, hardware tasks composing an adaptive system can be allocated and relocated on demand or depending on the dynamically changing environment. Existing design flows and commercial tools have evolved to meet the requirements of reconfigurables architectures, but that are limited in functionality. These tools do not allow an efficient placement and relocation of variable-sized hardware tasks. The main objective of this thesis is to propose a new methodology and a new approaches to facilitate to the designers the design phase of an adaptive and reconfigurable system and to make it operational, valid, optimized and adapted to dynamic changes in the environment.

The first contribution of this thesis deals with the issues of relocation of variable-sized hardware tasks. A design methodology is proposed to address a major problem of relocation mechanisms : storing a single configuration bitstream to reduce memory requirements and increasing the reusability of generating hardware modules. A reconfigurable region partitioning technique is applied in this proposed relocation methodology to increase the efficiency of use of hardware resources in the case of reconfigurable tasks of variable sizes. This methodology also takes into account communication between different reconfigurable regions and the static region. To validate the design method, several cases studies are implemented. This validation shows an efficient use of hardware resources and a significant reduction in reconfiguration time. The second part of this thesis presents and details a mathematical formulations in order to automate the floorplanning of the reconfigurable regions in the FPGAs. The algorithms presented in this thesis are based on the optimization technique MILP (mixed integer linear programming). These algorithms allow to define automatically the location, the size and the shape of the dynamic reconfigurable region. We are mainly interested in this research to satisfy the constraints of placement of the reconfigurable zones and those related to the relocation. In addition, we consider the optimization of the hardware resources in the FPGA taking into account the tasks of variable sizes. Finally, an evaluation of the proposed approach is presented.

Keywords: dynamic partial reconfiguration, FPGA, relocation, floorplanning, variable sized tasks, partitioning, mixed integer linear programming