N° d'ordre : **42529**

UNIVERSITÉ DE LILLE 1 - FRANCE

THÈSE

présentée pour obtenir le titre de

DOCTEUR SPÉCIALITÉ INFORMATIQUE

par

Wissem Chouchene

Vers une reconfiguration dynamique partielle parallèle par prise en compte de la régularité des architectures FPGA-Xilinx

Thèse soutenue le 07 Décembre 2017, devant la commission d'examen formée de :

Carlos VALDERRAMA	Professeur – Université de Mons – Belgique	**Rapporteur**
Fabrice MULLER	MdC HDR – Polytech Nice Sophia – France	**Rapporteur**
Bertrand Geranado	Professeur – Université Pierre et Marie Curie – France	**Examinateur**
Laetitia Jourdan	Professeur - Université Lille 1 Sciences et Technologies – France	**Examinateur**
Jean-Luc DEKEYSER	Professeur - Université Lille 1 Sciences et Technologies – France	**Directeur**
Rabie BEN ATITALLAH	MdC HDR – Université de Valenciennes – France	**Co-Directeur**

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
CRIStAL - UMR 9189- Cité Scientifique, Bât. M3 - 59655 Villeneuve d'Ascq Cedex

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES LISTAGES

CHAPITRE 1 : Introduction Générale

1.1 Contexte de la thèse

La micro-électronique est un domaine vaste et révolutionnaire. Les progrès technologiques visant la réduction de la largeur de canal des transistors jusqu'à 10 nm [63] [90] et la capacité de leur intégration de plus en plus dense a engendré des inventions qui se multiplient jusqu'à nos jours. Une particularité de ce domaine se manifeste dans les systèmes sur puces ou SoC (System on Chip) qui ne cessent d'évoluer depuis l'année 1980 surtout avec la loi de Moore a assisté la résolution des problèmes de conception des circuits intégrés 2D au point de vue surface et intégration des composants jusqu'à la première décennie du XXI siècle. De nos jours, l'évolution dimensionnelle des circuits intégrés jusqu'à la technologie 3D a dépassé cette loi permettant une intégration de composants plus dense avec plus de performance.

L'une des plus importantes évolutions sont les FPGAs (Field Programmable Gate Arrays). Les FPGA sont des circuits reconfigurables émergeants comme cibles privilégiées pour l'implémentation des applications divers tels que le traitement de signal intensif et parallèle. Ils offrent une plateforme matérielle programmable, performante et à coût réduit. En plus ils permettent d'implanter des architectures modulaires à haute performance dû au grand nombre de composants logiques programmables qu'ils intègrent. Ces circuits ont pris le même rang que les ASIC (Application Specific Integrated Circuit) caractérisés par leur haute performance et leur faible consommation d'énergie.

1.1.1 Les FPGAs et la reconfiguration dynamique partielle (RDP)

Une caractéristique essentielle des circuits FPGA est la reconfigurabilité. Cette caractéristique est classée selon plusieurs aspects. Elle peut être interne ou externe, statique ou dynamique, complète ou partielle.

La reconfiguration est externe quand elle s'effectue par un module en dehors du FPGA tandis qu'une reconfiguration interne est effectuée dans le composant FPGA et gérée par un contrôleur bien spécifique qui peut être un processeur hard/soft ou un IP (Intellectual Property). En effet, la reconfiguration externe permet d'éviter l'utilisation des ressources FPGA mais nécessite un temps de reconfiguration qui peut entraîner une latence considérable. D'autre part, la reconfiguration interne nécessite l'instanciation d'un contrôleur bien spécifique qui prend part des ressources FPGA mais assure un bon débit de transfert des données de reconfiguration.

Un autre aspect de la reconfiguration est qu'elle peut être statique ou dynamique. La reconfiguration statique est effectuée lorsque le composant est inactif. D'autre part, la

reconfiguration est dite dynamique lorsque le FPGA est reconfiguré en cours d'exécution. La reconfiguration dynamique permet d'éviter les délais engendrés par la désactivation et la réinitialisation du circuit.

Enfin, la reconfiguration peut être complète ou partielle. La reconfiguration complète se manifeste par la configuration de l'intégralité du composant permettant de mettre en œuvre un système déjà implémenté. En effet, elle s'effectue grâce au chargement d'une description spécifique (bitstream) via une interface particulière qui permet d'allouer toutes les ressources du FPGA à ce système. D'autre part, la reconfiguration partielle se manifeste par la modification d'une partie du FPGA tout en laissant fonctionnelle une autre partie. Elle est réalisée pour permettre au FPGA de s'adapter aux algorithmes de changement de matériel, améliorer la tolérance aux pannes, améliorer les performances ou de réduire la consommation d'énergie [6] [7]. En effet, la reconfiguration partielle a un impact positif sur la performance puisqu'elle modifie une partie du FPGA pour une durée largement inférieure à celle de la reconfiguration complète.

En s'appuyant sur cette classification, la Reconfiguration interne Dynamique et Partielle (RDP) est clairement avantageuse par rapport aux autres types de reconfiguration. Pour cela, nous nous intéressons à ce type de configuration dans notre travail.

1.1.2 Les FPGA Xilinx réguliers

En se référant à la signification de son appellation et à la Figure 1.1, le terme 'Field' indique la capacité de programmer les matrices de portes logiques par un utilisateur plutôt que par le fabricant. Le mot 'Array' est utilisé pour indiquer la présence d'une série de colonnes et de rangées de portes qui peuvent être programmées aussi par l'utilisateur.

Nous nous intéressons dans cette thèse aux structures des plateformes FPGA-Xilinx présentes sur le marché qui intègrent la fonctionnalité de la RDP, notamment les familles FPGAs récentes. Ces FPGAs sont construits à base de plusieurs ressources hétérogènes qui sont les CLBs (Blocs Logiques Configurables), les DSPs (Digital Signal Processing) et les BRAMs (Blocs RAMs appelées parfois RAMB). Ces ressources sont appelées « Blocs Fonctionnels » qui sont disposées sous forme de lignes verticales les unes à côté des autres tout le long de la largeur de la plateforme. L'ensemble de ces lignes verticales est sectionnée en deux colonnes qui ne sont pas forcement symétriques horizontalement. Chacune de ces colonnes est subdivisée en plusieurs régions appelées Régions d'Horloges qui sont réparties de manière uniforme, équitable et symétrique verticalement. Ces régions d'horloges sont

référenciées par leurs coordonnées (XY) et comportent des ressources sous forme de Frames. Ces Frames se diversifient par trois types essentiels qui sont Frame_CLB, Frame_DSP et Frame_RAMB comme illustré dans la Figure 1.1.

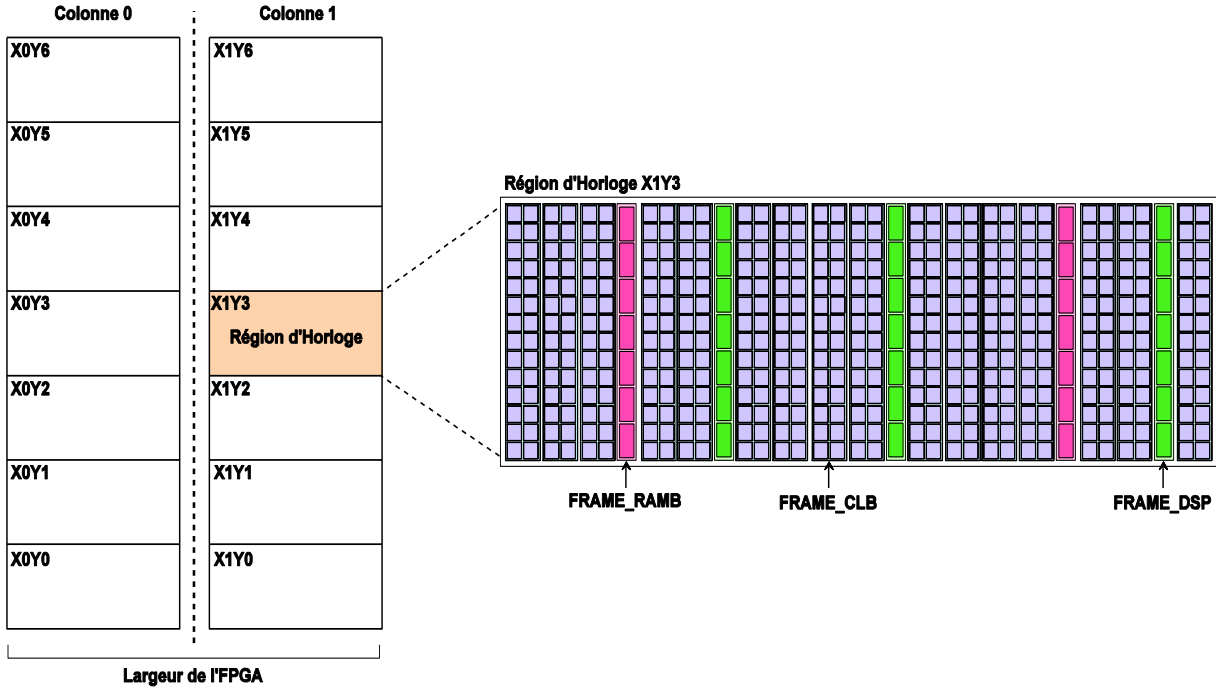


Figure 1.1 : Constitution de la structure FPGA de Xilinx

La Figure 1.1 montre bien la régularité structurelle des circuits FPGAs au niveau des régions d'horloges et plus précisément au niveau des différents types de Frames. En effet, les trois types de Frames gardent toujours la même taille ainsi que leur position au niveau de chaque région d'horloge. Cependant, la taille de Frame est une constante qui représente une caractéristique fondamentale pour la construction d'un FPGA donné. De plus, chaque Frame est constitué par un ensemble de blocs fonctionnels dont chacun est référencié par ces coordonnées XY, ce qui donne une structure symétrique facilement exploitable.

1.1.3 Environnement de travail

Nous nous intéressons aux architectures parallèles qui disposent d'un ensemble d'unités de calcul permettant la réduction du temps de l'exécution d'une application. Ces architectures telles que les architectures SPMD (Single Program Multiple Data) sont utilisées dans différents domaines d'applications qui requièrent une grande puissance de calcul distribué pour satisfaire les contraintes temps-réel tels que le traitement de signal, le traitement d'image/vidéo, etc.

En exploitant les avantages de la reconfiguration dynamique partielle en vue de l'optimisation de la surface et de la consommation d'énergie, ces systèmes parallèles peuvent évoluer donnant naissance à des systèmes plus flexibles supportant plusieurs applications sur la même architecture. Ceci se manifeste par la spécification d'une Région Partiellement Reconfigurable (RPR) au sein de chaque unité de calcul et qui est partagée par diverses tâches appelées Modules Partiellement Reconfigurables (MPR) portant chacun une fonctionnalité bien particulière. En effet, chaque RPR doit satisfaire les contraintes de ressources requises par cet ensemble de MPRs ainsi que leurs interfaces qui doivent être communes avec la partie statique du système. En s'appuyant sur le flot de la reconfiguration dynamique partielle de Xilinx qui exige que chaque MPR (bitstream) soit relatif à une seule RPR, une seule configuration du système parallèle nécessite que chaque RPR soit reconfigurée dynamiquement par un MPR relatif réalisant chacun la même fonction. Dans ce cadre nous envisageons exploiter la régularité et la symétrie des composants FPGAs pour proposer des mécanismes de reconfiguration qui sont adaptés à des architectures parallèles dynamiquement reconfigurables.

1.1.4 Vers une maîtrise de la régularité des FPGAs et de la RDP dans les architectures à multi-régions dynamiquement reconfigurables

Les attraits de la RDP et de son déploiement pour les architectures parallèles dynamiquement reconfigurables, nous a mené à proposer la factorisation de l'ensemble des reconfigurations d'un système multi-RPRs en une seule configuration compatible. En effet, la factorisation des MPRs sert à optimiser la taille et la gestion de la mémoire des bitstreams partiels. Cet aspect favorise l'implantation d'un mécanisme de diffusion d'un MPR vers chacune des RPRs d'une manière parallèle, nous introduisons alors la notion de la Reconfiguration Dynamique Partielle Parallèle (RDPP).

Agissant sur la structure régulière des FPGAs Xilinx, le placement régulier des RPRs et leurs routages uniformes au niveau de leurs interfaces est une étape essentielle qui vise à assurer la compatibilité des MPRs avec l'ensemble des RPRs. Il s'agit bien de la réutilisation de la technique de relocalisation de bitstream partiel. Cette technique nécessite en effet deux étapes essentielles qui sont :

- Le placement uniforme et symétrique des RPRs qui requière un Floorplanning [77] (Planification) identique en termes de formes des RPRs en fonction de leurs ressources requises.
- Le routage uniforme des interfaces qui séparent la partie statique et chacune des RPRs. Ces interfaces doivent être ainsi pour assurer que la reconfiguration d'un MPR par un autre au sein d'une même RPR ne conduit pas à la modification de ce routage qui engendre la rupture de connexion entre la RPR et la partie statique.

La mise en œuvre du mécanisme de factorisation permettant la RDPP d'un MPR vers l'ensemble des RPRs nécessite par conséquent un effort supplémentaire de conception pour unifier les formes de chacune des PRPs au niveau du placement ainsi que l'uniformisation des signaux d'interfaçage de chaque RPR au niveau du routage. Cet effort est dû aux problématiques liées d'un côté au flot traditionnel de la RDP de Xilinx et ces limites et de l'autre côté à la nécessité de maitriser la complexité matérielle par une vision structurelle et régulière des FPGAs Xilinx.

1.2 Problématiques

1.2.1 Problématique 1 : Le flot traditionnel de la RDP-Xilinx et ces limites

Les FPGAs-Xilinx permettent l'implantation des systèmes dynamiquement reconfigurables grâce au flot de la RDP. Ce flot constitue plusieurs étapes de conception en partant de la spécification jusqu'à la génération des descriptions physiques (bitstreams) relatifs à la partie statique et la partie dynamique du système. Il est donc évident que la mise en œuvre de la RDP dans les circuits FPGA-Xilinx introduit plus de complexité dans la conception des systèmes dynamiquement reconfigurables. Ceci implique l'instanciation de divers composants préconçus comprenant au moins un Softcore (MicroBlaze) [1], un bus (AXI ou PLB) [2] [3], un contrôleur de port ICAP [4] avec des interfaces encombrantes et des FIFOs. Afin d'optimiser ces systèmes, certains travaux ont été basés la mise en œuvre matérielle de la RDP dans le but de réduire la complexité de ces l'architectures et de diminuer le temps de reconfiguration correspondant au transfert de bitstreams partiels à travers l'interface de l'ICAP. Ces travaux se sont focalisés dans la conception des contrôleurs matériels dédiés pour la RDP.

L'utilisation de cette technique reste encore limitée dans les milieux industriels car elle requière l'intervention manuelle du concepteur, notamment pendant la phase de

floorplanning, et d'expertise de l'architecture interne des circuits FPGAs. Ceci se traduit par la détermination optimisée des ressources requises pour chaque région reconfigurable et son emplacement sur la structure du FPGA. Par conséquent, un temps considérable de conception est nécessaire quand il s'agit de spécifier un grand nombre de RPRs.

La spécification d'un grand nombre de RPRs dont chacune offre une multitude de fonctionnalités (multitude de MPRs) est soumise aux clauses de ce flot de la RDP. En effet, puisque chaque MPR (bitstream) ne doit être placé que dans sa RPR relative, un grand nombre de bitstreams partiels sont générés nécessitant une gestion mémoire relativement complexe. Donc, suivant notre approche, pendant une configuration du système pour une fonctionnalité précise, un ensemble de MPRs doit être transféré vers l'ensemble des RPRs. Le nombre des MPRs est égale au nombre des RPRs. Ceci cause d'une part, un temps considérable pour effectuer la reconfiguration, et d'une autre une gestion de mémoire complexe de l'ensemble des MRPs avec leurs fonctionnalités diverses.

Une autre limitation de ce flot de la RDP de Xilinx, c'est qu'il ne permet pas de garantir la reconfigurabilité quand nous spécifions un nombre précis de RPRs de même caractéristiques (fonction, taille, forme, …). En Conséquent, les bitstreams générés à partir de ces RPRs ne sont pas identiques en termes de placement et routage des composants au niveau de leurs contenus et leurs interfaces. Ceci se traduit par le fait que les processus de MAP (Mapper la conception logique sur le composant FPGA Xilinx suivant les règles de la description NGD[1]) et P&R (Placement et Routage) s'effectuent séparément au niveau de chaque RPR. En effet, les outils Xilinx conçus à cet effet tel que Vivado (qui intègre l'outil PlanAhead) [78] favorisent la performance au cours de l'implémentation plutôt que la reconfigurabilité. En résultat, ceci implique une redondance de fonctionnalité se traduisant par la génération d'un ensemble de bitstreams partiels ayant la même fonctionnalité.

1.2.2 Problématique 2 : La maîtrise de la complexité matérielle passant par une vision structurelle et régulière du FPGA

D'après notre description de la régularité des circuits FPGA-Xilinx dans la Section 1.1.2, les blocs fonctionnels constituent chacun une architecture assez complexe. Prenons par exemple un bloc CLB, il est composé de deux éléments SLICEs. Chaque SLICE peut être Logique ou Mémoire et soit côté droite soit côté gauche. Chaque SLICE contient quatre LUTs

[1] Native Generic Database

(Look Up Table) dont chacun présente une architecture interne plus complexe. Grâce à la symétrie des différents composants au niveau de chaque région d'horloge, il est possible de déterminer la composition de chaque région d'horloge voisine ou située sur le même axe vertical à partir d'une région d'horloge de référence. Xilinx a mis en œuvre pour ces FPGAs, un mécanisme de récupération des informations de chaque bloc fonctionnel en termes de nom, type, emplacement, orientation, etc. Ceci pour permettre l'exploration de ces circuits FPGAs. Le langage TCL (Tool Command Langage) [64] [65] est le langage utilisé pour exploiter ce mécanisme.

En effet, nous utilisons ce langage qui nous permet d'extraire la composition de chaque circuit FPGA dans le but de rechercher le bon emplacement de chaque RPR en fonction de ces ressources requises. Grâce à la régularité de ces circuits, nous pourrions placer l'ensemble des RPRs de manière uniforme et régulière dans le but de préparer l'étape de la réutilisation de bitstream partiel nécessaire pour la factorisation des reconfigurations. De plus, cette tâche de placement (Floorplanning) est automatisable si nous effectuons une analyse structurelle des FPGAs-Xilinx en se référant à leurs régularités. Cette analyse sert à la mise en place d'un algorithme de Floorplanning automatique qui permet de réduire considérablement le temps de conceptions.

1.2.3 Problématique 3 : La relocalisation de bitstream partiel sur floorplanning régulier en vue d'obtenir une diffusion paramétrée

La maîtrise de la structure régulière de l'FPGA est devenue possible grâce au mécanisme d'exploration structurelle des circuits FPGAs-Xilinx à travers le langage TCL. En effet, ce langage est utilisé pour permettre la gestion automatique des projets Xilinx, l'extraction des paramètres technologiques du composant FPGA utilisé et des paramètres architecturaux de la conception à implanter et l'exploration de différentes caractéristiques des blocs fonctionnels du FPGA. Ceci relève plusieurs défis non seulement pour automatiser le flot de la RDP basée sur l'automatisation de l'étape de floorplanning régulier, mais aussi pour factoriser/broadcast la fonctionnalité d'un bitstream partiel pour l'ensemble de régions partiellement reconfigurables. La factorisation d'une fonctionnalité repose sur la réutilisation de la relocalisation de bitstream. Deux obstacles essentiels se présentent : le premier se manifeste par l'absence de mécanisme d'automatisation de la technique de relocalisation qui consomme énormément de temps de conception. Le deuxième se manifeste par la nécessité d'une architecture matérielle spécifique à la RDPP pour effectuer le mécanisme de diffusion de

bitstream partiel (MPR) vers l'ensemble des RPRs. Cependant, il est essentiel de connaître l'emplacement physique de chaque RPR. Nous parlons alors d'une diffusion paramétrée au point de vue localisation des RPRs.

1.3 Contributions

La contribution majeure de cette thèse concerne la proposition d'un flot de conception qui vise à établir dans un système multi-RPRs, un broadcast paramétré d'un bitstream partiel vers un ensemble de RPRs sur FPGA. Ce flot de conception est assuré par un environnement qui automatise deux sous flots complémentaires : le flot de la reconfiguration dynamique partielle visant à effectuer un floorplanning régulier favorisant l'exécution d'une même fonction (bitstream partiel) dans chaque RPR. Le deuxième visant à son tour l'uniformisation des interfaces de toute les RPRs à partir d'une RPR de référence favorisant la relocalisation de bitstream. Cet environnement permet d'assurer la flexibilité, la réutilisabilité et l'automatisation afin de faciliter la tâche des concepteurs et d'améliorer leur productivité. La Figure 1.2 montre le déroulement des deux mécanismes visant à automatiser ces deux flots de conception.

Le premier mécanisme vise à automatiser le flot de la RDP en se basant sur le mécanisme d'exploration des plateformes FPGA-Xilinx en effectuant un placement (Floorplanning) régulier des RPRs sur leur structure physique. Ce mécanisme doit en même temps permettre d'obtenir des implémentations efficaces des différentes configurations du système multi-RPRs grâce à :

- L'automatisation de l'étape de placement des RPRs en tenant compte des réquisitions en termes de ressources (CLB, BRAM et DSP) des différentes configurations du système.
- La mise en œuvre d'un algorithme de floorplanning (placement) régulier, paramétrable et applicable pour les plateformes FPGAs pour garantir la généricité et la flexibilité.
- L'allocation régulière et identique des RPRs au point de vue forme pour permettre la portabilité d'un bitstream partiel (fonctionnalité) d'une RPR à une autre.

Ce mécanisme permet de générer une description UCF (User Constraint File) à introduire dans le flot de la RDP Xilinx.

Le deuxième mécanisme vise à assurer la factorisation de plusieurs fonctions à exécuter dans l'ensemble des RPRs. Etant donné qu'un MPR est propre à sa RPR, l'implémentation

d'une seule fonction dans un système multi-RPRs comportant N*RPRs nécessite un nombre égal à N de MPRs de même fonction mais comportant des informations de P&R différentes au niveau de leurs interfaces avec la partie statique. Ceci pose une problématique d'incompatibilité si nous voudrions placer un MPR (bitstream partiel) dans une RPR qui n'est pas propre à lui.

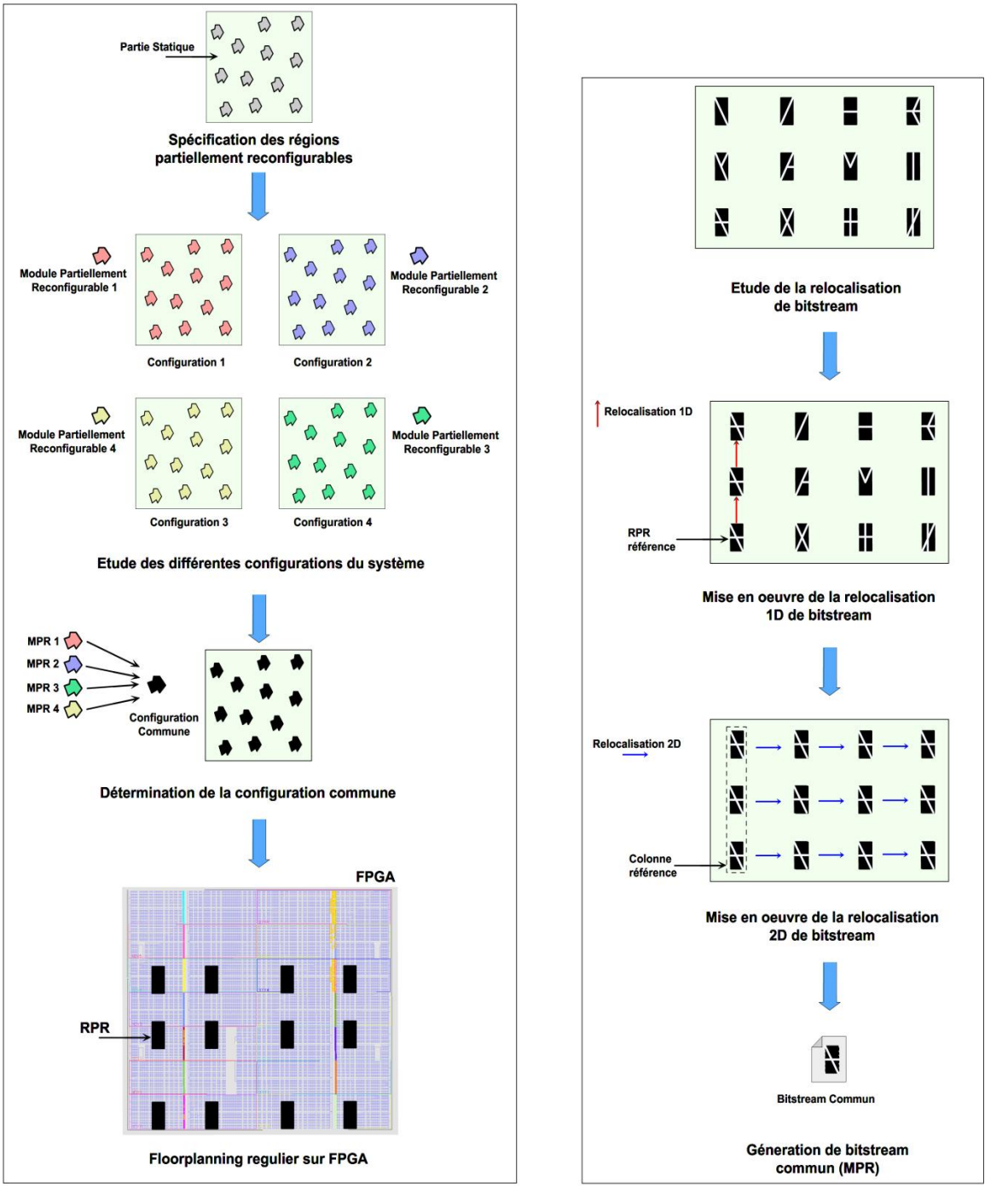


Figure 1.2 : Les mécanismes d'automatisation des deux flots de conception proposés

En effet, la factorisation d'une fonction vise à générer pour une seule fonction un seul MPR compatible à tout l'ensemble des RPRs. La factorisation se manifeste par l'uniformisation des données de P&R de toutes les RPRs pour permettre la diffusion/broadcast d'un seul MPR vers l'ensemble des RPRs. Nous proposons en effet un mécanisme permettant d'automatiser le flot de conception d'uniformisation qui prend en entrée le floorplanning régulier obtenu à partir du premier flot de conception dans le but d'optimiser la taille de la mémoire de bitstream partiels et augmenter la productivité des concepteurs. La mise en œuvre de la diffusion/broadcast des MPRs (fonctions) au sein du système multi-RPRs nécessite à son tour la conception d'une architecture matérielle de la RDPP qui tient compte des emplacements des RPRs suite à la génération des différents bitstreams partiels des MPRs.

1.3.1 Méthode de partitionnement automatique de la structure du composant FPGA pour la diffusion de bitstream

Dans le but d'automatiser l'étape de Floorplanning manuel des RPRs, nous présentons dans ce travail, un algorithme heuristique (AFLORA : Automatic Floorplanning For Multi-RPRs Architectures) qui exploite la régularité de la structure de FPGA au niveau technologique pour chercher et allouer un ensemble de RPRs identiques dans les architectures parallèle. Cette heuristique se base sur l'exploitation des paramètres technologiques de la structure du composant FPGA-Xilinx et des paramètres architecturaux. Les paramètres architecturaux se manifestent par la détermination des réquisitions en termes de ressources relatives à la partie statique et dynamique du système pour étudier la faisabilité de l'implémentation. La complexité réduite *O(n)* et la stabilité de notre algorithme montrent son efficacité pour automatiser la phase floorplanning avec le moindre coût en termes de temps d'exécution. Le flot d'automatisation permet de générer une description sous forme des contraintes utilisateur à introduire dans le flot de la RDP-Xilinx.

1.3.2 Construction d'un bitstream unique par réutilisation de la technique de relocalisation:

Le déploiement de notre algorithme vise à localiser d'une manière régulière sur la structure FPGA un ensemble de RPRs de même forme et de même taille qui facilite la mise en œuvre de la technique de relocalisation de bitstream. Ceci nécessite des directives supplémentaires sous forme de contraintes physiques qui ont pour but d'uniformiser le placement des

composants d'interfaçage entre la partie statique et les parties dynamiques. Ce résultat permet de factoriser un bitstream donné sur l'ensemble des RPRs. Donc un mécanisme de broadcast pourrait être mis en œuvre en tenant compte de la localité de chaque région reconfigurable pour l'optimisation de la mémoire des bitstreams partiels en garantissant la flexibilité.

1.3.3 Conception d'un environnement RDPP compatible avec les outils Xilinx

Etant donné que l'obstacle majeur du flot de la reconfiguration dynamique partielle se manifeste par la mise en œuvre de l'étape de floorplanning manuelle, l'intégration de notre algorithme d'automatisation de cette étape AFLORA a garanti un gain au point de vue temps de conception. De même l'uniformisation des données physiques de P&R des RPRs aboutissant à la factorisation de divers fonctionnalités a permit d'effectuer leurs diffusion dans un système multi-RPRs suite à l'automatisation de la technique de la relocalisation de bitstream partiel. En exploitant le langage TCL, spécifique aux outils Xilinx, nous avons conçu un environnement automatisé appelé ADForMe (Automatic DPPR Flow For Multi-RPRs Architecture). ARForMe permet de :

1) Etudier la faisabilité de l'implémentation en fonction des ressources requises par les parties statique et dynamique en créant le projet correspondant à l'architecture avec les outils PlanAhead/Vivado.
2) Effectuer le floorplanning automatique à l'aide d'AFLORA.
3) Etablir la première implémentation du système.
4) Récupérer les informations de placement et routage des RPRs pour chaque configuration à l'aide de l'outil FPGA_Editor [52].
5) Utiliser la technique de la relocalisation 1D et 2D de bitstream pour la factorisation de fonctionnalités pour chaque configuration.
6) Générer des bitstreams correspondants à chaque fonctionnalité et gestion des emplacements des bitstreams partiels.

Les résultats expérimentaux ont montré l'efficacité de notre outil pour augmenter la productivité de conception des architectures parallèles sur les plateformes FPGA-Xilinx.

1.4 Plan de la thèse

Notre manuscrit est organisé comme suit :

Le chapitre 2 : Etat de l'art, ce chapitre présente le contexte de nos travaux qui abordent les différents travaux portant sur les architectures parallèles reconfigurables sur FPGA, les travaux de recherche sur la RDP séquentielle et parallèle et enfin les travaux qui visent la conception des algorithmes de floorplanning automatiques des circuits FPGAs favorisant la technique de relocalisation de bitstream partiel. Cette étude permet de nous positionner nos travaux et identifier les verrous scientifiques pour nos contributions.

Le chapitre 3 : Méthode de partitionnement automatique de la structure du composant FPGA pour la diffusion de bitstream partiel, ce chapitre porte sur notre flot de conception de floorplanning automatique régulier sur la structure de l'FPGA qui favorise la relocalisation de bitstream suite à une première implémentation du système. Nous commençons par souligner les exigences et les spécifications de notre flot de conception ADForMe puis nous décrivons le déroulement des étapes qu'ils le constituent. L'automatisation est assurée par le langage TCL. L'exécutable ADForMe permet de générer la description UCF de floorplanning régulier. Enfin, nous analysons les résultats en termes de complexité et de qualité des solutions.

Chapitre 4 : La réutilisation de la technique de relocalisation et le broadcast de bitstream partiel, nous présentons dans ce chapitre la technique de la relocalisation de bitstream 1D et 2D utilisée et nous décrivons les différentes étapes du mécanisme d'uniformisation des données physiques de placement et de routage des interfaces situées entre les régions partiellement reconfigurables et la partie statique. Les étapes de ce mécanisme ont été automatisées en utilisant le langage JAVA et le langage script de l'outil FPGA_Editor. Le mécanisme d'automatisation permet de générer une description UCF comportant les contraintes physiques de routage et placement des RPRs suite à une deuxième implémentation du système. Enfin, le programme BitGen (l'outil Xilinx utilisé pour la génération des bitstreams) permet de générer les différents bitstreams partiels communs avec leurs différentes fonctionnalités.

Chapitre 5 Contrôleur de la reconfiguration dynamique dédié pour la diffusion de bitstream partiel, ce chapitre présente notre contrôleur de la RDPP proposé qui gère le broadcast de bitstream partiel en fonction des emplacements des régions partiellement reconfigurables. Ce contrôleur est un accélérateur matériel développé en VHDL et procédant trois interfaces principales qui sont l'interface avec une mémoire de bitstreams partiels, l'interface avec la mémoire de configuration du FPGA et l'interface de pilotage IP/Processeur. Notre contrôleur permet de garantir un débit de transfert élevé de 1.5 Go/s.

Chapitre 6 : Validation expérimentale sur différentes plateformes FPGA Xilinx, ce chapitre présente la validation expérimentale sur différentes plateformes FPGAs Xilinx en fonction de différentes granulites de parallélisme. Ceci, suivant plusieurs implémentations d'architectures parallèles intégrant la fonction de la DPR sur FPGA. Nous analysons ces résultats expérimentaux à la fin de ce chapitre.

Chapitre 7 conclusion et perspectives, nous concluons cette thèse avec par un récapitulatif portant sur les travaux effectués avant de donner quelques perspectives à nos contributions.

CHAPITRE 2 : Etat de L'art

2.1 Introduction

La motivation principale de notre travail est d'augmenter la flexibilité dans les systèmes parallèles dynamiquement reconfigurables ainsi que l'amélioration de la productivité des concepteurs en mettant en œuvre des mécanismes d'automatisation. En effet, les difficultés pour développer ces systèmes multiprocesseurs intégrant des régions dynamiquement reconfigurables au sein de chaque processeur se manifestent par le coût en termes de temps de développement, la maitrise des différents outils utilisés pour constituer le flot de la RDPP à partir de la RDP et enfin la nécessité d'une expertise sur la structure des circuits FPGAs. En soulignant ces difficultés, cette thèse vise à :

- Réduire le temps de développement des systèmes parallèles reconfigurables dynamiquement tout en unifiant l'utilisation des outils pour la mise en œuvre de la RDPP. Ceci s'effectue par l'utilisation du langage TCL qui permet de gérer les différents projets de manière automatique et intuitive.
- La mise en œuvre d'un algorithme rapide de floorplanning régulier adapté pour la structure régulière du FPGA permettant de réaliser des implémentations efficaces.
- Réduire le temps d'intervention manuelle des concepteurs par la mise en œuvre des mécanismes d'automatisation des étapes de floorplanning et de la relocalisation de bitstream.

Ces objectifs nous ont amené à structurer ce chapitre en 7 sections principales. La deuxième section introduit les différents travaux de recherches portant sur les architectures parallèles reconfigurables sur FPGA. La section 3 détaille la structure régulière des FPGAs. La section 4 présente les travaux de recherche sur la RDP séquentielle et la RDP parallèle (RDPP). La section 5 introduit les travaux de recherche portant sur les algorithmes de floorplanning. La section 6 présente les travaux effectués sur la relocalisation de bitstream. Enfin, la section 7 présente une analyse des points décrits ci-dessus avant de conclure ce chapitre.

2.2 Les architectures parallèles reconfigurables sur FPGA

Parmi les motivations de l'exécution parallèle est le gain en performance. Plusieurs travaux concurrents se sont spécialisés dans la conception des architectures parallèles pour disposer d'une puissance de calcul élevée applicable dans différents domaines. Dans cette partie, nous allons présenter quelques architectures parallèles HPC (High Performance Computing) implantées sur FPGA. En ce qui concerne le HPC, le marché d'accélérateurs matériels est

largement dominé par les GPUs (Graphical Processing Unit), qui sont connus par leur performance en calcul flottant. D'autre part les FPGAs demeurent un concurrent redoutable face aux GPUs puisqu'ils ont envahi le marché des systèmes embarqués en touchant divers domaines d'applications grâce à leur performance et leur programmabilité. L'une des inventions les plus intéressantes en 2015 est le SRC Saturne 1 [66]. Le serveur SRC présente une solution alternative à l'utilisation des cœurs X86 [82] qui a dominé les systèmes HPCs. La machine SRC permet l'exécution de toutes les instructions en un seul cycle d'horloge qui représente une accélération de 500 fois par rapport à la machine X86 traditionnelle.

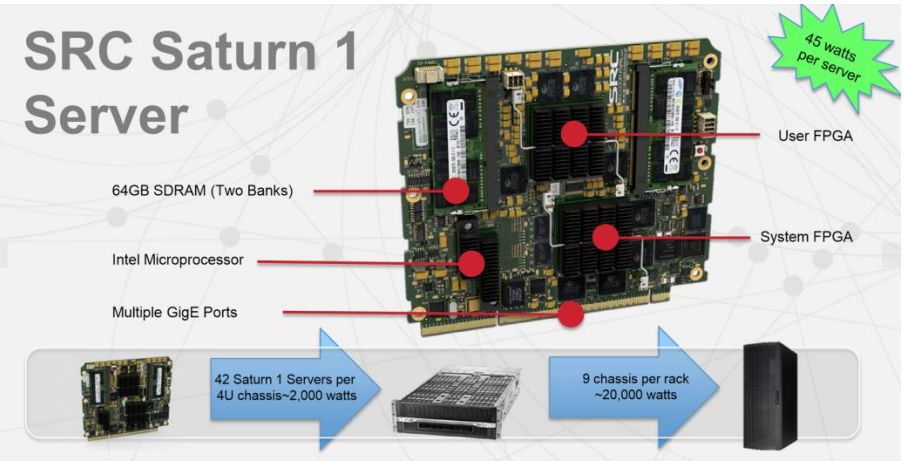


Figure 2.1 : Le Serveur SRC Saturn 1

Le serveur Saturn 1 est disponible à partir de SRC Computers par HP. Comme le montre la Figure 2.1, il y a deux FPGA Stratix-IV [67] sur une carte. Le premier FPGA est disponible pour les concepteurs. Le second FPGA est un FPGA système qui exécute un logiciel conçu par SRC qui relie tout l'ensemble des composants dans un système unifié. Il y a aussi deux bancs de mémoire, un processeur Intel à quatre cœurs (Xeon W3565) [84], et un connecteur qui regroupe plusieurs ports Ethernet. Une Pile de serveur Saturn 1 est déposée dans un châssis de Moonshot HP. Sachant que chaque serveur fonctionne à 45 watts, la pile contient 42 serveurs Saturne 1 avec une charge d'alimentation est de 2000 Watts. Un rack équipé de 9 châssis consomme environ 20.000 Watts. La nature déterministe de la plateforme signifie que les programmes exécutés sur Saturn 1 se déroulent exactement avec la même performance à chaque instant. D'autres avantages par rapport aux processeurs x86 traditionnels se manifestent par la réduction de la consommation de puissance et la surface réduite. Un autre avantage que le SRC souligne est la facilité d'utilisation. En effet, un gain en temps de conception pour programmer les FPGA avec les langages de haut niveau. Donc, au lieu d'être relégué à un langage de bas niveau comme Verilog ou VHDL, les clients SRC peuvent utiliser

l'environnement de développement personnalisé intitulé CARTE, pour écrire des programmes avec un langage familier, tel que C++, Python ou Ruby, de sorte que le serveur peut être déployé juste comme un serveur régulier x86. Quant aux cas d'utilisation spécifiques HPC, Saturn 1 est idéal pour traiter un programme ou un algorithme qui consomme jusqu'à 90-95% de cycles CPU. En utilisant les circuits FPGAs, les processeurs peuvent être complètement reprogrammés en un peu moins d'une seconde. Cela signifie que les serveurs peuvent être optimisés à la volée pour répondre à la charge du processus. Par exemple, en utilisant un Saturn 1, une entreprise comme Google serait en mesure de conFigurer les processeurs pour traiter les requêtes de recherche pendant des heures pendant le jour. Par contre, pendant la nuit, lorsque les recherches sont moins fréquentes, l'environnement du serveur pourrait être optimisé pour l'analyse Web.

Etant donné que le HPC touche plusieurs domaines tel que le BIG Data, l'apprentissage machine et l'apprentissage profond (Deep Learning), plusieurs centre de données avec leurs architectures classiques n'ont pas suivi cette évolution. Pour répondre à ces nouveaux besoins, IBM a adopté un modèle d'entreprise innovant grâce à OpenPower [5]. Cette initiative comporte essentiellement l'utilisation des dispositifs de NVIDIA [68] avec sa technologie NVLink [72], Mellanox [69], les FPGAs Xilinx et Altera, le processeur POWER [70], les mémoires systèmes à accès direct (DMA) et l'interface Coherent Accelerator Processor (CAPI) [71] pour constituer un système complet. Cet appairage est une avancée puissante du calcul numérique intensif, elle offre les performances, la programmation et l'accessibilité avancées requises pour traiter de gros volumes de données rapidement. L'initiative OpenPower d'IBM a aiguisé l'accent sur le HPC en particulier. Elle a conçu le processeur POWER qui est un CPU rapide en technologie 7nm. Ce processeur établit une connexion serrée avec les accélérateurs comme NVIDIA et Xilinx, les dispositifs tels que Mellanox et les mémoires de stockage Flash, en utilisant CAPI et les interfaces NVLink. Plusieurs architectures HPC ont été développées à base de ce processeur garantissant une puissance de calcul importante comparée à l'utilisation du X86. Les architectures les plus connues sont IBM Power System S822LC, S812L, S822L, S824L (avec la technologie NVIDIA) [73].

Intel considère que les FPGAs sont un moyen efficace d'accélération des applications et a conçu une puce hybride qui associe un FPGA (Altera) à un processeur Xeon E5 sur le même socket. Pour confronter le nombre croissant de fournisseurs des puces ARM qui sont rapides et paramétrables, Intel a fourni des versions personnalisées de ses processeurs Xeon. La puce

hybride Xeon-FPGA [74] utilisera un réseau cohérent fourni par QuickPath Interconnect sur la puce qui interconnecte plusieurs processeurs comme indiqué dans la Figure 2.2.

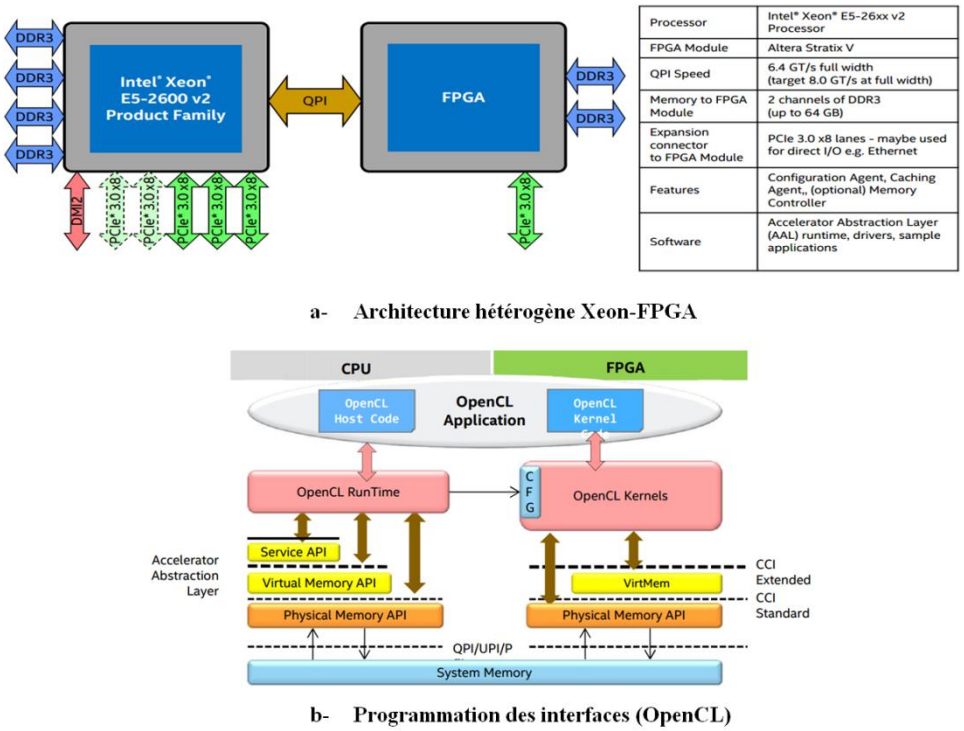


Figure 2.2 : Architecture Xeon-FPGA proposée par Intel

Ceci permet au composant de lire et d'écrire dans la hiérarchie du cache et la mémoire principale sur le processeur Xeon. Cette proximité est importante, car elle réduit la latence et augmente les performances. L'utilisation de la puce hybride Xeon-FPGA présente un avantage essentiel. Elle sert comme un moyen de prototypage et d'essai pour savoir quels types de fonctions pouvant être d'accélérés à l'intérieur d'une puce Xeon. L'idée est de prendre une combinaison de Xeon-FPGA, créer le code HDL (Verylog) pour accélérer une fonction, puis éliminer cette fonctionnalité à partir du code en cours d'exécution sur le Xeon et le céder à l'FPGA sous forme d'une description compatible (bitstream) pour montrer quel genre d'accélération est possible.

2.3 Les architectures régulières des FPGAs

Nous décrivons dans cette partie les structures des plateformes FPGAs présentes sur le marché qui intègrent la fonction de la RDP. Xilinx et Altera, sont les constructeurs les plus connus mondialement pour ce propos. Les familles FPGA Virtex et Spartan sont les familles les plus connues chez Xilinx qui offrent la possibilité d'implémenter des systèmes à base de la DPR. De même nous distinguons la famille Startix de la firme Altera. Ces FPGAs sont

construits à base de plusieurs ressources hétérogènes (CLB, DSP et RAMB) disposées sous forme de lignes verticales les unes à coté des autres tout le long de la largueur de la plateforme ainsi que plusieurs autres ressources positionnées sur le contour de la plateforme tels que les IOBs (Input Output Banks), les PLLs (Phase-Locked Loops), les ports de communication à haut débit (PCI, Ethernet, …) etc.

Comme l'indique la Figure 1.1, un FPGA est structuré en plusieurs colonnes (S) dont chacune est subdivisée en plusieurs régions appelées Régions d'Horloges. Ces dernières sont réparties de manière uniforme, équitable et symétrique verticalement. Elles sont référenciées par leurs coordonnées (XY) et comportent des ressources sous forme de Frames. Ces Frames se diversifient par trois types essentiels qui sont Frame_CLB, Frame_DSP et Frame_RAMB.

2.3.1 Les frames

2.3.1.1 Les Frame_CLB

Comme illustré dans la Figure 3.2, la Frame_CLB est une colonne composée de plusieurs CLBs (Configurable Logic Block) référenciés par leurs coordonnées XY et qui représentent l'élément dominant pour construire une région d'horloge. La Frame_CLB est caractérisée, d'un FPGA à un autre, par sa taille qui est égale au nombre de CLBs qu'elle comporte. Par exemple, dans la Virtex6, la taille de la Frame_CLB est de 40 tandis que sa taille dans la Virtex7 est de 50. Chaque CLB comporte deux SLICEs de nature identique ou différente (SLICEL et/ou SLICEM). Chaque SLICE contient quatre éléments principaux qui sont : 4 x LUT (Look Up Tables), 8 x éléments de stockage, des éléments de multiplexage et des éléments de retenue.

Les SLICEs se diffèrent par leurs fonctions, on trouve :

- Les SLICELs : ce sont des éléments qui permettent d'implanter des fonctions logiques, arithmétiques et des fonctions mémoire ROM.
- Les SLICEMs : ce sont des SLICEL qui intègrent de plus deux fonctions supplémentaires qui sont : stockage en mémoire distribuée RAM et registre de décalage de 32 bits.

Cependant, les deux types de CLB sont : Le premier type est le CLBLL qui comporte deux SLICELs. Le deuxième type est le CLBLM qui comporte un SLICEL et un SLICEM. Nous distinguons aussi deux types d'orientations des CLBs dans les FPGAs : les CLBs côté gauches (CLB_L) et les CLBs côté droite (CLB_R).

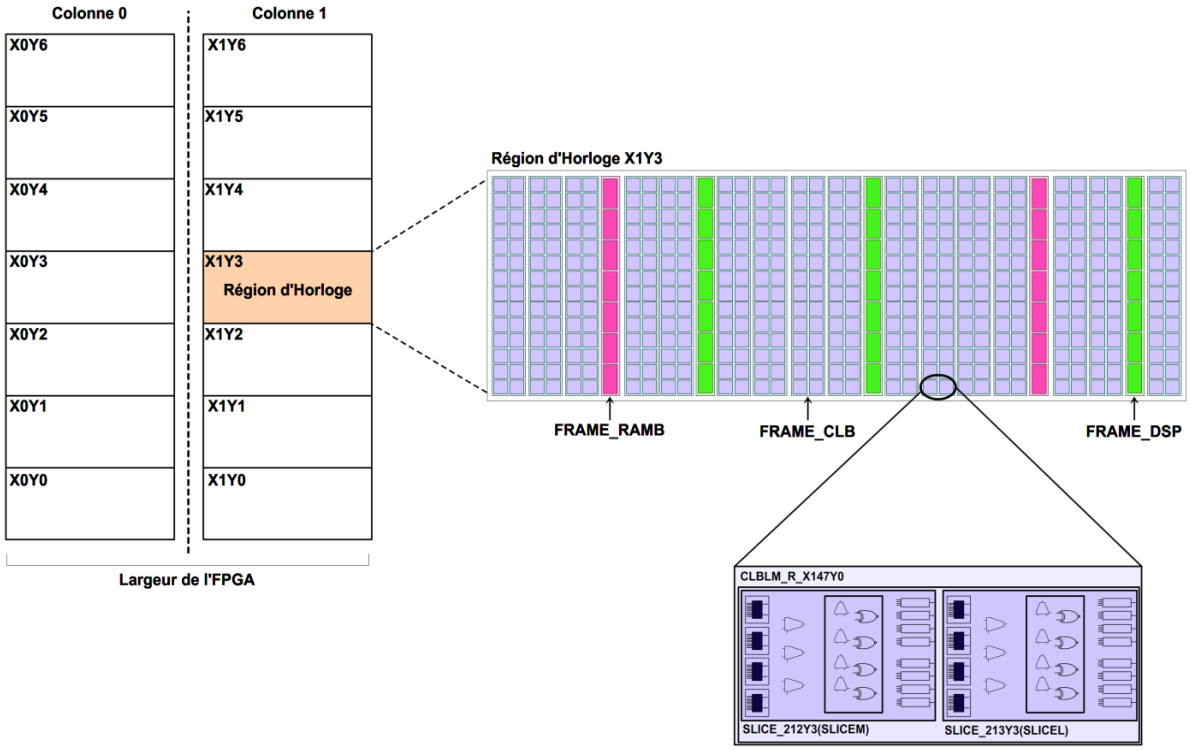


Figure 2.3 : Structure d'un élément CLB

2.3.1.2 Les Frame_RAMB

Comme les Frame_CLBs, les Frame_RAMB sont des colonnes des BRAM (Blocks RAM) caractérisées par une taille et représentent des éléments de construction d'une région d'horloge. Chacune des BRAMs est identifiée par ses coordonnées XY et une taille mémoire de 9Kbits. Ce composant supporte deux types d'opérations (simple ou double) dual-port en lecture et écriture. Une opération simple de lecture ou d'écriture est effectuée sur un bus de 18 bits pour une BRAM de 9Kbits. Une opération double en lecture ou en écriture est effectuée sur un bus de 36 bits pour un double BRAM de 9Kbits (18Kbits). Une BRAM est considérée soit comme un FIFO pour le stockage temporaire des données, soit d'une RAM qui par exemple gère une machine d'états finis.

2.3.1.3 Les Frame_DSP

Les Frame_DSP sont des colonnes de DSP48 [85] qui sont présentes dans les régions d'horloges. Chaque DSP48 est caractérisé par ses coordonnées XY et sert à effectuer des opérations de multiplication. Leur utilisation est couteuse et peuvent engendrer la baisse de performance d'un système implanté sur FPGA. Souvent, les opérations de multiplication sont implantées à base de SLICE grâce à des registres de décalage.

2.4 La reconfiguration dynamique partielle : son principe, son flot de conception et ces types

Dans cette section, nous présentons les concepts de base des systèmes reconfigurables sur FPGA : le concept de la reconfiguration dynamique partielle, le flot de conception des systèmes correspondant et les types de transfert de donnée de configuration séquentielle/parallèle.

2.4.1 La reconfiguration dynamique partielle

Le mécanisme de la reconfiguration dynamique partielle, illustré dans la Figure 2.4, permet la modification de certaines régions (RPRs) du FPGA durant l'exécution sans perturber le comportement des modules statiques. Ce concept permet le partage temporel des ressources matérielles entre les tâches mutuellement exclusives au sien d'une RPR. La reconfiguration dynamique partielle est utilisée dans plusieurs contextes tels que l'adaptation au changement de l'environnement, la réduction de la consommation d'énergie, etc.

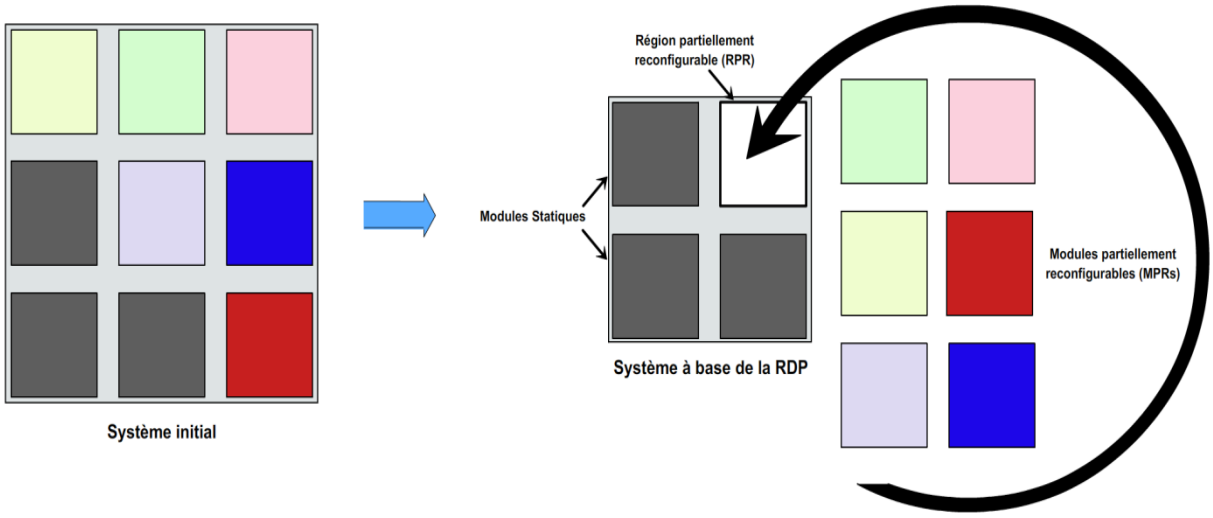


Figure 2.4 : Vue globale d'un système partiellement reconfigurable

Le support commercial de la reconfiguration dynamique partielle a commencé avec l'apparition de la famille des FPGAs Virtex de Xilinx à la fin des années 90. En 2010, Altera a commercialisé son premier FPGA supportant la reconfiguration dynamique partielle, le Startix-V [8]. Dans notre travail, nous avons utilisé les FPGAs de Xilinx sur lesquels beaucoup de travaux ont été effectués. Xilinx a commencé par présenter deux méthodologies de reconfiguration partielle (basée-différence et basée-module) [9] [10], suivies de la méthodologie EAPR (Early Access Partial Reconfiguration) en 2006 [11], et de la

reconfiguration partielle basée sur le partitionnement (Partition-based Partial Reconfiguration) en 2010 [12].

2.4.2 Flot de conception des systèmes sur FPGA

2.4.2.1 Le flot de conception des systèmes statiques

Un système implémenté sur FPGA est composé d'une partie logicielle et d'une partie matérielle.

- **La partie logicielle**

Le fichier MSS (Microprocessor Software Specification) contient une "déclaration" des librairies, des pilotes et éventuellement des OS utilisés pour les composants matériels du système. A l'aide de l'outil LibGen [60], les bibliothèques logicielles sont générées. L'application à exécuter par le microprocesseur est généralement écrite en langage assembleur ou C/C++. Le résultat de la compilation est un fichier ELF.

- **La partie matérielle**

La partie matérielle est décrite en utilisant soit des langages de description matérielle (HDL) tels que VHDL et Verilog, soit avec une description schématique. Les outils de Xilinx utilisent un fichier MHS (Microprocessor Hardware Specification) contenant entre autres une "déclaration" des composants à instancier dans le système. En cherchant les IPs correspondants dans la librairie d'IPs, la partie HDL du système est générée à l'aide de l'outil PlatGen [98]. Cet outil génère aussi une description des BRAMs qui seront utilisées pour les microprocesseurs (fichier d'extension BMM [99]). L'étape de synthèse permet d'obtenir des netlists (fichiers NGC) qui sont une description au niveau portes logiques du design. L'étape de traduction (par l'outil NGDBuild) permet de faire la fusion entre les fichiers NGC, le fichier BMM (Block Memory Map), et les fichiers de contraintes (UCF) pour générer un fichier design (NGD). L'étape de mapping permet de mapper la logique décrite dans les netlists sur les composants du FPGA (CLBs, I/Os, etc.). L'étape de placement et routage (PAR) permet de positionner les différents éléments du système implémenté sur FPGA et d'assurer leur interconnexion. Cette étape est réalisée en utilisant un outil propriétaire de placement et routage. Ensuite, le fichier de configuration (bitstream) est généré.

- **L'association entre logiciel et matériel**

A l'aide de l'outil BitInit [100], le bitstream correspondant à la partie matérielle du système est fusionné avec le fichier exécutable du logiciel, ce qui permet de décrire le code et les données qui seront stockées dans les BRAMs pour exécuter l'application. Le résultat de cette fusion est le fichier bitstream download.bit qui peut être chargé par la suite sur le FPGA à partir du PC en utilisant l'outil iMPACT. Ce bitstream peut être également stocké dans une mémoire externe du FPGA ou un compact flash pour qu'il soit chargé automatiquement dès la mise sous tension du FPGA.

- **La simulation**

Au cours de la conception du système, la simulation peut se faire à différents niveaux [13]. La simulation comportementale peut se faire avant la synthèse. Cette simulation permet de vérifier si le système fonctionne comme prévu. La simulation structurelle est appliquée après la synthèse. La différence entre le modèle de simulation structurelle et le modèle de simulation comportementale est que ce dernier utilise une abstraction des composants du système. Il contient des opérations à haut-niveau tel qu'un opérateur d'addition 4 bits, au lieu d'un additionneur Xilinx pour le modèle structurel. La simulation structurelle prend plus de temps que la simulation comportementale puisqu'elle intègre plus de détails. Pour cette raison, il est conseillé de commencer par une simulation comportementale afin de détecter le plus d'erreurs le plus tôt possible dans le flot de conception. La simulation temporelle est faite après le placement et routage. Elle permet de décrire le comportement du système sur le vrai circuit et de donner en plus des informations temporelles. Par exemple, pour un système synchrone, cette simulation permet de donner le temps t entre le front montant de l'horloge et le changement de la valeur d'un signal qui y est sensible. Ce temps semble être nul dans les simulations comportementales et structurelles qui ne prennent pas en compte les contraintes temporelles du système. Une telle information (le temps t par exemple), permet de déterminer si le système peut fonctionner avec une fréquence f donnée (vérifier que $t < 1/f$). Après la simulation temporelle, le bitstream du système peut être généré afin d'être chargé sur FPGA.

2.4.2.2 La conception des systèmes partiellement reconfigurables

Dans cette section, l'évolution technologique de la reconfiguration partielle pour les FPGAs de Xilinx est présentée avant de décrire brièvement le plus récent flot de conception

lié à la reconfiguration dynamique. Plus de détails sur ce flot seront présentés dans le chapitre 6 dans la partie qui décrit l'implémentation physique de notre modèle de contrôle.

- **Reconfiguration partielle basée-différence**

Ce type de reconfiguration convient pour de petits changements. Il est basé sur la comparaison de deux designs (le design avant et après la reconfiguration). Il détermine les trames différentes et crée un bitstream partiel qui modifie seulement les trames qui sont différentes, ce qui donne un temps de reconfiguration réduit. Les modifications peuvent affecter les standards I/O, le contenu de mémoires ou la configuration des LUTs.
L'inconvénient de cette méthode est que le concepteur doit manipuler des éditeurs de bas niveau tels que FPGA Editor, pour effectuer ce genre de modification, ce qui nécessite une grande connaissance de l'architecture du FPGA. Cette méthode ne convient pas pour les designs complexes qui nécessitent de plus grandes régions reconfigurables.

- **Reconfiguration partielle basée-module**

Cette méthode permet de diviser le design en modules. Ces modules peuvent être statiques (chargés seulement lors de la mise sous tension) ou dynamiquement reconfigurables.
Cette division permet aux concepteurs de travailler indépendamment sur différents modules puis de fusionner ces modules pour obtenir un système complet. La séparation permet également de modifier un module indépendamment des autres lors de la reconfiguration. Pour ceci, un bitstream initial est nécessaire pour conFigurer tout le FPGA, et des bitstreams partiels sont utilisés pour conFigurer chaque région reconfigurable.
Pour remplacer un module reconfigurable par un autre, il faut qu'ils aient la même interface.
L'inconvénient de cette méthode est au niveau du floorplanning se manifestant par le fait que la hauteur d'une région reconfigurable doit occuper toute la hauteur du FPGA, ce qui constitue une perte de ressources surtout pour les FPGAs de grande taille.

- **Le flot Early Access Partial Reconfiguration**

En 2006, Xilinx a introduit un flot de conception qui s'appelle Early Access Partial Reconfiguration (EAPR) et qui permet d'utiliser des modules reconfigurables à 2 dimensions.
Un module reconfigurable à 2 dimensions est un module qui peut prendre une forme rectangulaire quelconque, à la différence des modules à une dimension qui occupaient toute la hauteur du FPGA avec l'aspect modulaire de la reconfiguration partielle. L'EAPR permet aussi aux fils statiques de passer directement à travers les modules reconfigurables sans

l'obligation d'utiliser les bus macros [57], ce qui permet d'améliorer la performance et simplifier la conception. Selon Xilinx, une région reconfigurable s'appelle PRR (Partial Reconfigurable Region). Une PRR peut avoir plusieurs implémentations possibles ou PRMs (Partial Reconfigurable Modules). Tous les PRMs d'une PRR ont la même interface externe pour faciliter la compatibilité. Tous les PRMs d'une PRR doivent être prédéterminés. Au début, un bitstream initial est chargé sur le FPGA. Ce bitstream contient la partie statique et les PRMs initiaux des PRRs. Ensuite au cours d'une reconfiguration partielle, le contrôleur charge un bitstream partiel correspondant à un autre PRM de la même PRR. A l'aide du mécanisme Read-Modify-Write, les trames différentes entre les deux PRMs sont modifiées est réécrites dans la mémoire de configuration.

- **La reconfiguration partielle basée sur le partitionnement (Partition-based Partial Reconfiguration)**

L'apport du nouveau flot de conception de RDP de Xilinx se voit notamment au niveau de la phase de placement du système. Dans le nouveau flot, l'utilisation du bus macros est remplacée par l'utilisation des broches de partition (Partition Pins). Ces broches sont crées automatiquement pour les ports des partitions reconfigurables, alors que les bus macros se plaçaient manuellement sur le système en utilisant l'outil de placement PlanAhead de Xilinx. Dans le nouveau flot, les termes RP (Reconfigurable Partition) et RM (Reconfigurable Module) ont remplacé les termes PRR et PRM d'EAPR. Dans ce mémoire, nous avons utilisé ce flot de conception de systèmes reconfigurables pour la validation du modèle de contrôle proposé. Cependant, nous utilisons les anciens termes (PRR et PRM) qui sont les plus utilisés dans les travaux traitant la RDP. L'implémentation d'un système partiellement reconfigurable sur FPGA est similaire à l'implémentation de plusieurs systèmes statiques qui partagent des parties logiques. Les partitions sont utilisées pour s'assurer que cette partie commune est identique pour tous ces systèmes. Ce concept est illustré dans la Figure 2.3 [9].

Cette Figure donne l'exemple d'un système ayant une seule région reconfigurable qui a N implémentations différentes (N modules reconfigurables). Le flot de conception de ce système suit les étapes suivantes :

- Ecriture en HDL du module TOP décrivant tout le système avec la déclaration des instances des modules statiques et d'une instance des modules reconfigurables parce qu'ils seront implémentés dans la même région reconfigurable dans l'exemple de la Figure 2.3.

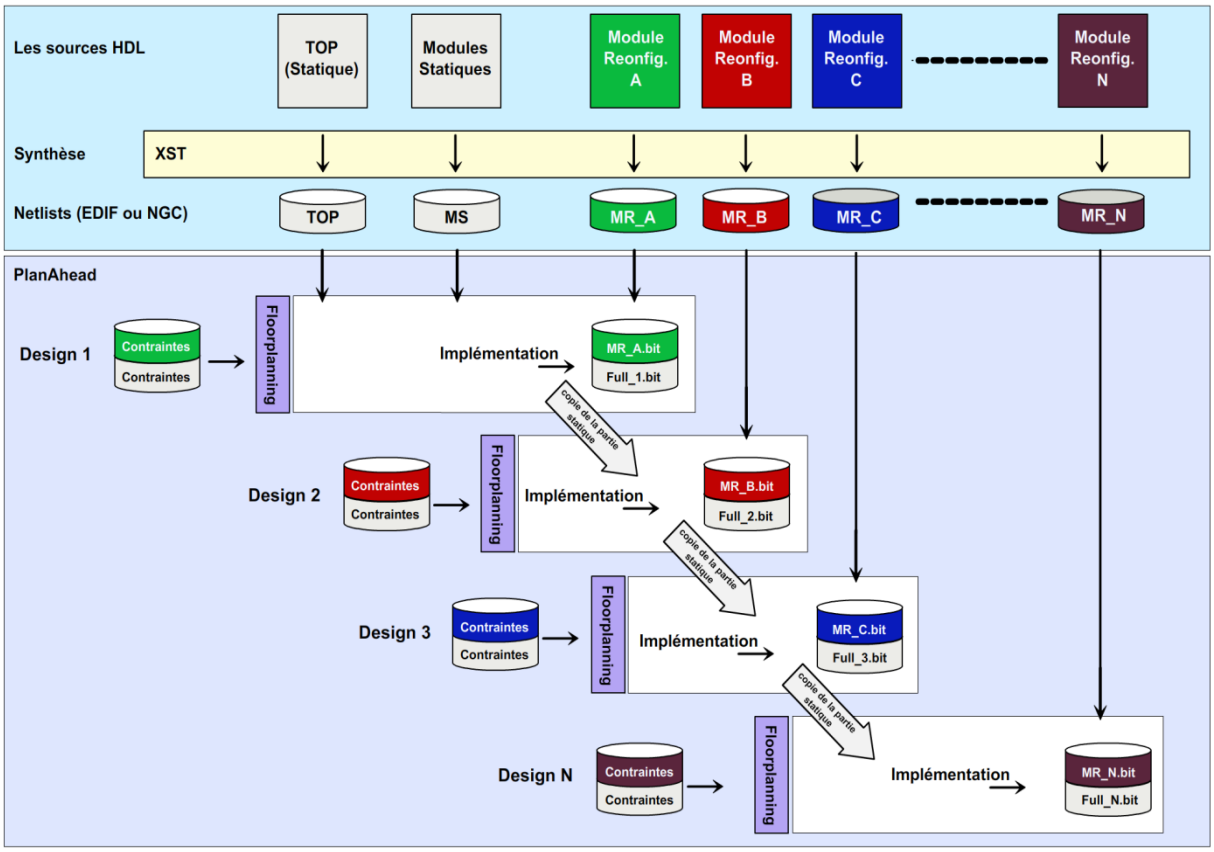


Figure 2.3 : Flot de conception d'un système reconfigurable sur FPGA de Xilinx

- Ecriture en HDL des modules statiques et des modules reconfigurables. Ces derniers doivent avoir le même nom puisqu'ils sont représentés par un seul composant dans le fichier TOP.
- La synthèse des fichiers HDL pour le TOP, les modules statiques et les modules reconfigurables. Ces derniers ayant le même nom, doivent être synthétisés dans des projets Xilinx séparés.
- L'implémentation du système est lancée N fois en prenant en compte à chaque fois des contraintes statiques, et les contraintes des modules reconfigurables. En d'autres termes, à chaque fois, nous lançons les phases NGDBuid (Translate), MAP et PAR pour un système constitué des netlists du TOP, des modules statiques et d'un module reconfigurable, avec les contraintes statiques et les contraintes du module reconfigurable comme le montre la Figure 2.3. Pour éviter la réexécution de l'implémentation sur les parties qui ne sont pas modifiées entre les représentations du système, ces parties sont copiées à partir de la première implémentation. Cette copie est utilisée pour le reste des implémentations comme le montre la Figure 2.3. Pour

générer les autres implémentations du système, il suffit de faire la fusion entre cette copie et les différentes implémentations des modules reconfigurables. Au cours de cette phase, la compatibilité entre celles-là et la copie de la partie statique est vérifiée afin de garantir le bon déroulement de la reconfiguration partielle au cours de l'exécution.

- A partir de chaque implémentation, deux bitstreams sont générés : le bitstream total et le bitstream partiel. Au début de l'exécution du système, un bitstream total est chargé. Pour pouvoir changer la configuration de la région reconfigurable au cours de l'exécution, les bitstreams partiels seront chargés dans cette région l'un après l'autre. Pour cette raison, il faut choisir le bitstream total à charger initialement dans le FPGA, ce qui détermine la configuration initiale de la région reconfigurable. Par exemple, si nous choisissons de charger le bitstream Full_2.bit, la configuration de la région reconfigurable sera équivalente à un chargement du bitstream RMB.bit dans cette région.
- Le bitstream total choisi est fusionné avec le fichier exécutable (le fichier ELF [91]) de la partie logicielle du système (si elle existe) comme nous avons montré dans la Figure 2.6. Le fichier binaire qui en résulte peut être chargé dans le FPGA directement de l'ordinateur en utilisant le logiciel iMPACT ou en le sauvegardant dans une mémoire externe du FPGA ou un System ACE (utilisant une Compact Flash). La mémoire externe ou le compact Flash doit donc contenir en plus de la configuration totale du système, tous les bitstreams partiels à charger durant l'exécution. Dans ce cas, à la mise sous tension, le bitstream total et le programme du processeur (s'il existe) sont chargés dans le FPGA et l'exécution de l'application commence.

2.4.3 Les types de reconfigurations

2.4.3.1 Le processus de la reconfiguration dynamique partielle séquentiel

Pour reconfigurer partiellement un FPGA, il est nécessaire d'isoler la zone à reconfigurer et charger les mots de configuration (bitstream) correspondants à cette zone. Un outil appelé PARBIT [14] a été développé par Xilinx afin de générer les bitstreams partiels pour implémenter la reconfiguration partielle.

Pour charger un bitstream sur FPGA, nous pouvons passer par un contrôleur externe ou un contrôleur interne. Pour la reconfiguration partielle externe, nous utilisons des interfaces telles que JTAG et SelectMap [15]. Pour la reconfiguration partielle interne, les FPGAs de Xilinx

utilisent l'ICAP (Internal Configuration Access Port) [16], qui est un sous ensemble de l'interface parallèle à 8-bits SelectMAP. L'ICAP permet donc d'accélérer le processus de reconfiguration dynamique partielle par rapport à l'interface SelectMAP et l'interface série JTAG [17]. L'ICAP est présent presque dans tous les FPGAs Xilinx. Pour les Virtex-II et Virtex-II Pro, l'ICAP fournit des ports d'entrée/sortie de 8 bits, alors que pour Virtex-4, l'interface d'ICAP est de 32 bits avec une possibilité d'alternance entre 8 et 32 bits [18]. Pour les versions ultérieures de Virtex, l'interface d'ICAP offre la possibilité d'alternance entre 8, 16 et 32 bits. La fréquence maximale avec laquelle l'ICAP peut fonctionner est 100Mhz pour les Virtex-4, 5, 6 et 7 et 20Mhz pour Spartan-6 [19].

Le débit théorique de l'ICAP est la taille d'une donnée traitée divisée par la période du temps, ce qui fait un débit de 100 Ko/ms pour une interface d'ICAP de 8 bit et une fréquence de 100Mhz. Cependant, cette valeur ne peut pas être atteinte en réalité. Par exemple, pour les Virtex-II Pro, quand la fréquence de l'horloge de l'ICAP est supérieure à 50 MHz (100 MHz pour Virtex-4), il est nécessaire de respecter le signal d'acquittement (busy) de l'ICAP, ce qui fait que la vitesse théorique maximale ne peut pas être atteinte [20]. Il y a plusieurs travaux qui ont ciblé l'augmentation du débit de l'ICAP. Le nombre d'ICAPs disponibles sur un FPGA est passé d'un seul pour les FPGAs Virtex-II Pro à deux pour les Virtex-4 et les FPGAs plus récents. Ces deux ICAPs sont placés généralement au centre du FPGA. Ils sont physiquement adjacents, l'un au-dessous de l'autre. Ces deux ICAPs ne peuvent pas être actifs simultanément. Le système doit commencer par l'ICAP situé au-dessus, qui est actif par défaut [21]. Au cours de l'exécution, nous pouvons commuter vers l'autre ICAP en écrivant un bit de contrôle. L'utilité de cette redondance est que, dans le cas d'un dysfonctionnement ou blocage du premier ICAP, nous pouvons toujours sélectionner le deuxième ICAP, ce qui laisse le temps pour réinitialiser le premier ICAP sans interrompre l'exécution de l'application. L'ICAP ne peut pas être directement connecté au bus système puisqu'il nécessite un contrôleur pour gérer le flot de données venant du contrôleur de reconfiguration.

Le mécanisme de reconfiguration utilisant ICAP s'appelle read-modify-write (RMW) [22]. Ce mécanisme permet au contrôleur de la reconfiguration de modifier le bitstream correspondant à un module dynamiquement reconfigurable à l'aide du module HwICAP. Un module HwICAP est composé d'un contrôleur de l'ICAP, d'une mémoire BRAM et du cœur de l'ICAP. Ce module permet au contrôleur de la reconfiguration d'accéder à l'ICAP pour lire et écrire des données de configuration dans la mémoire de configuration. Ce contrôleur peut être commandé par le processeur du système. Le processus de reconfiguration commence quand le processeur détermine le bitstream partiel à charger et son emplacement dans la mémoire

externe. Le processeur envoie ensuite les trames une par une au contrôleur de l'ICAP. Celui-ci demande à l'ICAP de lire les trames correspondantes de la mémoire de configuration une par une. Les trames envoyées par l'ICAP sont stockées dans une mémoire BRAM du contrôleur. Cette mémoire a suffisamment de capacité pour stocker au moins une trame (Frame) de configuration [23]. Le contenu de chaque trame est fusionné avec celui de la trame correspondante envoyée par le processeur. L'ICAP doit donc réécrire cette trame de la BRAM vers la mémoire de configuration suivant le mécanisme read-modify-write. Quand une trame est réécrite dans la mémoire de configuration, les composants logiques correspondants aux données non modifiées de la trame continuent leurs opérations sans aucun changement [24].

HwICAP est capable de contrôler l'interface de l'ICAP. Son utilisation nécessite l'utilisation du processeur MicroBlaze ou PowerPC [25] dans une architecture reconfigurable. Ce système présente un ordre de complexité important vu le goulot d'étranglement en connexions qu'il comporte. En effet, il ne permet de garantir qu'un débit réduit. Xilinx a pensé ensuite à une architecture alternative de reconfiguration dynamique basée sur une solution matérielle [26] [27]. Ceci rentre dans l'objectif de réduire la surface occupée et le grand nombre de connexions mais en gardant le même débit fourni par l'architecture de base. Cette dernière idée a poussé les concepteurs à chercher d'autres alternatives pour consolider la réduction de la surface occupée par cet IP tout en augmentant la vitesse de transfert des bitstreams partiels vers la mémoire reconfigurable. Ceci tout en forçant l'ICAP à fonctionner au-delà de sa capacité en termes de fréquence d'horloge pour atteindre la réponse maximale qu'il peut supporter. Divers travaux se sont focalisés sur cet aspect, nous citons le contrôleur FaRM [28] qui est un contrôleur rapide qui permet de réduire le surcoût de la reconfiguration. Cet IP utilise divers techniques, notamment les mémoires DMA, le pré-chargement des bitstream partiels et la sur-cadence de l'horloge. Il peut fonctionner à une horloge de 200 Mhz et garantir un débit de 800 Mo/s. D'autre part, ce contrôleur est soumis à des limites telles que la limitation en termes de fréquences permises en fonction de la technique de compression de fichier utilisée.

Nous citons aussi le contrôleur UPaRC [29], qui est un contrôleur très rapide permettant de garantir un débit élevé égal à 1.433Go/s grâce une sur-cadence d'horloge de 362.5Mhz. Ce contrôleur est construit à base de :

- Une mémoire BRAM qui joue le rôle de synchronisation entre le domaine d'horloge côté ICAP et le domaine d'horloge côté processeur d'initiation (MicroBlaze) et le pré-chargement du bitstream.

- Le contrôleur de la reconfiguration appelé UReC joue le rôle fondamental dans ce système : il permet de gérer le début et la fin de la phase de reconfiguration, la gestion de la lecture des mots de reconfiguration depuis la BRAM et enfin l'interfaçage avec l'ICAP pour gérer son activation.

D'autre part, un mécanisme de pipeline peut améliorer la gestion d'écriture/lecture dans la BRAM et qui aurait être effectué pour augmenter encore la valeur du débit. L'utilisation des mémoires internes de l'FPGA pour une telle méthodologie est assez couteuse car la taille d'un bitstream varie en fonction de la taille du module à reconfigurer. Ceci pousse les concepteurs à intégrer un mécanisme de compression/décompression des bitstreams pour réduire la taille occupée par la BRAM sur puce. Ce fait nécessite toutefois l'allocation d'une mémoire de bitstream et un décompresseur puisque la compression se fait en dehors de l'FPGA. Le temps de décompression peut effectivement influencer la rapidité du transfert du bitstream.
Dans [30], les auteurs décrivent l'utilisation de la mémoire externe DDR dans leur contrôleur de reconfiguration à base de MPMC [31]. En effet, l'utilisation de telle mémoire permet de stocker un grand nombre de bitstream partiels et assurer un débit ultra-rapide de 1,48Go/S pour les transférer vers la mémoire reconfigurable via l'ICAP. Dans ce cas, inutile de prévoir les mécanismes de compression et de décompression. En effet la mémoire DDR utilisée permet le transfert des mots de reconfiguration de 64 bits à une vitesse d'horloge de 200 Mhz. Etant donné que le port de données de l'ICAP est sur 32 bits, le débit fourni à 200 Mhz peut être multiplié par 2 si on divise les données transférées sur 2 mots de 32 bits. Ceci permet d'assurer un débit de transfert théorique de 1.6 Go/sec à 400Mhz, mais dû à des contraintes physiques, ce contrôleur fournit un débit réel de 1.48 Go/s à une fréquence de 370 Mhz. l'architecture du contrôleur proposée est caractérisée par deux flux de données : 1) le flux d'écriture des bitstreams dans la DDR en utilisant le contrôleur MPMC. Ce flux est géré par le processeur MicroBlaze. 2) le flux de lecture qui permet de gérer l'envoi des mots de reconfiguration de la DDR vers la mémoire reconfigurable de l'FPGA qui est à son tour géré par le Microblaze pour doubler la fréquence d'horloge et gérer l'écriture à travers l'ICAP.

2.4.3.2 La reconfiguration dynamique partielle parallèle

Cette section traite la façon dont une reconfiguration parallèle peut être initiée dans un système donné. Il y a deux façons pour se faire. Tout d'abord, la reconfiguration parallèle peut être initiée par l'envoi du flux binaire à partir d'un processeur hôte pour chaque FPGA en

parallèle par l'intermédiaire des flux DMA à travers le bus PCIe comme le montre la Figure 2.4.

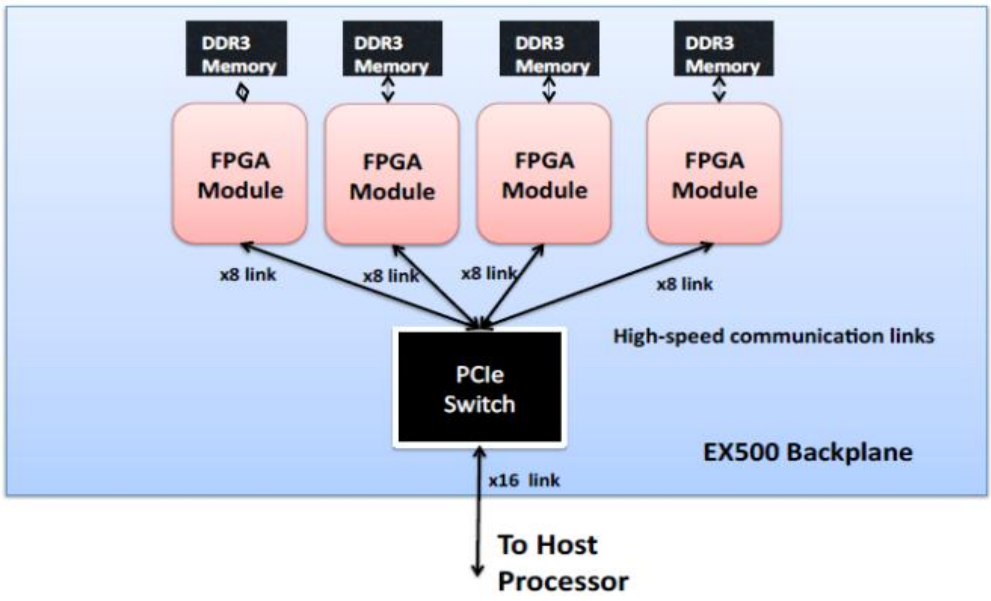


Figure 2.4 : La communication entre le processeur Hôte et l'FPGA

Elle est suivie par une émission d'une commande de démarrage à tous les FPGA en utilisant les registres PCIe. La deuxième option consiste à stocker le bitstream dans chaque DDR3 locale FPGA individuellement, puis diffusé une commande de reconfiguration de démarrage à tous les FPGA en même temps, de sorte que chaque FPGA peut commencer à lire à partir de la mémoire locale respective et commencer le processus de reconfiguration [32].

En outre, il est essentiel de concevoir un contrôleur ICAP pour chacun des deux cas afin qu'il puisse lire le bitstream partiel de l'hôte ainsi que la mémoire DDR3 locale.

Plusieurs travaux ont été élaborés pour assurer la RDP parallèle. [33] propose un FPGA bien adapté à la RDP parallèle. Il s'est focalisé sur l'intégration d'une couche mémoire jouant le rôle d'accélérateur de la reconfiguration dynamique. Cette couche contient une mémoire cache et une DRAM de taille 64 Mbit capable de supporter 289 configurations différentes. Contrairement dans les architectures FPGAs classiques, les DRAMs sont placées en dehors des la puce dû au goulot d'étranglement que peut engendrer les interconnexions. Les configurations sont chargées depuis la DRAM à travers une mémoire cache vers l'FPGA. Le délai entre deux configurations est de 5 cycles (60ns) et la latence d'écriture entre la DRAM et la mémoire cache est de 8.42µs. Ce mécanisme est très efficace au point de vue

optimisation par rapport à l'architecture standard de l'FPGA qui emploi un processeur logiciel pour le contrôle. La Figure 2.5 montre l'architecture proposée.

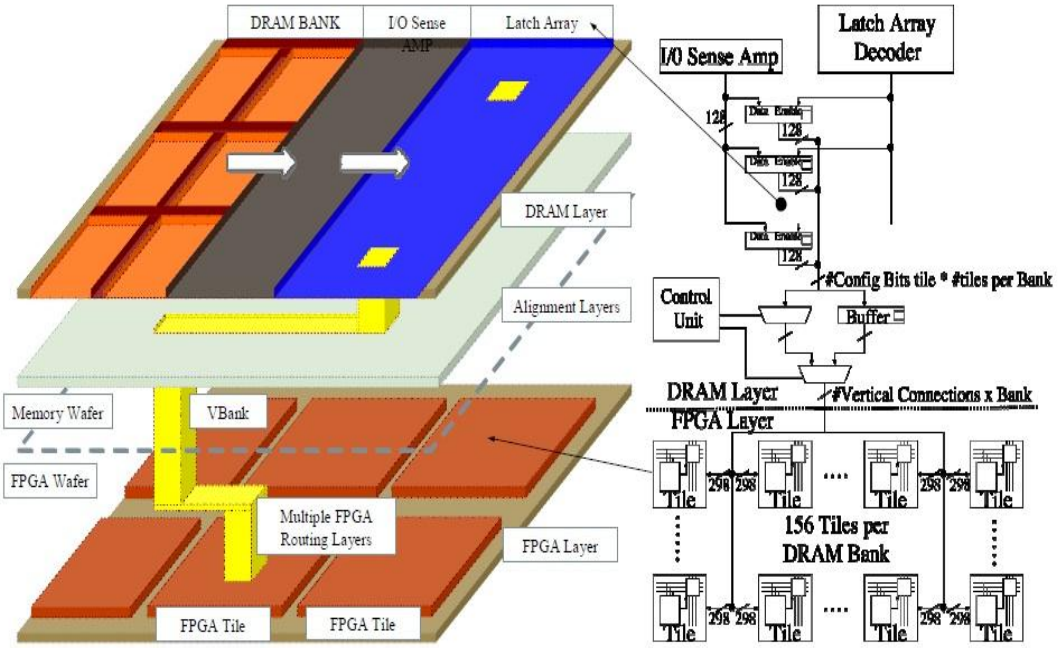


Figure 2.5 : Architecture globale de la Plateforme FPGA

Le mécanisme de reconfiguration s'effectue comme suit : initialement les configurations sont stockées dans la DRAM. Le début de la reconfiguration commence par la lecture de la configuration depuis la DRAM vers la mémoire cache via les entrées sorties *I/O sens Amp*, le cache est composé d'un ensemble de registres de 128 bits contrôlés par le Latch Array Decoder. Les ports de sortie des registres forment un bus de configuration de taille égal au nombre de bits d'une configuration par tuile (1488 bits) multiplié par le nombre des tuiles par banc (156 tuiles). Ce bus est contrôlé par le signal *Enable*. Ensuite, les bits de configuration sont redirigés soit vers un tampon mémoire dans le cas d'une configuration maximale ou vers un MUX géré par le bloc Control Unit pour une configuration minimale. Le MUX de sortie gère ces deux types de configurations. Le bus de sortie de ce MUX est de largeur relative au nombre des connexions verticaux (298) multiplié par la taille d'un banc mémoire (64 Mbits). Enfin les bits de reconfiguration sont diffusés à 156 tuiles à travers leurs connexions verticales de 289 bits. Cette technique de reconfiguration a montré des résultats significatifs par rapport à d'autres architectures en termes de vitesse et de surface grâce l'intégration d'une couche de mémoire au dessus des autres couches [34].

2.5 Les algorithmes de floorplanning automatiques existants pour FPGA

L'un des objectifs que nous visons est l'automatisation de la phase de floorplanning applicable sur les FPGAs. Cette phase nécessite une bonne connaissance de la structure granulaire de l'FPGA. L'automatisation s'effectue par la conception d'un algorithme qui prend en compte les différentes entrées de l'architecture parallèle reconfigurable qu'on veut implanter. Nous visons aussi à effectuer un floorplanning qui favorise la technique de relocalisation de bitstream dans le but de mettre en ouvre le mécanisme de reconfiguration parallèle d'un certain nombre de RPRs. Ces dernières doivent être identiques en termes de forme et contenu pour favoriser un tel mécanisme.

2.5.1 Les heuristiques et méthodes de floorplanning

Les travaux récents sur le floorplanning ont mis l'accent sur deux aspects différents [35] [36] : Le premier met l'accent sur la réduction de la taille de bitstream en utilisant la méthode *Columnar Kernel Tessellation*. Cette méthode sert à minimiser le coût des ressources gaspillées. Alors que le second aspect vise à optimiser la longueur des signaux d'interconnexions entre les différents modules d'un système reconfigurable en utilisant le principe de *recuit simulant* [75]. La formulation *Mixed Integer-Programmation linéaire* présentée dans [37] a été utilisée pour améliorer les méthodes précédentes en termes d'optimisation de l'espace des solutions qui englobe l'ensemble des régions partiellement reconfigurables. [37] présente deux algorithmes d'optimisation. Le premier est le *Heuristic Optimal* (HO) qui est utilisé pour réduire le temps de recherche des régions éligibles pour la reconfiguration tout en considérant la première solution une contrainte qui sert à réduire au minimum l'espace de recherche de solutions. Le second, appelé *Optimal* (O), est utilisé pour explorer l'espace total des solutions. L'algorithme (O) nécessite clairement une durée plus longue que le (HO). Les deux algorithmes ont été adaptés pour supporter la relocalisation de bitstream [39]. La nouvelle version permet la réutilisation de la conception mais cause des pertes de performances au niveau de la longueur des signaux d'interconnexion et la taille des bitstreams partiels. Dans le même cadre, les auteurs proposent un mécanisme d'automatisation de floorplanning, intégré dans le flot de conception des outils de Xilinx, qui repose sur une énumération explicite des emplacements possibles de chaque région. En effet ils proposent un algorithme génétique (GA), amélioré avec une stratégie de recherche locale, pour automatiser l'activité de floorplanning [76].

Suivant la littérature, des outils d'automatisation de floorplanning ont été conçus et basés sur des méthodologies bien spécifiques qui tiennent compte de l'allocation des régions partiellement reconfigurables. Nous citons HETRIS [40], un outil de floorplanning automatisé pour les FPGA hétérogènes et intégrant les fonctionnalités de VPR [41]. HETRIS utilise une approche basée sur la légalité adaptative pour cibler des architectures FPGAs Altera. Il porte sur des améliorations lui permettant d'exécuter, en moyenne, 15,6 fois plus rapide que les travaux antérieurs [42 43 44], tout en produisant un floorplanning plus dense que les outils commerciaux. HETRIS permet d'étudier la relation entre le partitionnement, le floorplanning et de l'architecture FPGA. Par contre, cet outil ne permet pas d'optimiser les temps de propagations entre les différents blocs fonctionnels ainsi que l'impact du floorplanning sur les phases de placement et routage pour garantir la qualité des résultats (QoR : Quality of Result). Un autre travail à prendre en considération propose un outil appelé PRFloor [45]. Il prend en compte le placement de l'ensemble des modules statiques et dynamiques du système. PRFloor se base sur la récursivité heuristique pseudo-bipartitioning unique qui utilise la formulation *Nonlinear Integer bipartitioner* [46]. PRFloor supporte diverses configurations de système reconfigurables (jusqu'à 130 modules, 24 PRR et 85% de la ressource FPGA). La fréquence d'horloge maximale obtenue pour tels systèmes à l'aide de PRFloor est supérieure de 3% comparé à des systèmes statiques. Pour le moment cet outil ne supporte malheureusement pas encore le floorplanning dédié pour la relocalisation de bitstream.

2.6 La technique de relocalisation de bitstream

La technique de relocation de bitstream est une technique utilisée pour déplacer un bitstream partiel d'une région partiellement reconfigurable vers une autre. Les deux régions reconfigurables doivent respecter plusieurs aspects pour être compatibles.

La forme est le premier aspect nécessaire pour l'application de cette technique. En effet, les deux régions reconfigurables doivent avoir les mêmes dimensions en longueur et largeur. Ceci explique que les deux régions ne peuvent contenir que des bitstreams de même taille.
La même disposition des ressources en termes des CLB, DSP et BRAM doit être présente dans les régions compatibles à la relocalisation des bitstreams. Ceci représente le deuxième aspect à prendre en considération. En effet, la même disposition des ressources est la contrainte la plus simple à tenir compte pour assurer cette compatibilité. Le routage à travers les différents types de ressources utilisées est aussi un facteur important pour obtenir des bitstreams partiels compatibles.

La Figure 2.6 présente ces différents aspects mentionnés ci-dessus. Nous remarquons que les régions R0.0 et R0.1 (de même pour R1.0, R1.1 et R1.2) respectent les exigences de la technique, par contre les régions R0.0 et R0.2 ne sont pas compatibles car la disposition des ressources ne sont pas identiques.

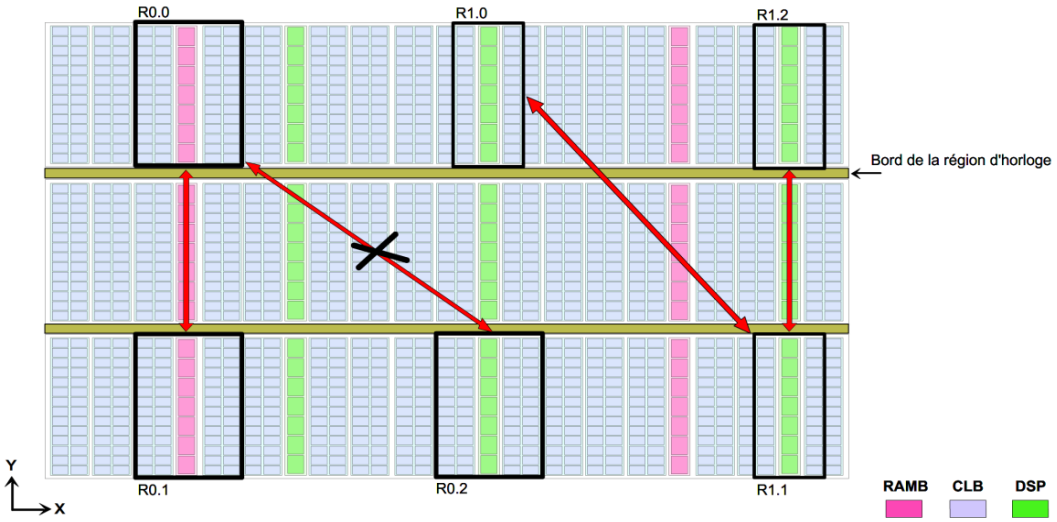


Figure 2.6 : Les différents aspects de la mise en œuvre de la technique de relocalisation de bitstream.

Il existe deux types de relocalisation de bitstreams : la relocalisation 1D et la relocalisation 2D. La relocalisation 1D consiste à assurer la compatibilité entre deux RPRs positionnées sur un même axe X ou Y. Par exemple, les régions R0.0 et R0.1 sont compatibles suivant l'axe des Y tandis que les régions R1.0 et R1.2 sont compatibles suivant l'axe des X. D'autre part la relocalisation 2D consiste à assurer la compatibilité entre deux RPRs identiques non positionnées sur le même axe X ou Y. Par exemple, les régions R1.0 et R1.1 sont compatibles suivant la relocalisation 2D.

Les bitstreams partiels obtenus à partir de la technique de relocalisation sont caractérisés par leur localisation. En effet chaque bitstream partiel contient cette propre information qui permet de conFigurer le registre FAR (Frame Address Register) [47] de l'FPGA à chaque fois la configuration commence.

Plusieurs travaux se sont focalisés sur la mise en œuvre de cette technique qui permet l'optimisation de l'espace mémoire utilisée pour leur stockage et essentiellement la réutilisation de bitstream partiel [48] [49] [50] [53]. La réutilisation du bitstream consiste premièrement à récupérer les données de reconfiguration d'un bitstream partiel déjà localisé dans une région reconfigurable R1. La récupération de ces données s'effectue à travers le port de l'ICAP en mode lecture. Les données de reconfiguration sont ensuite mises en forme pour

être transférées vers une autre région R2 reconfigurable compatible à la région d'origine R1. Pour cela la valeur du registre FAR relative à la région R2 doit être déterminée. Le transfert des données de reconfiguration s'effectue lors de la réception du mot de synchronisation par le contrôleur du port ICAP et se termine par la capture de la valeur de désynchronisation.

Cette technique ne peut être applicable que pour les FPGAs qui intègrent la fonction de la RDP. La technique de la RDP nécessite l'utilisation des interfaces entre les parties reconfigurables et la partie statique. Ces interfaces sont implémentées dans les régions partiellement reconfigurables sous forme logique (LUT) et qui s'appellent Proxys [51]. Ces éléments assurent l'isolation entre la partie statique et la partie dynamique et servent à sauvegarder le contexte d'exécution suite à une phase de reconfiguration donnée.

Suivant l'état de l'art, cette technique est passée par plusieurs améliorations dans le but de réduire le surcoût de ressources qui se manifeste par l'instanciation d'un LUT (de la partie statique) pour chaque élément Proxy. Cet aspect consiste d'une part à garder l'uniformité de routage interne des régions reconfigurables et d'autre part de forcer les signaux de la partie statique de ne pas passer à travers les régions reconfigurables pour ne pas modifier le contenu des bitstreams partiels générés. Dans le but de garder le même routage des signaux d'interfaçage de la partie statique et la partie dynamique, il est essentiel de récupérer les informations de routage de ces signaux suite à une première implémentation. Cette première implémentation sert à récupérer en effet la position de chaque élément proxy en termes de coordonnées XY, son élément et sa broche de connexion puisque les fonctions de placement et routage ne sont pas prédictibles par les outils de Xilinx. La position de chaque proxy est indiquée par les coordonnées XY du Slice utilisé. La connexion de chaque proxy est indiquée par l'un des LUTs contenu dans le Slice de position. L'identification de ces informations se fait grâce à des commandes bien spécifiques sous forme de contraintes. En effet, pour localiser un Slice, on utilise la contrainte « LOC » tandis que la commande « BEL » est utilisée pour spécifier un LUT. Sachant qu'un LUT comporte 2 broches de sortie appelées O5 et O6, alors la commande « PIN » [101] est alors utilisée pour spécifier l'une d'elles. Les proxys logiques sont placés dans des slices qui se situent au niveau des bords intérieurs d'une RPR. Chaque proxy logique consomme un seul élément LUT. Ce LUT est connecté en entrée avec un LUT appartenant à la partie dynamique et en sortie avec un LUT appartenant à la partie statique. La connexion de sortie s'effectue en effet grâce à l'une des deux broches O5 ou O6. Pour chaque deux éléments LUT connectés, les informations de routage du signal d'interconnexion entre eux sont déterminées d'après l'adresse de l'élément source, les adresses des blocks de routage utilisés qui décrivent le chemin traversé par le signal jusqu'à la

connexion de destination et l'adresse de l'élément de destination. Ces adresses sont représentées sous forme de deux adresses : une adresse absolue et une adresse relative. Ces adresses sont récupérées et traitées comme suit :

Tout d'abord, les informations de chemin sont extraites à partir du fichier NCD en utilisant "Directed Routing Constraints (DRC)" qui est un programme intégré dans l'outil FPGA_Editor. Ce programme permet à l'utilisateur d'obtenir les adresses de routage de n'importe quel signal sélectionné. Dans le but d'aboutir à la compatibilité de la relocalisation il faudrait uniformiser les chemins de routage entre les RPRs et la partie statique, tous les signaux entre les proxys et les modules statiques doivent donc être analysés.

La Figure 2.7 illustre l'intervention que l'utilisateur doit effectuer pour uniformiser le routage entre les proxys et les modules interfaçant une RPR. La Figure 2.7.a montre les positions de ces connexions suite à la première implémentation d'un système disposant de deux régions reconfigurables. Le placement des proxys est effectué d'une manière automatique suivant une stratégie déjà définie dans les outils Xilinx de P&R. Nous remarquons bien que les positions des proxys au niveau de chacune des deux RPRs sont différentes. Donc les chemins de routage issus des proxys vers la partie statique ne sont pas identiques. Dans ce cas les régions RPR1 et RPR2 ne sont pas compatibles.

Pour permettre l'application de la technique de la relocalisation de bitstream, l'utilisateur doit intervenir pour uniformiser les positions des proxys dans chaque RPR à partir d'un ensemble de postions des proxys de référence. En effet, le calcul des nouvelles positions s'effectue grâce à l'alignement des positions des Silces entre la position de référence et la position à déterminer grâce à des vecteurs de translation/projection. L'uniformisation des positions de proxy facilite par la suite l'uniformisation des chemins de routage des signaux. De même, l'uniformisation des chemins de routage des signaux d'une RPR donnée s'effectue à partir de l'ensemble des chenins de routage de références. En effet, le calcul des nouveaux chemins de routage s'effectue grâce à la différence d'adresse de routage entre le Switch (bloc de routage) de référence et le Switch à déterminer.

Dans la Figure 2.7.b, la région RPR1 est considérée comme région de référence. Les positions de ces proxys sont reproduites au niveau de la région RPR2 avec un certain vecteur de translation qui dépend de la taille du frame utilisée dans le circuit FPGA. Les positions des proxys de la région RPR1 sont extraites suite à la première implémentation. Ces positions sont utilisées pour recalculer les postions des proxys relatifs à la RPR2.

Nous citons par la suite quelques travaux qui visent à favoriser la technique de la relocalisation de bitstream avec le moindre gaspillage de ressources. Dans [45], les auteurs présentent une technique de conception qui sert à uniformiser les régions partiellement reconfigurables sur l'FPGA Xilinx-Virtex6.

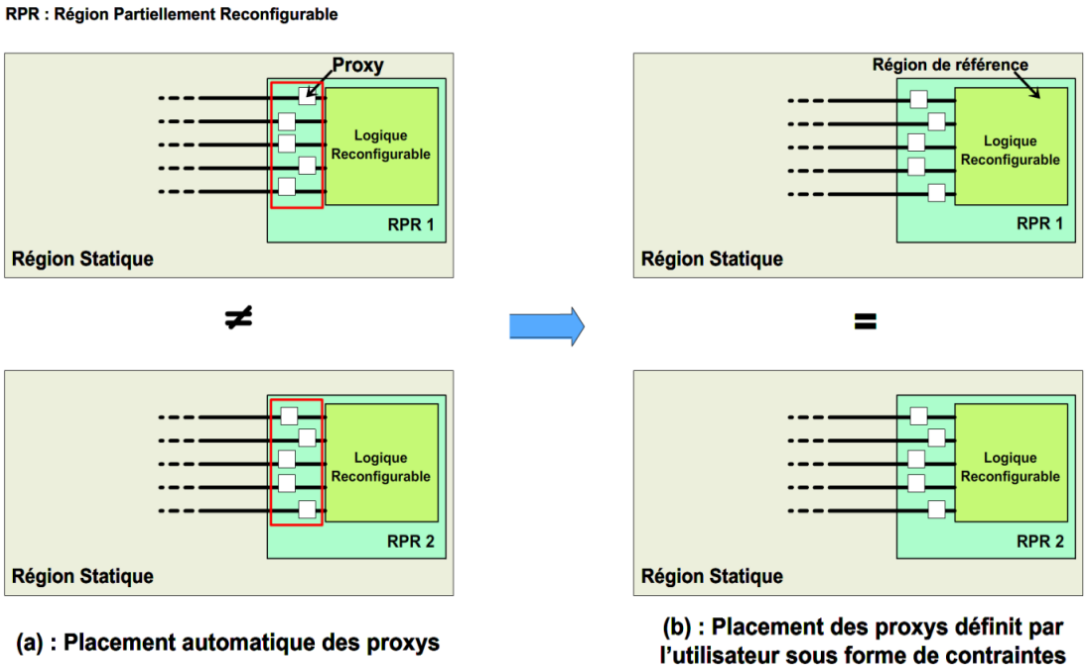


Figure 2.7 : Exemple de placement des proxys

En effet, leur technique permet de mettre en œuvre des modules partiellement reconfigurables de grande taille tout en combinant les régions reconfigurables voisines. Leur technique de relocalisation est basée sur les restrictions de placement des ressources reconfigurables, le placement des interfaces proxys et le routage des d'interconnexions. Les auteurs présentent aussi une solution qui vise à éviter le passage des signaux qui appartiennent à la partie statique à travers les régions dynamiquement reconfigurables lors du processus de placement et routage comme l'illustre la Figure 2.8.

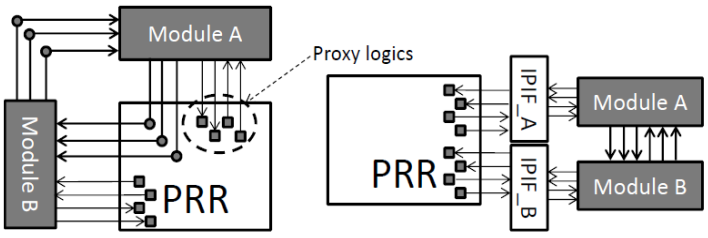


Figure 2.8 : Mise en œuvre des interfaces d'E/S

Leur méthodologie s'est basée sur l'instanciation des interfaces supplémentaires pour chaque région reconfigurable dans l'architecture d'un système donné. La taille de ces interfaces statiques pour une RPR est justement égale à la taille des proxys qu'elle comporte (égal au nombre des entrées/sorties du MPR) : chaque élément logique proxy (LUT) qui se trouve dans une RPR est connecté directement à un LUT se trouvant dans l'interface statique et qui est associée à cette région. Donc, un surcoût considérable mais négligeable pour appliquer la relocalisation de bitstream. Cette solution peut engendrer la réduction de performance du système dû à l'intervention de l'utilisateur dans la phase de placement et routage des signaux du système et perturber le processus d'implémentation de *NGDBUILD*. De plus, dans cette solution proposée l'allocation des interfaces statiques nécessite l'uniformisation de placement et routage des Proxys Logiques au sein de zone reconfigurable ce qui demande encore un effort en termes de temps de conception puisque ceci se fait de manière manuelle avec l'outil *FPGA_EDITOR*.

Dans [55] [56], la taille des interfaces statiques est divisée par deux. Etant donné qu'un LUT comporte deux sorties, il est possible d'associer deux proxys (deux LUTs) dans un seul LUT qui se trouve dans la partie statique. En fait, les auteurs utilisent le mécanisme de bus macro anciennement utilisée pour la mise en œuvre des systèmes reconfigurables dans les FPGAs Xilinx. Cette technique se manifeste par modifier la propriété à chaque zone reconfigurable sans utiliser les interfaces d'E/S. Dans [58], les auteurs proposent une nouvelle technique pour limiter la communication entre les ressources statiques et dynamiques. Leur technique s'est basée sur l'utilisation des macros pour éliminer définitivement le surcoût en LUTs d'interfaçage.

2.7 Synthèse bibliographique

Dans cette partie, nous avons présenté quelques architectures parallèles qui exploitent les fonctionnalités des FPGAs. Aucun de ces des travaux n'utilisent la fonctionnalité de la reconfiguration dynamique partielle, ce qui nous a motivé à concevoir des systèmes à base d'architectures régulières parallèles dynamiquement reconfigurables. Nous avons également mis l'accent sur la mise en œuvre du flot traditionnel de la RDP de Xilinx qui souffre du manque d'automatisation de ces différentes étapes. En effet, ce flot de conception s'effectue manuellement par les concepteurs et plus précisément pendant la phase de floorplanning qui requière des compétences techniques approfondies sur les structures des FPGAs [79] [80]. Les travaux présentés dans la section 2.2.3 se sont focalisés sur le développement d'algorithmes

qui visent l'automatisation de la phase de floorplanning mais malheureusement ces algorithmes ne favorisent pas l'application de la technique de la relocalisation de bitstreams qui vise à son tour l'augmentation des fonctionnalités d'un bitstream partiel. Les travaux déjà présentés se sont basés sur des algorithmes complexes qui consomment un temps d'exécution assez élevé par rapport à notre démarche pour la mise en ouvre de notre algorithme AFLORA. L'heuristique AFLORA est caractérisée par sa simplicité et son temps d'exécution réduit, il permet l'application de la technique de relocalisation de bitstream grâce à un floorplanning régulier. Notre algorithme est intégré dans le flot de conception automatisé ADForMe. Notre outil exploite différents paramètres architecturaux et technologiques pour générer un fichier de configuration (UCF) pour une architecture parallèle donnée.

Pour appliquer la technique de relocalisation de bitstream, nous optons pour une solution hybride qui consiste à employer les interfaces d'E/S se situant au dessus des zones reconfigurables dans la même région d'horloge en tenant compte de leurs positions sur les régions d'horloges supérieures. De plus, pour renforcer l'homogénéité des régions reconfigurables et éviter le routage des signaux à travers les régions reconfigurables, nous forçons les propriétés des régions partiellement reconfigurables pour qu'elles soient insensibles à l'algorithme de placement et routage de Xilinx (NGBUILD). Dans le but d'augmenter la productivité, nous avons conçu un flot de conception qui vise à automatiser la technique de relocalisation de bitstream puisqu'elle s'effectue manuellement dans les outils existants.

2.8 Conclusion

Dans ce chapitre nous avons présenté une étude détaillée sur les approches que nous allons entamer dans les chapitres suivants à savoir : l'automatisation totale du flot de la RDP-Xilinx grâce à l'automatisation de l'étape de floorplanning à travers l'algorithme AFLORA, la mise en œuvre d'un mécanisme d'automatisation de la technique de relocalisation de bitstream permettant la factorisation d'un bitstream partiel sur un ensemble de RPRs identiques, enfin la conception d'une architecture matérielle adéquate pour le broadcast/diffusion d'un bitstream factorisé. L'analyse effectuée dans ce chapitre a montré une forte complémentarité de ces différentes approches pour aboutir à la mise en place d'un flot de la RDPP ainsi que son application dans les systèmes parallèles dynamiquement reconfigurables.

CHAPITRE 3 : Méthode de Partitionnement Automatique de la Structure de l'FPGA pour le Broadcast de Bitstream Partiel

3.1 Introduction

De nos jours les plateformes FPGAs modernes sont suffisamment denses pour permettre la conception d'un système à base de multiprocesseurs sur une seule puce. Les architectures parallèles rentrent parfaitement dans le paradigme des systèmes embarqués à haute performance et s'appliquant grâce à l'assemblage d'un nombre d'unités de calculs (Processeur/IP) qui peuvent être homogènes comme les machines SMPD (Single Programme Multiple Data) [59]. D'autre part, en tirant profit de la fonction de reconfiguration dynamique partielle offerte par certains composants FPGA, ce type de système de calcul peut évoluer donnant naissance à des architectures parallèles reconfigurables dynamiquement. Cette architecture se caractérise par son changement de comportement en cours d'exécution ce qui lui permet de supporter de nombreuses applications à la fois grâce au partage de ressources à travers les régions reconfigurables au sein des unités de calcul. En effet, l'instanciation grandissante des unités de calcul, engendre relativement l'instanciation identique du nombre des régions partiellement reconfigurables. L'allocation de ces régions nécessite l'intervention manuelle du concepteur pour spécifier géométriquement leurs dimensions pour satisfaire les réquisitions en ressources. Ceci nécessite à son tour un effort important et aussi un temps de conception considérable.

Dans cette partie, nous nous intéressons à la conception d'un mécanisme de floorplanning automatisé sous forme d'un algorithme heuristique (AFLORA) qui permet l'amélioration de la productivité d'un concepteur. Ce mécanisme est basé sur la régularité de la structure FPGA, d'où des paramètres architecturaux et technologiques sont nécessaires pour sa mise en œuvre. Notre algorithme vise à établir tout d'abord une configuration commune à partir d'un ensemble de configurations (applications) en fonction de diverses fonctionnalités sur la même architecture. Ensuite, nous envisageons effectuer le floorplanning compatible à la relocalisation permettant la réutilisation d'un même bitstream pour l'ensemble des RPRs et assurer la RDPP basée sur le broadcast. Pour ce faire, trois étapes essentielles doivent être réalisées :

La première étape consiste à retrouver une configuration commune qui définit la taille de chaque RPR en fonction de l'ensemble de fonctionnalités (MPRs) que nous voulons intégrer dans l'architecture parallèle.

La deuxième étape consiste à chercher dans la structure du FPGA des régions identiques en termes de : **Volume** (disposition des blocs fonctionnels CLB, BRAM et DSP) et de **Forme géométrique** (permet d'avoir des bitstreams de même taille physique sur disque). Cela permet de satisfaire les réquisitions en ressources de la configuration commune et assurer le placement uniforme des RPRs. Le placement des RPRs s'effectue en deux phases: la première vise à repérer des sous-groupes de RPRs, considérés comme RPRs de référence, positionnées dans une section arbitraire de la structure du FPGA et la deuxième consiste à placer symétriquement le reste des RPRs en fonction des RPRs de référence. A ce stade, les informations de routage restent encore inconnues, donc il est impossible d'établir l'uniformisation des interfaces entre la partie statique et dynamique. En effet, une première implémentation du système permet la récupération des ces informations pour qu'elles soient traitées.

La dernière étape permet de récupérer à partir de la première implémentation les informations de routage des RPRs de référence pour les rétablir sur le reste des RPRs. Ceci permet à la fois d'avoir un placement et un routage des signaux de connexions communs entre les régions partiellement reconfigurables. Donc, avoir des bitstreams identiques à la fin du processus BitGen suite à la deuxième implémentation du système.

Nous nous intéressons dans ce chapitre aux deux premières étapes qui présentent la détermination de la configuration commune et notre mécanisme d'automatisation du floorplanning à travers l'algorithme AFLORA. Ces deux étapes sont assurées par notre flot de conception ADForMe.

3.2 Les Exigences et les spécifications de l'heuristique AFLORA

En se basant sur les plateformes FPGA qui supportent la technique de la RDP du constructeur Xilinx, nous nous sommes intéressés à l'automatisation de l'étape de placement physique des régions partiellement reconfigurables. En effet, nous avons développé un mécanisme automatique basé sur une heuristique bien déterminée qui consiste à effectuer la phase de floorplanning des régions partiellement reconfigurables. La particularité de cette heuristique est de garantir et de réaliser une allocation équitable des RPRs et les placer symétriquement sur la structure du composant FPGA. Elle s'applique sur les architectures parallèles comportant au moins deux régions dynamiquement reconfigurables compatibles avec la technique de la relocalisation de bitstream. Ce mécanisme a été intégré dans notre méthodologie d'automatisation du flot de la RDPP. La Figure 3.1 résume notre méthodologie

de partitionnement (ADForMe). Notre méthodologie est composée de trois phases essentielles capables de générer une description UCF compatible avec les outils Xilinx.

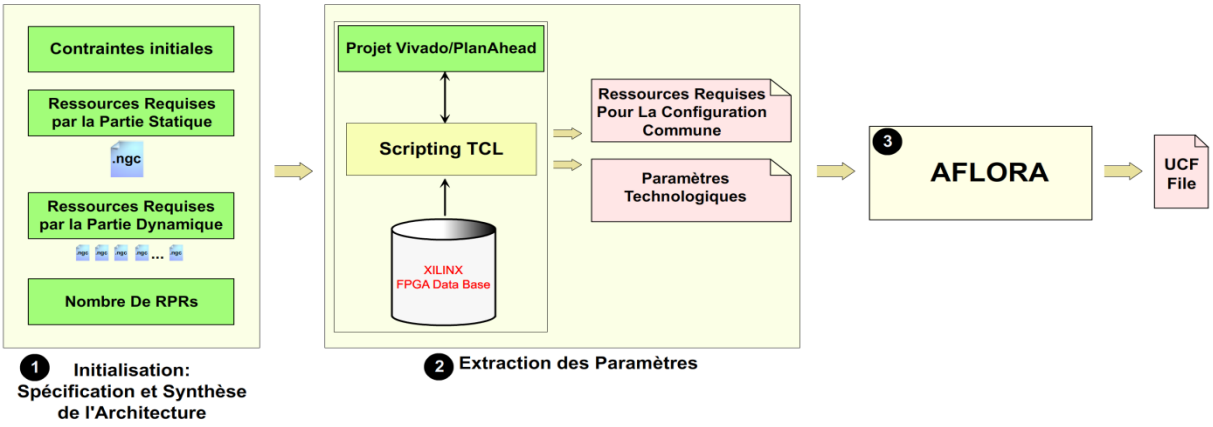


Figure 3.1 : Le Flot de Conception d'ADForMe

Tout d'abord une phase d'initialisation qui consiste à la création de différents projets correspondants à la partie statique ainsi qu'aux modules reconfigurables et de récupérer leurs résultats de synthèse en utilisant l'outil ISE-Xilinx. La seconde phase commence par la création d'un projet PlanAhead/Vivado et d'en extraire les paramètres architecturaux ainsi que l'extraction des paramètres technologiques à partir de la base de données FPGA Xilinx. La troisième phase consiste à récupérer ces différents paramètres dans le but d'établir automatiquement la phase de Floorplanning en se basant sur l'algorithme AFLORA. Nous détaillions ces phases dans les sections suivantes.

3.3 ADForMe

3.3.1 Intégration de la fonction de la RDPP

Nous considérons une architecture parallèle simplifiée telle qu'elle est présentée dans la Figure 3.2. Cette architecture comporte généralement un élément de contrôle global « NET_MP » composé d'une unité de calcul (UC) maître qui gère le contrôle d'accès mémoire RAM/DMA qui contient les données à traiter par le réseau de multiprocesseurs esclaves. Le réseau multiprocesseur est un ensemble de sous-réseaux de multiprocesseurs (MP) connectés à la mémoire RAM/DMA. La mémoire cache (L1) est utilisée pour un effectuer des accès beaucoup plus rapide que les accès à la mémoire globale. Chaque sous-réseau de multiprocesseurs est composée par un ensemble d'UCs qui partagent une mémoire partagée qui peut être à son tour une mémoire cache de niveau 2 (L2). Nous considérons que l'ensemble des unités de calculs (UC) s'exécutent en parallèle et englobent un ensemble de

composants dédiés pour effectuer des fonctionnalités diverses : contrôle, mémoire, accélérateurs matériels (IP : Intellectual Propriety), etc.

Nous nous intéressons dans ce cadre aux composants IPs au sein de chaque processeur pour intégrer la fonction de la RDP. En effet, nous considérons que tous les IPs possèdent la même interface de communication au niveau de chaque unité de calcul.

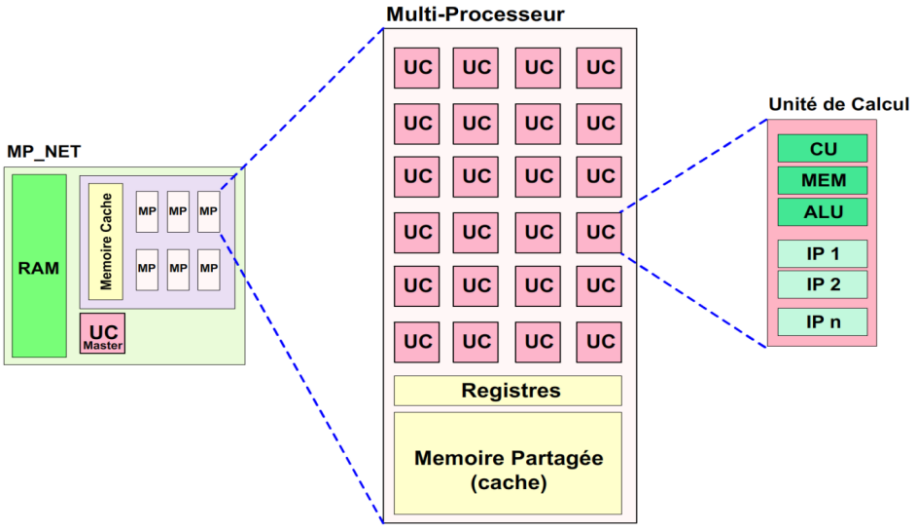


Figure 3.2 : Vue simplifiée de l'architecture parallèle ciblée

Notre approche se base premièrement sur la spécification de la nature des IPs (statique ou dynamique) au niveau de chaque UC. En effet, nous spécifions un IP (IP_{dyn}) comme étant un IP dynamiquement reconfigurable avec des IPs statiques. La sollicitation de chaque type de ces IPs se diffère au niveau de chargement du bitstream partiel. En effet, l'exécution d'un IP_{dyn} nécessite le chargement de son bitstream relatif de la mémoire des bitstreams vers son RPR correspondante pour qu'il soit exécuté enfin grâce à une instruction spécifique. D'autre part l'exécution d'un IP statique se fait seulement grâce à son instruction spécifique sans chargement intermédiaire. De plus, l'instanciation d'un IP_{dyn} sert à partager l'espace physique sur FPGA à travers une RPR. Théoriquement, une infinité d'IPs (MPRs) peuvent partager exclusivement une RPR pour réaliser une infinité de fonctionnalités. Le partage de l'espace physique mène à son tour à partager une instruction d'exécution unique entre tous les IPs venant s'exécuter dans la même RPR, d'où une exécution simple à gérer. Par contre, quand il s'agit d'une UC comportant plusieurs fonctionnalités, tous les IPs statiques doivent être instanciés séparément et disposant chacun de son instruction d'exécution. D'où plus de flexibilité est assurée et un gain en surface et en consommation d'énergie si nous comparons ces deux types d'IPs. Notre approche ne paraît pas très efficace quand il s'agit d'un nombre

limité de fonctionnalités dans une seule UC, par contre, l'efficacité d'un tel système peut prouver son existence dans le cas contraire. A cet avantage, il n'est pas souhaitable de rendre dynamiquement reconfigurable tous les IPs se trouvant dans une UC car les IPs se diffèrent par leur temps exécution. C'est-à-dire que c'est inutile de spécifier un IP_{dyn} qui s'exécute pendant une durée inferieure que le temps de chargement de son bitstream depuis la mémoire vers sa RPR relative. Pour cela, les deux types d'IPs peuvent exister en même temps au sein d'une UC.

La Figure 3.3 montre un exemple d'architecture simplifiée que nous visions à travers notre approche. Le mécanisme de la RDPP est assuré par un IP dédié à cet effet ($IP_{reconfig}$). Son architecture sera détaillée dans le Chapitre 4. Il effectue un ensemble d'étapes pour assurer la reconfiguration de toutes les régions avec le même bitstream. L'$IP_{reconfig}$ est centralisé et peut être géré par une UC maître. L'$IP_{reconfig}$ est directement lié à l'ICAP de l'FPGA. Une seule RPR (IP_{dyn}) est spécifiée au sein de chaque UC tandis que le reste des IPs sont statiques. L'ensemble des UCs peuvent être connectées sur un réseau de communication qui peut être régulier ou irrégulier.

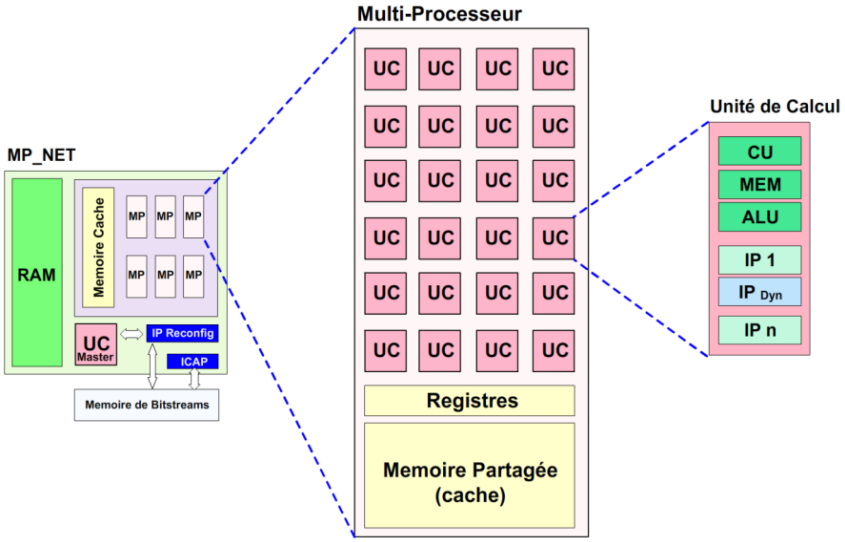


Figure 3.3 : Architecture parallèle proposée

3.3.2 Phase 1 : Initialisation

Cette étape se manifeste par la collecte des descriptions nécessaires pour l'initialisation du processus d'automatisation. Cette phase s'exécute suite à la phase de spécification des régions reconfigurables dans l'architecture parallèle puis sa synthèse. De même cette phase prend en compte le résultat de synthèse des différents MPRs. En fait, cette phase consiste à récupérer

les éléments générés par la synthèse de notre système reconfigurable suivant le flot de la reconfiguration dynamique partielle Xilinx. En effet, Cette étape prend en compte les préplacements du système (Mappage de mémoires, l'assignement des IOs,…), ensuite la description NGC (Native Generic Curcuit) relative à la partie statique TOP, enfin les fichiers NGC des modules reconfigurables synthétisés. Un autre type de paramètre est encore nécessaire pour notre méthodologie qui est le paramètre applicatif qui indique le nombre de RPRs « ***Net_D*** ». Ces différents types de paramètres constituent les entrées de base pour la création d'un projet PlanAhead relatif au système dynamiquement reconfigurable.

3.3.3 Phase 2: Extraction et définition des paramètres architecturaux et technologiques

Suite à la phase d'initialisation, l'extraction et la définition des paramètres architecturaux et technologiques devient possible suite à la création du projet PlanAhead.

3.3.3.1 La Création du projet PlanAhead

Afin d'extraire les ressources nécessaires du système à implanter et les paramètres technologiques de l'FPGA utilisé, un projet PlanAhead doit être créé pour permettre l'analyse architecturale du système et les propriétés de la plateforme FPGA. L'analyse architecturale permet de déterminer les besoins en ressources du système en termes de CLB, RAMB et DSP correspondant à la partie statique et aux modules partiellement reconfigurables. Le projet est créé en mode TCL en se basant sur la commande en ligne suivant:

PlanAhead –mode tcl –source cmd1.tcl

Le fichier script TCL appelé ***cmd1.tcl*** contient dans une première partie, la liste des commandes nécessaires pour permettre le choix du FPGA à utiliser (Device, Package, Speed, …), le nom du projet PlanAhead et sa localisation. Ensuite, nous fixons les propriétés nécessaires du projet pour qu'il puisse supporter la fonction de la RDP en utilisant la propriété ***is_partiel_reconfig*** qui doit être égale à **true**. Nous spécifions aussi le nom de la configuration et le nom des contraintes initiales grâce à l'importation des fichiers NGC (relatif à la partie statique) et UCF initial. Nous fixons la propriété TOP à la description statique (NGC) du système. Nous importons par la suite, la liste des fichiers NGC relatifs aux MPRs. Le projet devient actif et accessible à travers le ***netlist*** principal (TOP.ngc) grâce à la

commande TCL ***link_design***. List1 résume l'ensemble de ces étapes intégrées dans la première partie du script ***cmd1.tcl.***

```
1-  create_project -part xc7vx485t-3ffg1761 <chemin et nom du projet> -force      // création du projet (nom, FPGA utilisé)
2-  set_property design_mode GateLvl [current_fileset]                           // Mode de conception (niveau porte logique)
3-  set_property name config_B [current_run]                                     // spécifier le nom de la première configuration
4-  set_property is_partial_reconfig true [current_project]                      // activer la fonction de RDP pour ce projet
5-  set projDir [file dirname [info script]]                                     // récupérer le chemin absolu du projet
6-  import_files -norecurse [glob $projDir/*.ngc]          // importer les fichiers NGC de la partie statique (TOP, IP_Core, ….)
7-  set_property top TOP [current_fileset]       // spécifier le fichier TOP.NGC comme conception hiérarchique de premier niveau
8-  add_files -fileset constrs_1 -norecurse <fichier.ucf>          // importer et ajout du fichier de contraintes initial dans le projet
9-  import_files -fileset constrs_1 <fichier.ucf>
10- link_design -name netlist_1                                                 // activer le netlist principal
```

List1 : Création du projet PlanAhead

Étant donné que l'architecture parallèle que nous visons favorise l'exécution de la même tâche reconfigurable dans chaque UC, une configuration unique relative à un seul MPR est alors exécutée au sein de chacune RPR. Suivant le flot de la RDP intégré dans l'outil PlanAhead, la mise en œuvre d'une configuration nécessite au moins l'association d'un MPR à une RPR, donc, dans notre cas, l'association s'effectue entre un seul MPR et tout l'ensemble des RPRs. Cela signifie que le nombre des configurations est égal au nombre des MPRs. Nous rajoutons une configuration initiale en utilisant un MPR vide appelé BLANK. Cette configuration est utilisée avant l'appel de la fonction de la RDP car elle vise à économiser la consommation d'énergie puisque l'ensemble des RPRs ne consommeront initialement que l'énergie statique du FPGA.

Suivant la technique de la RDP, une RPR donnée a une forme physique rectangulaire qui peut supporter un nombre illimité MPRs. Nous rappelons qu'un PBLOCK est une abstraction d'un RPR et utilisé pour spécifier l'emplacement d'une région physique donnée. Dans notre cas, le nombre de PBLOCKs est égal à la valeur du paramètre Net_D puisque nous prenons une hypothèse qu'une UC contient une seule RPR. Initialement, dans le projet de PlanAhead, une RPR est représentée comme une entité boite noire (BLACKBOX). Une boite noire est un module déclaré au cours de la phase de la conception qui ne contient aucune description matérielle. Une boite noire est considérée comme un conteneur pour un ensemble de RPRs. Suivant notre démarche d'automatisation, nous devons récupérer la liste des boites noires du système pour qu'elles soient associées chacune à un PBLOCK et un MPR. La description List2 montre une façon de récupérer la liste des BLACKBOX d'un projet donné. L'ensemble des boites noires est généré dans un fichier spécifique que nous allons utiliser dans l'étape d'automatisation du floorplanning.

```
1-  set  Net_D < Nombre des RPRs>        //déclaration du nombre des RPRs
2-  set list_bbx {}                      // liste des boites noires
3-  set BBX {}                           //variable boite noire
4-  set file [open "BBX.txt" w]  //création et ouverture d'un fichier en mode écriture
5-  for {set i 0} {$i<$Net_D} {incr i} {
6-  BBX [lindex [get_cells -hier -filter {IS_BLACKBOX}] $i]   //récupérer les boites noires une par une de la conception TOP
7-  lappend list_bbx $BBX                 // mise à jour de la liste des boites noires
8-  puts $file $BBX                       // mise à jour du fichier de sortie
9-  }
10-  close $file
```

List2 : Récupération des modules BLACKBOX

L'étape suivante consiste à créer les modules partiellement reconfigurables au sein de chaque RPR (boite noire). Cette étape s'effectue en fonction du nombre des RPRs (NET_D) et le nombre des MPRs. Leur création s'effectue en deux phases. La première consiste à créer le MPR dans l'arborescence là où une boite noire se situe. En effet, chaque boite noire du système constituera elle-même une arborescence de MPRs relative à chaque configuration. La deuxième consiste à associer à ce MPR le fichier NGC convenable qui contient sa description matérielle. Les fichiers NGC relatifs aux MPRs sont situés dans le dossier **.\MPR** qui se trouve dans le même répertoire que le projet PlanAhead. Au fur et à mesure, nous générons les ressources requises par chaque MPR en termes de CLB, BRAM et DSP. Ces ressources sont générées séparément dans des fichiers de trace qui seront utilisés ultérieurement pendant la phase d'automatisation du floorplanning. La description List3 illustre les étapes de cette mise en œuvre.

```
// Créer les MPRs
1-  for {set i 0} {$i<$Net_D} {incr i} {                              // Nombre des MPRs
2-  for {set j 0} {$j<[llength [glob $projDir/MPR/*.ngc]]} {incr j} {        //Nombre des MPRs
3-  create_reconfig_module -name RR$i$j -cell [lindex $list_bbx  $i] –force            //création d'un MPR dans une boite noire
4-  }
5-  }

// Associer à chaque MPR le fichier NGC
1-  for {set i 0} {$i<$Net_D} {incr i} {
2-  for {set j 0} {$j<[llength [glob $projDir/MPR/*.ngc]]} {incr j} {
3-  set RPR [get_filesets *#RR$i$j]                                  //récupérer le non du MPR déjà créé
4-  set_property edif_top_file [lindex [glob $projDir/MPR/*.ngc] $j] [get_filesets $RPR ]
5-  import_files -fileset $RPR -norecurse [lindex [glob $projDir/MPR/*.ngc] $j] -force
6-  }
7-  save_constraints -force
8-  }

// Générer les ressources requises par chaque MPR
1-  for {set j 0} {$j<[llength [glob $projDir/MPR/*.ngc]]} {incr j} {
2-  for {set i 0} {$i<$Net_D} {incr i} {
3-  load_reconfig_modules -reconfig_modules [lindex $list_bbx  $i]:RR$i$j //activer un MPR
4-  save_constraints -force
5-  }
6-  report_stats -all  -tables   physicalResource  -pblock  [lindex [get_pblocks] 0] -file RPRs_reports($j).txt            //générer le rapport
7-  }
```

List3 : Création des modules partiellement reconfigurables

Finalement, nous créons les différentes configurations du système sachant que nous sommes menés à spécifier un MPR pour chaque configuration, donc un total de RPRs égale à Net_D pour chaque configuration. La liste des configurations peut être visible dans le dossier **Desing.Runs** créé automatiquement lors de la création du projet. Comme nous avons indiqué précédemment, nous initialisons notre système avec une configuration Blank (config_B) comme décrit dans List1. La mise en œuvre de cette configuration initiale nécessite la création d'un MPR vide (Blank) dans chaque arborescence de RPR. Chaque MPR Blank ne nécessite pas une description NGC particulière. La description TCL dans la List4 illustre ces différentes phases.

```
// Créer les MPRs Blank
1-  for {set i 0} {$i<$Net_D} {incr i} {
2-  create_reconfig_module -name blank -cell [lindex $list_bbx  $i] -blackbox
3-  save_constraints -force
4-  }
// Mettre le MPR Blank dans chaque arborescence de RPR comme MPR actif
1-  for {set i 0} {$i<$Net_D} {incr i} {
2-  load_reconfig_modules -reconfig_modules [lindex $list_bbx  $i]:blank
3-  save_constraints -force
4-  }
// Associer les MPR Blank à la configuration initiale config_B
1-  for {set i 0} {$i<$Net_D} {incr i} {
2-  config_partition -run config_B -cell [lindex $list_bbx  $i] -reconfig_module blank -implement
3-  save_constraints -force
4-  }
// Construire les configurations du système en fonction de chaque MPR de chaque //arborescence de RPR
1-  for {set j 0} {$j<[llength [glob $projDir/MPR/*.ngc]]} {incr j} {
2-  create_run config_$j -flow {ISE 14} -strategy my_st
3-  config_partition -run config_$j -implement
4-  for {set i 0} {$i<$Net_D} {incr i} {
5-  config_partition -run config_$j -cell [lindex $list_bbx  $i] -reconfig_module RR$i$j -implement
6-  }
7-  }
```

List4 : Mise en œuvre des configurations du système

3.3.3.2 Extraction des paramètres architecturaux du système

Une fois les différentes configurations sont créées, nous procédons par l'analyse de l'architecture multiprocesseur que nous voulons implémenter. Notre première étape se manifeste par la détermination des ressources requises par la partie statique et les différents modules reconfigurables. Toutes ces tâches sont élaborées de manière automatique grâce à un ensemble de commandes TCL dans le script ***cmd1.tcl***. Ces commandes génèrent des résultats sous forme de fichiers. La Figure 3.4 illustre les différentes étapes qui conduisent à la génération des différents fichiers de traces.

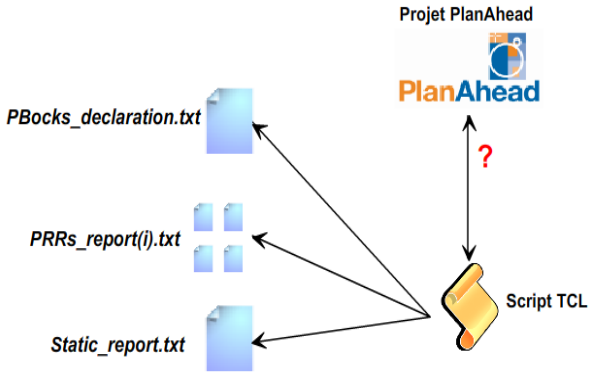


Figure 3.4 : Génération des traces visant la détermination des ressources requises par le système

Nous citons les principaux fichiers générés : Le fichier *static_report.txt* qui comporte les différentes ressources requises par la partie statique. Sa génération est effectuée suite à l'exécution de la commande TCL :

Report_state –all –file static_report.txt

L'ensemble des fichiers *RPRs_reports(i).txt* comporte les ressources requises par les MPRs. Cet ensemble de fichiers est généré à partir de la List3 présentée précédemment. Le fichier *PBLOCKs_declaration.txt* contient les noms des PBlocks créés avec leurs régions reconfigurables associées. Le contenu de ce fichier servira au traçage des RPRs dans la phase de floorplanning automatique, c'est-à-dire une source de mappage entre PRP et PBlock. La création de ce fichier s'effectue comme illustré dans la List5.

```
1- set file [open " PBLOCKs_declaration.txt " w]
2- set pb_count [llength $list_bbx]  //récupérer le nombre des RPRs
3- for {set i 0} {$i<$pb_count} {incr i} {
4- puts $file "[lindex $list_bbx  $i] [get_property PBLOCK [lindex $list_bbx  $i]]" //récupérer chaque PBlock associé à chaque RPR
5- }
6- close $file
```

List5 : Génération du fichier de mappage

3.3.3.3 Extraction des paramètres technologiques

Les paramètres technologiques sont des paramètres qui caractérisent la structure de la plateforme FPGA. La collecte de ce type d'informations se traduit par la mise en œuvre d'un ensemble de requêtes à la base de données FPGA Xilinx. L'extraction de ces paramètres se manifeste par l'exploration de la structure de la plateforme FPGA et les des ressources disponibles utilisables par le concepteur pour lui permettre l'implantation de son système reconfigurable. En effet, le concepteur a besoin d'identifier les types de ressources et leurs

emplacements pour effectuer l'allocation des régions partiellement reconfigurables. Pour ceci, nous avons exploité le langage TCL Xilinx pour récupérer tout l'ensemble de ces paramètres. Les résultats de ces requêtes sont générés sous forme d'une structure de données en listes. Ces listes sont stockées dans des fichiers d'une manière organisée dans le but de classifier les différents types de paramètres technologiques. Ce mécanisme est décrit dans la Figure 3.5. Ces traces comportent des résultats des requêtes qui seront utilisés par notre méthodologie AFLORA.

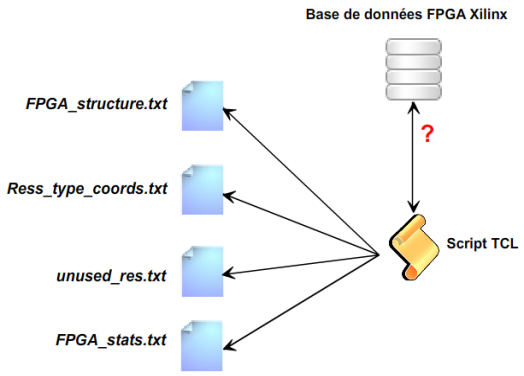


Figure 3.5 : Génération des traces visant à identifier les paramètres technologiques

La structure de l'FPGA utilisé est décrite dans le fichier *FPGA_structure.txt.* Les FPGAs Xilinx présentent une structure régulière, ils sont divisés en plusieurs colonnes (C) qui peuvent être symétriques ou non. Les colonnes sont composées par un ensemble de lignes verticales hétérogènes de tuiles (en anglais : Tile). Dans la Figure 3.6, les lignes verticales Tile_CLB, Tile_RAMB et Tile_DSP sont représentées respectivement en bleu, rose et vert.

Les colonnes sont également divisées d'une manière équitable en régions d'horloge (CR). Chaque CR est définie par son emplacement suivant les coordonnées XY et organisée comme un ensemble de sous-lignes verticales (Frames).

Une Frame est définie comme un ensemble de tuiles, caractérisée par un type et une taille fixe. La taille d'une Frame diffère d'un FPGA à l'autre en fonction de la technique de référence correspondante. Avec une telle structure, il est possible de déterminer l'ensemble des ressources disponibles dans l'FPGA en termes de tuiles (lieux et types) tout en analysant une seule ligne horizontale. Pour ce faire, il est important de calculer la largeur totale de l'FPGA qui est égale à la somme des largeurs des colonnes. Ensuite, en fonction de la taille et du type des différentes Frames et le nombre de CR, le total des ressources disponibles peut être calculé.

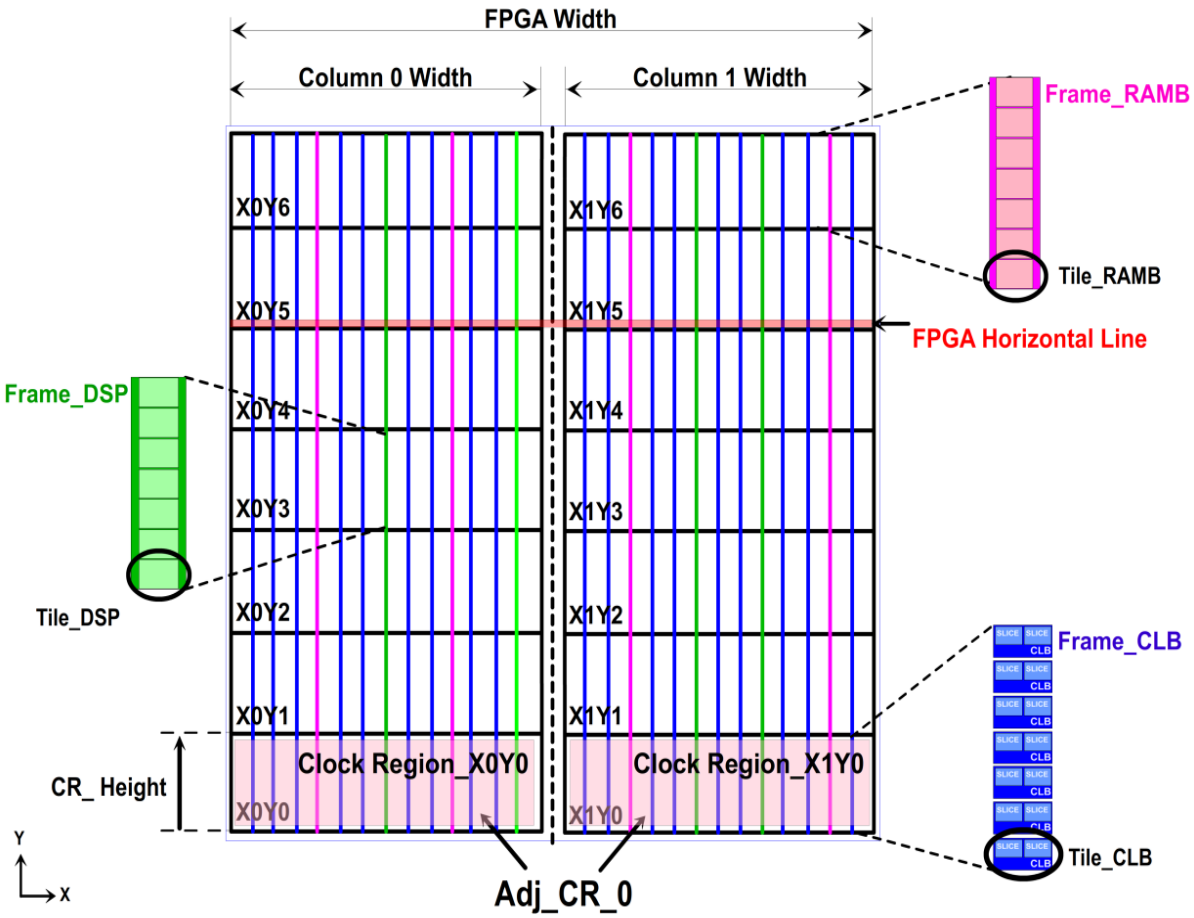


Figure 3.6 : La structure des FPGAs Xilinx

Une commande TCL peut générer un rapport qui contient les ressources disponibles d'un FPGA donné qui est :

Report_property [get_parts <référence FPGA>].

Les commandes TCL qui permettent la génération du fichier de traces *FPGA_structure.txt* contenant tous ces paramètres technologiques sont décrites dans List6.

```
// Vérifier la hauteur du FPGA : récupèrer la taille de chaque ligne verticale de l'FPGA
1-  set siteslen [llength [get_sites *X0Y* -filter {SITE_TYPE=~SLICE*}]] //nombre de SLICE sur la ligne X0
2-  set nbr_dsp [llength [get_sites *X*Y0 -filter {SITE_TYPE=~DSP* }]] //nombre de DSP sur la ligne X0
3-  set nbr_bram [llength [get_sites *X*Y0 -filter {SITE_TYPE=~RAMB* }]] //nombre de BRAM sur la ligne X0
4-  set compo_ROW_list {} // le vecteur des hauteurs
5-  set var 0

6-  for {set i 0} {$i<$siteslen} {incr i} {
7-  if { [expr $i%5] == 0 } { incr $var} // hauteur d'une BRAM équivalent SLICE
// récupérer l'occurrence de chaque composant en ligne horizontale
8-  set sites_SLICE  [llength [get_sites *X*Y${i} -filter {SITE_TYPE=~SLICE* }]]
9-  set sitesDSP  [llength [get_sites *X*Y${var} -filter {SITE_TYPE=~DSP*}]]
10- set sitesBRAM  [llength [get_sites *X*Y${var} -filter {SITE_TYPE=~RAMB*}]]
11- if {$sitesBRAM > $nbr_bram} { set sitesBRAM $nbr_bram}
12- if {$sitesDSP > $nbr_dsp} { set sitesDSP $nbr_dsp}
13- set  sites [ expr $sites_SLICE + $sitesDSP + $sitesBRAM] //taille de chaque ligne horizontale
14- lappend compo_ROW_list $sites
15- }
```

```
//-------------------------------------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------------------------------
1- set file [open " FPGA_structure.txt " w]
// longueur de la Frame CLB
2- foreach region [get_clock_regions X0Y0] { puts $file "[llength [get_sites -filter "CLOCK_REGION==$region" SLICE_X0Y*]]"}
// longueur de la Frame BRAM
3- foreach region [get_clock_regions X0Y0] {puts $file "[llength [get_sites -filter "CLOCK_REGION==$region" RAMB36_X0Y*]]"}
// longueur de la Frame DSP
4- foreach region [get_clock_regions X0Y0] { puts $file "[llength [get_sites -filter "CLOCK_REGION==$region" DSP48_X0Y*]]"}

5- puts $file "[llength [get_sites *X*Y0 -filter {SITE_TYPE=~SLICE*}]]" //largeur de l'FPGA
6- puts $file "[llength [$compo_ROW_list]]" // hauteur de l'FPGA
7- puts $file "[llength [get_clock_regions]]" // nombre des régions d'horloges
// déterminer le nombre des colonnes
8- set fpga_cols 0
9- for {set i 0} {$i< 10} {incr i} {
10- if {[llength [get_clock_regions X${i}*]] != 0 } {incr fpga_cols}
11- }
// récupérer la largeur de chaque colonne
12- puts $file $fpga_cols
13- for {set i 0} {$i< $fpga_cols} {incr i} {
14- set region [get_clock_regions X${i}Y0]
15- puts $file "[llength [ get_sites  -filter "CLOCK_REGION==$region" SLICE_X*Y0]]"
16- }
17- close $file
```

List6 : Mise en œuvre des commandes TCL visant la récupération des paramètres technologiques

Dans le but de récupérer la composition de chaque ligne horizontale, la description List7 permet de récupérer les ressources en termes de **Nom, coordonnées** et **Type** dans le fichier *Ress_type_coords.txt*. Etant donné que la plateforme FPGA est symétrique en termes de régions d'horloges sur la même colonne, alors l'identification des ressources disponibles sur une ligne horizontale (par exemple la ligne X*Y0) revient à identifier le type et le nom de chaque composant en colonne. La Figure 3.7 illustre le contenu du fichier *Ress_type_coords.txt.*

```
1- set file [open "Ress_type_coords.dat" w]
2- set compo_list_name {} // vecteur des noms des composants
3- set compo_list_type {} // vecteur des types des composants
4- set siteslen [llength [get_sites *X*Y0 -filter {SITE_TYPE=~RAMB* || SITE_TYPE=~SLICE* || SITE_TYPE=~DSP*}]]

5- for {set i 0} {$i<$siteslen} {incr i} {
6- set sites  [lindex [get_sites *X*Y0 -filter {SITE_TYPE=~RAMB* || SITE_TYPE=~SLICE* || SITE_TYPE=~DSP*}] $i ]
7- set prop [get_property NAME  $sites] // récupérer les noms de composants
8- lappend compo_list_name $sites // mise à jour du vecteur (noms)
9- set prop1 [get_property SITE_TYPE $sites] // récupérer les types de composants
10- lappend compo_list_type $sites // mise à jour du vecteur (types)
11- puts $file "$prop $prop1"
12- }
13- close $file
```

List7 : Mise en œuvre des commandes TCL visant la récupération de la composition de chaque ligne horizontale

```
cmd.tcl | mapping.txt | BBX_bis.txt | first
1  SLICE_X0Y0 SLICEL
2  SLICE_X1Y0 SLICEL
3  SLICE_X3Y0 SLICEL
4  SLICE_X2Y0 SLICEM
5  SLICE_X4Y0 SLICEL
6  SLICE_X5Y0 SLICEL
7  SLICE_X7Y0 SLICEL
8  SLICE_X6Y0 SLICEM
9  RAMB36_X0Y0 RAMBFIFO36E1
10 SLICE_X9Y0 SLICEL
11 SLICE_X8Y0 SLICEL
12 SLICE_X10Y0 SLICEM
13 SLICE_X11Y0 SLICEL
14 SLICE_X13Y0 SLICEL
15 SLICE_X12Y0 SLICEM
16 SLICE_X14Y0 SLICEM
17 SLICE_X15Y0 SLICEL
18 SLICE_X17Y0 SLICEL
19 SLICE_X16Y0 SLICEL
20 RAMB36_X1Y0 RAMBFIFO36E1
21 SLICE_X19Y0 SLICEL
22 SLICE_X18Y0 SLICEM
23 SLICE_X20Y0 SLICEM
24 SLICE_X21Y0 SLICEL
25 DSP48_X0Y0 DSP48E1
26 SLICE_X22Y0 SLICEM
```

Figure 3.7 : Résultat de la requête d'identification des composants se trouvant sur une même ligne horizontale

Dans certains FPGAs, des régions inutilisées peuvent se manifester sur la puce, ce qui est considéré comme une discontinuité dans la structure régulière du FPGA. La Figure 3.8 montre un exemple de ces régions inutilisées dans le dispositif Xilinx-Virtex7. Cela conduit à envisager une nouvelle contrainte pour la phase de floorplanning automatique. En se référant à l'exemple de dispositif Virtex7, notre floorplanning doit éviter l'utilisation des intervalles {[B1], [B2], [B3]} suivant l'axe des X. Cet ensemble d'intervalles est généré dans un fichier intitulé *unused_res.txt* suite à l'exécution d'un algorithme qui consiste à parcourir la plateforme FPGA pour détecter les tuiles non référenciées (coordonnées).

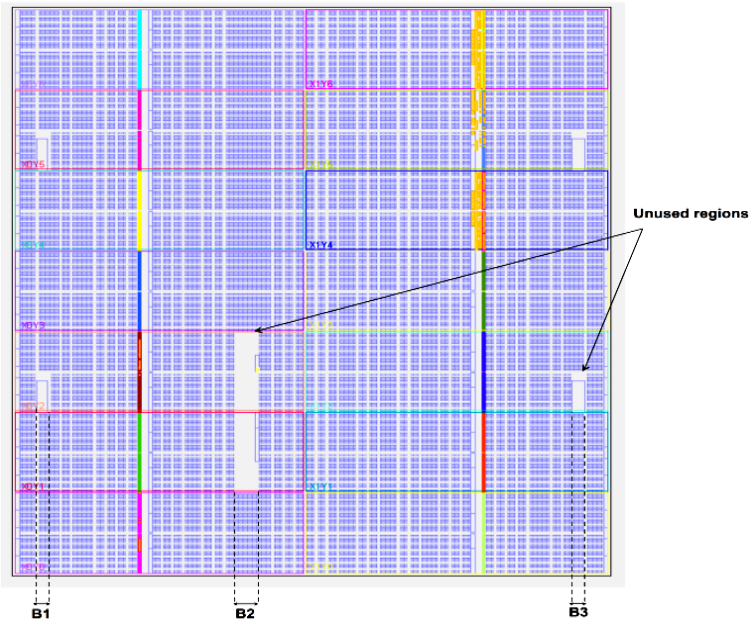


Figure 3.8 : Représentation des zones non exploitables par les concepteurs

La description List8 décrit les commandes TCL correspondantes à détermination des SLICEs non référenciés dans la structure FPGA.

```
1- set nbr_slice [llength [get_sites *X*Y0 -filter {SITE_TYPE=~SLICE* }]]      // larger du FPGA
2- set max_H "[llength [$compo_ROW_list]]"                                    // hauteur du FPGA
3- set unused_SLICE {}                                                        // vecteur des références non utilisées
4- for {set i 0} {$i<$max_Slice} {incr i} {
5- for {set j 0} {$j<$nbr_slice} {incr j} {
6- if {[get_sites SLICE_X${j}Y${i}] == "" } {
7- lappend unused_SLICE SLICE_X${j}Y${i}
8- puts $file "$j $i"
9- }}}
```

List8 : Récupération des références des SLICEs introuvables

Les fichiers de traces générés sont organisés selon une structure bien déterminée et exploitables par notre mécanisme de floorplanning automatique AFLORA. Ces fichiers seront traités par la suite de manière logicielle en utilisant le langage Orienté Objet (JAVA) qui permet de bien classifier ces différents paramètres ainsi que la construction et référenciations des différentes PBLOCKs er RPRs (fichier de mappage *PBLOCKs_declaration.txt*) sous forme de contraintes utilisateur.

3.3.3.4 Classification des paramètres architecturaux et technologiques

Avant de décrire les détails de notre algorithme proposé, il est essentiel de définir et classifier les paramètres technologiques liés à la structure du FPGA ainsi que les paramètres de l'architecture visée.

Les paramètres technologiques :

Nous énumérons dans le tableau 3.1 les différents paramètres technologiques extraits depuis les traces générées. En se référant à la Figure 3.6, ces paramètres sont présentés comme suit.

Désignation	Définition
C	Le Nombre des colonnes de l'FPGA utilisé
nbCR	Le Nombre total des régions d'horloges
Frame_C	La taille de la Frame CLB
Frame_D	La taille de la Frame DSP
Frame_R	La taille de la Frame BRAM
	Un vecteur de translation qui comporte trois éléments qui sont les

V_{trans}	différentes tailles de Frames: $V_{trans} = \begin{pmatrix} Frame\_C \\ Frame\_R \\ Frame\_D \end{pmatrix}$
X_max	La largeur de l'FPGA en termes de tuiles hétérogènes
CR_Height	Un paramètre géométrique qui représente la hauteur d'une région d'horloge en fonction de chaque type de ressource (CLB, BRAM et DSP)
Unsd_Range	L'ensemble des intervalles suivant l'axe des X des ressources non référenciés

Tableau 3.1 : Les paramètres technologiques

Nous définissons le terme Adj_CR toutes régions d'horloges adjacentes suivant l'axe des X. Dans notre cas Adj_CR = 7 (Adj_CR_0, Adj_CR_1, Adj_CR_0, … Adj_CR_6). Puisque les Adj_CRs sont toutes symétriques suivant l'axe des Y, nous choisissons arbitrairement Adj_CR_0 qui sera considérée comme Adj_CR de référence de notre circuit FPGA. Ainsi, le reste des Adj_CRs peuvent être identifiées à partir de Adj_CR_0 à l'aide des paramètres X_max et le vecteur V_{trans} suivant l'axe des Y.

Comme une hypothèse globale nous considérons qu'une RPR ne peut pas dépasser une région d'horloge donnée en cohérence avec l'objectif de construire une machine reconfigurable parallèle reposant sur l'ensemble des IPs de Fine ou moyen grain (modules). En effet, l'utilisation des IPs à gros grains ne correspond pas à la structure régulière du FPGA et permet de réduire les performances de notre machine en termes de niveau de parallélisme. Par conséquent, l'exploration d'un seul Adj_CR est suffisant pour trouver les emplacements disponibles des RPRs répondant aux contraintes de nos MPRs (emplacement, le type et l'orientation des tuiles nécessaires pour mettre en œuvre les modules). Le résultat de cette exploration sera reproduit à partir de l'Adj_CR de référence aux Adj_CRs restants, ce qui réduit la complexité de l'algorithme de floorplanning.

Les paramètres architecturaux :

Comme les paramètres technologiques, le Tableau 3.2 présente les différents paramètres architecturaux de notre méthodologie.

Désignation	Définition
AR	Un vecteur qui comporte trois éléments et qui représente toutes les ressources disponibles sur l'FPGA en termes de CLB, BRAM et DSP

SR	Un vecteur qui comporte trois éléments et qui représente les ressources utilisées par la partie statique en termes de CLB, BRAM et DSP
DR	Un vecteur qui comporte trois éléments et qui représente les ressources utilisées par la partie dynamique en termes de CLB, BRAM et DSP
Net_D	La dimension du réseau MULTI-RPRS : Net_D = NX_Slaves x NY_Slaves.
MPR_C	Un vecteur qui comporte trois éléments et qui représente les ressources requises par le MPR commun pour une seule RPR.
TR	Un vecteur qui comporte trois éléments et qui représente les ressources requises par tous les MPRs : TR = Net_D x RPR_C
F	Le paramètre de faisabilité.

Tableau 3.2 : Les paramètres architecturaux

Le paramètre Net_D est fourni par l'utilisateur pour fixer le nombre des PEs reconfigurables dans l'architecture. A partir des fichiers de traces, nous déterminons et nous classifions les ressources disponibles sur FPGA (AR), ainsi que les ressources requises par la partie statique (SR). Comme première étape, nous commençons par soustraire (SR) de (AR) pour obtenir les ressources disponibles pour tous les RPR. Ensuite nous déterminons la taille requise par un MPR commun (MPR_C). La détermination de ce paramètre sera détaillée dans la Section suivante. Avant de finir, nous calculons le paramètre DR en fonction de Net_D et PRM_C. Enfin, nous étudions la faisabilité de l'implémentation tout en calculant le paramètre F. Ce paramètre est calculé à partir de l'expression suivante :

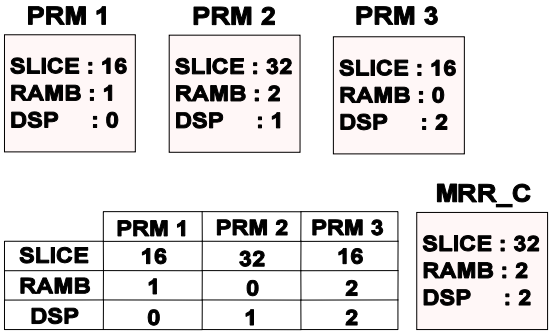
$$F = (AR\text{-}SR) – (Net\_D \times RPR\_C)$$

Dans le cas où $F < 0$, ceci indique une infaisabilité de l'implémentation dû au défaut de ressources nécessaires pour satisfaire les réquisitions de l'implémentation. Sinon, l'algorithme de floorplanning automatique peut être exécuté.

3.3.3.5 Détermination de la configuration commune de l'architecture

Avant de décrire les détails de l'algorithme, étant donné que tous les RPRs sont similaires dans notre architecture et chacun d'eux permet de supporter une infinité de MPRs avec différentes exigences en termes de ressources, il est nécessaire de définir une RPR commune (RPR_C). Un RPR_C est l'union des ressources maximales en termes de SLICE, RAMB et

DSP requis par chaque MPR comme le montre l'exemple de la Figure 3.9. Le tableau présenté dans l'exemple montre l'exigence des ressources des modules MPR1, MPR2 et MPR3. Le RPR_C contient l'exigence maximum des ressources de chaque type (SLICE = 32, BRAM = 2 et DSP = 2).



	PRM 1	PRM 2	PRM 3
SLICE	16	32	16
RAMB	1	0	2
DSP	0	1	2

Figure 3.9 : Détermination de la taille de la MPR_C size suivant les différentes réquisitions de MPRs

Nous mentionnons que des ressources supplémentaires en termes de LUT sont ajoutées à une MPR_C en raison de la nécessité des Proxy Logic au niveau de la composante CLB. De ce fait, un MPR_C devient une RPR_C. Le taux de ressources en termes de Proxy Logic est récupéré à partir des fichiers de traces *RPRs_reports(i).txt* au niveau de la valeur du paramètre ***Boundary-crossing Nets***. En effet, ce taux égal au nombre d'entrées/sorties des MPR en termes de LUTs.

3.3.4 Phase 3: Floorplanning régulier des régions reconfigurables sur la structure de l'FPGA (AFLORA)

Nous présentons d'abord les étapes nécessaires qui permettent le déroulement de notre mécanisme de floorplanning AFLORA.

Les règles de l'algorithme heuristique sont décrites dans la Figure 3.10. Notre algorithme présente différentes étapes visant le traçage automatiquement un nombre de Net_D de RPR_Cs identiques. Il consiste à trouver un ensemble de RPR de références dans un seul Adj_CR (Adj_CR_0) puis les répliquer sur les Adj_CRs restantes en utilisant le vecteur de translation V_{trans}. Cet algorithme est basé sur la réplication pour favoriser la relocalisation bitstream en tenant compte de la symétrie des FPGAs. Il prend en compte les paramètres technologiques et architecturaux comme des entrées et génère une configuration UCF (User

Constraint File) décrivant un groupe de RPRs pour différentes RPR_Cs. Les étapes de l'algorithme sont présentées comme suit.

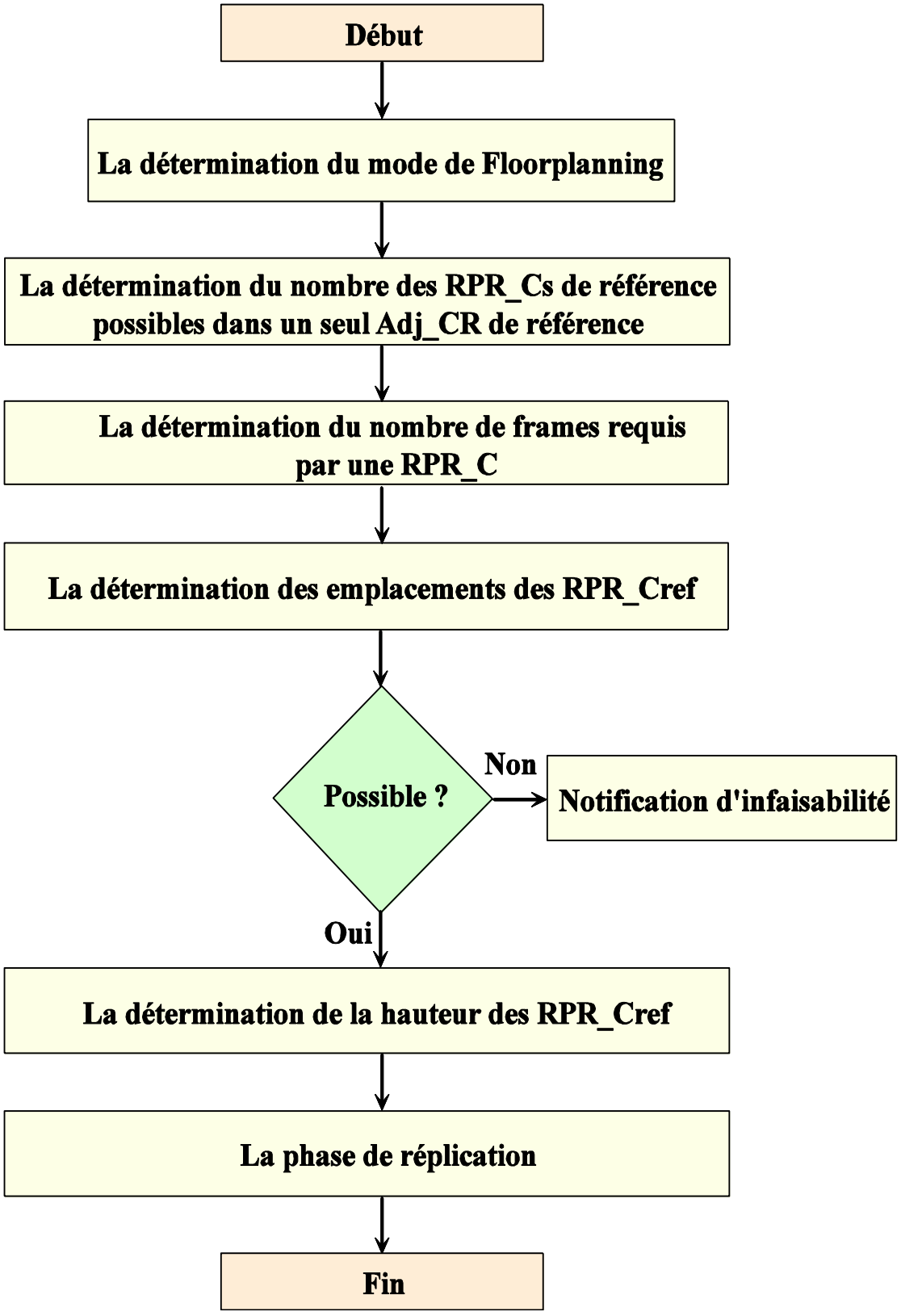


Figure 3.10 : Les étapes de mise en œuvre d'AFLORA

3.3.4.1 La détermination du mode de Floorplanning

Nous avons défini quatre modes de floorplanning en fonction des types de ressources requis par les RPR_Cs en tenant compte du montant supplémentaire de LUTs pour Proxy Logic. Ces modes sont les suivants: Mode 1 (CLB), Mode 2 (CLB + BRAM), mode 3 (CLB + DSP) et le mode 4 (CLB + BRAM + DSP).

3.3.4.2 La détermination du nombre des RPR_Cs de référence dans un seul Adj_CR de référence

Afin de connaitre le nombre des RPR_Cs de référence (RPR_C_{ref}) qui peuvent être positionnées dans un Adj_CR de référence (Adj_CR_{ref}), il est nécessaire de déterminer un taux de répartition équitable des RPR_Cs dans chaque Adj_CR. Nous représentons ce nombre par la notation Z. Z est calculé à partir de l'expression suivante.

$$Z = \left\lceil \frac{Net\_D * C}{nbCR} \right\rceil$$

Où C est le nombre des colonnes de l'FPGA utilisé et nbCR représente le nombre total des régions d'horloges. Nous prenons l'exemple d'une architecture contenant 24 unités de calcul et un FPGA-Virtex 7 qui est caractérisé par les paramètres technologiques suivants : C = 2 et nbCR = 14. Sachant que le nombre total des Adj_CRs est égal à nbCR/C, d'où :

$$Z = \frac{Net\_D}{Nombre\ des\ Adj\_CR} = \frac{24}{7}$$

Si la partie fractionnaire du résultat est non nul, alors Z est incrémenté de 1. D'où Z = 4. La Figure 3.11 illustre la répartition équitable que nous visons à effectuer à travers la détermination de la valeur du paramètre Z. La Figure 3.11 montre dans sa première partie l'allocation des quatre RPR_C_{ref} au niveau de l'Adj_CR_{ref} (Adj_CR0). A partir des emplacements des RPR_Cs que nous allons déterminer par la suite, nous établissons la réplication de chacune de ces postions vers les Adj_CRs restantes en fonction du nombre des unités de calculs Net_D. La deuxième partie de la Figure illustre ce que nous voulons obtenir comme positions des RPR_Cs restantes conformément aux règles indiquées par relocalisation de bitstream grâce à un floorplanning régulier.

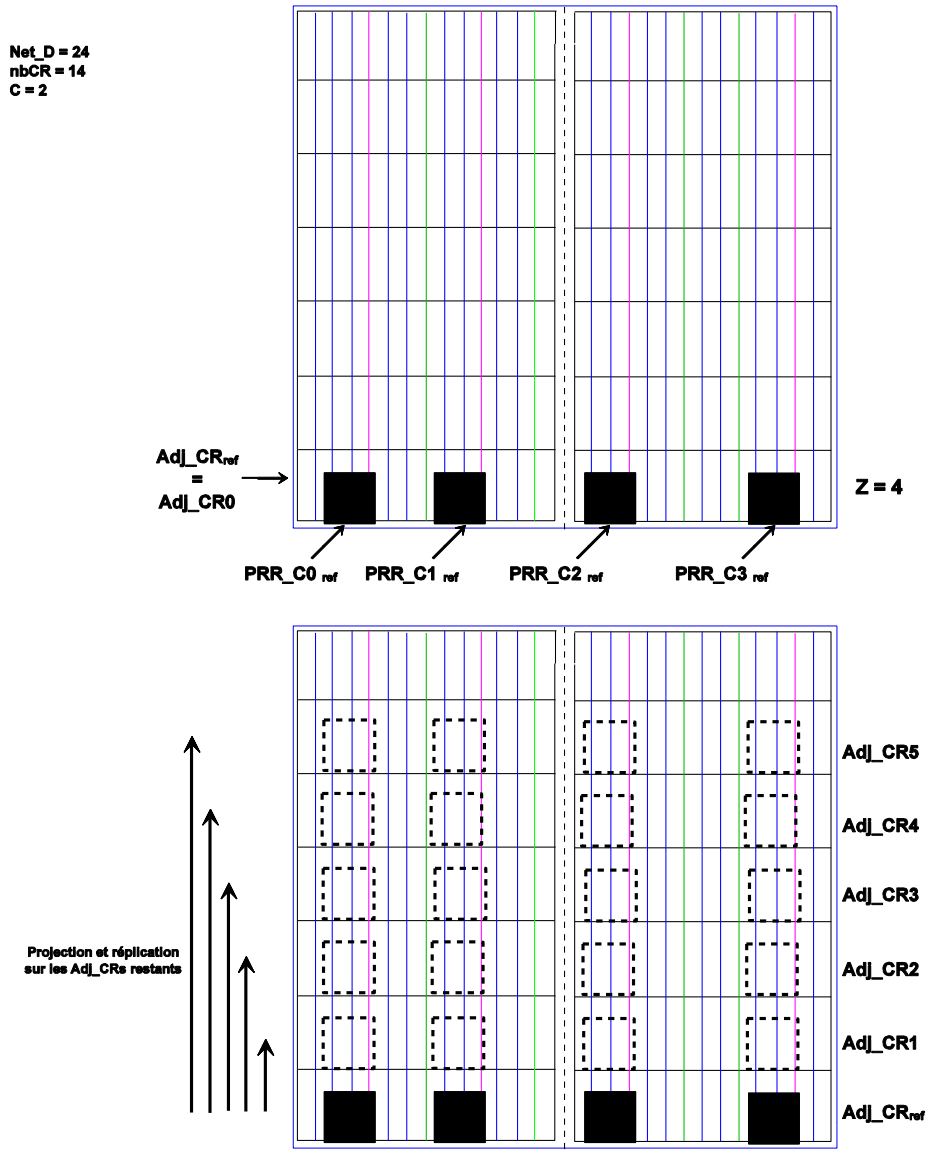


Figure 3.11 : Floorplanning régulier en fonction de la valeur de Z

3.3.4.3 La détermination du nombre de frames requis par une RPR_C

Etant donné que les tailles des frames CLB, BRAM et DSP sont fixes, le calcul du nombre nécessaire des frames pour une RPR_C est donnée par l'équation suivante:

$$\text{Vect} = \begin{pmatrix} \text{PRR\_C}_{\text{CLB}} \\ \text{PRR\_C}_{\text{BRAM}} \\ \text{PRR\_C}_{\text{DSP}} \end{pmatrix} / \begin{pmatrix} \text{Frame\_C} \\ \text{Frame\_R} \\ \text{Frame\_D} \end{pmatrix} = \begin{pmatrix} \text{N}_{\text{CLB}} \\ \text{N}_{\text{BRAM}} \\ \text{N}_{\text{DSP}} \end{pmatrix}$$

Où $\text{RPR\_C}_{\text{CLB}}$, $\text{RPR\_C}_{\text{BRAM}}$ et $\text{RPR\_C}_{\text{DSP}}$ sont les ressources requises par une seule RPR_C. Frame_C, Frame_B et Frame_D sont des paramètres technologiques qui sont respectivement les tailles des frames CLB, BRAM et DSP. Etant donné qu'une RPR nécessite

au moins 2 éléments CLBs (R et L) positionnés sur ces bords suivant l'axe des X, le paramètre technologique Frame_C est multiplié par 2. Donc Frame_C = Frame_C x 2.
Vect est le vecteur résultant qui indique le nombre de frames requis par une RPR_C. Nous mentionnons que si l'une des valeurs du Vect est supérieure à 1, la hauteur de la RPR_C sera égale à CR_Height puisque Frame_C, Frame_B et Frame_D sont géométriquement égales. Sinon la hauteur RPR_C sera ajustée.

3.3.4.4 La détermination des emplacements des RPR_C_{ref}

Une fois le nombre nécessaire des frames requis par une RPR_C est calculé, nous pouvons entamer la phase suivante qui consiste à trouver les emplacements des Z x RPR_C_{ref} au sein de l'Adj_CR_{ref}. Pour ce faire, notre algorithme effectue un parcours de chaque tuile de la largeur de l'FPGA (X_MAX) suivant une seule ligne de tuiles dans le but de chercher la largeur minimale d'une RPR_C en fonction des valeurs données par le vecteur Vect. Chaque largeur de RPR_C doit être comprise entre deux tuiles : la première de type CLB_L côté gauche et la deuxième de type CLB_R côté droite suivant le flot de DPR Xilinx. Par conséquent, la largeur d'une RPR_C est déduite entre les positions de la tuile CLB_L et la tuile CLB_R.

En général, pour un FPGA donné, les BRAMs sont les ressources les plus critiques tandis que les CLBs sont les moins critiques. En outre, la distribution de ces ressources tout le long de l'axe des X n'est pas régulière en général. Cet aspect nous conduit à optimiser la largeur de la RPR_C puisque la taille physique d'un bitstream partiel dépend de la largeur de sa RPR correspondante.

La première étape de ce processus est de trouver une première RPR_C_{ref} puis sauvegarder dans un tableau (SEQ) sa séquence de tuiles formant sa largeur. Ensuite, la séquence SEQ sert à rechercher la même occurrence pour les (Z – 1) x RPR_C_{ref} restants dans la même Adj_CR_{ref}.

Suivant les modes de floorplanning, trois tableaux sont générés (CLB_loc, BRAM_loc et DSP_loc) pour contenir les emplacements des Z RPR_C_{ref}. En effet cette classification est déduite de l'hétérogénéité de la séquence SEQ. L'exemple présenté dans la Figure 3.12 montre la classification de la séquence SEQ dans les tableaux CLB_loc_0, BRAM_loc_0 pour la RPR_$C0_{ref}$ ainsi de suite pour les trois RPR_Ci_{ref} restantes.

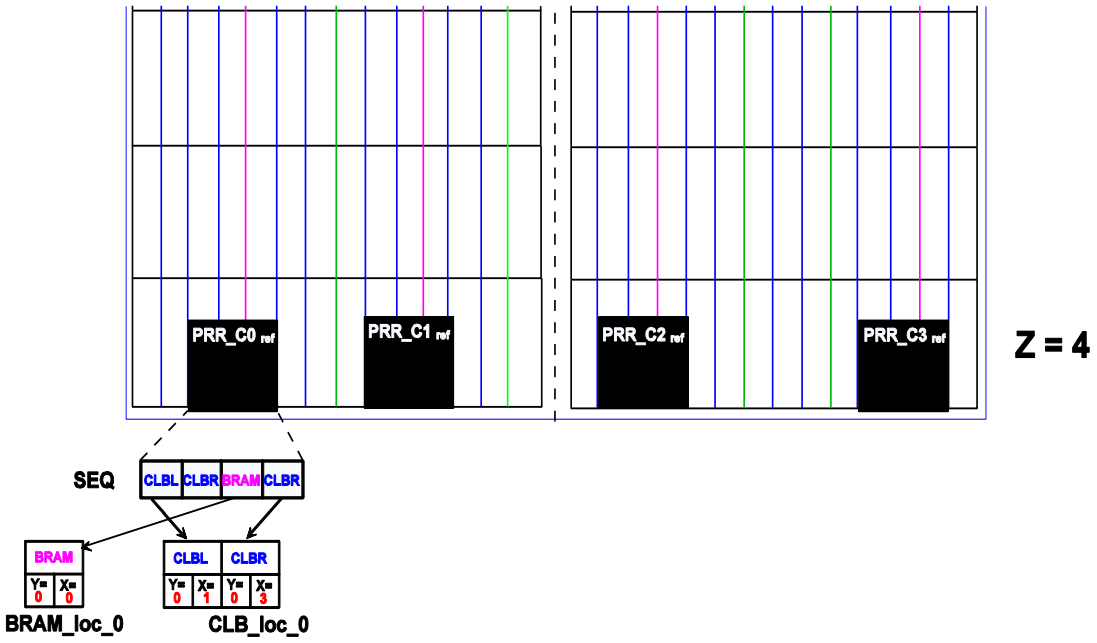


Figure 3.12 : Classification de la séquence SEQ

Nous remarquons bien que seulement les valeurs des coordonnées X sont affectées puisque jusqu'à ce stade nous n'avons pas encore calculé la valeur de l'hauteur d'une RPR_C_{ref}.
Dans le cas où l'Adj_CR_{ref} ne peut pas contenir le nombre Z des RPR_Ci_{ref} avec leurs emplacements, une notification est générée à l'utilisateur pour indiquer une infaisabilité d'implantation de sa configuration du système.

3.3.4.5 La détermination de la hauteur des RPR_C_{ref}

La hauteur RPR_C (H) est déterminée à partir des équations ci-dessous. Les termes N_{CLB}, N_{BRAM} et N_{DSP} sont donnés précédemment par le vecteur Vect. La fonction MAX(H0) est détaillée par la suite.

$$H = \begin{cases} CR\_Height, & \text{if } N_{CLB} > 1 \text{ or } N_{BRAM} > 1 \text{ or } N_{DSP} > 1 \\ \lceil Max(H_0) \rceil, & \text{otherwise} \end{cases}$$

La hauteur H est géométriquement égale à la valeur maximale des éléments N_{CLB}, N_{BRAM} et N_{DSP}. En fait, puisque tous les types frames sont géométriquement égaux, il est essentiel de faire une représentation commune qui exprime un type de frame en fonction des autres types de frames. Elle est réalisée par le biais de la normalisation du vecteur $Frame_{Size}$ = (Frame_C, Frame_B, Frame_D) où Frame_C représente la taille du frame CLB, Frame_B représente la taille du frame BRAM et Frame_D représente la taille du frame DSP. Par exemple dans le FPGA Virtex7, la valeur du vecteur $Frame_{Size}$ est égale à (Frame_C = 50, Frame_B = 10,

Frame_D = 20). La normalisation s'effectue en exprimant (par division) l'un des types des frames en fonction des autres. Dans cet exemple la normalisation en fonction de Frame_B est décrite comme suit :

$$\begin{pmatrix} \text{Frame\_C/Frame\_B} \\ \text{Frame\_B/Frame\_B} \\ \text{Frame\_D/Frame\_B} \end{pmatrix}_{\text{BRAM}} = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}_{\text{BRAM}}$$

Nous appelons H0 le vecteur résultant de l'opération de division des ressources requises par une RPR_C par le vecteur $\text{Frame}_{\text{Size}}$ normalisé. H0 est donné par l'expression suivante :

$$H0 = \begin{pmatrix} \text{PRR\_C}_{\text{CLB}} \\ \text{PRR\_C}_{\text{BRAM}} \\ \text{PRR\_C}_{\text{DSP}} \end{pmatrix} / \begin{pmatrix} \text{Frame\_C/Frame\_B} \\ \text{Frame\_B/Frame\_B} \\ \text{Frame\_D/Frame\_B} \end{pmatrix}_{\text{BRAM}}$$

De cette façon nous obtenons la valeur de H suivant la fonction MAX(H0). Une fois que H est déterminé, tous les emplacements et coordonnées de chacune des Z x $\text{RPR\_C}_{\text{ref}}$ en (X, **Y**) peuvent être mis à jour dans les tableaux CLB_loc, BRAM_loc et DSP_loc.

3.3.4.6 La phase de réplication

L'étape de réplication consiste à dupliquer les emplacements indiqués dans les tableaux CLB_loc, BRAM_loc et DSP_loc de chaque $\text{RPR\_C}_{\text{ref}}$ suivant le vecteur $\text{Frame}_{\text{Size}}$ vers les Adj_CRs restants. La Figure 3.13 illustre l'application de la réplication des $\text{RPR\_C}_{\text{ref}}$ sur le reste des Ajd_CRs.

Au cours de la boucle de réplication une description UCF est générée au niveau de chaque passage d'un Adj_CR pour mettre en œuvre les contraintes de localisation en coordonnées XY des (Net_D - Z) x RPR_Cs avec leurs PBLOCKs correspondants. Cette description est générée dans un fichier à introduire dans le flot de RDP Xilinx.

L'étape suivante se manifeste par l'association de chaque module reconfigurable (.ngc partiel) à toutes les régions reconfigurables. Ceci se fait par un ensemble de commandes TCL. Les différentes configurations se trouvent dans l'interface *« Design Runs »* de l'outil PlanAhead.

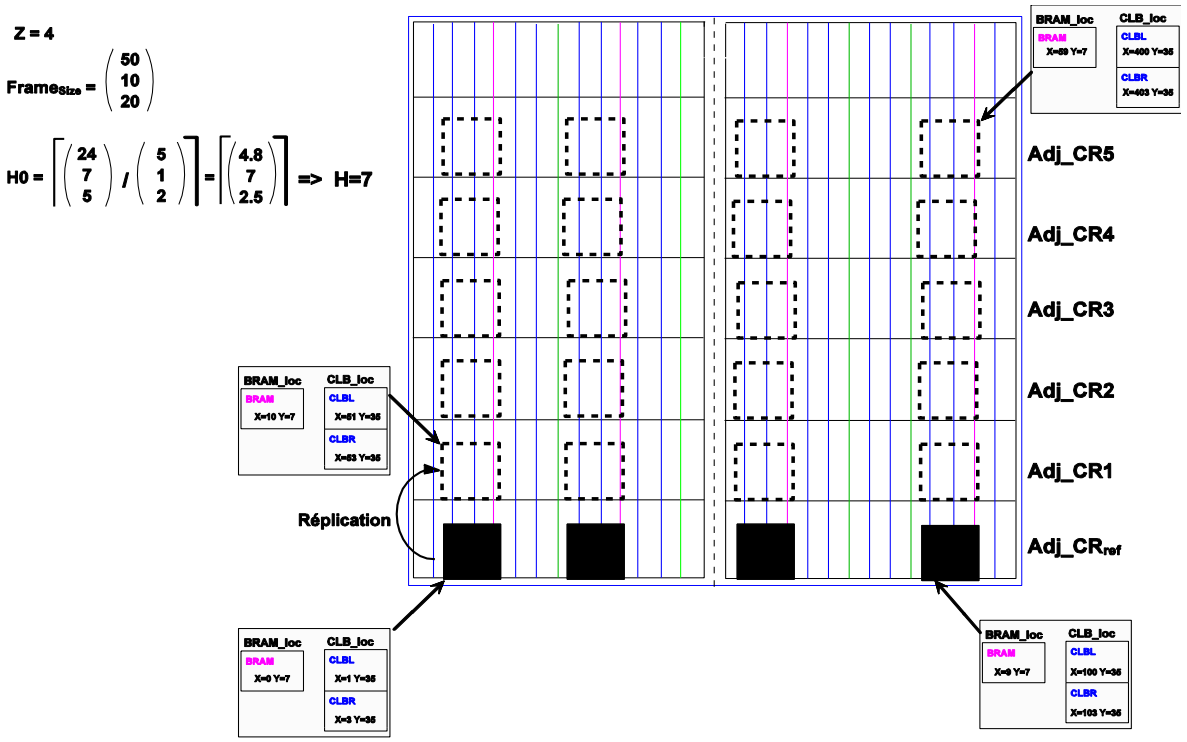


Figure 3.13 : La phase de réplication

3.4 Analyse du résultat : Complexité et qualité des solutions

La complexité de l'algorithme AFLORA linéaire O(Z) et déterminée comme suit : Nous considérons la variable SEQ_L qui représente la largeur de la séquence SEQ. Nous démontrons le coût en termes d'opérations (principalement de comparaison) lors du floorplanning des PRR_C_{ref} tout le long de la largeur Adj_CR_{ref} (largeur du FPGA = X_max). Soit nbIT le nombre de comparaisons. Nous considérons l'étude de cas le plus complexe qui est le mode 4 (voir section 3.3.4.1) avec des RPR_C_{ref} qui nécessitent l'ajustement de leurs hauteur.

La première étape consiste à trouver une BRAM la plus proche que possible d'un DSP. Le nombre de comparaisons maximal ici est égal à Z * X_max. La deuxième étape consiste à satisfaire la mise en place des contraintes CLB_L/CLB_R. Le CLB_L doit être positionné à gauche de la BRAM tandis que le CLB_R doit être positionné à droite du DSP trouvés. Il suffit de balayer sur deux côtés droit et gauche autour de l'intervalle I = [BRAM DSP]. Dans le cas le plus critique, Il peut être positionné au milieu de l'Adj_CR. Par conséquent, le coût des deux itérations à droite et à gauche est égal à Z * X_max/2. D'où, le nombre d'opérations totales pour trouver la première PRR_C_{ref} est :

$$nbI\ T_1 = 2 * Z * X\_max$$

Une fois la première PRR_C_{ref} est allouée, nous récupérons la séquence SEQ qui constitue les éléments de largeur de cette PRR_C_{ref}. La séquence SEQ nous permet de chercher les Z-1 RPR_C_{refs} restantes. Ce mécanisme est illustré dans la Figure 3.14 et se déroule comme suit :

- Trouver la position suivante de tuile de BRAM ($BRAM_{suivant}$) sur la largeur de l'FPGA en commençant la recherche par la position X_max – [Position (SEQ_{i-1}) + SEQ_L], où 1<i<Z-1 et Position (SEQ_{i-1}) est la position du CLBL se trouvant sur l'extrémité gauche d'une RPR_C_{ref}.
- Vérifier à partir des positions de l'élément BRAM ($BRAM_{SEQ}$) alloué dans la séquence SEQ et la position de $BRAM_{suivant}$, que les voisinages de $BRAM_{suivant}$ sont identiques par rapport aux voisinages de $BRAM_{SEQ}$ en termes de type et d'occurrence.
- Si le résultat est valide, la séquence trouvée sur la largeur du FPGA est allouée pour une RPR_C_{ref}, sinon l'algorithme poursuit la recherche d'un autre $BRAM_{suivant}$.

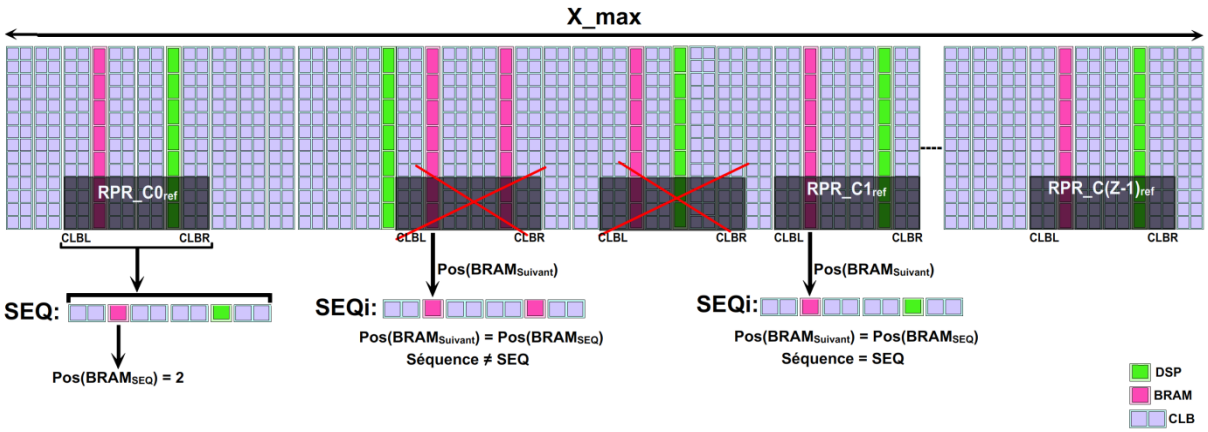


Figure 3.14 : Recherche des RPR_C_{ref} sur la largeur du FPGA

En effet, le nombre des opérations nécessaires au pire cas pour retrouver et localiser chacune des RPR_C_{refs} est présenté comme suit:

$nbIT_2 = X\_max – [Position\ (SEQ_1) + SEQ\_L]$

$nbIT_3 = X\_max – [Position\ (SEQ_2) + SEQ\_L]$

$nbIT_4 = X\_max – [Position\ (SEQ_3) + SEQ\_L]$

(…)

$nbIT_z = X\_max – [Position\ (SEQ_{Z-1}) + SEQ\_L]$

D'une manière plus générale, le coût en Nombre Total d'Opérations (NTO) à base de comparaison est donné comme suit :

$$\mathrm{NTO} = \sum_{i=1}^{Z} \mathrm{nbIT_i}$$

$$\mathrm{NTO} = 2.\mathrm{Z}.\mathrm{X\_max} + (\mathrm{Z}-1).\mathrm{X\_max} - [\sum_{i=2}^{Z-1} \mathrm{Position}\ (\mathrm{SEQ_i}) + \mathrm{SEQ\_L}]$$

L'expression [Position (SEQ_i) + SEQ_L] tel que 1<i<Z, est la largeur restante sur laquelle la recherche de la SEQ est effectuée suite à la localisation d'une PRR_C_{ref}. Suite à la localisation de la première PRR_C_{ref} cette largeur est presque égale à X_max et elle tend vers 0 si Z tend vers l'infini. Dans le pire cas la somme de ces largeurs est égale à X_max d'où :

$$\mathrm{NTO} = 2.\mathrm{Z}.\mathrm{X\_max} + (\mathrm{Z}-1).\mathrm{X\_max} - \mathrm{X\_max}$$

$$\mathrm{NTO} = 2.\mathrm{Z}.\mathrm{X\_max} + \mathrm{Z}.\mathrm{X\_max} - \mathrm{X\_max} - \mathrm{X\_max}$$

$$\mathbf{NTO = 3.Z.X\_max - 2.X\_max}$$

3.5 Conclusion

Dans ce chapitre, nous avons décrit notre mécanisme de floorplanning automatique et régulier des régions partiellement reconfigurables. En effet, suite à la première implémentation en utilisant la description UCF générée introduite dans le flot de conception de la RDP Xilinx, il nous sera possible d'unifier le placement des blocs fonctionnels au sein de chaque région reconfigurable à partir des RPR_{ref}. De plus, nous exploiterons le routage de ces régions de référence pour unifier aussi le routage des signaux de connexion de ces blocs. Ceci nécessite l'extraction et le traitement de ces informations à partir du fichier NCD du projet. De plus, le forçage de quelques paramètres d'outillage de PlanAhead pour éviter que les régions partiellement reconfigurables soient traversées par des signaux de connexion de la partie statique. Dans le chapitre quatre nous détaillons le traitement de la description NCD qui nous permet d'appliquer la technique de la relocalisation de bitstream suivant les dimensions 1D et 2D.

CHAPITRE 4 : La Réutilisation de la Technique de Relocalisation et le Broadcast de Bitstream Partiel

4.1 Introduction

Suite à la phase de floorplanning automatique assurée par notre flot d'automatisation ADForMe, la mise à jour du projet PlanAhead est effectuée en associant la nouvelle spécification (UCF). Après avoir lancé la première implémentation du système, il est rare d'obtenir le même placement et routage (P&R) des blocs fonctionnels dans chacune des régions partiellement reconfigurables. Ceci s'explique du fait que les algorithmes de mappage et de P&R intégrés dans l'outil PlanAhead/Vivado favorisent la performance plutôt que la reconfigurabilité même si les régions reconfigurables sont de composition identique. Donc cela pose une problématique pour assurer la technique de relocalisation de bitstream. Deux aspects se présentent à cet effet :

Premièrement, nous pouvons remarquer que les signaux qui appartiennent à la partie statique sont routés à travers les régions dynamiquement reconfigurables. En effet, ceci engendre un disfonctionnement du système puisque ces signaux statiques peuvent être rompus lors de la phase de relocalisation étant donné que le P&R dans chaque RPR est isolé même si ces dernières sont identiques en termes de forme et de fonction.

Deuxièmement, les interfaces des régions partiellement reconfigurables ne sont pas symétriques vu que le placement des Proxys Logiques n'est pas identique. Ceci engendre aussi l'incompatibilité lorsque nous voulons reconfigurer l'ensemble des régions partiellement reconfigurables par un même bitstream partiel.

Dans ce chapitre nous décrivons les solutions susceptibles de résoudre ces deux problématiques. Nous commençons par détailler ces deux aspects et les primitives à tenir compte pour assurer la technique de relocalisation, puis nous décrivons notre intervention en différentes étapes pour assurer cette technique en mode 1D et 2D. Nous proposons ainsi un mécanisme qui vise l'automatisation de ces différentes étapes favorisant cette technique.

4.2 Problématiques

4.2.1 Le routage des signaux de la partie statique à travers les régions partiellement reconfigurables.

Le flot de la reconfiguration dynamique partielle de Xilinx place des cellules (LUT) appartenant à la logique statique à l'extérieur et à l'intérieure des partitions reconfigurables. Un LUT de la logique statique se trouvant dans une région partiellement reconfigurable est connecté à plusieurs broches appelées « NETs ». En effet, lors de la génération de bitstream partiel relatif à cette région, les informations de placement et routage de la cellule statique y

sont incluent. Ces informations de placement et routage sont décrites dans le fichier de configuration XDL (*Xilinx Design Language*) généré à partir de la description NCD. Un tel aspect se manifeste au niveau de chaque région partiellement reconfigurable avec des placements et routages divers. En effet, il est impossible de garantir le bon déroulement de la relocalisation de bitstream partiel puisque l'interface d'une RPR est sensible aux signaux qui la traversent. La Figure 4.1 illustre en détail cet aspect.

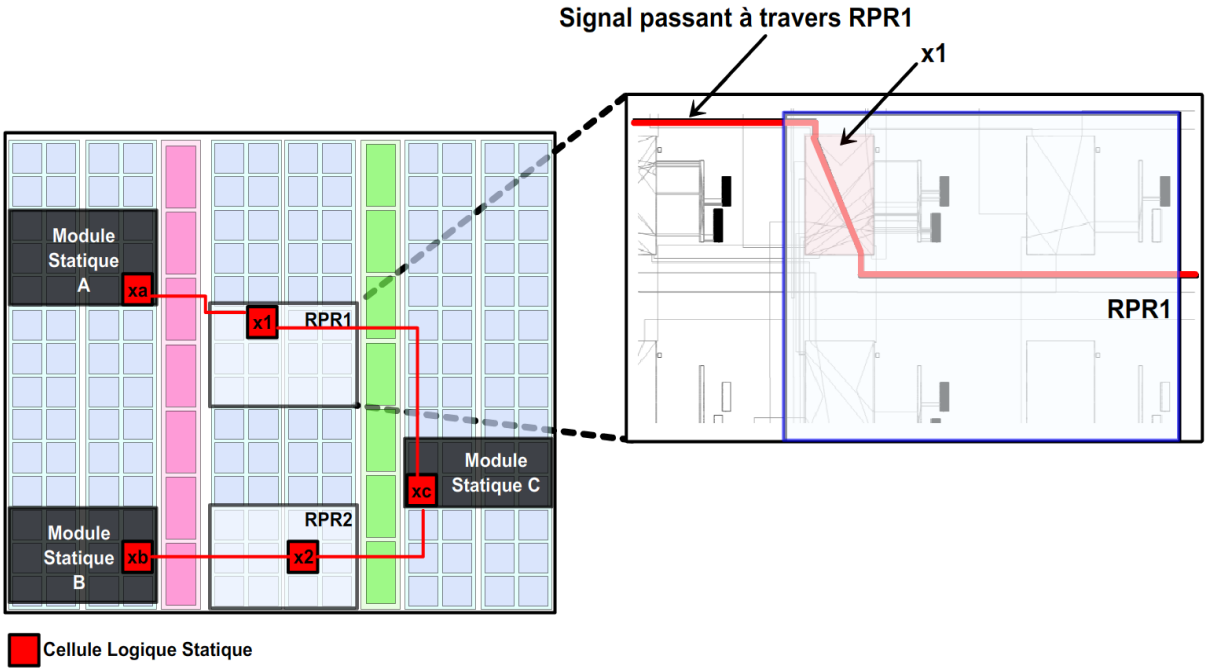


Figure 4.1 : Le placement et le routage des cellules statiques à travers les régions partiellement reconfigurables

La reconfiguration de la région partiellement reconfigurable RPR1 avec un bitstream généré à partir d'une la RPR2 engendre d'une part le risque de rupture des connexions statiques (xa, x1) et (x1, xc) qui interconnectent l'élément x1, et d'autre part le risque de la modification de la fonction *f(x1)* produite par la cellule logique statique x1 par une autre fonction $f_1(x1)$ relative à celle produite par la cellule qui occupe le même placement dans la région RPR2.

Pour remédier à ces deux problèmes il suffit de forcer l'outil de mappage et placement et routage (NGBUILD) à éviter le placement de cellules logiques statiques dans les régions partiellement reconfigurables. Ceci se manifeste par l'ajout d'un ensemble de primitives en deux contraintes pour chaque région qui modifient leurs propriétés en termes d'accès. Ces contraintes sont:

```
AREA_GROUP "Nom de la RPR" PRIVATE=ROUTE;
AREA_GROUP "Nom de la RPR" CONTAINED =ROUTE;
```

4.2.2 Uniformisation des interfaces des régions partiellement reconfigurables

Le flot de la reconfiguration partielle de Xilinx introduit pour chaque entrée/sortie d'une RPR un ensemble de cellules logiques dynamiques appelées Proxys Logiques. Un Proxy Logique est un élément LUT inséré automatiquement par l'outil d'implémentation pour chaque broche (PIN) passant d'une région statique vers une région dynamiquement reconfigurable. Un seul LUT est requis par un signal binaire. D'après notre conception de l'architecture Multi-RPRs, toutes les régions dynamiques possèdent le même nombre d'entrées/sorties. D'où, chacune d'elle a le même nombre de Proxys Logiques. En revanche, l'outil *NGBUILD* ne donne pas d'importance au placement et routage de ces composants d'une manière uniforme au sein de chaque région partiellement reconfigurable car il ne favorise que la performance. A cet effet, suite à la génération des bitstreams partiels, le contenu de chacun de ces derniers n'est pas identique.

Dans ce cadre, nous intervenons dans le but d'uniformiser le placement des Proxys Logiques et leur routage au sein de toutes les régions reconfigurables comme l'indique la Figure 2.7. L'ensemble des Proxys Logiques dans une région dynamiquement reconfigurable constitue une interface d'isolation entre l'espace reconfigurable et l'espace statique.

D'autre part, il n'est pas toujours garanti de contrôler les connexions directes entre les Proxys Logiques et les composants de la partie statique. En effet, l'outil *NGBUILD* peut placer ces composants statiques différemment aux alentours de chaque région reconfigurable. Ceci engendre des routages non uniformes avec les Proxys Logiques comme illustré dans les Figures 4.2 et 4.3.a.

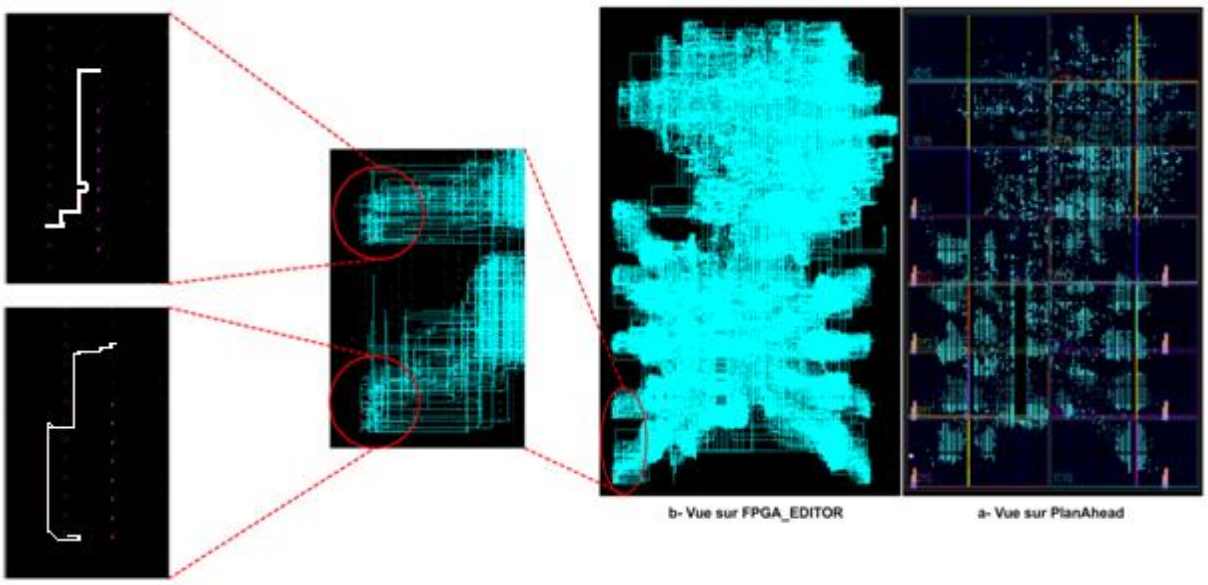
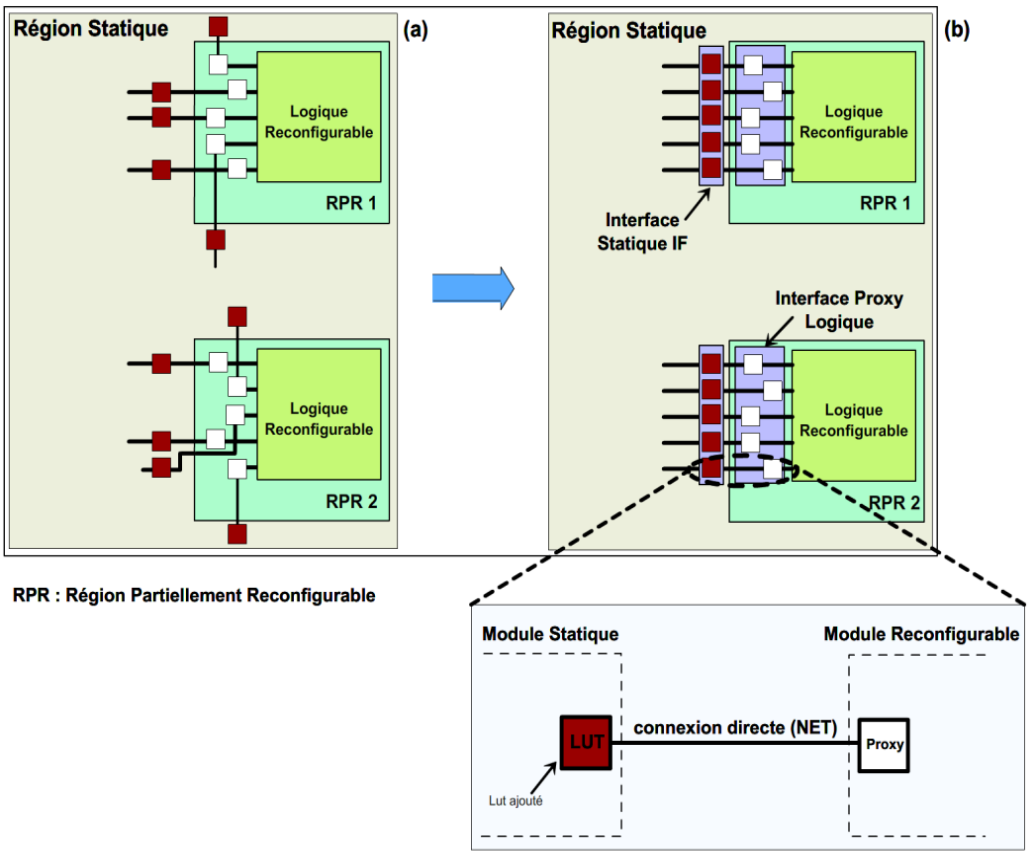


Figure 4.2 : Résultat de routage suite à la première implémentation



Figue 4.3. Uniformisation des interfaces des régions partiellement reconfigurables

La Figure 4.2 montre le résultat de placement et routage d'une configuration de notre architecture MULTI-RPRS en dimension 3x3 en utilisant les outils PlanAhead (a) et FPGA_Editor (b). Nous remarquons la différence de routage de la même référence de signal

d'interconnexion entre deux RPRs et la partie statique. Ceci est à cause de la différence de placement et de routage des composants Proxys Logiques au niveau de chaque RPR.

Il est donc nécessaire de construire une interface statique *IF* pour chaque interface de Proxy Logique. Ceci permet d'uniformiser le routage des connexions directes tout en les faisant orienter vers une même direction. La distance qui sépare les deux interfaces doit être minimale pour éviter qu'une connexion directe passe par des nœuds de routage intermédiaires comme illustré dans la Figure 4.3.b.

4.2.2.1 Les interfaces statiques

Les interfaces statiques jouent le rôle de bus macro pour les technologies FPGA antérieures. Leur composition est formée par des LUTs dont chacun est associé à un Proxy Logique. Il est donc nécessaire de prévoir leur conception au niveau de la définition de l'architecture Multi-RPRS. Nous employons deux types d'interfaces, les interfaces d'entrée *IPIF_IN* et les interfaces de sorties *IPIF_OUT*. Une interface statique d'entrée est associée à un bus d'entrée d'un module reconfigurable à travers l'interface de Proxys Logiques d'entrée. De même, une interface statique de sortie est associée à un bus de sortie d'un module reconfigurable à travers l'interface de Proxys Logiques de sortie.

Nous considérons dans cette thèse des modules reconfigurables qui disposent de 3 bus d'entrée, ainsi que 3 bus de sortie. Chaque bus d'entrée ou de sortie est de largeur de 32 bits.
Le surcoût engendré par l'instanciation des interfaces statiques pour chaque module reconfigurable en termes de LUTs est de 192 (= 48 Slices = 24 CLBs) puisque le processus de mappage exécuté par *NGBUILD* est conçu pour associer un Proxy dans un LUT. Par contre, il est possible de placer deux Proxys dans un même LUT grâce à un forçage effectué par des contraintes utilisateurs. L'application de ce forçage n'est possible que si tous les modules du système Multi-RPRs partagent le même signal d'horloge. Ce qui est le cas pour notre spécification d'architecture Multi-RPRs. Ce mécanisme permet d'optimiser la moitié du surcoût des interfaces statiques. D'où un surcoût de 96 LUTs (= 24 Slices = 12 CLBs) pour chaque région dynamiquement reconfigurable. Les contraintes utilisateurs relatives aux interfaces statiques d'entrée/sorties s'effectuent grâce à la contrainte *AREA_GROUP* [101]. C'est une contrainte d'implémentation qui permet le partitionnement du système sous forme de régions physiques sur l'FPGA et prend effet pendant la phase de mappage. Un exemple d'utilisation de cette contrainte est présenté comme suit :
AREA_GROUP "NOM" RANGE=SLICE_XnYn:SLICE_XmYm;

Le terme NOM représente le nom de l'interface statique d'entrée ou de sortie, tandis que le terme RANGE indique l'intervalle de placement de cette interface.

4.2.2.2 Les interfaces de Proxys Logiques

Les Proxys Logiques sont automatiquement insérés par l'outil (NGDBUILD) pendant la phase de P&R. Suivant notre approche, ces composants doivent être placés dans des zones particulières en utilisant les contraintes LOC [101] et BEL [101]. La contrainte LOCK_PINS [101] doit être utilisée aussi pour verrouiller toutes les broches pour qu'elles restent insensibles à l'algorithme de placement et routage de NGBUILD. Un exemple d'utilisation de ces contraintes est présenté comme suit :

PIN "le nom de la pin" LOC=SLICE_XnYn;
PIN "le nom de la pin" BEL=A6LUT;
INST "le nom de l'instance" LOCK_PINS=ALL;

4.3 La technique de relocalisation de bitstream

La technique de relocalisation de bitstream des modules reconfigurables est une technique qui se base sur le flot de la RDP. Elle permet le déplacement un modules partiellement reconfigurable initialement situés dans une RPR vers une autre RPR de propriétés similaires (taille, longueur, largeur et attributs) mais de localisation différente sur FPGA.

Suivant le floorplanning régulier effectué par l'algorithme AFLORA, la relocalisation se manifeste par deux types :

- Pour deux RPRs se trouvant sur la même colonne, soit $RPR_{départ}$ la première RPR qui est considérée comme RPR de départ et soit la deuxième $RPR_{destination}$ qui est considérée comme RPR de destination. Si nous déplaçons un MPR (son bitstream) de la $RPR_{départ}$ vers la $RPR_{destination,}$ nous parlon alors de la relocalisation 1D. ce type de relocalisation s'effectue tout en translatant verticalement les localisations LOC et les adresses de routages des Proxy Logiques.
- La relocalisation 2D définie comme tout déplacement de MPR d'une $RPR_{départ}$ vers une $RPR_{destination}$ non placées sur la même colonne. Dans ce cas la translation s'effectue verticalement et horizontalement.

Dans notre démarche, les modules reconfigurables ne doivent pas avoir une complexité élevée pour permettre l'application de relocalisation 2D dans l'architecture Multi-

RPRS. Malheureusement, malgré les avantages de la relocalisation partielle de bitstream, les outils de conception pour les FPGA ne prennent pas en charge la relocalisation des modules reconfigurables.

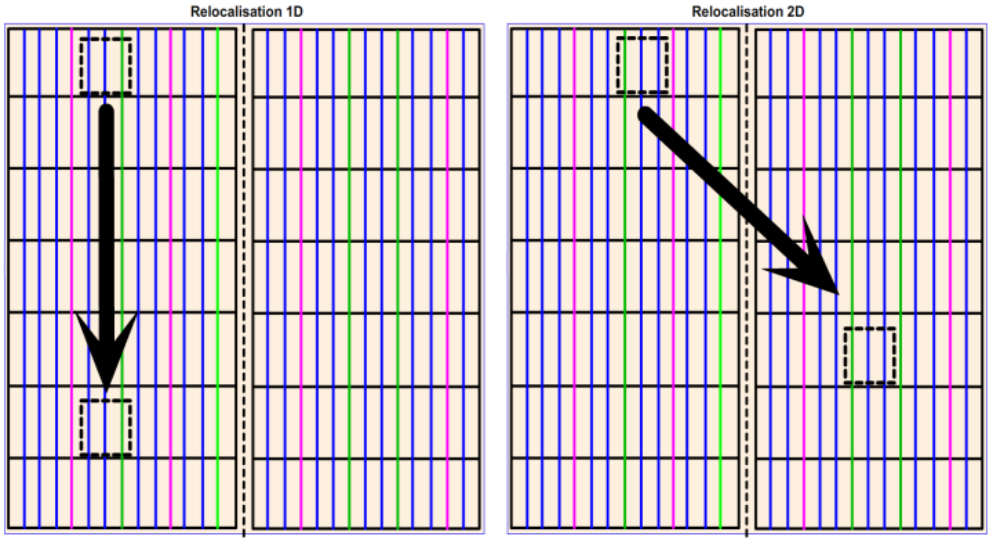


Figure 4.4 : Les techniques de relocalisation 1D et 2D

4.3.1 La relocalisation 1D

Notre démarche pour appliquer la technique de la relocalisation 1D et 2D de bitstream consiste à unifier le placement et le routage des interfaces statiques et des interfaces Proxys Logiques. L'étape de relocalisation commence dès la fin de la première implémentation effectuée par le flot de conception ADForMe. Une analyse des résultats de la première implémentation est nécessaire pour récupérer les informations des deux types d'interfaces. Cette analyse s'effectue en utilisant les deux outils Xilinx : *FPGA_Editor* et Direct Routing Constraints (DRC). Il est possible aussi d'utiliser l'interpréteur de commande Xilinx pour récupérer les informations physiques de placement et de routage qui est la commande *pr2ucf* qui est présentée avec la syntaxe suivante :

Pr2ucf –bel –o <fichier de sortie>.ucf <fichier d'entrée>.ncd

L'exécution de cette commande permet de générer dans le fichier de sortie l'ensemble des emplacements des Proxys Logiques de chaque RPR décrite dans le fichier d'entrée d'extension NCD. Les détails de placement et routage de chaque Proxy Logique est décrit sous forme de contraintes UCF à partir du programme DRC. La Figure 4.5 montre le résultat d'un exemple de rapport généré. Les informations de routage sont décrites en utilisant les

contraintes "NET" et "ROUTE". La contrainte "NET" indique le nom du signal cible à contraindre et la contrainte "ROUTE" indique les informations du chemin de routage.

Le chemin de routage est composé par deux adresses qui sont l'adresse absolue du signal source et l'adresse relative qui comprend les adresses des blocs de routage à travers lesquels le signal source est routé. L'adresse absolue du signal source est décrite suivant les coordonnées (x, y), ces deux coordonnés sont encliquetés entre les symboles **"!-1"** et **"S!0;"**. L'adresse relative est composée par une suite ordonnée d'adresses des commutateurs de routage dont chacune est suivie par le symbole "**!**", puis par son numéro. La dernière connexion est référenciée par le symbole "**L!}**". La contrainte "INST" décrit la hiérarchie du composant Proxy Logique instancié. La contrainte "LOC" est utilisée pour indiquer le Slice alloué pour le placement de ce composant Proxy, tandis que la contrainte "BEL" est utilisée pour adresser le composant LUT qui se trouve dans ce Slice.

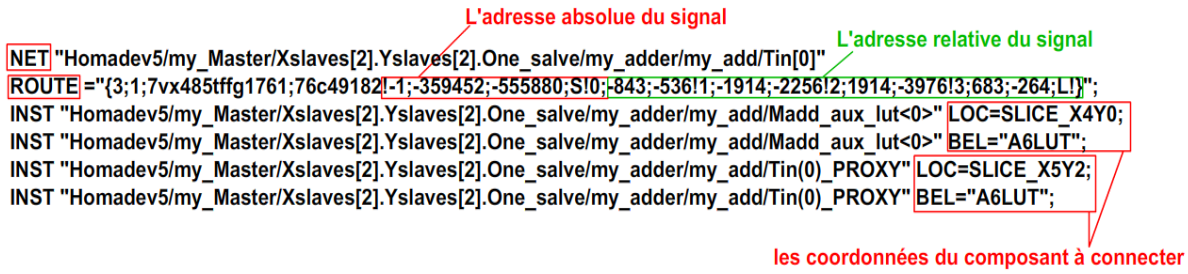


Figure 4.5 : Exemple de rapport « Directed Routing Constraints »

4.3.1.1 Uniformisation du placement des Proxys Logiques

Dans le but d'uniformiser le placement des Proxys Logiques dans chacune des régions partiellement reconfigurables, nous appliquons les étapes suivantes :

- Premièrement nous considérons arbitrairement une région partiellement reconfigurable de référence RPR_{ref} à travers laquelle nous extrairons ces informations de placement des Proxys Logiques en termes de LUT et Slice alloués.
- Deuxièmement nous remplaçons les informations de placement des RPRs situées sur la même colonne par les informations de placement relatives à la RPR_{ref}.
- Troisièmement, nous utilisons le vecteur de translation $\overrightarrow{Frame\_CLB}$, déjà présenté dans le chapitre précèdent, pour retrouver les emplacements des images de chaque Proxy Logique dans chaque RPR à partir de la RPR_{ref}. Chaque image est

l'emplacement d'un Proxy Logique appartenant à une RPR suivant une multiple du vecteur $\overrightarrow{Frame\_CLB}$.

Suivant la relocalisation 1D, l'uniformisation se fait verticalement. La deuxième étape s'effectue par copier les informations de placement des instances de Proxys Logiques à partir de la RPR_{ref} (les valeurs de LOC et BEL) puis les reproduire pour toutes les informations de placement des instances de Proxys Logiques des autres RPRs. Pendant la troisième étape, il suffit de modifier la composante Y de la contrainte LOC de chaque instance de Proxy Logique appartenant à chaque RPRs suivant la formule :

$$Y_{RPR} = Y_{PRR_{ref}} + n * \|Frame\_CLB\|$$

Où $Y_{PRR_{ref}}$ est la valeur de la composante Y de la RPR_{ref}, $\|Frame\_CLB\|$ est la norme du vecteur $\overrightarrow{Frame\_CLB}$ et ***n*** est entier qui représente le nombre de multiplication du vecteur $\overrightarrow{Frame\_CLB}$. Suivant notre démarche présentée dans le chapitre précèdent, la distance qui sépare toute RPR sur la même colonne en termes de CLB est toujours égal à un multiple de la hauteur d'une région d'horloge, de même, cette règle doit être applicable pour les Proxys Logiques. La Figure 4.6 illustre le déroulement simplifié de ce mécanisme pour une Frame_CLB = 50 (propriété technologique de l'FPGA Xilinx Virtex 7).

Comme le montre la Figure 4.6, l'uniformisation se fait au niveau de placement en se référant au placement de la RPR_{ref} et au vecteur $\overrightarrow{Frame\_CLB}$. La Figure présente les étapes de mise en œuvre qui visent à déterminer le placement des images d'un Proxy Logique appartenant à la RPR_{ref} par projection sur les régions partiellement reconfigurables RPR1 et RPR2 suivant le vecteur $\overrightarrow{Frame\_CLB}$. Les deux premières étapes permettent de placer tous les Proxys Logiques sur la même coordonnée X que celle de la RPR_{ref} ainsi que d'unifier la sélection des LUTs utilisés en agissant sur la contrainte BEL (le cas pour la transformation de la RPR1). La troisième étape se manifeste par la mise à jour des composantes Y de chaque emplacement de SLICE relatif à chaque Proxy Logique. Ceci s'effectue en fonction du vecteur $\overrightarrow{Frame\_CLB}$ ainsi qu'en fonction des coordonnées de la région d'horloge dans laquelle la RPR appartient.

Nous prenons l'exemple présenté dans la Figure 4.6, la détermination de la coordonnée Y de l'emplacement du Proxy Logique de la région RPR2 est déduit suivant:

- Le paramètre n qui est égal à la différence entre les coordonnées Y de la région d'horloge de coordonnée X0Y2 (où la région RPR2 est située) et celle de la région de référence X0Y0 (où la région RPR_{ref} est située). Ici n = 2.
- La composante Y de placement de la RPR_{ref} : $Y_{PRR_{ref}}$=5.
- Le vecteur $\overrightarrow{Frame\_CLB}$ de norme égale à 50.

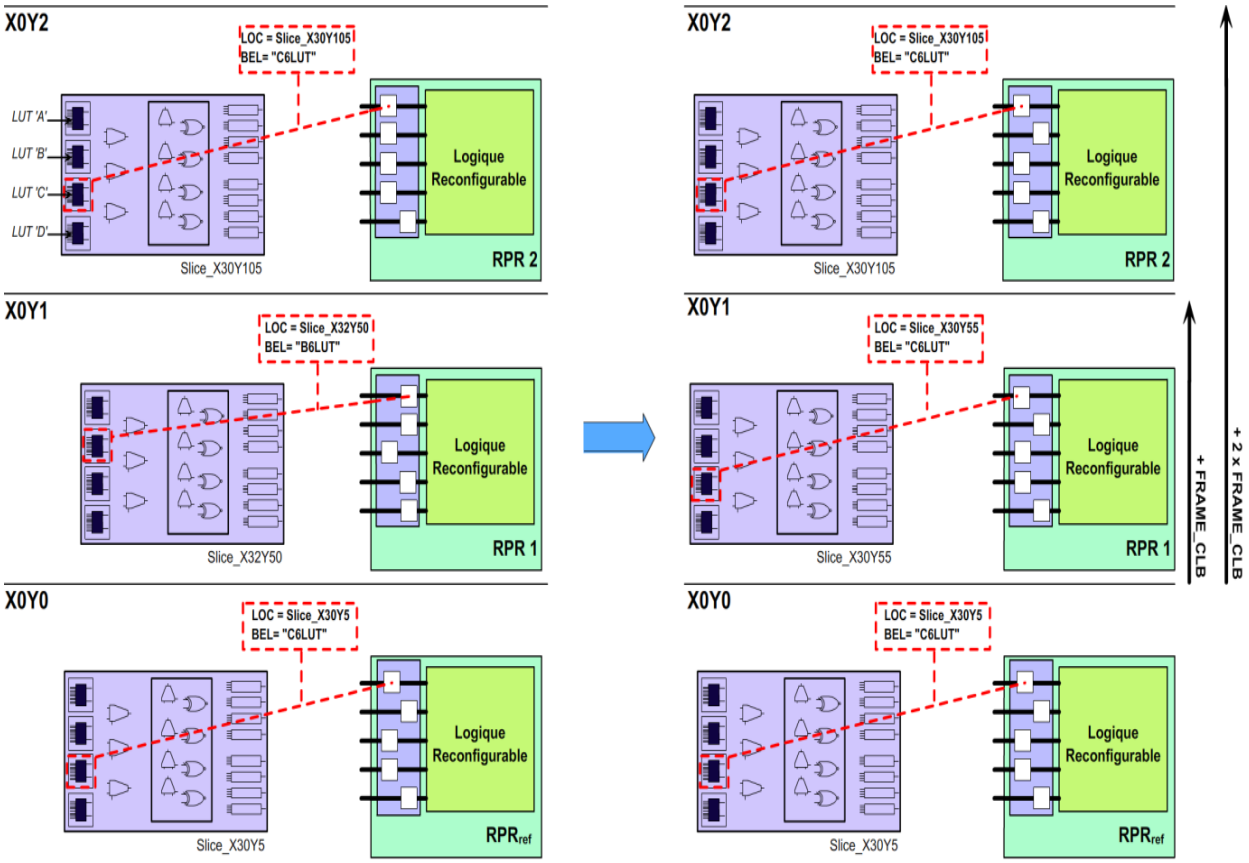


Figure 4.6 : Uniformisation du placement des Proxys Logiques

4.3.1.2 Uniformisation le routage entre les Proxys Logiques et la partie statique

Vu que les Proxys Logiques font partie de la région partiellement reconfigurable, alors les informations de routage des signaux d'interconnexion entre les Proxys Logiques et la région statique sont aussi incluses dans le bitstream partiellement reconfigurable. Donc il est nécessaire d'uniformiser aussi ces connexions. La mise en ouvre de l'uniformisation s'effectue suivant les étapes ci-dessous:

- Extraire les informations de routage en utilisant l'outil FPGA_editor et DRC.
- Copier les informations de routage de la RPR_{ref} puis les affecter au reste des RPRs.
- Mettre à jour les adresses absolues des RPRs sauf celles de la RPR_{ref}.

- Exécuter le processus de placement et routage pour appliquer les contraintes.

Les informations de routage sont extraites à partir du fichier NCD déjà généré depuis la première implémentation à l'aide du programme DRC intégré dans l'outil FPGA_editor. Ce programme permet d'obtenir les adresses de routage pour tout signal sélectionné. D'après l'analyse faite dans la section 4.3.1.

Après l'analyse du chemin de routage, les adresses de routage des Proxys Logiques dans les RPRs doivent être modifiées pour devenir uniformes avec celles des RPR_{ref}. Il suffit de remplacer les informations de routage de chaque image de Proxy Logique (adresse absolue et adresse relative) par une copie des informations du Proxy Logique correspondant se trouvant dans la RPR_{ref}. Etant donné que tous les Proxys Logiques de la RPR_{ref} et leurs images sont uniformisés en termes de placement (Section 4.3.2) vertical, la composante X de l'adresse absolue doit rester inchangée. De même pour l'adresse relative. D'autre part, l'adresse absolue au niveau de la coordonnée Y doit être modifiée pour relier le chemin de routage avec le Proxy Logique image au niveau de chaque RPR. Cela s'effectue à l'aide du vecteur $\overrightarrow{VS}$. Vu la symétrie des deux régions d'horloge adjacentes verticalement, $\overrightarrow{VS}$ est un vecteur de translation vertical qui permet de retrouver l'adresse d'un élément Switch (image) à partir de l'adresse d'un élément Switch de référence. Pour adresser des éléments Switch (image) dans des régions d'horloges distantes, nous employons le paramètre n déjà présenté dans la section précédente suivant la formule suivante :

$$Y_{SA} = Y_{SA_{ref}} + n * \|VS\|$$

Où Y_{SA}est l'adresse de l'élément Switch image, $Y_{SA_{ref}}$est l'adresse de l'élément Switch de référence et $\|VS\|$est la norme du vecteur $\overrightarrow{VS}$.

Pour déterminer la valeur de la norme $\|VS\|$, il est nécessaire de connaitre les coordonnées XY (adresse absolue) de la première broche de chaque type Slice (SliceL et SliceM) de l'FPGA utilisé. Ceci nous permet de connaitre la différence d'adresse entre deux Slices positionnés horizontalement ou verticalement. Nous présentons dans la Figure 4.7 un exemple de mappage d'adresse des Slices sur la plateforme FPGA. Ainsi, ce mappage permet de déterminer la différence d'adresse qui sépare deux Slices situés dans deux régions d'horloges disposées verticalement. Donc, suivant notre floorplanning régulier, uniforme et symétrique, la différence d'adresse est toujours constante entre deux régions d'horloges adjacentes verticalement. Cette différence d'adresses représente la norme du vecteur $\overrightarrow{VS}$.

Par la suite, nous présentons un exemple d'uniformisation de deux Proxys Logiques se trouvant dans deux régions partiellement reconfigurables RPR [0] [0] et RPR [2] [2]. Le résultat de floorplanning des deux régions est donné par notre algorithme de floorplanning présenté dans le chapitre précédent. Le module implémenté dans les deux régions est un additionneur 32 bits dont **Tin** et **Nin** sont les entrées et **Tout** est la sortie. Nous implémentons une configuration d'architecture MULTI-RPRS de taille 5x1, et nous ne traitons qu'une seule colonne de régions partiellement reconfigurables. Les deux régions RPR [0] [0] et RPR [2] [2] se situent verticalement et compatibles à la relocalisation 1D comme le montre la Figure 4.8.

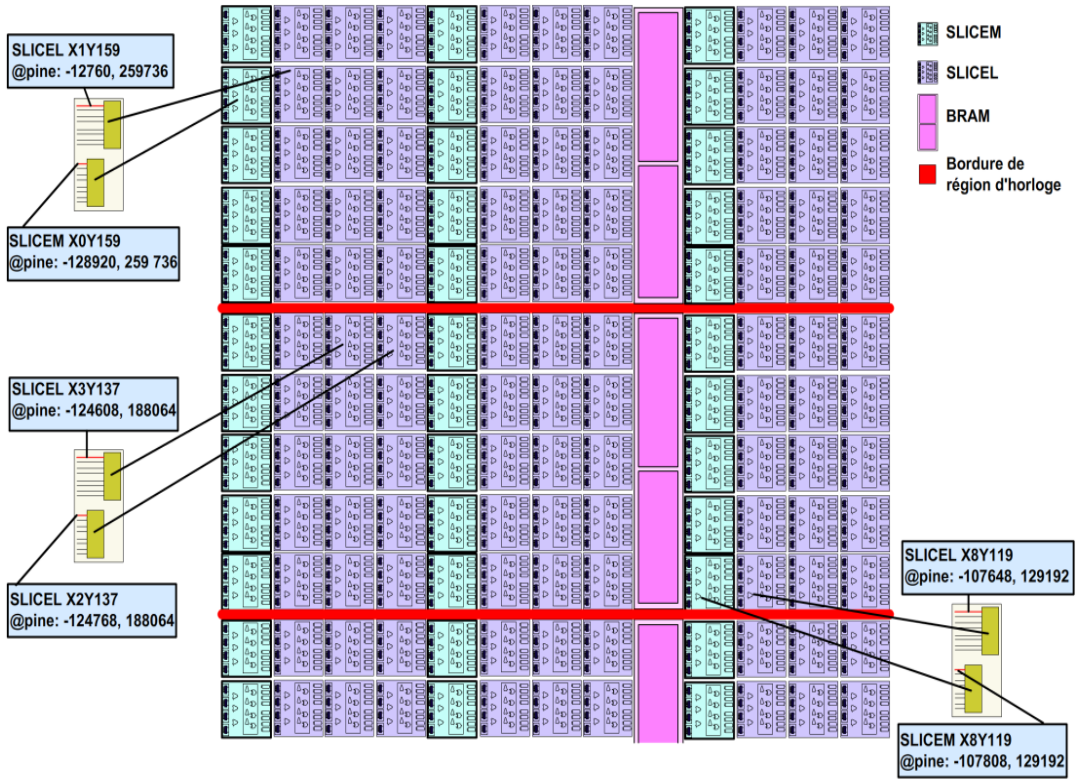


Figure 4.7 : Mappage des adresses des composants Switch

Nous voulons à travers cet exemple l'uniformisation du placement et du routage des Proxys Logiques connectés à l'entrée Tin [0] dans chacune des deux régions compatibles RPR [2] [2] et RPR [0] [0]. La région RPR [2] [2] est la région de référence (RPR_{ref}) qui est située dans la région d'horloge X0Y0. La région RPR [0] [0] est une région située dans région d'horloge X0Y4. Nous utilisons le FPGA Xilinx-Virtex7 pour réaliser cet exemple caractérisé par le paramètre technologique$\|\text{Frame_CLB}\| = 50$. La norme du vecteur de $\overrightarrow{VS}$ est: $\|\text{VS}\| = 161272$.

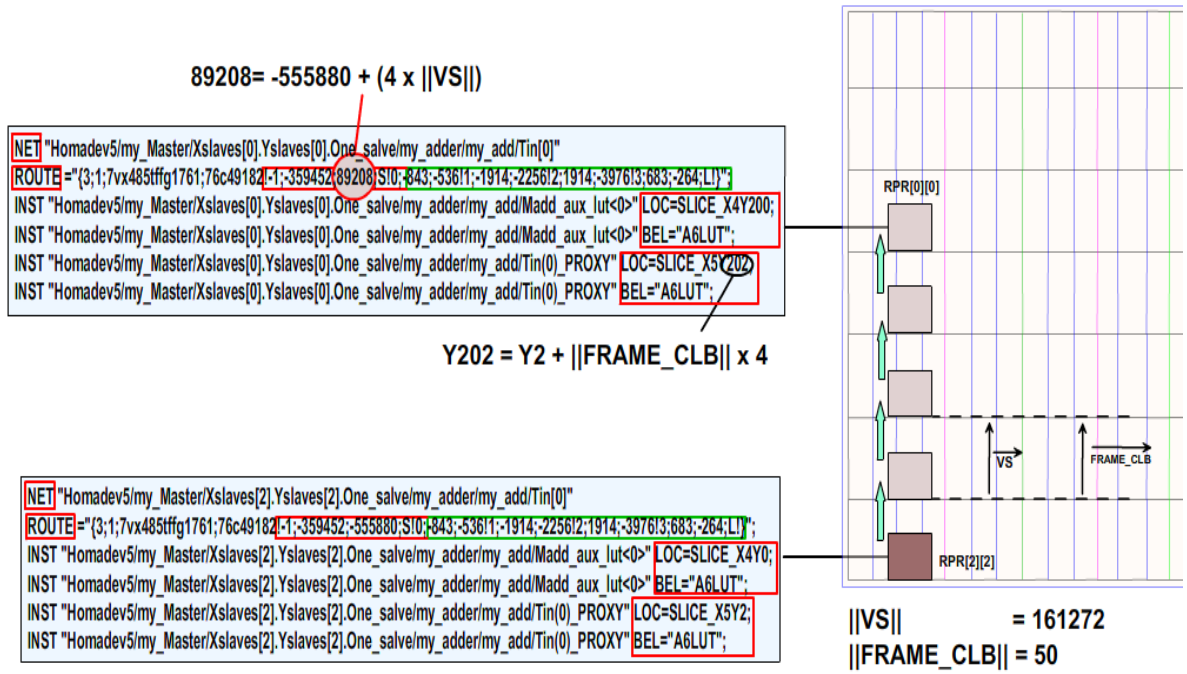
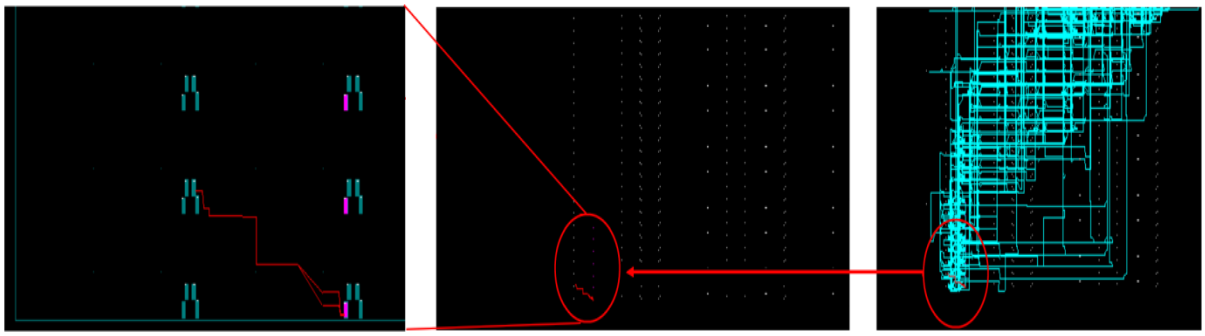
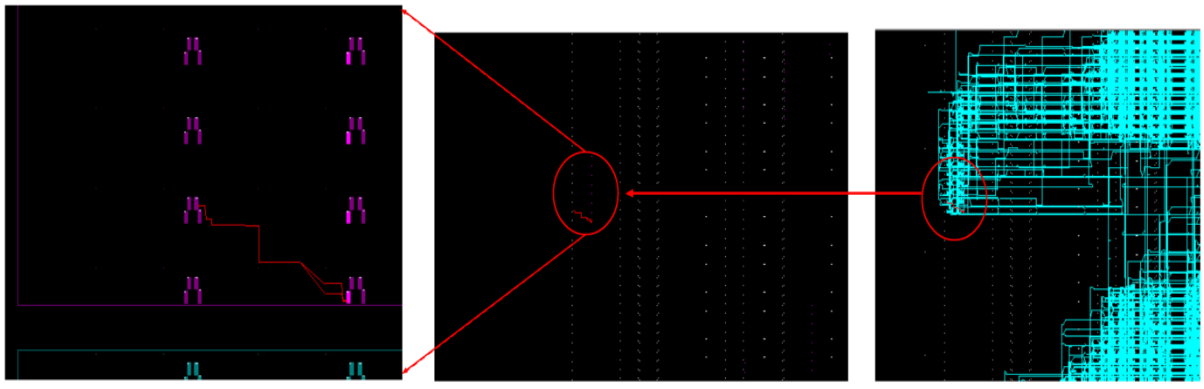


Figure 4.8 : Uniformisation de deux Proxys Logiques



(a) Routage du Proxy Logique avec la sortie Tin [0] dans la région RPR [2] [2]



(b) Routage du Proxy Logique avec la sortie Tin [0] dans la région RPR [0] [0]

Figure 4.9 : Résultat de l'uniformisation du routage des Proxys Logiques dans les régions RPR [0] [0] et RPR [2] [2]

La Figure 4.9 montre le résultat d'uniformisation de placement et routage des deux Proxys Logiques suite à la deuxième implémentation. En effet, nous remarquons que les chemins de routage entre les Proxy Logiques et les entrées T [0] sont identiques.

4.3.2 La relocalisation 2D

Notre démarche pour appliquer la relocalisation de bitstream partiel 2D se manifeste par uniformiser les informations de placement et routage de l'ensemble des régions partiellement reconfigurables à partir d'un ensemble de régions partiellement reconfigurables de référence compatibles à la relocalisation de bitstream 1D. L'ensemble de régions partiellement reconfigurables de référence est la colonne de RPRs déjà effectuée dans la section précédente. Donc, l'uniformisation s'effectue horizontalement à partir d'une colonne de référence vers l'ensemble de colonnes restantes appelées colonnes images comme le montre la Figure 4.10.

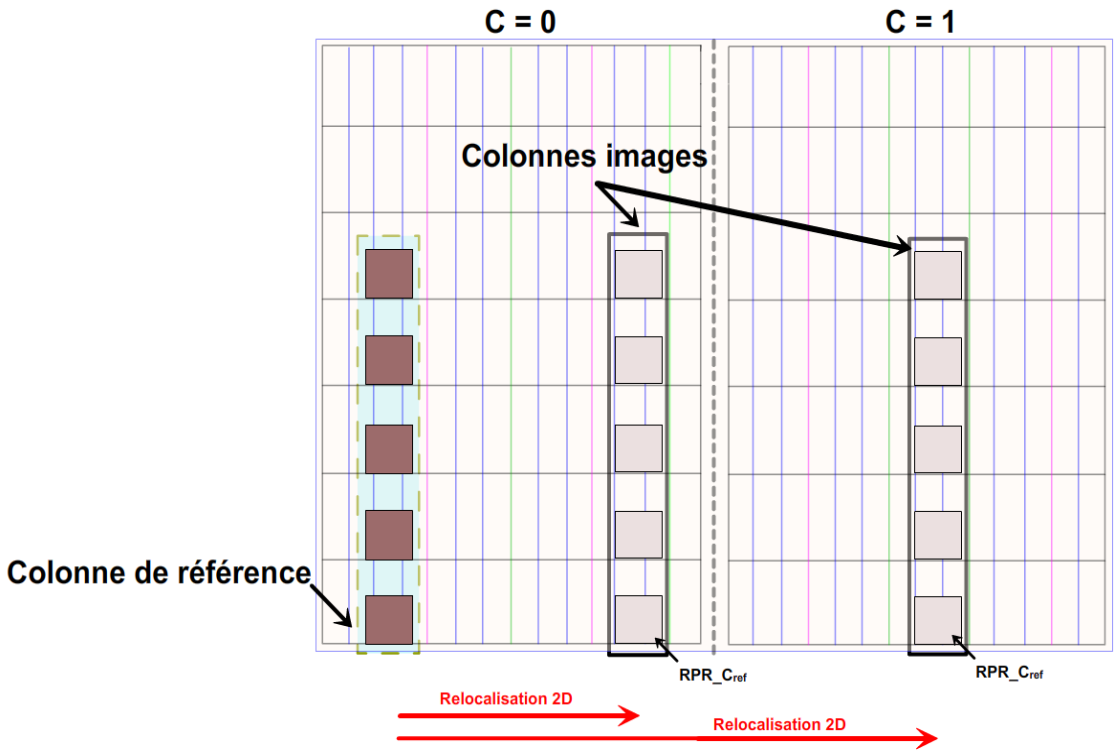


Figure 4.10 : Principe de relocalisation 2D

Suivant notre algorithme de floorplanning, les colonnes images sont réparties sur la largeur hétérogène de la plateforme FPGA. De plus, vu la dissymétrie des ressources fonctionnelles (CLB, DSP et BRAM) sur la largeur du FPGA, il n'est pas toujours évident de trouver un point de symétrie entre la colonne de référence les colonnes images. D'où l'utilité de

sauvegarder les informations de placement des RPR_C_{ref} résultants par notre algorithme de floorplanning dans les tableaux CLB_loc, BRAM_loc et DSP_loc. En effet, il existe Z-1 colonnes images dont chacune est identifiée par sa RPR_C_{ref} à travers laquelle il est possible d'en déduire le placement de toutes les régions partiellement reconfigurables d'une colonne image.

Dans ce cadre nous proposons une méthodologie générique basée sur la relocalisation 1D pour réaliser la relocalisation 2D. Cette méthodologie se compose de trois étapes essentielles qui sont détaillées comme suit :

- **Uniformiser le placement des Proxys Logiques des RPR_C_{ref} de chaque colonne image à partir de la RPR_C_{ref} de la colonne de référence :** Etant donné que les RPR_C_{ref} sont identiques en termes de forme et de composition, la première étape se manifeste par reproduire à partir de la RPR_C_{ref} de la colonne de référence les informations de placement des Proxys Logiques vers le reste des RPR_C_{ref} par simple clonage de positions. Le clonage se fait par la mise à jour des contraintes LOC et PIN au niveau de chaque Proxy. La Figure 4.11 illustre avec détails les transformations à effectuer pendent cette phase.

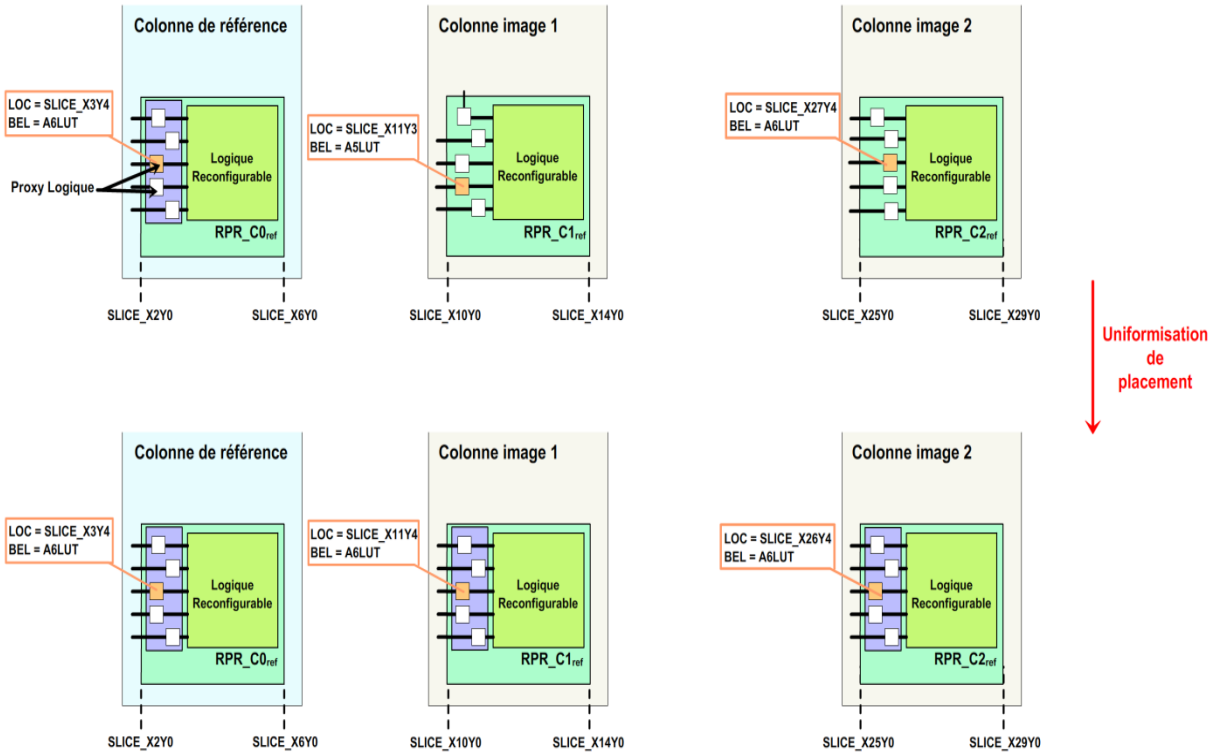


Figure 4.11 : Etape d'uniformisation de Placement des Proxys Logiques dans les RPR_C_{ref}

- **Uniformiser le routage des Proxys Logiques des RPR_C_{ref} de chaque colonne image à partir de la RPR_C_{ref} de la colonne de référence :** Durant cette étape l'uniformisation s'effectue au niveau de la description ROUTE de chaque Proxy Logique des RPR_C_{ref} images. Il suffit de remplacer l'adresse relative et la composante Y de l'adresse absolue de chaque Proxy Logique de chaque RPR_C_{ref} image par celles de la RPR_C_{ref} de référence. La composante X de l'adresse absolue de chaque RPR_C_{ref} est ensuite déterminée à partir d'un mécanisme de mappage d'adresse de routage bien spécifique que nous allons présenter dans la Section 4.3.3.2.
- **Reproduire la fonction de relocalisation 1D au niveau des Z-1 colonnes images :** Cette dernière étape s'effectue en répétant les étapes de mise en œuvre de la relocalisation 1D au niveau de chaque colonne image. Grace aux données de placement et routage déjà mises à jour de chaque RPR_C_{ref} de chaque colonne image, l'uniformisation verticale s'exécute en fonction du vecteur $\overrightarrow{Frame\_CLB}$ pour le placement et du vecteur $\overrightarrow{VS}$ dont la norme est déjà déterminée.

4.3.3 Automatisation de la relocalisation de bitstream partiel

La mise en œuvre de la technique de relocalisation de bitstream requière un effort énorme en termes de temps de conception qui engendre la diminution de la productivité que nous visons toujours à maximiser. A cet effet, nous proposons un flot de conception qui vise à automatiser la mise en œuvre de la technique de la relocalisation de bitstream pour les FPGA du constructeur Xilinx. Ce flot de conception permet de générer une description UCF qui contient toutes les informations physiques de placement et routage des éléments Proxys Logiques de chaque région partiellement reconfigurable associée à l'ensemble de ces interfaces statiques d'entrées et de sorties. La Figure 4.12 illustre les différentes étapes de mise en œuvre du flot d'automatisation de la relocalisation de bitstream.

Ce flot de conception est divisé en quatre phases essentielles présentées comme suit :

- Récupération des informations de P&R générées par la première implémentation
- La relocalisation 1D de la colonne de référence
- La relocalisation des colonnes images en fonction de relocalisation 1D
- La génération de la description physique de P&R des Proxys Logiques et leurs interfaces statiques correspondantes sous forme de contraintes.

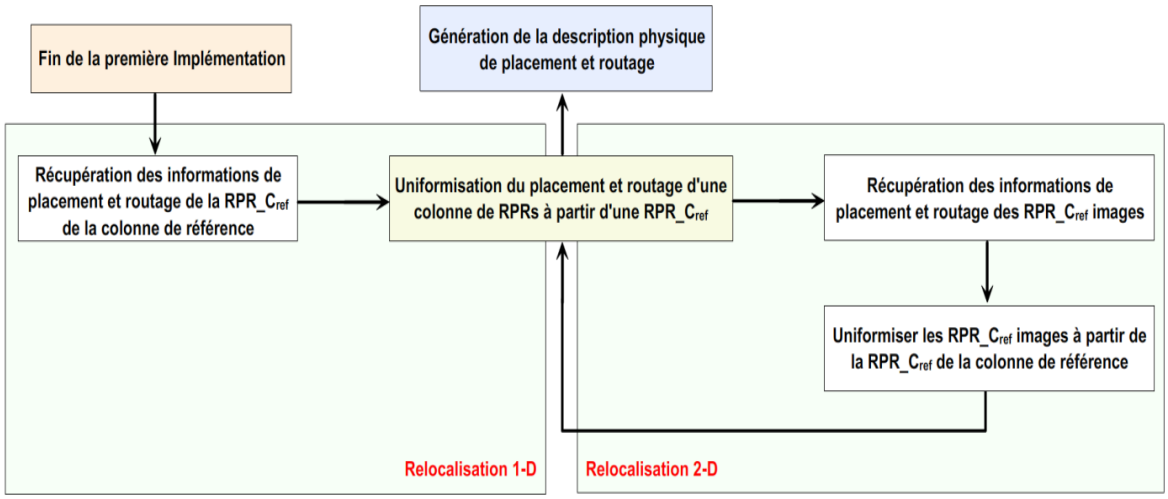


Figure 4.12 : Flot d'automatisation de la relocalisation de bitstream

La Figure 4.13 montre le déroulement de ce flot de conception dont nous allons détailler dans les paragraphes suivants :

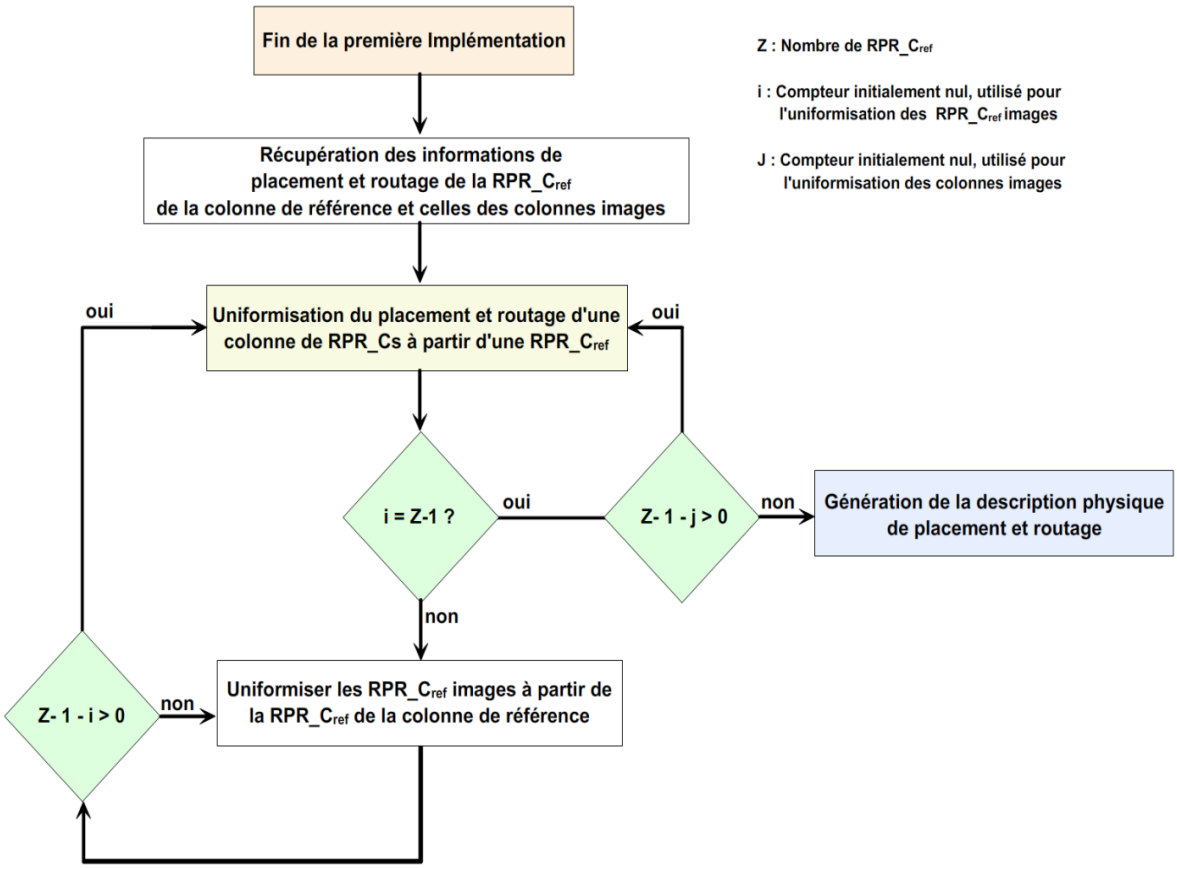


Figure 4.13 : Déroulement de la mise en œuvre d'automatisation de la technique de relocalisation de bitstream

4.3.3.1 Récupération des informations de P&R générées par la première implémentation

Suite à la première implémentation, le fichier NCD est le résultat du processus de MAP du flot de conception Xilinx. Ce fichier contient la description physique du système à implémenter. Ce fichier est ensuite utilisé par le processus de placement et routage (P&R ou PAR) pour générer un autre fichier NCD qui comporte les résultats de deux processus cité ci-dessus. Ce fichier nous permet d'analyser le placement et le routage des différents composants Proxys Logiques et les interfaces statiques de chaque région dynamiquement reconfigurable à l'aide des outils FPGA_Editor et DRC.

4.3.3.2 Récupération des informations de P&R de la RPR_C_{ref} en termes de Proxys Logiques de la colonne de référence et celles des colonnes images ainsi que leurs interfaces statiques associées

- **Récupération des informations de P&R de la RPR_C_{ref} en termes de Proxys Logiques de la colonne de référence et celles des colonnes images**

La phase de floorplanning nous a permis de générer les différents emplacements des régions partiellement reconfigurables disposées d'une manière régulières suite à l'étape de réplication. Ces emplacements sont stockés dans une structure de données sous forme de matrice d'objets. Chaque objet est constitué par les éléments nom, associé avec son hiérarchie depuis le module père, et ces coordonnées. Chaque coordonnée est le couple de CLBs (CLBL, CLBR) comme l'indique la Figure 4.14 qui représente la largeur d'une RPR_C. La Figure 4.14 illustre la composition de la matrice des RPR_Cs pour une configuration 3x3 d'une architecture Multi-RPRs implantée sur la plateforme FPGA Xilinx-Virtex7.

Les régions RPR_C_{ref} sont situées sur la ligne qui correspond à la position de l'Adj_CR_{ref} comme nous l'avons indiqué dans le chapitre précédent. Pour cette configuration Z = 2, donc nous avons deux colonnes dont celle de l'extrémité gauche est la colonne de relocalisation de référence tandis que les restantes sont les colonnes de relocalisation images qui serons traitées pendant la phase de relocalisation 2D. Donc, la RPR_$C0_{ref}$ est la région partiellement reconfigurable de la colonne de référence. Dans le but de déterminer les informations de placement et routage nous avons exploité le langage script (fichier SCR) utilisé pour l'outil FPGA_Editor qui sert à sélectionner l'ensemble des broches (NET) de chaque Proxy de chacune des RPR_C_{ref} ainsi que leurs interfaces d'E/S correspondantes. Etant donné que le

programme DRC génère les informations de P&R seulement pour les signaux sélectionnés, nous utilisons les commandes *Select* et *Unselect* [81]. La commande *Routecst* [81] est utilisée pour créer des données de P&R. Nous rappelons que nous avons mentionné précédemment que nous utilisons les modules partiellement reconfigurables à trois bus d'entrée de 32 bits et trois bus de sorties 32 bits avec leurs interfaces statiques correspondantes comme illustré dans la Figure 4.15.

Pour chacune des RPR_C_{ref} identifiée par l'attribut « nom » dans la matrice des RPR_Cs, nous générons un fichier UCF à partir du script illustré dans la List9. Le script comporte une suite de commandes cohérentes pour sélectionner exclusivement un par un chaque bus d'entrée ou de sortie et générer ces informations de P&R. Nous obtenons donc deux fichiers UCF relatifs à chaque RPR_C_{ref} dont chacun comporte les informations de P&R (P&R des Proxys Logiques) des entrées/sorties Tin, Nin, N2in, Tout, Nout et N2out.

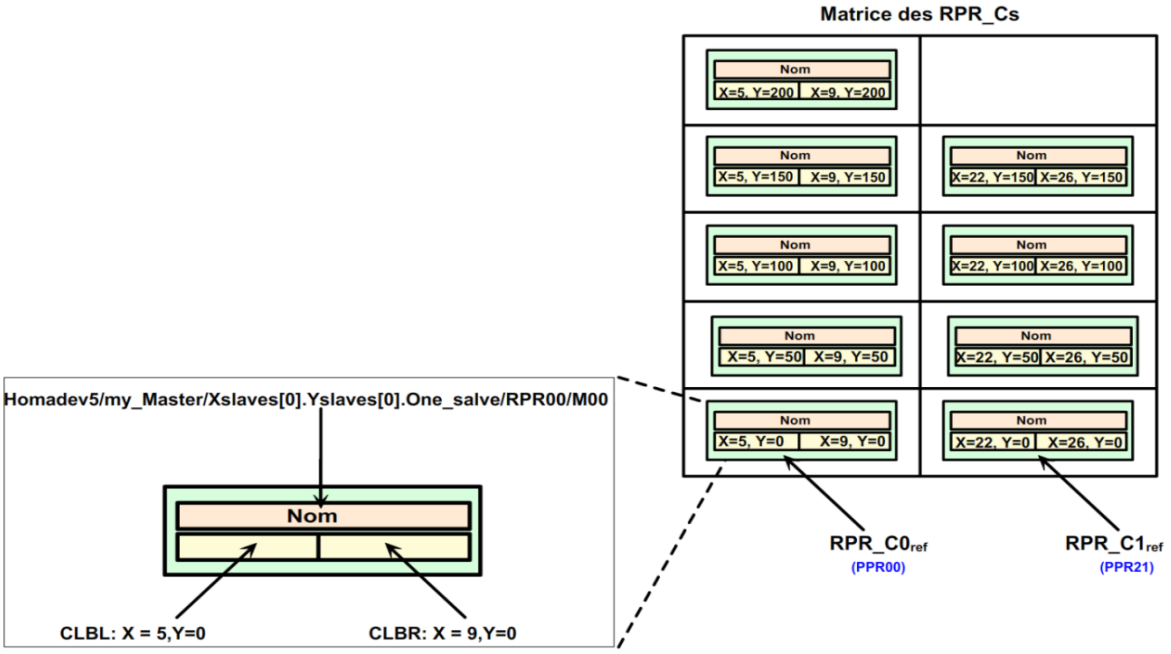


Figure 4.14 : La matrice des RPR_Cs

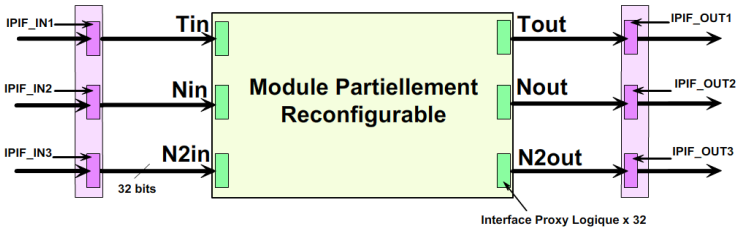


Figure 4.15 : Représentation de l'ensemble des régions partiellement reconfigurables avec les interfaces statiques

L'exécution du script par l'outil *FPGA_EDITOR* se fait en mode ligne de commande avec la commande suivante:

fpga_editor *–n* **<*nom_de _conception.ncd*>** **<*nom_de _conception.pcf*>** *–p* **<*script.scr*>**

Où l'option –n indique le nom des fichiers NCD et PCF (facultatif, le fichier de contraintes physiques) et l'option –p qui permet d'exécuter le script en mode playback.

```
                                                          Nom de la
                                                          RPR_Cref
unselect -all
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/Tin[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/Nin[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/N2in[*]          //Séléction des bus d'E/S
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/Tout[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/Nout[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR00/N2out[*]
routecst -l -cf Fichier_de_sortie_RPR_C0.ucf      // générer les information de P&R de la RPR_C0ref

unselect -all
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/Tin[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/Nin[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/N2in[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/Tout[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/Nout[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/RPR21/N2out[*]
routecst -l -cf Fichier_de_sortie_RPR_C1.ucf      // générer les information de P&R de la RPR_C1refimage

unselect -all
```

List9 : Script de génération des informations de P&R

Une autre façon d'exécuter ce script grâce à l'interface graphique de *FPGA_EDITOR* sous le menu ***Script -> Playback*** dans le menu ***Tools*** comme illustré dans la Figure 4.16.

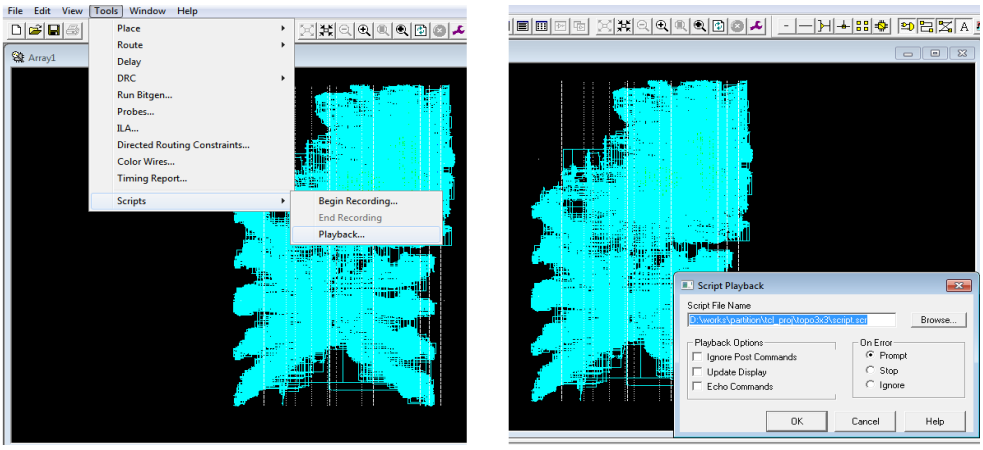


Figure 4.16: Exécution de Script sous FPGA_Editor

- **Récupération des informations de P&R de la RPR_C$_{ref}$ en termes d'interfaces statiques de la colonne de référence et celles des colonnes images**

Les interfaces statiques sont placées directement sur le bord intérieur droit de chaque région partiellement reconfigurable sous forme de modules statiques pendant la phase de floorplanning avant le début de la première implémentation. En s'appuyant sur la technique utilisée dans [55] [56], chaque interface consomme 16 LUTs (2 signaux par LUT). Ces interfaces sont réparties de manière régulière et symétrique au niveau de chaque Adj_CR et associées à chaque RPR_C en fonction du vecteur $\overrightarrow{Frame\_CLB}$. Leurs floorplanning se fait de manière automatique suite à la phase de floorplanning automatique des RPR_Cs. L'allocation et le floorplanning de ces interfaces se basent sur le même principe que l'algorithme AFLORA avec certaines différences :

- Ils prennent en considération les placements des Z x RPR_C$_{ref}$ à partir de la matrice des RPR_Cs.
- Ils ne prennent pas en considération l'étape de calcul de leur hauteur car leur taille est fixe en fonction d'un seul type de bloc fonctionnel (CLB).

La génération de leurs descriptions physiques de placement en fonction de leurs PBlocks respectifs sont associés avec les informations de contraintes physiques des régions partiellement reconfigurables avant d'exécuter la première implémentation du système. Les informations de placement des interfaces statiques sont stockées dans une matrice spécifique dont chaque élément, qui est un ensemble de contraintes de 6 interfaces, est associé à l'élément de la matrice des RPR_Cs par correspondance de coordonnées.

Au cours de la phase de relocalisation de bitstream, les interfaces statiques subissent la même démarche que celle des régions partiellement reconfigurables au niveau du placement (LOC et BEL) et du routage de leurs signaux d'interfaçage avec les Proxys Logiques. Les informations de placement sont extraites à partir des descriptions NCD et PCF en utilisant un script spécifique à exécuter à l'aide de l'outil FPGA_Editor. La List10 montre un exemple de script qui permet de générer les informations de P&R des LUTs appartenant aux 6 interfaces statiques IPIF_IN et IPIF_OUT de la RPR_C0$_{ref}$.

```
unselect -all
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_IN1_00/in[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_IN2/_00in[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_IN3_00/in[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_OUT1_00/out[*]
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_OUT2_00/out[*]      //Séléction des bus d'E/S
select Homadev5/my_Master/Xslaves[0].Yslaves[0].One_salve/IPIF_OUT3_00/out[*]
routecst -l -cf Fichier_de_sortie_IPIF00.ucf      // générer les information de P&R des interfaces statiques associées à la RPR_C0ref
```

List10 : Script de génération des informations de P&R des interfaces statiques

- **Mécanisme de mappage des adresses de routage**

L'irrégularité de placement des colonnes de RPR_Cs sur l'axe des X d'un FPGA Xilinx et la difficulté de récupérer les coordonnées de chaque entrée des composants logiques LUTs, appelés « NODE » suivant l'outil FPGA_Editor, posent un obstacle devant la récupération automatique des adresses de routage absolues de chaque signal (NET) au niveau des Proxys Logiques et des LUTs des interfaces statiques avant d'exécuter la deuxième implémentation. D'une part, l'irrégularité de placement des colonnes de RPR_Cs ne conduit pas à déterminer un vecteur de translation de norme constante (pour éviter des calculs intermédiaires) qui permet de déduire les informations de routage de chaque NET d'une RPR_C image à partir d'une RPR_C de la colonne de référence pendant la phase de la relocalisation 2D. En revanche, l'outil FPGA_Editor ne dispose pas de commandes en ligne qui permettent de générer les informations détaillées sur les Matrices Générales de Routage (GRM [61], [62]) de l'FPGA. Cependant, les blocs de routage (Switch Box) que nous pouvons voir sur l'interface de l'outil ne génèrent que les coordonnées de routage des NODEs (adresse absolue d'un signal) par simple sélection ou suivant la commande en ligne *Select*. Ces NODEs sont les signaux de connexion qui relient un bloc de routage avec une paire de blocs fonctionnels SLICEs (1 x CLB) comme le montre la Figure 4.17. Dans le but de résoudre ce problème présenté par ces deux aspects, nous avons cherché à développer un mécanisme de mappage des adresses de routage qui permet de déterminer les adresses absolues de chaque NODE. Ce mécanisme se base sur la considération d'un bloc élémentaire composé de 4 CLBs adjacents sur l'axe des X du FPGA avec Y = 0 et dont nous calculons les différences d'adresses absolues en passant d'un bloc CLB à un autre. L'outil FPGA_Editor représente cette suite de 4 CLBs sous forme de 2 blocs de CLBs séparés dont chacun est connecté à 2 blocs de routage gauche et droit comme illustré dans la Figure 4.17. En effet, ce mécanisme s'applique pour que le premier sous-bloc de CLB appartienne à une région dynamiquement reconfigurable sur la colonne CLBR de son extrémité droite où les Proxys Logiques doivent être placés, tandis

que le deuxième sous-bloc appartient à l'une des interfaces statiques associée à cette RPR_C. Ceci justifie la courte distance entre une RPR_C ces interfaces statiques associées.

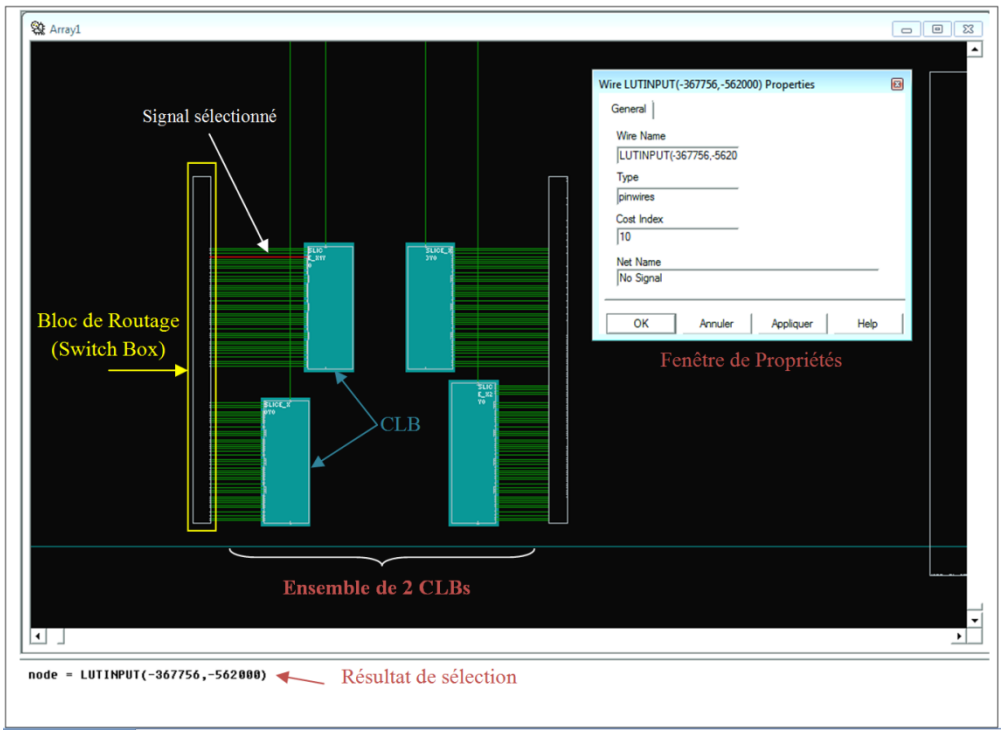


Figure 4.17 : Représentation des informations de routage disponibles pour un signal suite à une action de sélection

Dans une première étape, nous considérons 4 blocs de CLBs adjacents et nous identifions chaque SLICE de chaque CLB par les adresses absolues de la sortie O6 du LUT 'D'. Cette sortie s'intitule suivant l'outil FPGA_Editor par OUTPUT comme l'indique la Figure 4.18. Etant donné que chaque LUT comporte 2 sorties (O6 et O5), cette identification permet de déterminer automatiquement les adresses absolues de toutes les sorties des 4 LUTs d'un même SLICE. La Figure 4.18 illustre un balayage des adresses absolues de chaque sortie des 4 LUTs faisant partie du SLICE_X0Y0 de la technologie Xilinx-Virtex7. Suivant cette Figure nous remarquons bien que la Différence d'Adresse Absolue (DAA) entre les sorties d'un même LUT reste constante et égale à (0,-8), tandis que la DAA entre les sorties O6 de chaque LUT est égale à (0,-104). Nous remarquons aussi que la la valeur de la cordonnée Y des NODEs reste toujours constante puisqu'il s'agit d'un même SLICE. Donc, nous pouvons confirmer qu'il suffit de connaitre l'adresse absolue de la sortie O6 du LUT 'D' qui est égale à (-367916,-562536) pour en déduire les adresses absolues des sorties restantes du même

SLICE. Maintenant, essayons de chercher si une corrélation existe entre 2 SLICEs du même CLB.

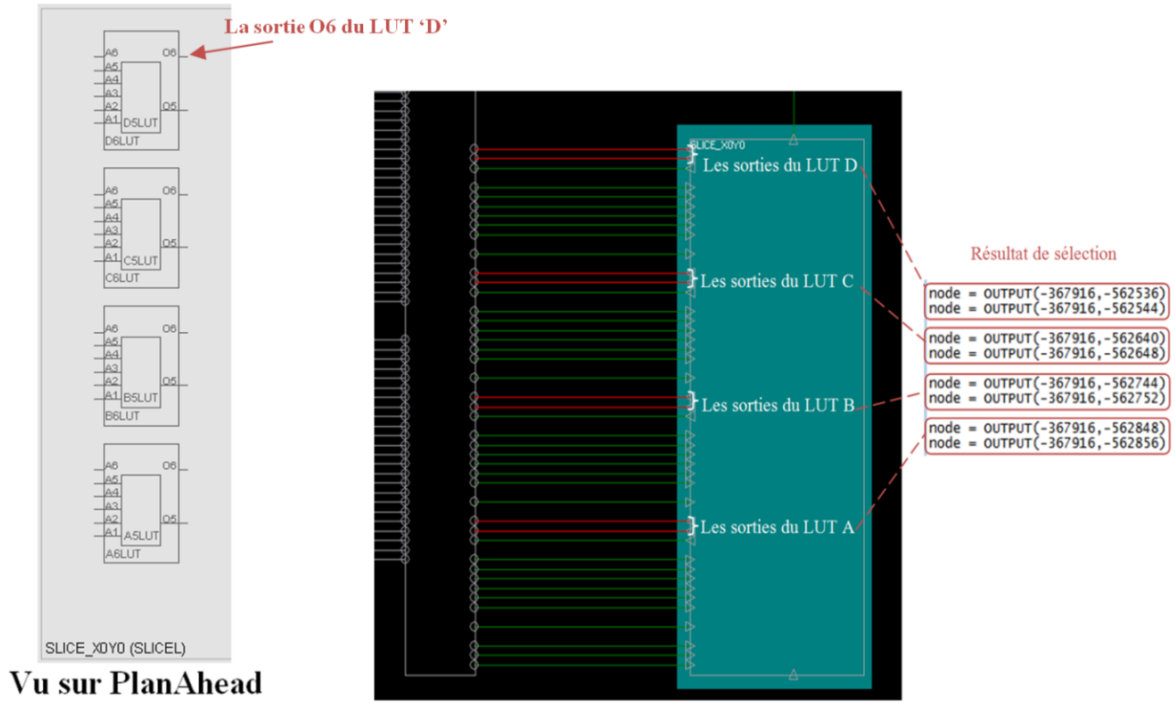


Figure 4.18 : Balayage des adresses absolues des sorties du SLICE_X0Y0

Suivant la Figure 4.19, nous avons sélectionné dans l'ordre les sorties O6 des SILICEs : SLICE_X0Y0 et SLICE_X1Y0 qui appartiennent au CLB_X1Y0 puis celles des SLICEs : SLICE_X2Y0 et SLICE_X3Y0 qui appartiennent au CLB_X2Y0. Le résultat de la sélection est affiché dans la barre d'affichage de FPGA_Editor. Nous remarquons que le DAA entre 2 SLICEs d'un même CLB est égale à (160,568). Ce qui revient à prouver que si nous disposons de l'adresse absolue d'une sortie O6 d'un LUT 'D', nous pouvons déterminer les adresses absolues de toutes les sorties d'un même CLB en fonction des DAAs : (0,-8), (0,-104) et (160,568). Jusqu'à cette phase, nous n'avons pas trouvé une irrégularité de DAAs des sorties dans un même CLB dû à la régularité de la structure FPGA suivant l'axe des Y. En revanche, l'irrégularité d'adressage se trouve suivant l'axe des X dû à l'hétérogénéité des composants qui s'y trouvent (CLB, BRAM, DSP). Mais en traitant seulement les suites de bloc CLBs nous remarquons qu'il existe une corrélation entre chaque série de blocs composés de 4 CLBs. Ceci justifie aussi notre considération pour traiter chaque 4 CLBs comme étant un bloc élémentaire. En effet, nous allons montrer la régularité des DAAs entre chacun de ces blocs élémentaires comme suit :

1. Nous considérons l'ensemble des 4 CLBs : CLB_X1Y0, CLB_X2Y0, CLB_X3Y0 et CLB_X4Y0.
2. Identifier les DAAs entre les différentes sorties O6 de chaque LUT.
3. Vérifier que les DAAs restent constants comparés aux blocs élémentaires suivants.

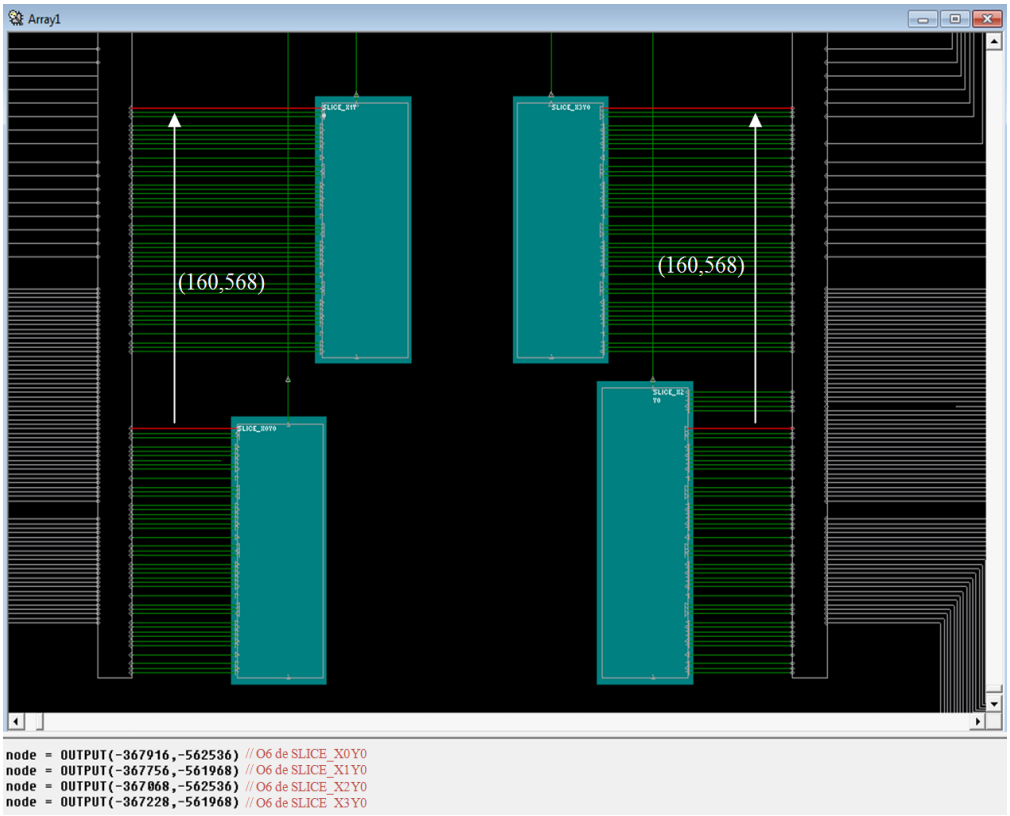


Figure 4.19 : Détermination de la DAA entre deux SLICEs du même CLB

Dans la première phase nous déterminons suivant l'axe des X la DAA entre les sorties O6 des SLICEs SLICE_X1Y0 (CLB_X1Y0) et SLICE_X2Y0 (CLB_X2Y0), ensuite la DAA entre les sorties O6 des SLICEs SLICE_X5Y0 (CLB_X3Y0) et SLICE_X6Y0 (CLB_X4Y0). Ceci pour déterminer l'irrégularité qui se trouve entre les valeurs des DAAs au niveau de la composante X. Enfin, nous déterminons la DAA entre les deux sous-blocs de paire de CLBs. Cette dernière se calcule entre les sorties O6 du LUT 'D' des SLICEs SLICE_X0Y0 et SLICE_X4Y0. Cette DAA représente le pas d'adressage à partir de laquelle nous pouvons déterminer l'adresse absolue image (@O6 du SLICE_X4Y0) en partant de l'adresse absolue origine (@O6 du SLICE_X0Y0). La Figure 4.20 montre les résultats de calcul des DAAs pour un bloc élémentaire de 4 CLBs. Elle montre aussi le résultat de calcul de la DAA entre les entre les deux sous-blocs de paire de CLBs qui est égal à (8304,0).

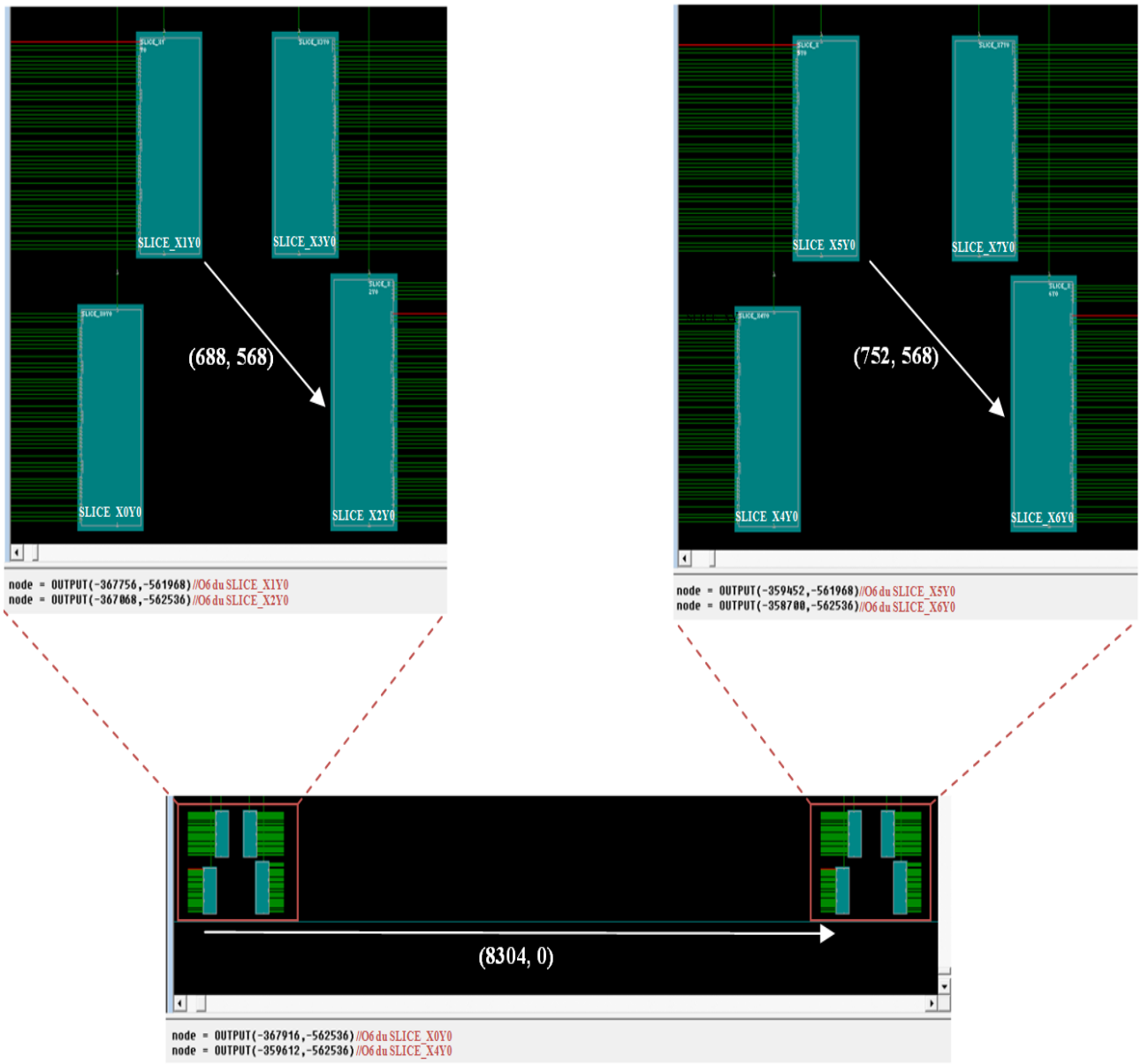


Figure 4.20 : Calcul des DAAs pour un bloc élémentaire de 4 CLBs

Enfin, dans le but de s'assurer que les DAAs restent constants pour chaque bloc élémentaire, nous avons parcouru la largeur de l'FPGA (Xilinx-Virtex7 dans notre cas) avec l'outil FPGA_Editor et nous avons vérifié que ces dernières restent inchangées pour chaque suite de 4 CLBs.

D'une manière générale, nous pouvons constater qu'un bloc élémentaire peut être identifié par une seule adresse absolue. En effet, cette adresse permet de déterminer les adresses absolues de toutes les sorties O6 et O5 d'un ensemble de 4 CLBs. Ceci permet le P&R visant à connecter 32 signaux différents dont 16 sont spécifiques pour les Proxys Logiques et 16 pour l'une des interfaces statiques.

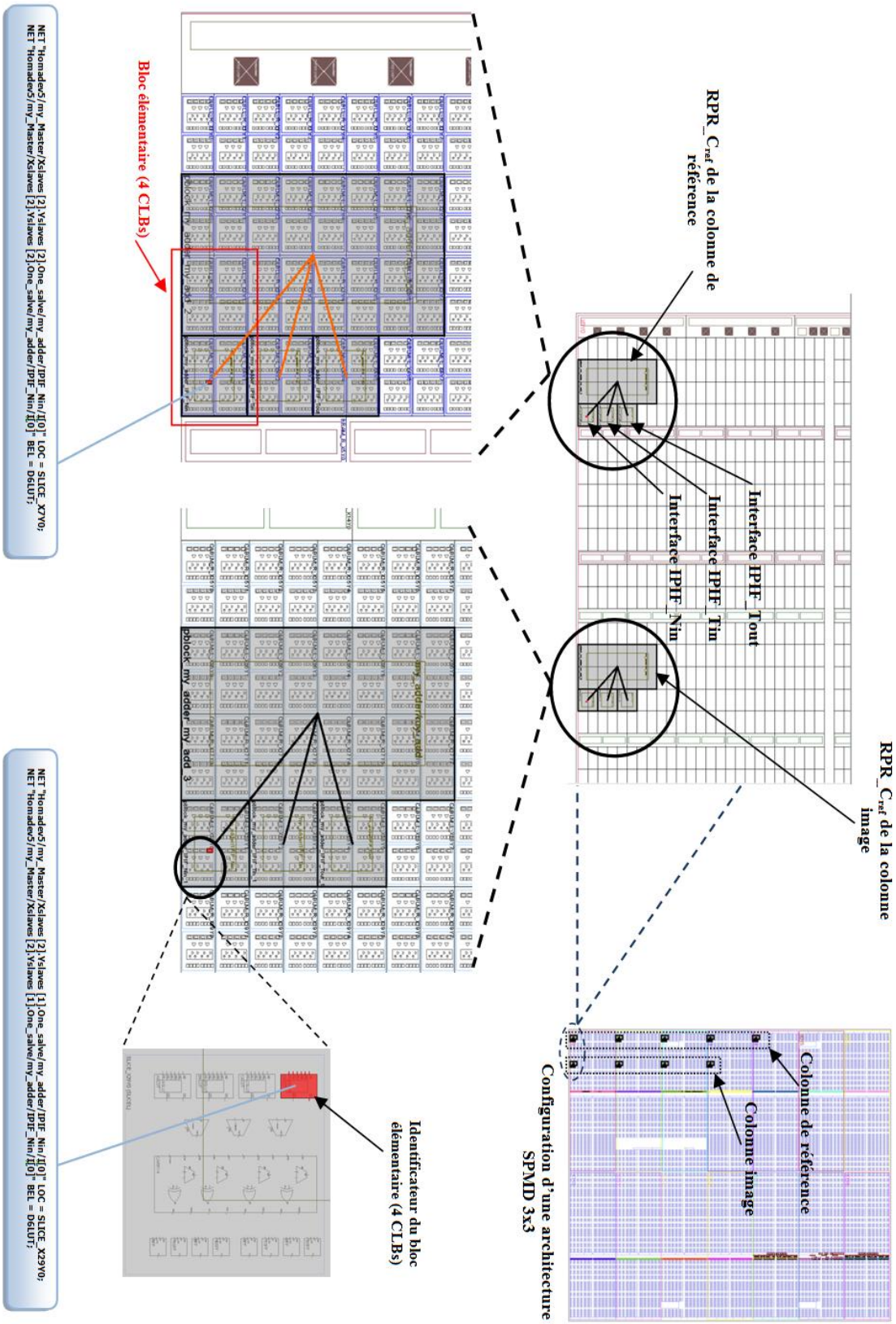


Figure 4.21 : Mise en œuvre des identifications des blocs élémentaires CLBs

Ce mécanisme s'effectue en fonction des DAAs que nous avons identifiées et qui forment le mappage des adresses absolues des sorties des LUTs. Un autre obstacle se présente et qui se manifeste par la manière de récupérer automatiquement l'adresse absolue qui identifie le bloc élémentaire de 4 CLBs sans procéder à l'identification manuelle sur FPGA_Editor.

Pour ce faire, nous intervenons avant de commencer la première implémentation. Il suffit de mettre une contrainte de placement sur un LUT faisant partie d'une interface statique associée à une RPR_C_{ref}. L'interface considérée doit englober deux CLB d'ordonné Y = 0. Ce signal est placé exactement dans un SLICE de l'extrême droite de l'interface statique ayant la composante Y = 0, et au niveau de la sortie O6 du LUT 'D'. Donc, un total de Z contraintes de placement supplémentaires s'ajoute à la description UCF initiale. La Figure 4.21 illustre avec détails la mise en œuvre des blocs élémentaires de 4 CLBs. En utilisant le FPGA Xilinx Virtex7 pour implémenter une architecture Multi-RPRS de taille 3x3, nous avons spécifié une région dynamiquement reconfigurable au sein de chaque unité de calcul. Chacune de ces RPR_C comporte 2 bus d'entrée connectés aux interfaces d'entrée IPIF_Nin et IPIF_Tin et un bus de sortie connecté à l'interface de sortie IPIF_Tout. Nous avons opté de placer les interfaces statiques d'entrées IPIF_Nin sur la ligne de CLB d'ordonné = 0 associées à leurs RPR_C_{ref} correspondantes. Chaque interface statique requière donc 2 CLBs (= 4 x SLICEs = 16 LUTs), en effet, chacune d'elle nécessite l'instanciation d'un PBlock de largeur égal à 2 SLICEs et de même pour sa hauteur. En considérant les blocs élémentaires de 4 CLBs, nous allouons les deux CLBs de l'interface statique IPIF_Nin et les deux CLBs adjacents qui appartiennent à la RPR_C_{ref} pour les Proxys Logiques. Enfin, nous spécifions une contrainte de placement pour l'une des signaux d'entrée de cette interface (I [0] suivant la Figure 4.21) dans le LUT 'D' du SLICE de l'extrémité droite de l'interface statique ayant la composante Y = 0 (selon la Figure 4.21, le SLICE_X7Y0 pour la RPR_C_{ref} de la colonne de référence et le SLICE_X29Y0 pour la RPR_C_{ref} de la colonne image) au niveau de la sortie O6 du LUT. La spécification de ce placement s'effectue identiquement au niveau de chaque interface IPIF_Nin associée à chaque RPR_C_{ref}.

Cette mise en ouvre s'effectue automatiquement suite au placement des interfaces statiques. Suite à la première implémentation, cette mise en œuvre nous permet facilement de récupérer l'adresse absolue de la sortie O6 du LUT 'D' que nous avons spécifiée à l'aide du programme DRC sous FPGA_Editor grâce aux commandes en ligne ***select*** et ***routecst*** en spécifiant toutes les hiérarchies des signaux I [0] des interfaces statiques IPIF_Nin.

4.3.3.3 Uniformisation d'une colonne de RPR_Cs à partir de RPR_C_{ref}

Cette étape prend en entrée un fichier de contraintes qui contient les informations de P&R des Proxys Logiques déjà générés à la fin de l'étape précédente. En effet, chaque fichier comporte 96 informations de P&R relatives à chaque signal d'entrées/sorties. Suivant notre flot de conception, cette étape traite premièrement le fichier relatif à la région RPR_$C0_{ref}$ pour effectuer la relocalisation 1D au niveau de la colonne de référence. Dans une seconde étape, suite à l'uniformisation du P&R des RPR_Cs_{ref} des colonnes images à partir de la RPR_$C0_{ref}$, nous effectuons la relocalisation 1D d'une manière séquentielle au niveau des colonnes images à partir des fichiers de contraintes physiques relatifs aux RPR_C_{ref} restants (le fichier généré relatif à la région RPR_$C1_{ref}$ pour cet exemple).

Cette étape considère également le nombre des RPR_Cs situées sur la même colonne pour savoir le nombre de translations et d'uniformisations à réaliser. Elle comporte deux sous-étapes qui consistent à effectuer l'uniformisation des RPR_Cs sur la même colonne, puis placer uniformément les interfaces statiques sur les bords des RPR_C à partir de la RPR_C_{ref} associée.

- **L'uniformisation des RPR_Cs sur la même colonne**

Cette tâche consiste premièrement à extraire les informations de placement et routage se trouvant dans le fichier de contraintes physiques. L'extraction s'effectue grâce à un mécanisme de traitement de fichier développé avec un langage orienté objet (Java). En effet, ces informations sont classées suivant un diagramme de classes (UML 2.0) qui facilite leur traitement d'une manière générique pour effectuer les mécanismes de translation des placements et l'uniformisation des routages. Les mécanismes de translation et d'uniformisation, comme l'indique la Figure 4.18, sont assurés par la fonction *Translater (Frame_CLB, LOC)* et la fonction *Uniformiser (Route)* qui font partie d'association *Uniformisation* qui assure la relation entre la classe *Signal de référence* et la classe *Matrice des RPR_Cs*. En effet, la projection du signal de référence dans chaque RPR_C image (mise à jour du nom du signal) s'effectue par le recalcul, en fonction du vecteur $\overrightarrow{VS}$, du paramètre *@y*, qui représente la composante Y de l'adresse absolue du signal à traiter et fait partie de la classe *ROUTE*, et les paramètres de placement *LOC* et *Bel* de la classe *Signal de référence* à translater suivant la norme du vecteur Frame_CLB. Cette association permet enfin de construire les informations de P&R du signal image et les générer sous forme de contraintes dans un nouveau fichier UCF.

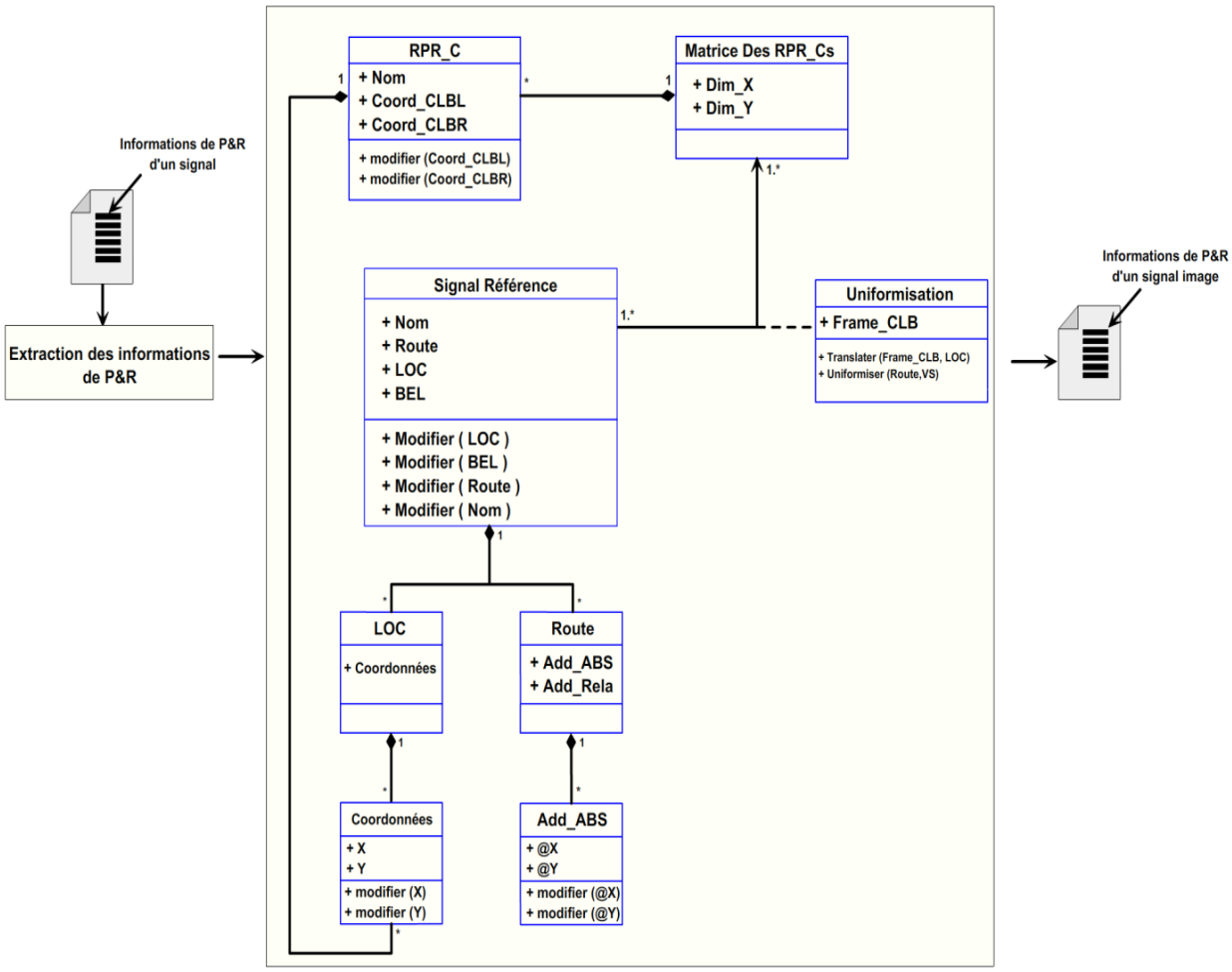


Figure 4.18 : Mécanisme d'uniformisation des RPR_Cs sur la même colonne

- **Placement uniforme des interfaces statiques**

Les interfaces associées au RPR_C_{ref} sont considérées comme des interfaces statiques de référence ($IPIF_{ref}$). Elles sont utilisées pour appliquer l'uniformisation des placements des autres IPIFs se situant sur la même colonne en se basant sur les fonctions de translations à l'aide du vecteur $\overrightarrow{Frame\_CLB}$. En effet, le mécanisme de relocalisation 1D prend en compte l'uniformisation des interfaces par l'extraction des informations de placements contenus dans le fichier de description des contraintes de placement (*Fichier_de_sortie_IPIF00.ucf* généré suite à l'exécution du script présenté dans List10). Ensuite la classification de ces contraintes, puis leurs reproductions à l'aide du mécanisme de translation. Enfin la génération des informations de placement de l'interface statique image. Les informations produites sont introduites dans un nouveau fichier de description physique.

4.3.3.4 Uniformiser les RPR_Cs$_{ref}$ images à partir de la RPR_C$_{ref}$ de la colonne de référence

Cette étape s'effectue sur deux niveaux, le premier au niveau du placement des Proxys Logiques à l'aide des opérations de translation et le deuxième au niveau de l'uniformisation de routage. Cette étape englobe aussi l'uniformisation des interfaces statiques relatives à chaque RPR_C$_{ref}$ image.

Au niveau du placement, cette étape commence par calculer la norme du vecteur $\overrightarrow{VH}$ de translation horizontale entre la RPR_C$_{ref}$ de la colonne de référence et chaque RPR_C$_{ref}$ image. Ceci s'effectue en utilisant la matrice des RPR_Cs. En effet, la norme est la distance qui sépare la limite droite de RPR_C$_{ref}$ de la colonne de référence et la limite droite de la RPR_C$_{ref}$ image comme illustré dans la Figure 4.19.

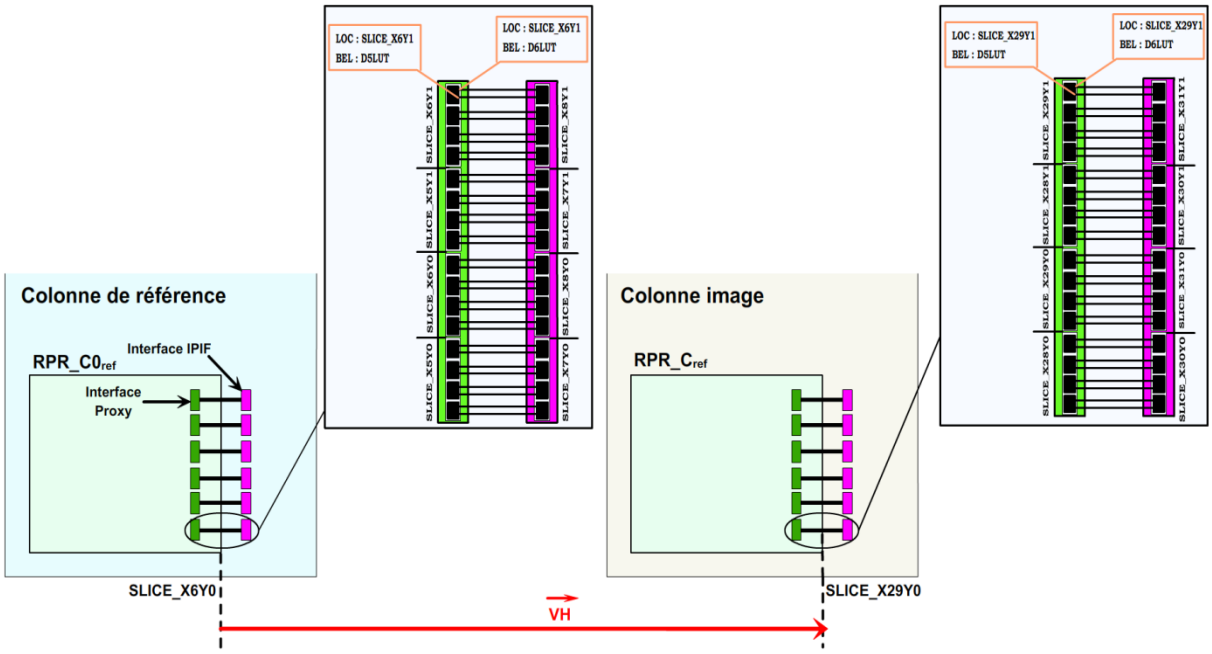


Figure 4.19 : Uniformisation des RPR_C$_{ref}$ images et leurs interfaces statiques associées

Ce vecteur permet de repositionner chaque Proxy Logique et chaque élément LUT appartenant à une interface statique, techniquement, il s'agit de recalculer le paramètre LOC de chacun de ces composants au niveau de la composante X du SLICE. D'autre part les valeurs du paramètre BEL restent inchangées en gardant celles de la RPR_C$_{ref}$ de la colonne de référence. Vu l'hétérogénéité des composants suivant l'axe des X de l'FPGA, la norme du vecteur $\overrightarrow{VH}$ varie en fonction du placement des RPR_Cs$_{ref}$.

Au niveau du routage, les signaux qui sont routés entre les interfaces des Proxys Logiques et les interfaces statiques au niveau de chaque RPR_C_{ref} images gardent les mêmes informations de routage que celles de la RPR_C_{ref} de la colonne de référence excepté pour la valeur de la composante X de l'adresse absolue. En effet, la mise à jour de cette composante s'effectue à partir du mappage d'adresse de routage que nous avons détaillée précédemment.

4.3.3.5 Uniformisation du placement et routage des RPR_Cs à partir d'une RPR_C_{ref} (Relocalisation 1D)

Comme nous avons vu dans la Section 4.3.3.2, la fonction de relocalisation 1D s'effectue dans cette étape au niveau des colonnes images en fonction des informations de placement et routage des RPR_C_{ref} respectives à l'aide des vecteurs de translation de placement et de routage respectivement $\overrightarrow{Frame\_CLB}$ et $\overrightarrow{VS}$. Cette étape s'effectue Z-1 fois tout en générant la description physique du P&R des Proxys Logiques et celle des interfaces statiques une fois que le traitement d'une colonne image est effectué. L'ensemble de ces descriptions est ensuite intégré dans le fichier de contraintes du système pour être à nouveau prêt à l'implémentation finale du système.

4.4 Conclusion

Nous avons présenté dans ce chapitre les détails de résolution des deux problématiques de placement et routage des RPR_C et leurs interfaces statiques associées pour aboutir à la relocalisation 1D et 2D de bitstream partiel. Ceci a été effectué grâce à des vecteurs de translation au niveau du placement et du routage ainsi qu'au mécanisme de mappage d'adresse absolue de routage. Nous avons également proposé un flot de conception qui vise à automatiser les étapes aboutissant à favoriser la technique de la relocalisation de bitstream partiel.

Dans le chapitre suivant, nous proposons l'architecture adéquate du contrôleur de reconfiguration dynamique dédié pour le broadcast de bitstream partiel.

CHAPITRE 5 : Contrôleur de la reconfiguration dynamique dédié au broadcast de bitstream partiel

5.1 Introduction

Dans le chapitre précédent, nous avons présenté la méthodologie d'automatisation de la technique de relocalisation de bitstream et sa réutilisation dans le but de favoriser la diffusion de bitstream partiel vers un ensemble des régions partiellement reconfigurables. Pour permettre l'exploitation de ce mécanisme dans les architectures multi-régions dynamiquement reconfigurables, nous avons proposé une architecture adéquate à cet effet, pouvant s'adapter rapidement à un large spectre d'applications de calcul reconfigurable. Nous avons opté pour une implémentation favorisant une RDP rapide s'appliquant sur les FPGA-Xilinx. L'architecture de l'accélérateur matériel comportant trois interfaces essentielles qui sont : l'interface d'écriture/lecture des bitstreams partiels dans la mémoire de bitstreams, l'interface processeur qui permet son pilotage et l'interface ICAP qui permet l'accès à l'espace reconfigurable du FPGA. Pour pouvoir développer une telle architecture, il est nécessaire de connaitre la structure d'un bitstream partiel et d'en extraire les paramètres essentiels tels que son emplacement dans l'espace reconfigurable, sa longueur, les registres de contrôle,…

Dans ce chapitre, nous présentons l'implémentation de l'architecture de l'accélérateur matériel de la RDPP. Il est structuré comme suit : La Section 5.2 présentera la méthode de conception à base d'assemblage de modules. L'expérimentation sur FPGA et l'évaluation des performances seront discutées dans la Section 5.3. Une étude comparative entre l'accélérateur matériel proposée et d'autres architectures sera présentée dans la Section 5.4. Finalement, la dernière section donnera une synthèse de ce chapitre.

5.2 Le Contrôleur de la RDP dédié pour le broadcaste de bitstream partiel

Nous proposons dans ce cadre, un contrôleur dédié pour la reconfiguration dynamique partielle parallèle utilisant une mémoire de bitstream externe de type DDR3. Pour permettre son utilisation, notre contrôleur de la RDPP dispose de trois interfaces essentielles qui sont présentées par la suite et illustrées dans la Figure 5.1.

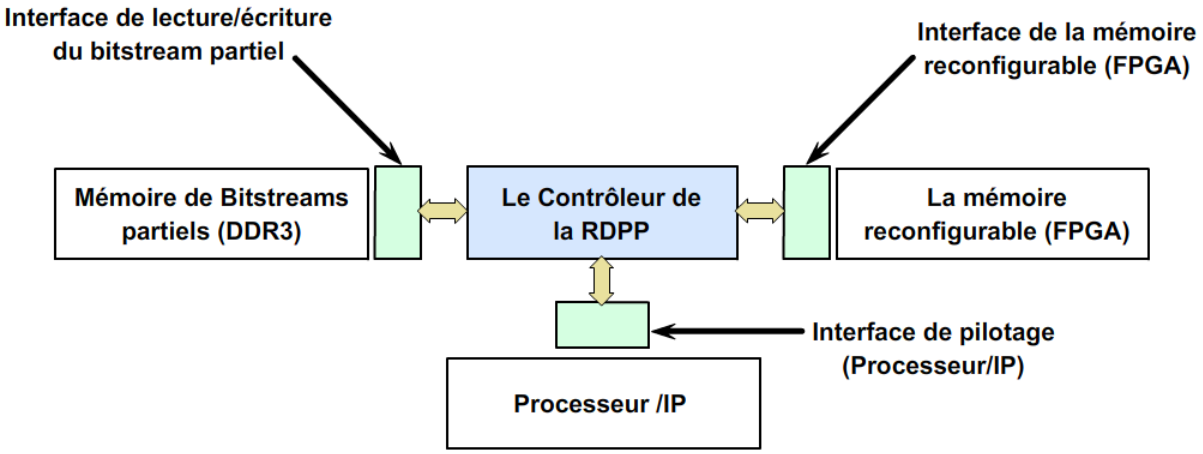


Figure 5.1 : Les interfaces du contrôleur de la RDPP

5.2.1 Les interfaces du contrôleur de la RDPP

5.2.1.1 L'interface de lecture/écriture du bitstream partiel et des emplacements des RPRs sur FPGA

Cette interface permet l'interconnexion de la mémoire de bitstream (DDR3) et le contrôleur de la RDPP à travers un contrôleur dédié. Ce contrôleur (MIG Controller) est généré à partir de l'outil Xilinx MIG (Memory Interface Generator) [86] qui prend en compte le type du composant DDR3 se trouvant sur la carte imprimée Xilinx. Cette interface comporte un ensemble de signaux de control, des bus de données d'entrée et de sortie et deux bus d'adresses pour l'écriture et la lecture des données. La largeur des bus de données varient en fonction des composants mémoires DDR3 avec différents modes de transferts en rafale (4, 8, 16, …). La phase d'écriture dans la mémoire DDR3 des bitstreams partiels communs relatifs à chaque configuration est la première phase effectuée. Elle s'effectue entre l'entrée série UART et l'interface d'écriture des bitstreams. Le contrôle des flux d'écriture est réalisé à travers un ensemble de modules que nous allons détailler dans les Sections suivantes. Ces modules permettant l'accumulation et la synchronisation des transferts. Une deuxième phase d'écriture s'effectue ensuite entre l'entrée série UART et la mémoire des emplacements des RPRs sur FPGA (FAR Memory). Différents niveau de synchronisation se présentent au cours de ces deux phases. En effet, plusieurs domaines d'horloges sont synchronisés. La mémoire DDR3 est fonctionnelle à une fréquence de 800 Mhz tandis que le contrôleur de la RDPP fonctionne à 200 Mhz. Le contrôleur MIG d'écriture/lecture permet en effet la synchronisation entre ces deux domaines d'horloges comme indiqué dans la Figure 5.2.

5.2.1.2 L'interface de la mémoire reconfigurable FPGA

La mémoire reconfigurable du FPGA est accessible exclusivement en cours d'exécution à travers le port ICAP. Notre interface est conçue pour contrôler et transférer les données de configuration vers les régions dynamiquement reconfigurables situées dans un emplacement précis. Le bus de données de l'ICAP est de largeur 32 bits dans les FPGAs récents. Deux signaux de contrôle sont utilisés pour gérer le transfert des mots de configuration qui sont CE et WR.

5.2.1.3 L'interface de pilotage (Processeur/IP)

Cette interface permet de piloter le processus de la reconfiguration dynamique parallèle à travers un ensemble de signaux. Comme indiqué dans la Figure 5.2, elle se base sur les deux signaux Start_Reconfig et Reconfig_Done qui limitent la phase de reconfiguration. En effet, le début du processus est indiqué par l'activation du signal Start_Reconfig. La fin de la reconfiguration est indiquée par la mise à un du signal Reconfig_Done. Lors de l'activation du signal Start_Reconfig, une adresse spécifique est fournie sur le bus Start_Address. C'est l'adresse initiale d'un bitstream se trouvant dans la mémoire DDR3. Un autre bus est spécifié dans cette interface pour gérer l'activité des RPRs. En effet, ce bus permet d'envoyer un masque d'activité par le processeur ou l'IP qui gère la RDPP pour permettre ou non la reconfiguration d'une RPR par un bitstream. Cependant, ce masque permet de reconfigurer un sous-ensemble de RPRs par un bitstream réalisant une fonctionnalité donnée. Il permet aussi de reconfigurer le reste des RPRs avec un autre bitstream possédant une autre fonctionnalité. Ce qui permet de multiplier les domaines d'exécution pour un ensemble de RPRs.

5.2.2 Le control des flux de données

L'architecture globale intégrant notre contrôleur est divisée en deux controls de flux de données comme illustré dans la Figure 5.2.

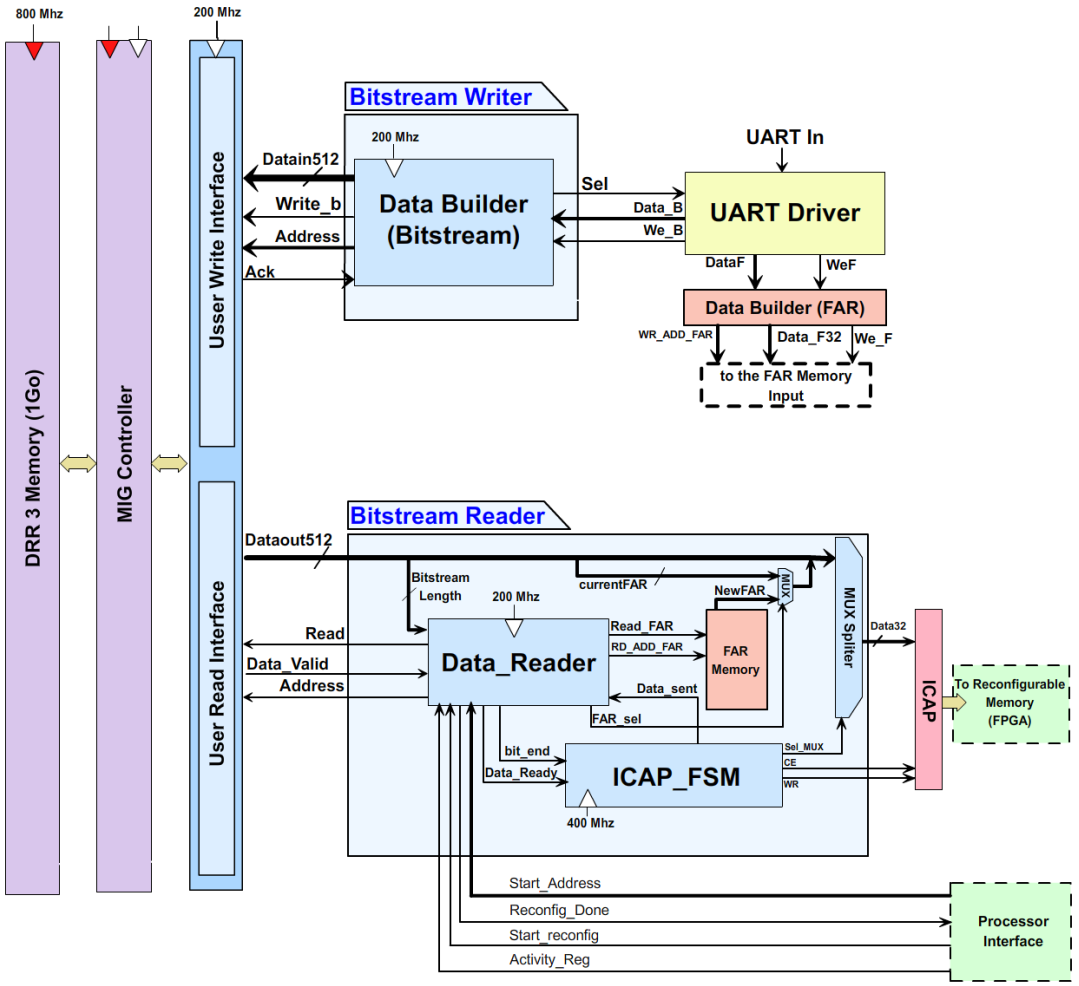


Figure 5.2 : Contrôleur de la reconfiguration dynamique dédié pour le broadcast de bitstream partiel

Le premier control de flux est responsable de l'écriture, à partir de l'entrée série UART, des bitstreams partiels dans la DDR3 ainsi que l'écriture des adresses d'emplacement des RPRs qui sont les valeurs des registres FARs. Les emplacements des RPRs sont écrits dans la mémoire FAR (FAR Memory). L'écriture de ces deux types de données passe par différents niveaux de synchronisation et modules de contrôle que nous allons détailler dans les paragraphes suivants.

Le deuxième control de flux est responsable de la lecture des bitstreams partiels depuis la mémoire de bitstreams. Ce deuxième control est assuré par le pilotage du processeur/IP qui gère le début et la fin de la RDPP. Il permet de transférer le bitstream partiel commun relatif à une configuration vers l'ensemble des RPRs. Etant donné que les plateformes FPGA-Xilinx actuelles ne permettent qu'un seul accès exclusif à la mémoire reconfigurable à travers le port ICAP, le transfert du bitstream partiel commun s'effectue en boucle suivant le nombre des

RPRs spécifiées en fonction de leurs emplacements et aussi en fonction de la valeur du masque donnée par le bus Activity_Reg.

5.2.2.1 Le contrôleur de flux de données d'écriture

Le control de flux d'écriture des différents types de données est assuré par les modules *"UART Driver"*, le module *"Bitstream Writer"*, l'interface *"User Write Interface"*, et le module *"Data Builder (FAR)"*.

Avant de commencer, essayons de comprendre le format du fichier bitstream que nous allons transfèrer dans la mémoire DDR3. Le format d'un bitstream Xilinx est assez simple. En effet, il est constitué par une entête, un corps et enfin des données de terminaison. Xilinx utilise des clés spécifiques pour construire ce fichier sous forme de champs de commandes et de champs de données. La Figure 5.3 illustre le contenu de l'entête d'un fichier bitstream codé en hexadécimal. Le tableau 5.1 présente les significations de la suite des champs qui constituent l'entête d'un fichier bitstream Xilinx. L'entête du bitstream est ensuite suivi par le corps qui comporte un ensemble de commandes de configuration des registres internes du FPGA.

```
00000000:  00 09 0f f0 0f f0 0f f0 0f f0 00 00 01 61 00 0a  *.............a..*
00000010:  78 66 6f 72 6d 2e 6e 63 64 00 62 00 0c 76 31 30  *xform.ncd.b..v10*
00000020:  30 30 65 66 67 38 36 30 00 63 00 0b 32 30 30 31  *00efg860.c..2001*
00000030:  2f 30 38 2f 31 30 00 64 00 09 30 36 3a 35 35 3a  */08/10.d..06:55:*
00000040:  30 34 00 65 00 0c 28 18 ff ff ff ff aa 99 55 66  *04.e..(.......Uf*
```

Figure 5.3 : L'entête d'un fichier bitstream Xilinx

Champs	Longueur	Utilité
Champ 1	0x000B	Ce champ est composé par deux sous-champs : - Une longueur de 2 octets égal à '0x0009'. - Un entête de 9 octets contient une sorte d'entête inutile et couramment ignorée par les contrôleurs ICAP et JTAG.
Champ 2	0x0003	Ce champ est composé par deux sous-champs : - Une longueur de 2 octets désignée pour spécifier une clé. - La clé 0x61 (symbole 'a') de longueur 1 octet.
Champ 3	0x000C	Ce champ est composé par deux sous-champs : - Une longueur de 2 octets désignée pour longueur du nom du fichier .ncd. - Une chaine de caractère (nom du fichier .ncd) sur 10 octets.
Champ 4	0x000F	Ce champ est composé par trois sous-champs : - La clé 0x62 (symbole 'b') de longueur 1 octet. - Une longueur de 2 octets contenant '0x000c' indiquant la référence du FPGA utilisé. - Une chaine de caractères sur 12 octets (référence du FPGA : "v1000efg860")

Champ 5	0x000E	Ce champ est composé par trois sous-champs : - La clé 0x63 (symbole 'c') de longueur 1 octet. - Une longueur de 2 octets contenant '0x000b' indiquant la date de création du fichier .bit. - Une chaine de caractères sur 11 octets contenant la date.
Champ 6	0x000C	Ce champ est composé par trois sous-champs : - La clé 0x64 (symbole 'd') de longueur 1 octet. - Une longueur de 2 octets contenant '0x0009' indiquant l'heure de création du fichier .bit. - Une chaine de caractères sur 9 octets contenant la date.
Champ 7	0x0005	Ce champ est composé par deux sous-champs : - La clé 0x65 (symbole 'e') de longueur 1 octet. - Une longueur de 4 octets indiquant le nombre de mots constituant le corps de fichier .bit.

Tableau 5.1 : Signification des données d'entête d'un bitstream Xilinx

Le corps est limité par deux mots de commande qui sont le mot de synchronisation et le mot de désynchronisation. Le mot de synchronisation **"0xaa995566"** est utilisé pour actionner la machine d'états interne qui gère le port ICAP. En effet, suite à la réception de cette commande, la machine interne attend la réception des mots de configurations de la mémoire reconfigurable du FPGA. Cette machine effectue un comptage qui est donné suivant la valeur du champ 7 se trouvant dans l'entête y compris le mot de désynchronisation. En effet, la valeur du champ 7 nous permet de connaître la taille de chaque fichier bitstream que nous voulons écrire/lire à partir de la mémoire de bitstream. D'autre part, le mot de désynchronisation **"0x000000D"** est utilisé pour que la machine d'états interne retrouve son état initial. Quelques mots de commande peuvent se trouver suite à la commande de désynchronisation et qui n'ont aucun effet sur la machine d'états. Ils sont utilisés pour l'alignement du fichier bitstream. Ces mots de commande sont des opérations NOP **"0x20000000".** L'ensemble des commandes des registres internes agissant sur la mémoire reconfigurable du FPGA est détaillée dans le guide Xilinx [47].

Une caractéristique essentielle de chacun des bitstreams partiels est leurs emplacements dans la mémoire reconfigurable du FPGA. Chaque emplacement est désigné par une adresse unique qui se situe au début du corps du fichier bitstream. Elle est précédée par un mot de configuration bien spécifique susceptible de commander le registre FAR (*Frame Address Register*) du FPGA. Le registre FAR est commandé à travers un mot de configuration de 32 bits égal à **"0x30002001"**. Dans le but de factoriser l'ensemble des bitstreams partiels compatibles à la relocalisation, il est nécessaire de disposer de leurs adresses d'emplacements. En effet, à chaque fois que nous voulons reconfigurer une RPR caractérisée par son adresse d'emplacement spécifique, nous effectuons une lecture du même fichier bitstream à partir de

la mémoire de bitstream suivant une boucle qui traite un certain nombre de RPRs compatibles. Ensuite pendant la lecture, lorsque nous apercevons le champ d'adressage du registre FAR, nous mettons à jour la valeur d'adresse d'emplacement relative à la RPR suivante. Enfin, nous envoyons le reste du flux binaire à travers l'interface de l'ICAP.

Etant donné que nous modifions un bitstream original à chaque configuration, une notification d'erreur d'intégrité des données est générée par le registre de control CRC. Dans le but de contourner cette notification, nous désactivons le control effectué par ce registre en modifiant sa commande relative. Le mot de commande qui gère ce registre est **"0x30000001"** qui se trouve juste avant le mot de désynchronisation. Cette valeur doit être modifiée par la valeur **"0x30008001"** suivi par la valeur **"0x00000007"**. La valeur **"0x30008001"** permet à commander le registre CMD pour agir sur le registre CRC et la valeur **"0x00000007"** permet de mettre à zéro ce registre. Une autre manière de désactiver le registre CRC se manifeste par rajouter l'option ***-g crc:disable*** à l'outil **BitGen** lors de la génération des bitstreams partiels.

a- Le Control de flux des données d'écriture des bitstreams et les informations de placement des RPRs

La première phase d'écriture des données se manifeste par l'écriture des bitstreams partiels communs relatifs à chaque configuration dans la mémoire DDR. Tous ces bitstreams élémentaires sont concaténés dans un seul bitstream global qui se caractérise par une spécification bien déterminée pour sa construction. Cette spécification est décrite dans la Figure 5.4. En effet, le bitstream global est structuré d'une manière permettant son écriture dans la mémoire DDR ainsi que pour faciliter la sélection d'un bitstream élémentaire à diffuser vers l'ensemble des RPRs. Il est constitué par une entête de 32 bits qui contient la taille des bitstreams élémentaires suivi par l'ensemble des bitstreams élémentaires dont chacun est écrit dans la DDR à partir d'une adresse spécifique. Chaque bitstream élémentaire est aussi structuré de la même façon. Il contient une entête de 32 bits qui spécifie sa taille suivie par l'ensemble des mots de commande et les mots de configurations. En effet, l'entête globale du fichier bitstream total est la somme des tailles contenues dans les entêtes de chaque bitstream élémentaire. La taille d'un bitstream élémentaire est donnée par la valeur du champ 7 comme indiquée dans le Tableau 5.1.

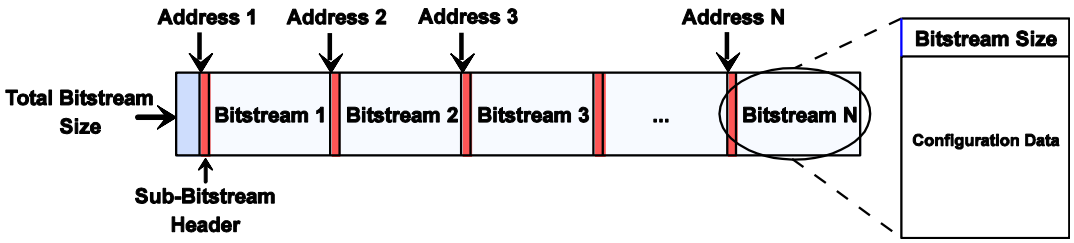


Figure 5.4 : Structure du bitstream global

Nous utilisons la mémoire DDR 3 pour stocker le bitstream global. Cette mémoire dispose de deux bus de données de lecture et d'écriture dont chacun est codé sur 64 bits. Le Contrôleur MIG généré par l'outil Xilinx, supporte au minimum un transfert en rafale de 8. Donc notre interface *User Write Interface* doit effectuer des transferts de données d'écriture sur 512 bits. Ce qui est aussi le cas pour l'interface de lecture *User Read Interface*. La mémoire DDR dispose également deux bus d'adresses de lecture et d'écriture codés chacun sur 28 bits.

- **Le Module : UART Driver**

Le bitstream total est envoyé à partir d'un ordinateur hôte via l'interface UART en utilisant l'outil *Tera Term*. La partie UART_RX (expéditeur) du module *UART Driver* envoie les mots de configuration par paquets de 8 bits à travers le bus **data8** vers l'entrée du démultiplexeur *DMUX_data8*. Ce bus de données est accompagné par un signal d'activation **enable8** connecté à l'entrée du démultiplexeur *DMUX_en8*. Ces deux démultiplexeurs sont initialement aiguillés par le signal Sel (**Sel** = '1') vers les sorties **Data_B** et **We_B** qui sont connectés au module *Bitstream Writer*. Lorsque l'envoi du bitstream global est achevé, ces deux démultiplexeurs sont aiguillées automatiquement (**Sel** = '0') vers les sorties **DataF** et **WeF** connectées au contrôleur d'écriture de la mémoire *FAR Memory*. L'architecture interne du module *UART Driver* est présentée dans la Figure 5.6.

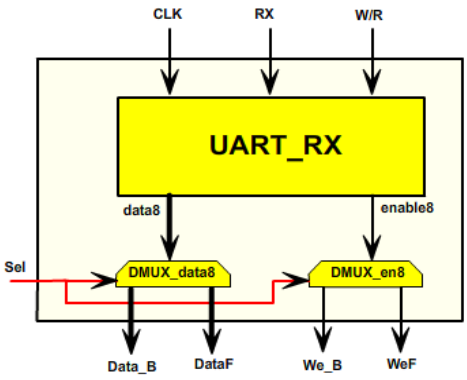


Figure 5.6 : Architecture interne du module UART Driver

- **Le Module : Bitstream Writer**

Le module *Bitstream Writer* permet d'accumuler des données de 8 bits provenant du module UART Driver dans un tampon de données de 512 bits pour les transférer vers l'interface de l'écriture avec une adresse valide. Ce module est composé de deux sous-modules fonctionnant comme suit :

- ***Le Module Constructeur des Données (Data Builder)***

Ce module est basé sur une machine d'états finis dédiée pour le contrôle et la construction des données de configuration des bitstreams élémentaires et leurs transferts. Cette machine d'états finis est synchronisée avec le signal d'horloge CLK_baud utilisé par le module *UART Driver* à une vitesse de 9600 baud. Ce module est présenté dans la Figure 5.7.

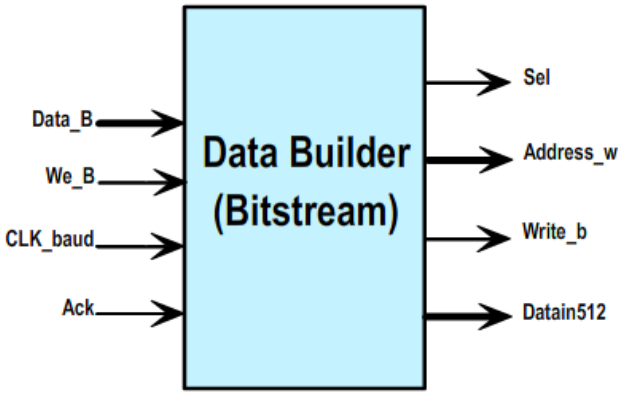


Figure 5.7 : Le Module Data Builder

Initialement, ce module reçoit les quatre premiers 8 bits sur le bus Data_B à chaque activation du signal We_B et reconstruit sur 32 bits l'entête du bitstream global qui sert à décompter le nombre de mots de 512 bits. Ensuite il entame la construction des données de configuration de 512 bits après 64 fronts d'horloge en utilisant un tampon de 512 bis. Le transfert de chaque donnée est effectué en suite sur le bus Datain512 tout en activant le signal Write_b et aussi en générant une adresse valide sur 28 bits. La génération des adresses s'effectue à partir de l'adresse 0x0000000 et incrémentée de 64 à chaque fois que la donnée est construite. L'acquittement de chaque donnée transférée est assuré par l'activation du signal Ack généré par le module « User Write Interface ». Lorsque le registre de décomptage contient une valeur nulle suite à l'opération décomptage, le signal Sel est basculé à la valeur '0' pour permettre le passage à la deuxième phase d'écriture (écriture des adresses d'emplacement des RPRs dans la mémoire FAR Memory).

- ***Le Synchroniseur des données:***

Le module de synchronisation permet d'adapter les deux domaines d'horloge entre l'interface d'écriture de 200 Mhz et le module UART Driver supportant une vitesse de 9600 baud. En effet il permet d'adapter la vitesse de transfert des données de 512 bits sur le bus Datain512 synchronisé avec le signal Write_B qu'il génère.

- ***Le Module : User Write Interface***

Le module User Write Interface est conçu avec le module User Read Interface dans le même module puisqu'ils partagent la même interface avec le module MIG controller. Nous nous intéressons en premier temps au module « User Write Interface » pour décrire la phase d'écriture du bitstream global. Ce module joue le rôle d'une interface d'écriture des données relatives au bitstream global à une fréquence de 200 Mhz. Elle permet l'adaptation de l'écriture suivant le protocole de transfert de données exigé par le contrôleur MIG. Elle effectue les différentes phases à base d'une machine d'états finis permettant l'écriture des données de 512 bits par la mise en œuvre de la commande correspondante et un ensemble de signaux de control présentés dan la Figure 5.8.

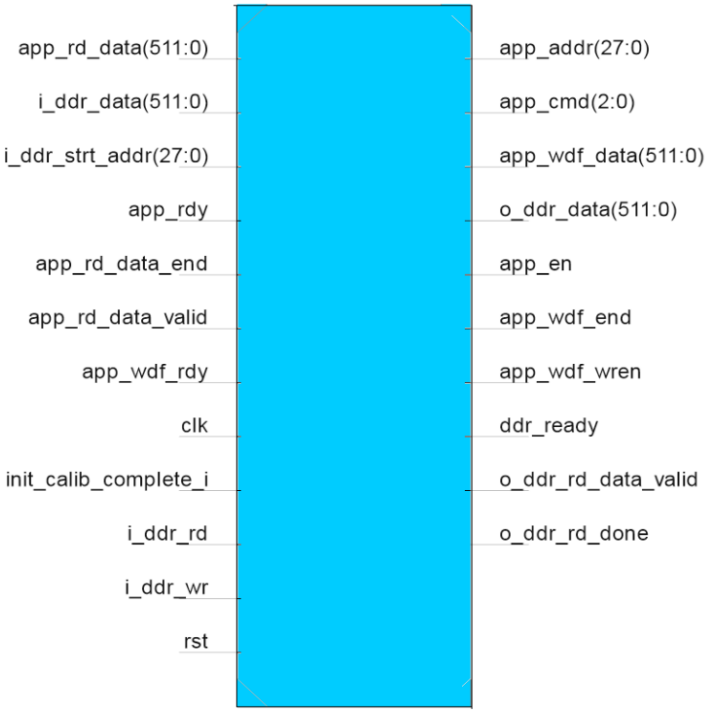


Figure 5.8 : Le Module d'écriture/lecture du bitstream global

En effet, la première phase est exécutée dès l'activation du signal write_B (connecté à l'entrée i_ddr_wr de l'interface), la commande d'écriture affectée au bus app_cmd est mise à la valeur "000" en même temps que le signal d'activation app_en = '1'. Tant que l'écriture

n'est pas encore effectuée, le signal ddr_ready esy mis à '0'. La deuxième phase se manifeste par l'affectation de la donnée provenant du bus Datain512 (connecté à l'entrée i_ddr_data) au bus de sortie app_wdf_data ainsi que l'affectation du bus d'adresse app_addr par l'adresse d'entrée provenant du bus d'entrée Address (connecté au bus ddr_wr_addr). Cette phase s'effectue aussi par le maintien des valeurs app_en = '1' et app_cmd = "000". Dès l'activation du signal app_ready = '1' provenant du contrôleur MIG, l'écriture effective est exécutée. Cependant, les signaux de control de fin d'écriture app_wdf_end (connecté au signal Ack) et app_wdf_wren sont mis à '1'. Finalement, le contrôleur MIG acquitte la réception de la donnée de 512 bits par l'activation du signal aap_wdf_rdy = '1'. La machine d'état finis reprend alors son état initial.

- **Le Module Data Writer (FAR)**

Suite à l'étape d'écriture du bitstream global, la phase d'écriture des adresses d'emplacement des RPRs est déclenchée avec Sel = '0'. Un fichier binaire est alors envoyé depuis l'entrée du module UART Driver pour être traité (accumulation est synchronisation) au niveau du module Data Writer (FAR). Ce fichier est structuré d'une manière spécifique pour permettre son envoi et l'écriture des valeurs FAR dans la mémoire Memory FAR. En effet, il est composé par une entête suivi par l'ensemble des adresses FAR. L'en-tête est un champ codé sur 8 bits (6 bits utiles) et contient le nombre des adresses FAR (64 mots). Tandis que chaque valeur d'emplacement d'une RPR est codée sur 32 bits.

Le module Data Builder (FAR) est conçu à base d'une machine d'états finis qui sert à compter et accumuler les adresses d'emplacement puis leurs envois. En effet, dés la réception des 8 bits premiers provenant du bus DataF et synchronisée avec le signal de validation WeF, la machine commence l'accumulation de 4x8 bits tout en effectuant le décomptage. Ces étapes s'effectuent à une vitesse de 9600 baud. L'envoi des données s'effectue suite à une phase de synchronisation à une fréquence de 200 Mhz. Ces données sont transférées sur le bus Data_F32 à chaque validation du signal We_F et génération d'adresse sur le bus WR_ADD_FAR de largeur 6 bits. La génération des adresses s'effectue à partir de l'adresse "0b000000".

b- Le control de flux des données de lecture des bitstreams et les informations de placement des RPRs (La phase de RDPP)

La phase de la reconfiguration est déclenchée à travers l'interface de pilotage (Processeur/IP) qui permet de gérer le transfèrt des bitstreams partiels communs et leurs mises

en place entre l'interface de lecture *"User Write Interface"* et l'interface de la mémoire reconfigurable (FPGA). En effet, la phase de la RDPP est activée lorsque le signal Start_reconfig est mis à la valeur '1' avec une adresse Start_Address valide relative à un bitstream (Figure 5.4) et la mise en place d'un masque d'activité sur le bus Activity_Reg. Le bus Start_Address est codé sur 28 bits tandis que le bus Activity_Reg est codé de manière générique à une valeur maximale de 64 bits. Ce dernier permet en effet de gérer au maximum une activité de reconfiguration de 64 RPRs. Sa largeur dépend en fait du nombre des RPRs que nous pouvons spécifier en fonction des ressources disponibles sur FPGA et leurs faisabilités d'implémentation. Le processus de la reconfiguration est géré par le module Bitstream Reader qui est constitué par les quatre modules suivants :

- **Le Module Data _Reader**

Le processus de RDPP est sollicité à partir de l'interface du (Processeur/IP) dont les signaux sont connectés au module Data_Reader. Ce module est basé sur une machine d'états finis qui permet la lecture des mots de configuration, mettre à jour la valeur du registre FAR à chaque reconfiguration d'une RPR en fonction du registre d'activité et synchroniser le transfert des mots de configuration à travers le module ICAP_FSM qui gère le port ICAP.
En effet, le module Data_Reader effectue la première lecture pour récupérer la première donnée de reconfiguration tout en activant le signal Read = '1' connecté à l'interface User Read Interface à travers le signal i_ddr_rd. La lecture s'effectue à l'aide d'une adresse valide qui correspond au début de la plage mémoire d'un bitstream élémentaire comme illustré dans la Figure 5.4. Cette adresse correspond en effet à l'en-tête du bitstream élémentaire qui contient sa propre taille. Donc la première donnée de 512 bits lue est alors utilisée pour le comptage des mots de configuration codés sur 32 bits dans l'intervalle Dataout512 [0..31] et affecté sur le bus Bitstream Length. Les données de 512 bits sont lues, comme nous venons de l'évoquer, en mettant à '1' le signal Read et effectuer une incrémentation de 64 pour adresser une nouvelle donnée dans la mémoire DDR. Le signal Data_Valid est utilisé pour acquitter la réception d'une donnée par le module User Read Interface. Cette première lecture ne peut se produire que si le masque Activity_REG permet la reconfiguration d'une RPR. En effet, la taille de la mémoire FAR Memory est générique et toujours égale à la valeur Net_D. De même, elle est égale à la largeur générique du bus Activity_REG. Chaque élément du bus Activity_REG correspond dans l'ordre à une case mémoire dans FAR Memory et qui représente l'activité de reconfiguration d'une RPR. C'est-à-dire que si la valeur de Activity_REG [i] = '1' ($1 < i < 64$), alors la RPR ayant la valeur du FAR dans FAR Memory

de rang i sera reconfigurée. Sinon cette RPR ne sera pas reconfigurée et le compteur i est incrémenté de 1 pour tester l'éligibilité de reconfiguration de la RPR suivante. La Figure 5.9 illustre ce mécanisme.

Pendant la lecture de la deuxième donnée, nous récupérons la valeur du FAR au niveau de la position Dataout512 [351..383] équivalent à la douzième position du bus Dataout512 en termes de mots de configuration de 32 bits. La valeur du FAR est récupérée sur le bus currentFAR pour un éventuel multiplexage par la valeur du bus de lecture NewFAR de la mémoire FAR Memory. Ce multiplexage est assuré par l'élément MUX qui est aiguillé à partir du signal FAR_Sel. Le signal FAR_Sel est géré par le module Data_Reader. Il est aiguillé vers le bus NewFAR lors de la lecture de la deuxième donnée de 512 bits sinon l'aiguillage s'effectue vers le bus currentFAR pour le reste des données d'un même bitstream commun.

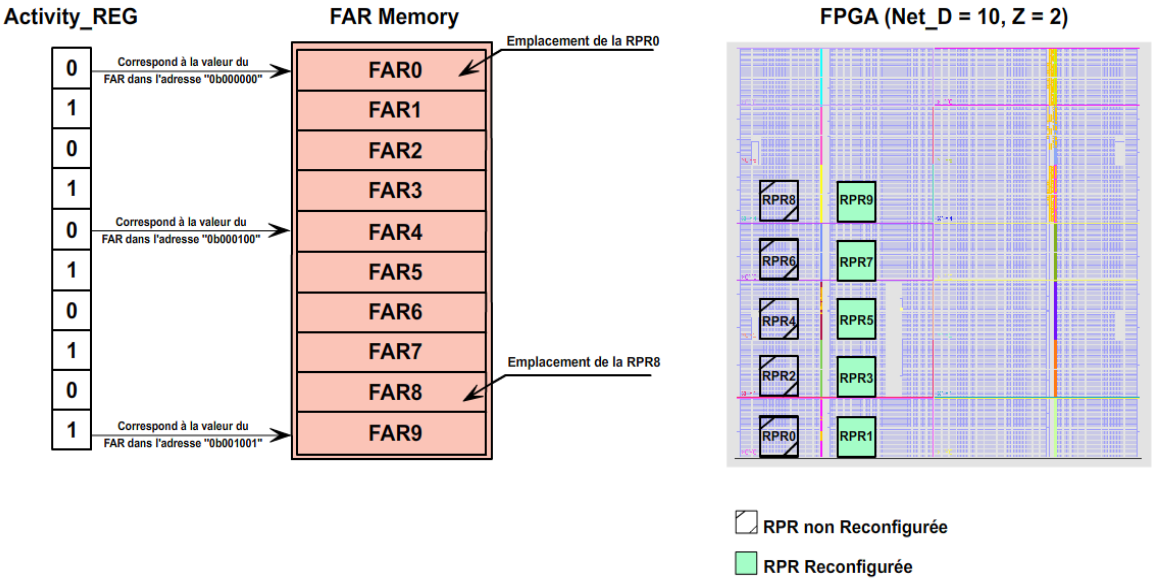


Figure 5.9 : La gestion de la reconfiguration des RPRs

De même, lors de lecture de la deuxième donnée, la lecture de la mémoire FAR Memory intervient. A partir du module Data_Reader, la lecture des données stockées dans la mémoire FAR Memory s'effectue à travers le bus d'adresses RD_ADD_FAR de largeur 6 bits et l'activation des signaux Read_FAR suivi par l'activation du signal FAR_sel. La valeur du bus d'adresse RD_ADD_FAR s'incrémente à chaque nouvelle reconfiguration d'une RPR. La lecture de chaque donnée effectuée par le module Data_Reader engendre l'activation du signal Data_Ready pour indiquer au module ICAP_FSM qu'une nouvelle donnée est présente sur le bus Dataout512. En effet, le signal Data_Ready permet de déclencher la machine d'états finis du module ICAP_DSM pour qu'il commence le transfert des données de

reconfiguration de 32 bits à partir du bus Dataout512 à travers le multiplexeur MUX_Spliter de 16 entrées. La fin de transfert d'une donnée de 512 bits est indiquée par l'activation du signal Data_sent par le module ICAP_FSM pour qu'une nouvelle donnée soit lue par le module Data_Reader. De cette manière, les données de configuration d'un bitstream partiel commun sont transférées dans une RPR tout en effectuant le décomptage à partir de la valeur donnée par le bus Bitstream_Length. Quand le registre de décomptage devient nul, le signal bit_end est activé pour que le module ICAP_FSM retrouve sont état initial. Ce processus se reproduit suivant le parcours de chaque case du registre Activity_REG en testant la possibilité de reconfiguration des RPRs. La fin de la RDPP est indiquée par le signal Reconfiguration_Done = '1' faisant partie de l'interface (Processeur/IP).

- **Le Module ICAP_FSM**

Lorsque le signal « Data_Ready » est activé, le module « ICAP_FSM » gère le transfert des données de reconfiguration tout en agissant sur le bus de sélection Sel_MUX du module MUX_Spliter. En même temps, il contrôle l'opération d'écriture à travers l'interface de l'ICAP par l'activation des signaux CE et WR. Ce module génère les valeurs de sélection sur 4 bits grâce à un comptage à chaque front d'horloge pendant 16 cycles à une vitesse de 400 MHz correspondant à 8 cycles d'horloge du module Data_Read fonctionnant à 200 MHz. Une phase de synchronisation est alors prévue pour les signaux d'interconnexion entre ces deux modules (Data_sent, Data_Ready et bit_end).

- **Les Modules MUX_Spliter et MUX**

Le module MUX_Spliter est un multiplexeur de 16 entrées dont chacune est codée sur 32 bits permettant de transférer une donnée de 512 bits. Il est adressé par le signal Sel_MUX codé sur 4 bits provenant du module ICAP_FSM. L'entrée numéro 10 de ce module est multiplexé par le module MUX. Le multiplexage assuré par le module MUX s'effectue lors de la lecture de la deuxième donnée de reconfiguration de 512 bits suite à l'activation du signal d'adressage FAR_sel dans le but de mettre à jour la valeur du registre FAR.

5.3 L'expérimentation sur FPGA et évaluation des performances

Les plateformes ML605 [87] et VC707 [88] de Xilinx sont utilisée comme plateformes de prototypage, de test et d'évaluation de l'architecture proposée de RDPP. La plateforme ML605 est utilisée pour comparer notre contrôleur de la RDPP avec les travaux existants. Il se compose d'un FPGA Virtex-6 XC6VLX240T-1FFG1156 et de plusieurs périphériques

tandis que la plateforme VC707 comporte un FPGA Virtex-7 XC7VX485T-FFG1761 [89]. Le FPGA Virtex-6 contient 24152 éléments logiques, 37680 slices CLB dont chacun contient 4 LUT et 8 flip-flops, 768 slices DSP48E1 et 416 blocs RAM de taille 36Kbits. Il fonctionne avec une fréquence utilisateur de 200 MHz. Le FPGA Virtex-7 contient 485760 éléments logiques, 75900 slices CLB dont chacun contient 4 LUT et 8 flip-flops, 2800 slices DSP48E1 et 1030 blocs RAM de taille 37Kbits. Il fonctionne avec une fréquence utilisateur de 200 MHz.

Les outils utilisés pour la conception, l'implémentation et l'évaluation des performances sont les outils Xilinx : ISIM 14.7 et Chipscope pour la simulation et ISE 14.7 pour la conception et la synthèse. Les outils PlanAhead 14.7 et FPGA_Editor sont utilisés pour l'implémentation du système. L'outil XPower Analyzer est utilisé pour mesurer la consommation d'énergie statique et dynamique. Les fichiers de configuration binaires sont générés à l'aide de l'outil BITGen. La partie statique est transférée dans l'FPGA à l'aide de l'outil iMPACT tandis que les bitstreams partiels et les informations de placement des RPR sont transférés à partir du porte de l'UART respectivement dans les mémoires DDR et FAR Memory en utilisant l'outil TeraTerm.

L'évaluation des performances du contrôleur de la RDPP concerne le temps d'exécution, la fréquence maximale, la bande passante, la consommation d'énergie, la surface occupée, etc. L'utilisation des outils de conception de Xilinx permet de prédire ces caractéristiques avant l'implémentation, afin de faire les modifications nécessaires pour améliorer les performances de l'accélérateur conçu. En effet, les résultats de synthèse donnent une estimation de la fréquence de 450 MHz relative au contrôleur du flux d'écriture (Bitstream Writer) et une fréquence de 373 Mhz relative au contrôleur du flux de lecture (Bitstream Reader). La fréquence maximale supportée par le module Bitstream Reader est indiquée par la fréquence du module Data_Reader qui est fonctionnel d'après notre conception à 200 Mhz. Tandis que le module ICAP_FSM est fonctionnel à une fréquence maximale de 451 Mhz.

Ces fréquences dépendent du plus long chemin critique dans la configuration. Un bon placement et routage des composants du contrôleur est nécessaire afin de réduire la longueur de ce chemin critique et d'accélérer la propagation des signaux. De plus, les résultats de synthèse détaillent la surface occupée (les éléments logiques, mémoires, blocs DSP, etc.) par les composants de l'architecture sur les FPGAs ciblés.

A notre connaissance, il n'existe pas dans la littérature un contrôleur de RDPP favorisant le broadcast d'un bitstream vers un ensemble de RPRs mais des contrôleurs qui permettent la

reconfiguration d'une seule RPR. Dans le but de comparer notre conception avec les accélérateurs existants, pour avons développé deux configurations. Une configuration basée sur la reconfiguration d'une seule RPR (mono-RPR). La deuxième configuration permettant la RDPP (multi-RPRs : 64 RPRs). Les Tableaux 5.2 et 5.3 détaillent la surface occupée et les fréquences estimées de chaque sous module par la configuration mono-RPR respectivement sur les FPGAs Virtex-6 puis Virtex-7. Les Tableaux 5.4 et 5.5 détaillent la surface occupée et les fréquences estimées de chaque sous module par la configuration multi-RPRs respectivement sur les FPGAs Virtex-6 puis Virtex-7.

Bitsream Writer	Sous-Module	Slices	BRAM	Fréquence (MAX)	Puissance Dynamique (W)
	Data_Builder	165	1	355.7 Mhz	0.00003
	User Read/Write Interface	167	1	637.5 Mhz	0.00085
	Le Synchroniseur des données	2	1	448.2 Mhz	0.00212
	Total	**334**	**2**	**355.7 Mhz**	**0.003**
Bitsream Reader					
	Data_Reader	32	2	329.15 Mhz	0.00280
	ICAP_FSM	2	1	448.16 Mhz	0.00210
	Mux_Spliter	30	0	-	
	Total	**64**	**2**	**329.15 Mhz**	**0.00490**

Tableau 5.2 : Evaluation de la surface occupée et les différentes fréquences estimées de la configuration mono_RPR (plateforme de prototypage Virtex-6)

Bitsream Writer	Sous-Module	Slices	BRAM	Fréquence (MAX)	Puissance Dynamique (W)
	Data_Builder	165	1	450 Mhz	0.00002
	User Read/Write Interface	167	1	701 Mhz	0.00080
	Le Synchroniseur des données	2	1	450 Mhz	0.00204
	Total	**333**	**2**	**450 Mhz**	**0.00287**
Bitsream Reader					
	Data_Reader	32	2	332 Mhz	0.00274
	ICAP_FSM	2	1	450 Mhz	0.00207
	Mux_Spliter	30	0	-	-
	Total	**64**	**2**	**332 Mhz**	**0.00484**

Tableau 5.3 : Evaluation de la surface occupée et les différentes fréquences estimées de la configuration mono_RPR (plateforme de prototypage Virtex-7)

Bitsream Writer	Sous-Module	Slices	BRAM	Fréquence (MAX)	Puissance Dynamique (W)
	Data_Builder	165	1	356 Mhz	0.00003
	User Read/Write Interface	167	1	637.5 Mhz	0.00085
	FAR_Builder	10	1	441.8 Mhz	0.00012
	Le Synchroniseur des données	2	1	448.16 Mhz	0.00212
	Total	**344**	**3**	**356 Mhz**	**0.00312**

Bitsream Reader					
	Data_Reader	52	2	299.3 Mhz	0.01080
	ICAP_FSM	2	1	448.16 Mhz	0.00210
	FAR Memory	0	1	373 Mhz	0.00282
	Mux_Spliter + MUX	38	0	-	
	Total	**92**	**3**	**299.3 Mhz**	**0.01572**

Tableau 5.4 : Evaluation de la surface occupée et les différentes fréquences estimées de la configuration multi_RPR (plateforme de prototypage Virtex-6)

Bitsream Writer	**Sous-Module**	**Slices**	**BRAM**	**Fréquence (MAX)**	**Puissance Dynamique (W)**
	Data_Builder	168	1	450 Mhz	0.00002
	User Read/Write Interface	167	1	701 Mhz	0.00080
	FAR_Builder	10	1	479.5 Mhz	0.00009
	Le Synchroniseur des données	2	1	450 Mhz	0.00204
	Total	**333**	**3**	**450 Mhz**	**0.00295**
Bitsream Reader					
	Data_Reader	43	2	373 Mhz	0.01011
	ICAP_FSM	2	1	451 Mhz	0.00206
	Mux_Spliter + MUX	38	0	-	-
	FAR Memory	0	1	373 Mhz	0.00281
	Total	**83**	**3**	**373 Mhz**	**0.01499**

Tableau 5.5 : Evaluation de la surface occupée et les différentes fréquences estimées de la configuration multi_RPR (plateforme de prototypage Virtex-7)

Les tableaux ci-dessus montrent que la consommation d'énergie dynamique totale (contrôleur d'écriture + contrôleur de lecture) des deux configurations implémentées sur les plateformes Virtex-6 et Virtex-7 ne dépassent pas 0.1W sachant que la consommation d'énergie statique de la plateforme Virtex-7 est de 1.337W et celle de la Virtex-6 est de 1.977W. En effet, cette faible consommation d'énergie indique l'intégration d'un ensemble de modules de taille réduite et rapides. Nous notons que pour l'intégration du mécanisme de la RDP dans une architecture, il est préférable d'intégrer des accélérateurs spécialisés que d'intégrer un processeur tel que MicroBlaze, afin d'optimiser la surface et d'accélérer l'exécution.

En se basant sur la taille occupée par les deux types de configurations implémentées sur les deux plateformes FPGA, le coût de la surface totale occupée par le contrôleur de la RDPP (contrôleur d'écriture + contrôleur de lecture) est relativement faible et ne dépasse pas 1% de la surface totale des deux FPGAs. Nous soulignons que la surface occupée par le module ICAP_FSM est assez réduite qui lui permet de fonctionner à une fréquence très élevée supérieure à 400 Mhz. En effet, ce module garantit un débit théorique de transfert de

bitstreams de 1.6 Go/s. Réellement, suite à l'implémentation le l'architecture du contrôleur de la RDPP, le module ICAP_FSM ne peut supporter qu'une fréquence maximale de fonctionnement de 380 Mhz qui garantit un débit de 1.52 Go/s.

5.4 Etude comparative

Dans cette section, nous comparons les performances de notre contrôleur de la RDPP et ceux proposés suivant la littérature. La comparaison est effectuée au niveau du débit de transfert des bitstreams et la surface occupée sur FPGA.

Dans un premier lieu, nous comparons à travers le Tableau 5.6 les différents débits assurés par les contrôleurs de la DRP existants [27][28][29] et [30] implémentés sur FPGA Virtex-6 avec celui que nous avons développé.

Contrôleur de la RDP	Débit (MB/s)	Largeur de Bitstream	Fréquence Max de l'ICAP (Mhz)
xps hwicap [27]	14.5	+++	120
FaRM [28]	800	++	200
UPaRC [29] (avec compression)	1008	++	255
UPaRC [29] (sans compression)	1433	-	362.5
Contrôleur à base de MPMC [30]	1480	+++	370
Contrôleur Mono-RPR Proposé	1520	+++	380

Tableau 5.6 : Tableau comparatif en termes de débit

Etant donné que les contrôleurs de la RDP existants ne supportent pas la technique de broadcast du bitstream qui se base sur la relocalisation, nous utilisons donc pour la comparaison la configuration mono-RPR implémentée sur l'FPGA Virtex-6. Le tableau montre que notre contrôleur de la RDP garantit un débit supérieur à ceux qui existent dans la littérature. Comparé à [27] et [28], notre contrôleur assure un débit largement élevé vu que la fréquence de transfert des bitstreams est plus élevée. Comparé à [29], notre contrôleur de flux de lecture intègre un mécanisme de pipeline dans le but d'éviter les attentes intermédiaires entre les mots de configuration. De plus, [29] intègre une mémoire BRAM pour le stockage temporaire des bitstreams de petite taille sur FPGA et un composant de compression permettant de réduire la taille de cette dernière ce qui explique la latence de transférer des mots de configuration à un débit de 1008MB/s. La suppression du module de compression dans la même configuration de [29] à engendré l'augmentation du débit à 1433 MB/s à une fréquence limite de 362.5 Mhz qui est la fréquence limite de la mémoire BRAM. Comparé à [30], notre contrôleur garantit un débit légèrement supérieur de 40MB/s. D'autre part, MPMC

est une interface simple à utiliser mais elle reste spécifique pour certains FPGAs (Virtex-5 et Virtex-6) et non supportée par les FPGAs récents.

Le Tableau 5.7 montre la surface occupée par notre contrôleur de la RDP et ceux présentés dans [29] et [30] sur l'FPGA Virtex-6. Les architectures de ces deux derniers emploient le processeur MicroBlaze pour piloter le transfert des données de configuration. Nous nous limitons dans notre étude comparative à présenter seulement les surfaces totales occupées par les modules élémentaires de chaque contrôleur.

Contrôleur de la RDP	**Contrôleur du Flux de :**	**SLICEs**	**BRAM**
UPaRC (avec compression)	Lecture	944	992 KBytes (6%)
	Ecriture	-	-
UPaRC (sans compression)	Lecture	44	2444 KBytes (16%)
	Ecriture	-	-
Contrôleur à base de MPMC	Lecture	812	-
	Ecriture		
Contrôleur Mono-RPR Proposé	Lecture	**64**	RAMBFIFO36E1 x 2 = **72 KBytes** (<1%)
	Ecriture	**334**	RAMBFIFO36E1 x 2 = **72 KBytes** (<1%)
Contrôleur Multi-RPR Proposé (64 RPR) :	Lecture	**92**	RAMBFIFO36E1 x 3 = **108 KBytes** (<1%)
	Ecriture	**344**	RAMBFIFO36E1 x 3 = **108 KBytes** (<1%)

Tableau 5.7 : Tableau comparatif en termes de surface occupée

Dans [29], Au niveau de lecture des bitstreams partiels, le tableau montre que le compresseur de données utilisé occupe 900 Slices pour permettre la réduction considérable de la taille de la mémoire BRAM qui occupe 6% de la totalité des BRAMs disponibles sur FPGA. Sans utilisation du compresseur de données, la surface occupée par la mémoire BRAM atteint 16% des ressources disponibles sur FPGA et seulement 44 Slices pour la logique. Cette dernière configuration dépend en effet du choix de l'application dans laquelle ce type de contrôleur peut exister. Comparé à notre architecture, la somme des ressources pour les deux contrôleurs de flux de lecture et d'écriture en termes de BRAM se limite également à moins de 1% des ressources disponibles pour les deux configurations mono-RPR (144 KBytes) et mono-RPR (216 KBytes). Dans [29], les ressources logiques en termes de Slices sont dominées par l'utilisation de l'interface MPMC qui requière 775 Slices nécessaires pour implémenter les interfaces de lecture et d'écriture des données depuis la mémoire DDRx. Tandis que 37 Slices sont utilisés pour implémenter les modules d'interfaçage de l'ICAP. Comparé aux réquisitions de ressources, notre contrôleur en mode mono-RPR nécessite presque la moitié de ces ressources avec un taux de 398 Slices. De même, 436 Slices

consommés par la configuration en mode multi-RPR. D'autre part, en faisant la comparaison entre les surfaces occupées par les deux configurations proposées, la différence en termes de Slices est de 36 Slice et 72 KBytes en termes de BRAM. Cette différence est relativement faible comparée aux ressources disponibles sur FPGA permettant d'intégrer la fonctionnalité de broadcast de bitstreams.

5.5 Conclusion

Dans ce chapitre, nous avons présenté l'architecture interne de l'accélérateur matériel de la RDPP. Notre contrôleur de la RDPP intègre un ensemble de modules constituant deux contrôles de flux de lecture et d'écriture. Il comporte trois interfaces essentielles pour permettre l'écriture des bitstreams communs dans la mémoire externe de type DDR puis leur lecture et transfert vers la mémoire reconfigurable du FPGA à travers l'interface de pilotage. L'évaluation des performances et l'étude comparative ont montré que notre accélérateur matériel de la RDPP occupe une surface assez réduite par rapport aux accélérateurs matériels existants. Il garantit également un débit de transfèrt des données de configuration de 1.52 Go/s à une fréquence de 380 Mhz. Les résultats ont montré l'efficacité de notre système au niveau de surface occupée sur FPGA et la vitesse de transfèrt des mots de configuration. Mais, le fonctionnement de notre contrôleur de la RDPP à une fréquence assez élevée engendre à son tour une augmentation de la puissance dynamique au cours de la phase de la reconfiguration.

CHAPITRE 6 : Validations Expérimentales Sur Différentes Plateformes FPGA Xilinx Pour Différents Grains De Parallélisme

6.1 Introduction

Nous avons détaillé dans le chapitre 3 les démarches à suivre pour automatiser le flot de la reconfiguration dynamique partielle à base de floorplanning régulier (AFLORA) en s'appuyant sur le flot d'automatisation d'ADForMe. A l'aide du floorplanning régulier, nous avons pu définir un autre flot d'automatisation de la relocalisation de bitstream partiels. Dans le Chapitre 4, ce flot d'automatisation vise à assurer la fonction de broadcast de bitstream partiel commun vers un ensemble de régions dynamiquement reconfigurables.

Dans ce chapitre, dans le but de mieux comprendre le déroulement des flots d'automatisation proposés, nous détaillons les différentes fonctionnalités et les étapes du mode opératoire sur différentes familles FPGA-Xilinx dans la Section 6.2. Nous effectuons dans la Section 6.3 un jeu de test sur différents FPGAs-Xilinx pour différents grains de parallélisme. Les résultats expérimentaux sont présentés dans la Section 6.4. Enfin la Section 6.5 présente l'étude analytique des résultats. Finalement, la dernière section donnera une synthèse de ce chapitre.

6.2 Plateformes de prototypage, fonctionnalités et mode opératoire

Dans cette partie, nous décrivons une démonstration de notre flot d'automatisation d'une manière globale afin de faciliter son utilisation avec l'outillage nécessaire. La Figure 6.1 détaille les différentes étapes à effectuer pour générer un bitstream commun à partir des éléments d'entrée du flot d'automatisation d'ADForMe. En effet, la constitution des éléments d'entrée du flot d'ADForMe est considérée comme une étape de préparation à la création du projet PlanAhead/Vivado et d'automatisation. Cette étape se manifeste par mettre dans le même dossier projet l'ensemble de descriptions .ngc de la partie statique et celles des différents MPRs, la description des contraintes initiales et le fichier script TCL. Le choix de la plateforme de prototypage FPGA est spécifié dans le fichier script lors de la création du projet comme indiqué dans List1. Des éléments supplémentaires doivent aussi exister dans le dossier projet qui sont représentés sous forme de boites jaunes : l'exécutable du flot AFLORA permettant de générer les emplacements réguliers des RPRs et leurs interfaces statiques, le script FPGA_Editor permettant de récupérer les informations de P&R des Proxys Logiques ainsi que les blocs fonctionnels se trouvant dans les interfaces statiques. Enfin, l'exécutable du flot d'uniformisation 1D et 2D des informations de P&R. En effet, le concepteur doit intervenir pendant quatre phases essentielles commençant par l'exécution du script TCL initial. L'exécution du script TCL comme indiqué dans la Section 3.2.3.1 dans le Chapitre 3

permet de générer les fichiers qui décrivent les ressources requises par la configuration commune du système et les paramètres technologiques du FPGA utilisé.

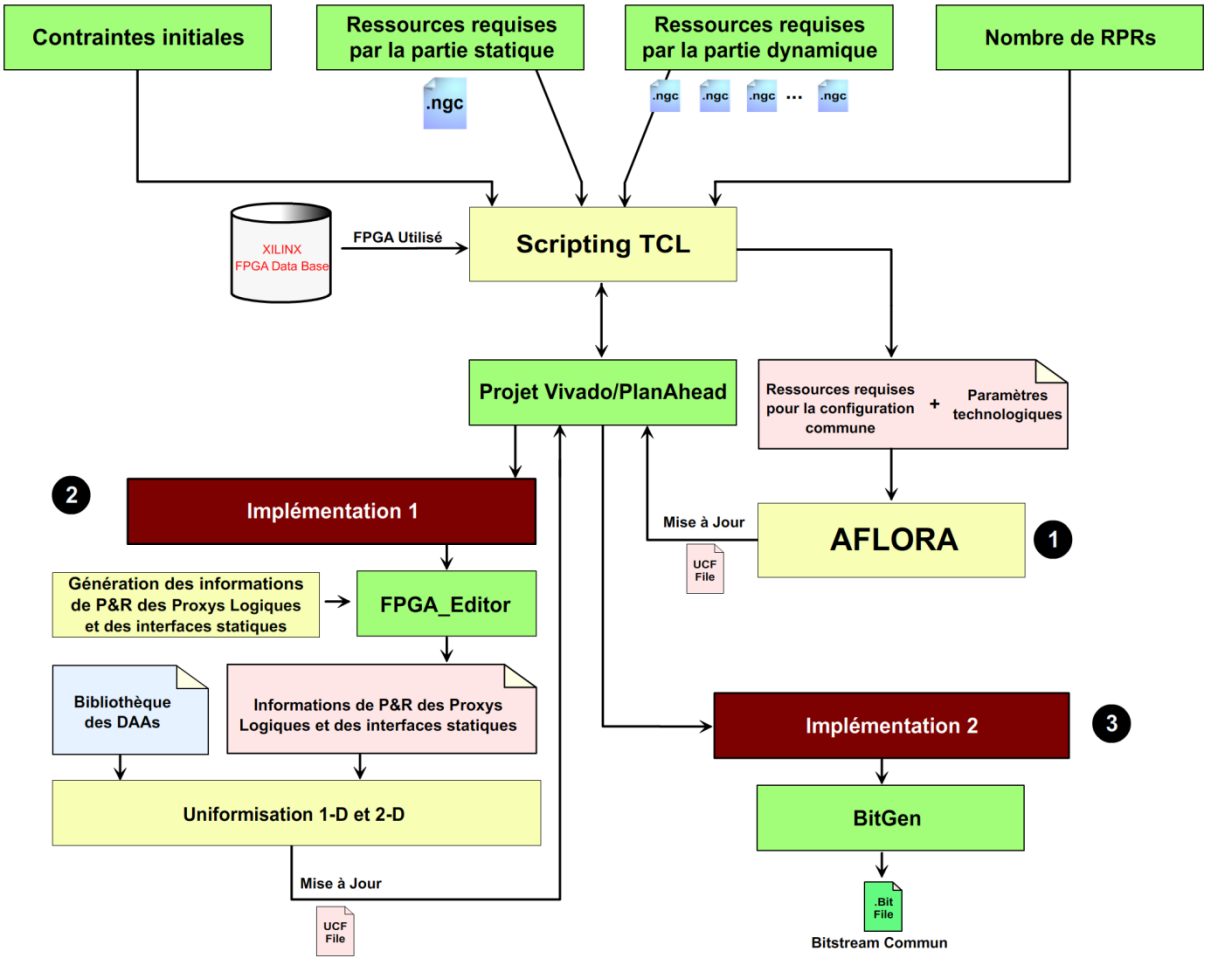


Figure 6.1 : Flot d'automatisation global

Dans la suite nous détaillons les étapes dont le concepteur doit intervenir pour assurer le déroulement du flot d'automatisation englobant l'automatisation de floorplanning régulier des RPRs et l'automatisation du flot d'uniformisation de P&R de Proxys Logiques et les interfaces statiques des RPRs.

6.2.1 Elaboration du floorplanning régulier

Suite à la génération des fichiers décrivant les ressources requises par la configuration commune du système et les paramètres technologiques du FPGA utilisé, les directives de traçage des RPRs assurés par le mécanisme AFLORA est exécuté. L'exécution est effectuée sous forme d'une commande en utilisant l'exécutable AFLORA.jar :

java –jar AFLORA.jar

AFLORA permet de générer les informations de traçage régulier des RPRs sous forme de contraintes .ucf. Ces informations sont introduites dans le fichier .ucf initial du projet PlanAhead/Vivado qui se trouve dans le répertoire du projet suivant le chemin : **.\<nom_du_projet>.src\constrs_1\imports\<nom_du_dossier_projet>\<fichier_de_contraintes_initiale >.ucf**. La mise à jour des contraintes s'effectue automatiquement lors de l'ouverture du projet.

6.2.2 Uniformisation automatique des informations de P&R des Proxys Logiques et des interfaces statiques

Cette étape se déroule suite à une première implémentation du système avant laquelle le concepteur précise sa stratégie d'implémentation suivant l'ensemble des options fournies par les outils PlanAhead et Vivado. La première implémentation concerne toutes les configurations du système afin d'assurer la compatibilité de reconfiguration d'une RPR avec un ensemble des MPRs. Suite à la première implémentation le concepteur doit ouvrir la description .ncd du système sous l'outil FPGA_Editor et du programme DRC pour procéder à la génération des informations de placement et routage des Proxys Logique et des interfaces statiques au niveau de la RPR de référence. La génération de ces informations est effectuée suite à l'exécution du script dédié comme indiqué dans la Section 4.3.3.2.1 du Chapitre 4. Le mécanisme d'uniformisation des informations de P&R des Proxys Logiques et des interfaces statiques est exécuté ensuite en utilisant l'exécutable Uniformisation.jar exécuté en ligne de commande :

java –jar Uniformisation.jar

Le mécanisme d'uniformisation exploite les fichiers relatifs aux informations de P&R et la bibliothèque des DAAs des blocs élémentaires pour appliquer l'uniformisation 1D et 2D sur la plateforme convenable. Elle permet enfin de générer les descriptions de placement est routage de l'ensemble des RPRs à partir des éléments de référence avec les positions des identificateurs des blocs élémentaires sous forme de contraintes LOC (LUT) et PIN (Bel).

6.2.3 Implémentation finale et génération du bitstream partiel commun

Dans cette étape, le concepteur reprend le projet au niveau de la phase de l'implémentation suite à la mise à jour de la description des contraintes finales. L'implémentation englobe les différentes configurations du système dont chacune permet d'obtenir une description

commune (.ncd) de P&R des Proxys Logiques et des interfaces statiques. L'exécution du programme BitGen permet d'obtenir un nombre de Net_D multiplié par le nombre de configurations du système dans le dossier du projet, notamment dans le repertoire : **.\<nom du projet>.runs**\. Le paramétrage du programme BitGen doit désactiver le registre de CRC en appliquant la commande ***-g crc:disable.*** Lorsque le processus BitGen est achevé, le concepteur récupère à partir de chaque dossier de configuration les valeurs des registres FAR situées dans chaque bitstream et choisit arbitrairement un bitstream partiel qui représente le bitstream commun. Puis il récupère le bitstream relatif à la partie statique se trouvant dans le dossier relatif à la configuration BLANK.

6.3 Jeux de tests sur différents FPGAs-Xilinx pour différents grains de parallélisme et résultats expérimentaux

Dans cette étape, nous effectuons un jeu de tests de notre méthodologie proposée sur différentes plateformes FPGA-Xilinx en fonction de différents grains de parallélisme. En effet nous étudions l'application de floorplanning régulier en fonction du nombre de RPRs sur les FPGAs qui supportent la fonctionnalité de la RDP. Nous présentons tout d'abord les différentes ressources requises par les MPRs afin d'en déduire les ressources requises par la RPR commune. Cette étape est intégrée dans le processus d'AFLORA suite à la récupération des rapports statistiques de chaque RPR pendant la phase 2 du flot ADForMe. Le Tableau 6.1 illustre ces informations en termes de BRAMs, DSPs et Slices. Les ressources en Slices sont déterminées automatiquement à travers les réquisitions de base suite à la synthèse d'un IP associée aux réquisitions des Proxys Logiques. Les ressources requises par les éléments Proxys Logiques sont récupérés en termes de LUTs à convertir en Slices. Nous utilisons des IPs disposant de trois entrées de 32 bits et une sortie de 32 bits. Donc un total de 128 LUTs par IP en termes de Proxys Logiques équivalent à 32 Slices (=128/4).

IPs	DSPs	BRAMs	Slices		
			Résultat de Synthèse	**Proxys Logiques**	Total
IP_Kramer	0	0	36	32	68
IP_Sobel	0	0	40	32	72
IP_Muladd	**1**	0	16	32	48
IP_FIR	0	**2**	75	32	**108**
RPR_C	**1**	**2**	**108**		

Tableau 6.1 : Détermination des ressources requises par la configuration commune RPR_C à partir de quatre configurations

Les ressources requises pour la PRR_C sont calculées d'après le Tableau 6.1. Le vecteur PRR_C est alors représenté comme suit sachant qu'un CLB est composé par 2 x Slices:

$$RPR\_C = \begin{pmatrix} 54 \\ 2 \\ 1 \end{pmatrix}$$

Nous utilisons le FPGA Xilinx-Virtex7 XC7VX485T-FFG1761 pour cette étude de cas. Ce FPGA est caractérisé par ces paramètres technologiques présentés dans le Tableau 6.2:

Désignation	Définition
C	2
nbCR	14
Frame_C	50
Frame_D	20
Frame_R	10
V_{trans}	$V_{trans} = \begin{pmatrix} 50 \\ 10 \\ 20 \end{pmatrix}$
X_max	222
CR_Height	$\begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}_{BRAM}$
Unusd_Range (X)	[8,13], [80,91] et [208,213]

Tableau 6.2 : Les paramètres technologiques du FPGA Xilinx-Virtex7

En se référent à la phase 3 énoncé dans le Chapitre 3 au niveau de la Section 3.2.4, le mode de floorplanning suivant AFLORA est bien le mode 4 (CLB + BRAM + DSP). Pour permettre le calcul de la distribution équitable dans les régions d'horloges adjacentes (Z), nous traitons l'exemple d'un système à base de 9 RPRs. En effet, Z = 2 et le floorplanning complet sera effectué à travers deux RPR_C_{ref}.

Ensuite, le calcul du nombre des frames requis pour une RPR_C est donné par le vecteur Vect qui est calculé à partir de l'equation énoncée dans la Section 3.2.4.3 tout en rappelant que Frame_C = Frame_C x 2. Vect est alors égal à :

$$Vect = \begin{pmatrix} 54 \\ 2 \\ 1 \end{pmatrix} / \begin{pmatrix} 100 \\ 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 0.54 \\ 0.2 \\ 0.05 \end{pmatrix}$$

Puisque chacun des éléments du vecteur Vect est strictement inférieur à 1, l'hauteur de chaque RPR sera ajustée. En plus chacun des éléments du vecteur Vect est différent de 0, donc chaque RPR nécessite en termes de frames au moins 2 x CLB, 1x BRAM et 1 x DSP.
Suite à l'exécution du mécanisme de recherche des emplacements des RPR_C_{ref} suivant la priorité haute des éléments BRAMs, les localisations en termes des coordonnées (CLB_loc, BRAM_loc et DSP_loc) suivant l'axe des X sont présentées dans le Tableau 6.3.

RPR_C_{ref}	**Coordonnées (x) en Slices**		**Coordonnées (x) en BRAM**		**Coordonnées (x) en DSP**	
RPR_C0_{ref}	**CLB_loc_0 :**	CLBL_X=16 CLBR_X=23	**BRAM_loc_0 :**	BRAM_X=1	**DSP_loc_0 :**	DSP_X=0
RPR_C1_{ref}	**CLB_loc_1 :**	CLBL_X=34 CLBR_X=41	**BRAM_loc_1 :**	BRAM_X=2	**DSP_loc_1 :**	DSP_X=2

Tableau 6.3: Localisations des RPR_C_{ref} suivant l'axe des X

La séquence des ressources constituant la largeur de la RPR_C0_{ref} et contenues dans le tableau SEQ est représentée dans la Figure 6.2.

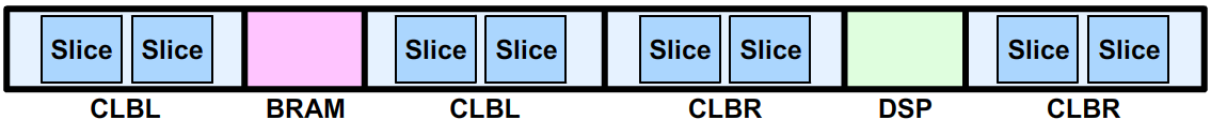


Figure 6.2 : Constitution du tableau SEQ

Le tableau SEQ est composé de 6 éléments de blocs fonctionnels. Nous remarquons que le résultat de recherche suivant la priorité haute accordée aux éléments BRAMs a nécessité l'allocation de deux CLBs supplémentaires ce qui représente malheureusement un gaspillage de ressources. Ce gaspillage se manifeste par l'allocation de deux Frame_C supplémentaires qui se trouvent entre les éléments BRAM et DSP d'après le tableau SEQ. Cet aspect est considéré comme un désavantage de la RDP Xilinx qui exige un CLBL et CLBR sur les bords d'une région partiellement reconfigurable. En revanche, la solution trouvée est considérée optimale pour le floorplanning des RPRs.
Dans le but d'optimiser et augmenter la précision de notre algorithme de recherche, la valeur du vecteur Vect est recalculée au niveau des ressources CLBs requises en fonction de la nouvelle configuration donnée par le tableau SEQ. En effet, le vecteur Vect sera égal à :

$$Vect = \begin{pmatrix} 54 \\ 2 \\ 1 \end{pmatrix} / \begin{pmatrix} 50x4 = 200 \\ 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 0.27 \\ 0.2 \\ 0.05 \end{pmatrix}$$

Une fois le vecteur Vect est recalculé, le calcul de la hauteur de chaque RPR est réalisé. Les éléments du vecteur Vect confirment que la hauteur de chaque RPR sera ajustée. Il faut donc calculer la valeur de H0 en fonction de la normalisation du vecteur V_{trans} pour donner le vecteur CR_Height. D'après la Section 3.2.4.5 du Chapitre 3, H0 est égal à :

$$H0 = \begin{pmatrix} 54 \\ 2 \\ 1 \end{pmatrix} / \begin{pmatrix} 200/10 \\ 10/10 \\ 20/10 \end{pmatrix}_{BRAM} = \begin{pmatrix} 2.7 \\ 2 \\ 0.5 \end{pmatrix}_{BRAM}$$

La valeur maximale du vecteur H0 est égale à 2.7. D'après la relation de H (même section), H est égal à 3. Il s'agit donc de RPRs de hauteur 3 BRAMs équivalent d'après le vecteur CR_Height à 15 CLBs et 6 DSPs. Une mise à jour des tableaux de localisation des deux RPR_C_{ref} au niveau des coordonnées Y est effectuée. Le résultat de la mise à jour est présenté dans le Tableau 6.4.

RPR_C_{ref}	**Coordonnées (x) en Slices**		**Coordonnées (x) en BRAM**		**Coordonnées (x) en DSP**	
RPR_$C0_{ref}$	**CLB_loc_0 :**	CLBL_X=16 CLBR_X=23 CLBL_Y=0 CLBR_Y=14	**BRAM_loc_0 :**	BRAM_X=1 BRAM_Y=0 BRAM_Y=2	**DSP_loc_0 :**	DSP_X=0 DSP_Y=0 DSP_Y=5
RPR_$C1_{ref}$	**CLB_loc_1 :**	CLBL_X=34 CLBR_X=41 CLBL_Y=0 CLBR_Y=14	**BRAM_loc_1 :**	BRAM_X=2 BRAM_Y=0 BRAM_Y=2	**DSP_loc_1 :**	DSP_X=2 DSP_Y=0 DSP_Y=5

Tableau 6.4: Localisations des RPR_C_{ref} suivant l'axe des X

Suite à la détermination complète des localisations des RPR_C_{ref}, les interfaces statiques peuvent être placées en superposition à droite de chacune des RPR_C_{ref}. Il s'agit bien de 4 interfaces statiques (3 d'entrées et une de sortie) dont chacune nécessite 4 Slices (32 bits). Les descriptions List11 et List12 illustrent la mise en œuvre des contraintes relatives à ces deux RPR_C_{ref} et leurs interfaces statiques.

```
# Slices ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region";
AREA_GROUP "pblock_my_MPR_my_region" RANGE = SLICE_X16Y0:SLICE_X23Y14;
AREA_GROUP "pblock_my_MPR_my_region" PRIVATE=ROUTE;
# Interfaces ----------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/IPIF_Nin" AREA_GROUP = "IPIF2";
AREA_GROUP "IPIF2" RANGE = SLICE_X24Y0:SLICE_X25Y1;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/IPIF_Tin" AREA_GROUP = "IPIF3";
AREA_GROUP "IPIF3" RANGE = SLICE_X24Y3:SLICE_X25Y4;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/IPIF_N2in" AREA_GROUP = "IPIF1";
AREA_GROUP "IPIF1" RANGE = SLICE_X24Y6:SLICE_X25Y7;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/IPIF_Tout" AREA_GROUP = "IPIF0";
AREA_GROUP "IPIF0" RANGE = SLICE_X24Y9:SLICE_X25Y10;
# BRAMs ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region";
AREA_GROUP "pblock_my_MPR_my_region" RANGE = RAMB36_X1Y0:RAMB36_X1Y2;
# DSPs ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST "Homadev5/my_Master/Xslaves[2].Yslaves[2].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region";
AREA_GROUP "pblock_my_MPR_my_region" RANGE = DSP48_X0Y0:DSP48_X0Y5;
```

List11 : Contraintes de localisation de la RPR_C0_{ref} est ses interfaces statiques

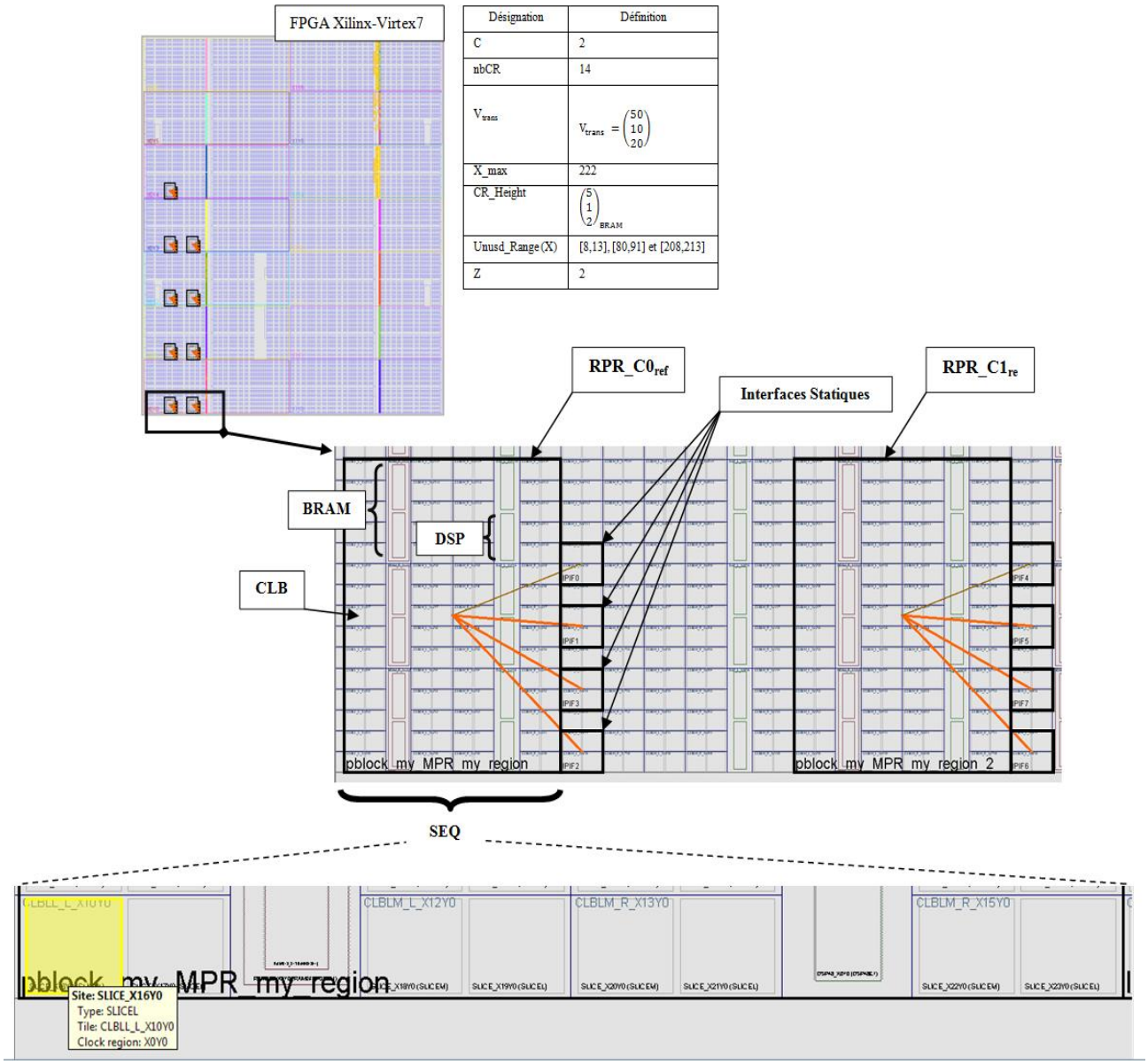
```
# Slices ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST"Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region_2";
AREA_GROUP "pblock_my_MPR_my_region_2" RANGE = SLICE_X34Y0:SLICE_X41Y14;
AREA_GROUP "pblock_my_MPR_my_region_2" PRIVATE=ROUTE;
# Interfaces ----------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/IPIF_Nin" AREA_GROUP = "IPIF6";
AREA_GROUP "IPIF6" RANGE = SLICE_X42Y0:SLICE_X43Y1;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/IPIF_Tin" AREA_GROUP = "IPIF7";
AREA_GROUP "IPIF7" RANGE = SLICE_X42Y3:SLICE_X43Y4;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/IPIF_N2in" AREA_GROUP = "IPIF5";
AREA_GROUP "IPIF5" RANGE = SLICE_X42Y6:SLICE_X43Y7;
#-------------------------------------------------------------------
INST "Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/IPIF_Tout" AREA_GROUP = "IPIF4";
AREA_GROUP "IPIF4" RANGE = SLICE_X42Y9:SLICE_X43Y10;
# BRAMs ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST"Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region_2";
AREA_GROUP "pblock_my_MPR_my_region_2" RANGE = RAMB36_X2Y0:RAMB36_X2Y2;
# DPSs ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INST"Homadev5/my_Master/Xslaves[2].Yslaves[1].One_salve/my_MPR/my_region" AREA_GROUP = "pblock_my_MPR_my_region_2";
AREA_GROUP "pblock_my_MPR_my_region_2" RANGE = DSP48_X2Y0:DSP48_X2Y5;
```

List12 : Contraintes de localisation de la RPR_C1_{ref} est ses interfaces statiques

La phase de réplication est exécutée ensuite en utilisant les informations de placements illustrées dans le Tableau 6.4 en fonction du vecteur de translation V_{trans}. La réplication est répartie sur les quatre régions d'horloges situées au dessus de la région d'horloge de référence Adj_CR_0. Au fur et à mesure, les contraintes de placement des RPRs sont générées et introduites dans le fichier de contraintes .ucf du système. Le résultat du floorplanning est présenté dans la Figure 6.3 suite à l'exécution de la fonction ***Open Synthesized Design*** dans le projet PlanAhead/Vivado puis l'affichage de l'onglet ***Device***.

La Figure 6.3 illustre le floorplanning régulier sur le FPGA Xilinx-Virtex7 - XC7VX485T-FFG1761 caractérisé par les paramètres technologiques illustrés dans le Tableau 6.2. En effet, le floorplanning est répartit sur cinq Adj_CRs. La Figure montre bien aussi que la hauteur de chaque RPR est égal à la hauteur de 3 BRAMs et d'une largeur composée par la séquence SEQ. La Figure montre aussi que la séquence SEQ débute par le Slice_X16Y0 comme indiqué dans le Tableau 6.4, de type SLICEL (Logique) et contenu dans le CLBLL_L_X10Y0 (CLB côté gauche) qui fait partie de la région d'horloge X0Y0.

Dans le but d'évaluer notre méthodologie d'automatisation ADForMe en termes de temps d'exécution et montrer son efficacité, nous avons élaboré différentes configurations d'un système multi-RPRs sur le FPGA Xilinx-Virtex7 - XC7VX485T-FFG1761 en fonction du nombre des RPRs (Net_D). En effet, nous avons implémenté des systèmes multi-RPRs de tailles: Net_D = 4, Net_D = 9, Net_D = 21, Net_D = 25, Net_D = 30, et Net_D = 42 en fonction du vecteur RPR_C = (54 2 1). Le résultat du floorplanning est représenté dans la Figure 6.4.



Désignation	Définition
C	2
nbCR	14
V_{trans}	$V_{trans} = \begin{pmatrix} 50 \\ 10 \\ 20 \end{pmatrix}$
X_max	222
CR_Height	$\begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}_{BRAM}$
Unusd_Range(X)	[8,13], [80,91] et [208,213]
Z	2

Figure 6.3 : Résultat du floorplanning d'un système multi_RPRs à base de 4 configurations, de taille Ned_D = 9 avec une configuration commune RPR_C = (45 2 1) sur le FPGA Xilinx-Virtex7 - XC7VX485T-FFG1761

Nous avons utilisé le même FPGA pour implémenter des systèmes de même taille : Net_D = 25 mais pour différentes ressources requises par la configuration RPR_C. Les résultats de floorplanning sont représentés dans la Figure 6.5.

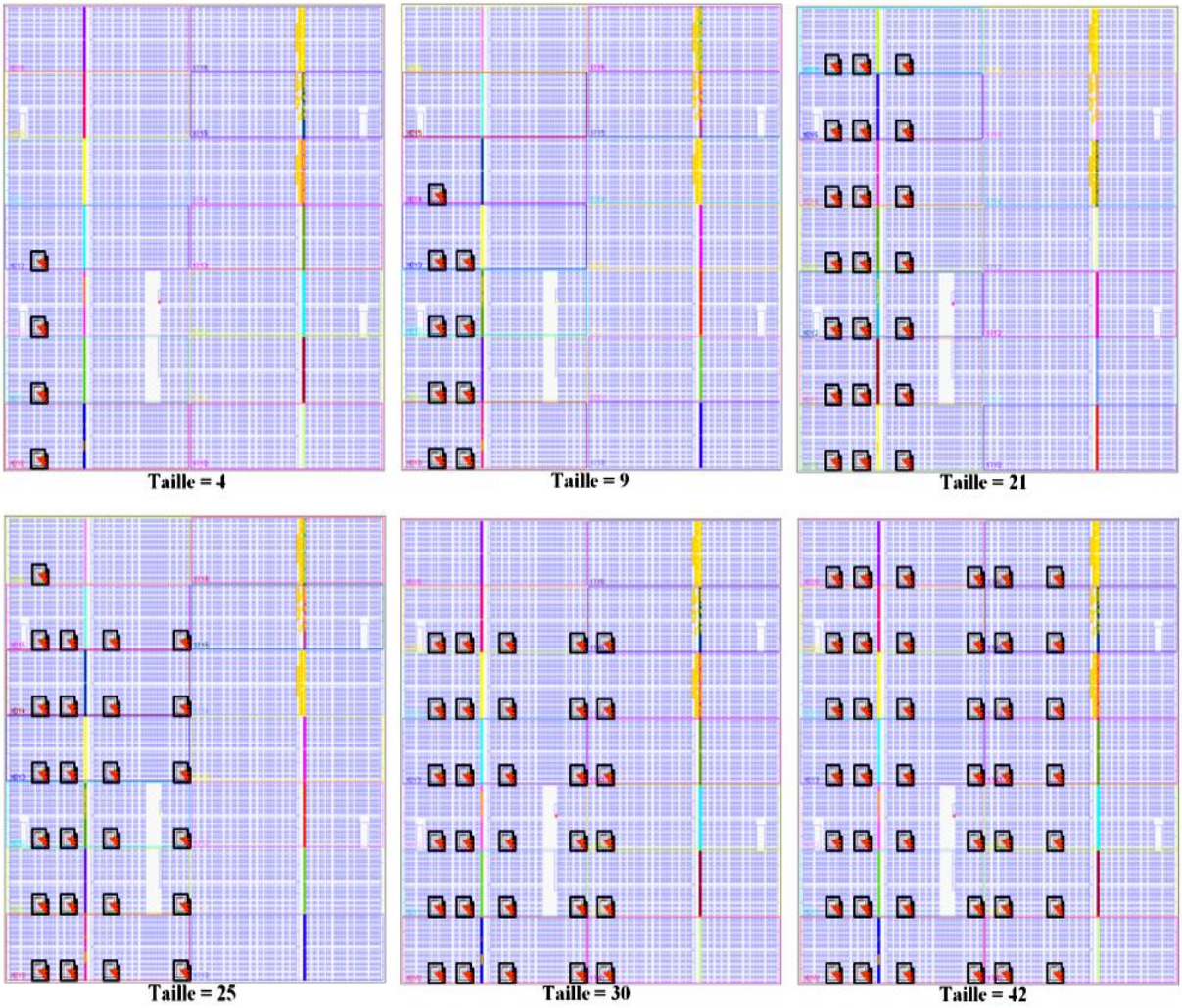


Figure 6.4 : Résultat de floorplanning d'un système Multi_RPRs ayant 4 configurations pour différentes tailles. Le floorplanning effectué en utilisant la configuration commune RPR_C = (45 2 1) sur FPGA Xilinx-Virtex7 - XC7VX485T-FFG1761

Le floorplanning effectué sur le FPGA Xilinx-Virtex7-XC7VX485T-FFG1761 ne permet qu'un traçage au maximum 42 RPRs avec la configuration RPR_C = (45 2 1). Ceci s'explique par le fait que l'algorithme de recherche n'arrive pas à trouver plus que 6 occurrences (Z = 6) de la séquence SEQ au niveau de l'Adj_CR_{ref}_0.

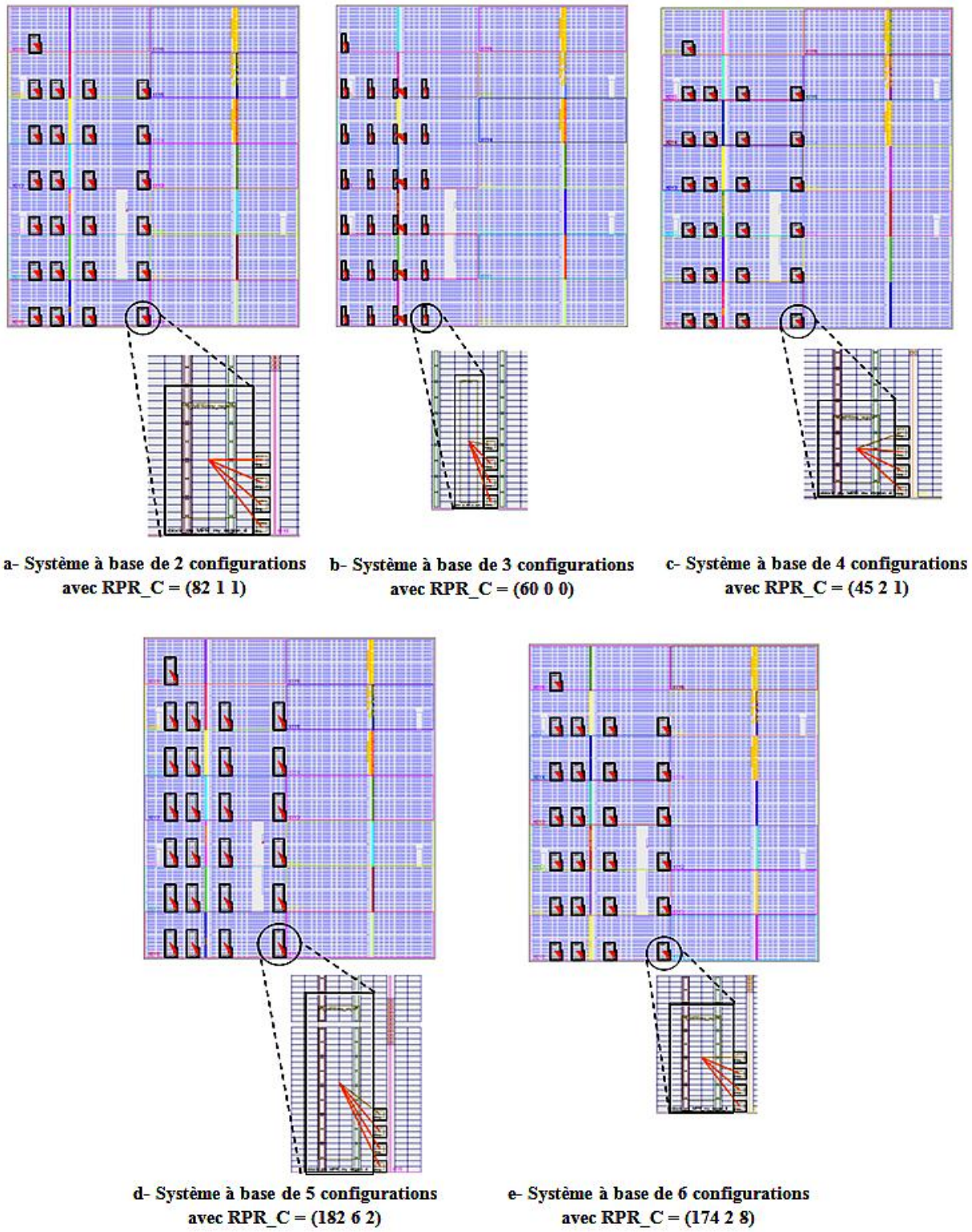


Figure 6.5 : Résultat de floorplanning d'un système Multi_RPRs de taille 25 pour différentes configurations sur FPGA Xilinx-Virtex7 - XC7VX485T-FFG1761

Nous avons également implémenté ce système sur différents FPGAs en fonction d'une même configuration commune RPR_C = (45 2 1) déduite à partir de 4 configurations avec

une taille de 16 RPRs. Nous avons utilisé le FPGA Virtex-6 XC6VLX240T-1FFG1156 et les FPGAs-Xilinx de la série 7 : Zynq (XC7Z100-2FFG1156), Kintex (XC7K480T-3FFG1156), Vitrex7 (XC7V2000T-2FTG1761) et Virtex7 (XC7VX485T-FFG1761). Nous présentons dans le Tableau 6.5 les différents paramètres technologiques de chacun de ces FPGAs avec le calcul de la distribution équitable dans les régions d'horloges adjacentes (Z) et la hauteur de chaque RPR. Les résultats de floorplanning sont représentés dans la Figure 6.6.

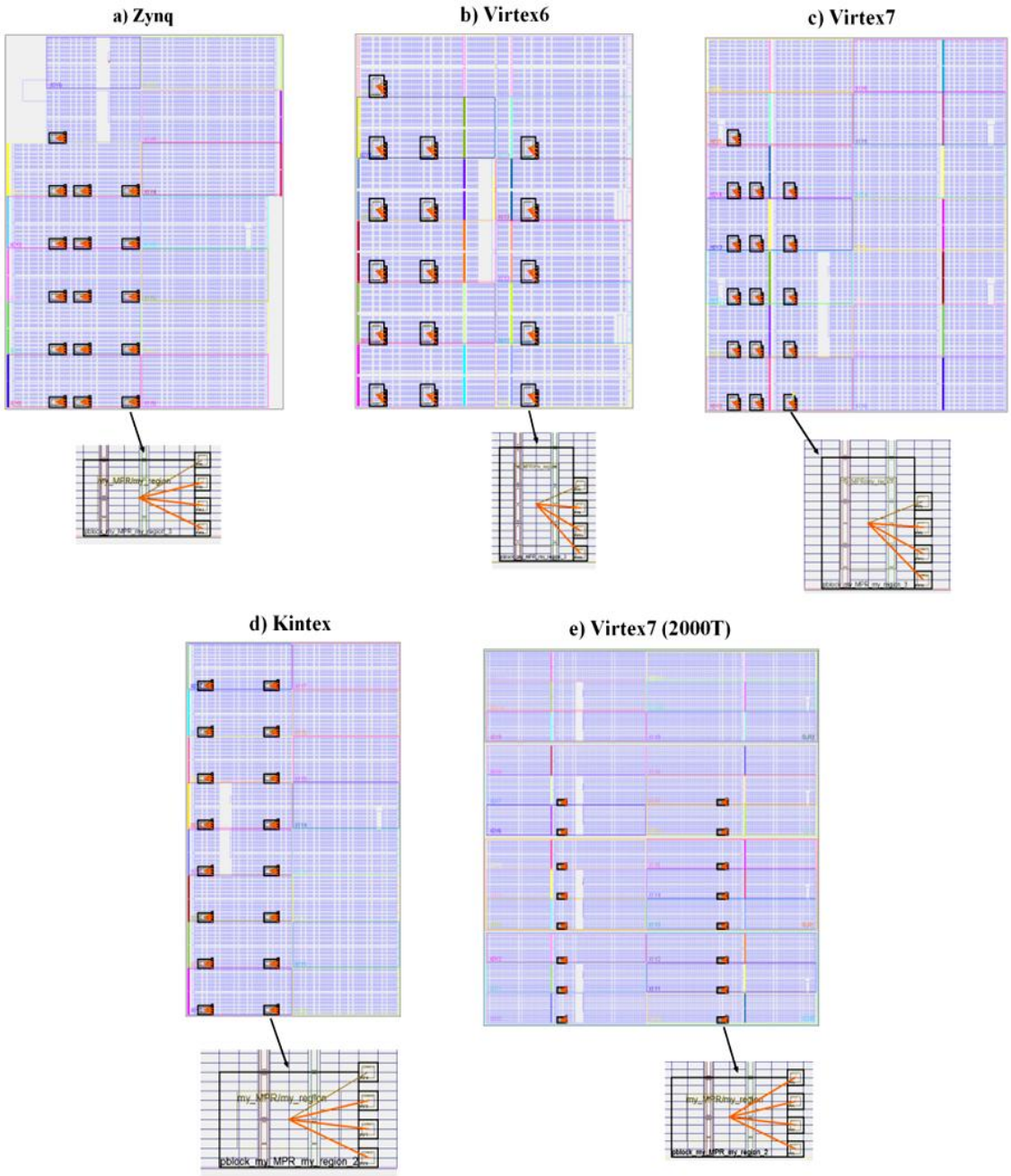


Figure 6.6 : Résultat de floorplanning d'un système Multi_RPRs de taille 16 ayant 4 configurations sur différents FPGAs-Xilinx

FPGAs / **Paramètres**	**Virtex7 (XC7VX485T-FFG1761)**	**Virtex-6 XC6VLX240T-1FFG1156**	**Zynq (XC7Z100-2FFG1156)**	**Kintex (XC7K480T-3FFG1156)**	**Vitrex7 (XC7V2000T-2FTG1761)**
C	2	2	2	2	2
nbCR	14	24	14	16	24
V_{trans}	$\begin{pmatrix}50\\10\\20\end{pmatrix}$	$\begin{pmatrix}40\\8\\16\end{pmatrix}$	$\begin{pmatrix}50\\10\\20\end{pmatrix}$	$\begin{pmatrix}50\\10\\20\end{pmatrix}$	$\begin{pmatrix}50\\10\\20\end{pmatrix}$
X_max	222	162	206	190	506
CR_Height	$\begin{pmatrix}5\\1\\2\end{pmatrix}_{BRAM}$	$\begin{pmatrix}5\\1\\2\end{pmatrix}_{BRAM}$	$\begin{pmatrix}5\\1\\2\end{pmatrix}_{BRAM}$	$\begin{pmatrix}5\\1\\2\end{pmatrix}_{BRAM}$	$\begin{pmatrix}5\\1\\2\end{pmatrix}_{BRAM}$
Unusd_Range (X)	[8,13], [80,91] et [208,213]	[72, 83] et [156, 161]	[0, 25], [66, 77] et [190, 195]	[24, 35] et [170, 175]	[136, 147] et [504,505]
Z	3	3	3	2	2
H_{BRAM}	3	3	2	2	2
Taille de la séquence SEQ	10	10	14	14	14

Tableau 6.5: Les Paramètres technologiques des FPGAs utilisés et les dimensions des RPRs_C pour un système multi_RPRs de taille 16 ayant 4 configurations.

Suite à cette étape, le concepteur procède à l'exécution de la première implémentation du système en utilisant la stratégie d'implémentation adéquate. Une implémentation d'une configuration suffit pour permettre la récupération des informations de P&R des Proxys Logiques et les interfaces statiques. Par exemple, pour un système à base de 4 configurations, il suffit d'implémenter une seule pour avoir ces informations. La phase d'implémentation se manifeste par l'étape de translate, ensuite l'étape de MAP et enfin l'étape de P&R. l'implémentation permet de générer les descriptions .ncd et .pcf. Ces descriptions sont exploitées en utilisant l'outil FPGA_Editor pour nous permettre l'extraction des informations de P&R dans le but de favoriser la relocalisation 1D et 2D en appliquant le mécanisme d'uniformisation. La Figure 6.7 montre le résultat d'implémentation sur le FPGA Xilinx-Virtex7 (XC7VX485T-FFG1761) d'un système à base de 16 RPRs ayant 4 configurations.

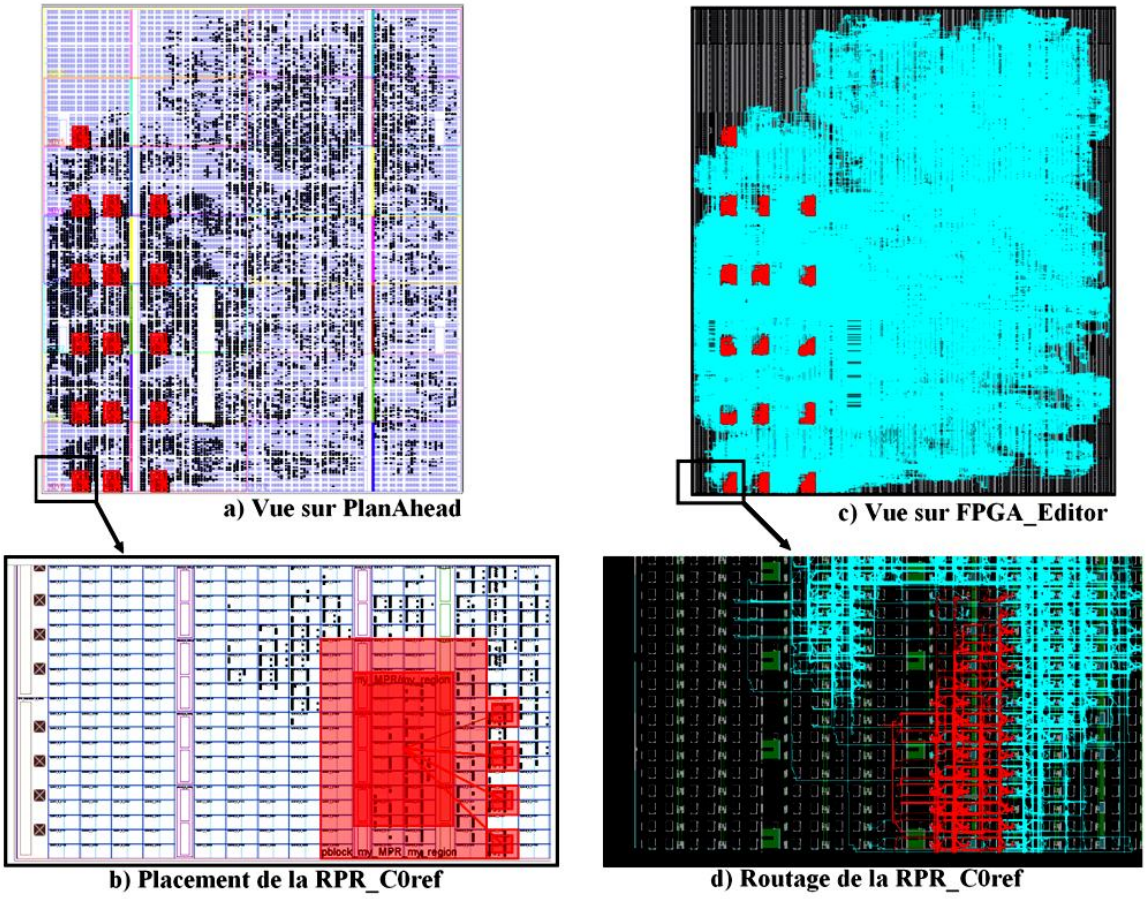


Figure 6.7 : Résultat de la première implémentation

La Figure 6.7 (b, d) montre aussi le placement et le routage de la RPR_C0$_{ref}$. Cette dernière est utilisée comme RPR_C de référence pour établir la relocalisation 1D au niveau de la colonne à laquelle elle appartient. D'autre part, dans le but de favoriser la relocalisation 2D, à partir de la matrice des RPR_Cs illustrée dans la Figure 4.14 et construite au fur et à mesure de l'exécution du mécanisme AFLORA, le script SCR compatible à l'outil FPGA_Editor est généré. Il sert à sélectionner et générer les informations de P&R de l'ensemble des broches (net) de chaque Proxy de chacune des RPR_C$_{ref}$ se trouvant sur la même ligne que la RPR_C0$_{ref}$ ainsi que leurs interfaces d'E/S correspondantes. L'exécution du script par l'outil *FPGA_EDITOR* se fait en mode ligne de commande avec la commande suivante:

fpga_editor –n config_1.ncd config_1.pcf –p RPRCREF.scr

Dans cet exemple, 3 x RPR_C$_{ref}$ sont situées dans Adj_CR_0. Deux fichiers de contraintes sont donc générés pour chaque RPR_C$_{ref}$. Le premier contenant les informations de P&R des Proxys Logiques et le deuxième contenant les informations de P&R des interfaces statiques

d'E/S. Suivant notre démarche d'uniformisation énoncée dans le Chapitre 4, le mécanisme favorisant la relocalisation 1D et 2D est exécuté en fonction de ces 6 fichiers de contraintes ainsi que la description de DAA correspondant à l'FPGA utilisé. Le résultat d'exécution se manifeste par la génération des informations de P&R des Proxys Logiques et les interfaces statique d'E/S uniformisés au niveau de toutes des RPR_Cs sous forme de contraintes. Le fichier UCF du système est alors mis à jour pour favoriser la deuxième implémentation du système. En effet, la deuxième implémentation se manifeste par la génération des 4 configurations du système suivie par production de la configuration initiale (BLANK). Suite à l'implémentation de ces 5 configurations, la phase de génération des bitstreams statiques et partielles est exécutée.

6.4 Analyse des résultats expérimentaux

Notre analyse des résultats expérimentaux est basée sur deux aspects essentiels. Dans le but d'évaluer l'amélioration de la productivité du concepteur, le premier aspect consiste à évaluer en termes de temps d'exécution du flot d'automatisation d'ADForMe qui englobe en premier le temps d'exécution de la méthodologie permettant de construire un système Multi-RPRs puis le temps d'exécution du processus AFLORA. Le deuxième aspect concerne l'impact de favoriser la reconfigurabilité sur la performance du système. Il englobe le surcoût en termes de ressources permettant la relocalisation de bitstreams ainsi que l'évaluation de la vitesse d'exécution du système.

6.4.1 Evaluation en termes de temps d'exécution du flot d'automatisation d'ADForMe

Dans le but d'évaluer le temps d'exécution des deux mécanismes constituant le flot d'automatisation ADForMe nous avons traité différents aspects d'un système Multi-RPRs en fonction de différents paramètres qui sont la taille du système (Net_D), le type du FPGA et le nombre de configurations aboutissant à une configuration commune. Nous avons également évalué le temps d'exécution du mécanisme AFLORA en fonction de sa complexité.

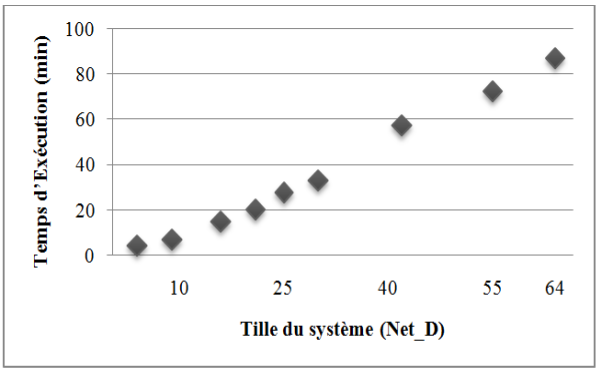
6.4.1.1 Evaluation en termes de temps d'exécution du mécanisme de construction du système Multi-RPRs

Pour permettre le calculer du temps d'exécution du mécanisme de construction du système reconfigurable, nous avons intégré au niveau du script TCL les commandes illustrées dans List13.

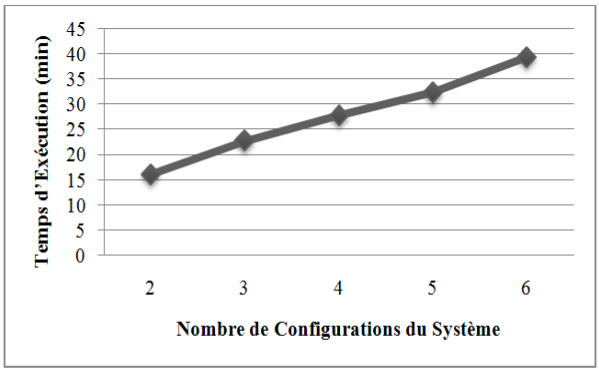
```
…
Set TIME_Start [clock clicks -millisecondes]                    // déclanchement du traitement
…
(Traitement : construction du système Multi-RPRs)
…
Set TIME_stamp [ exp [clock clicks -millisecondes] - $TIME_Start]  //calcule de la différence entre le temps de
                                                                   //déclanchement et le temps de fin de traitement
puts $TIME_stamp
```

List13 : Calcul du temps d'exécution du mécanisme de construction du système Multi-RPRs

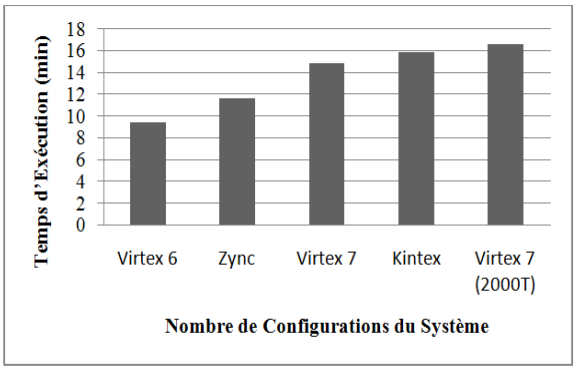
La Figure 6.8 illustre l'évolution du temps d'exécution en fonction des paramètres : taille (a), le nombre de configurations du système (b) et le FPGA utilisé (c).



a- Le temps d'exécution en fonction de la taille du système Multi-RPRs



b- Le temps d'exécution en fonction du nombre de configurations du système



c- Le temps d'exécution en fonction du FPGA utilisé

Figure 6.8 : Temps d'exécution du mécanisme de construction du système Multi-RPRs

En se référant au paramétrage à l'issu duquel nous avons obtenu le floorplanning illustré dans la Figure 6.4, la Figure 6.8.a montre l'évolution du temps d'exécution en fonction de la taille du système en fixant le nombre de configurations à 4 et le FPGA utilisé (Xilinx- Virtex7 XC7VX485T-FFG1761). Cette évaluation exprime le temps enduré pour créer les éléments PBlocks relatifs à chaque RPR ainsi que la création de leurs interfaces correspondantes. En gardant le même FPGA utilisé et en fixant la taille du système à 16 suivant le paramétrage

illustré dans la Figure 6.5, la Figure 6.8.b illustre l'évolution du temps d'exécution en fonction du nombre de configurations du système (égal au nombre des modules reconfigurables). Enfin, en fixant la taille du système à 16 et le nombre de configurations à 4 comme indiqué dans la Figure 6.6, la Figure 6.8.c illustre l'évolution du temps d'exécution en fonction des paramètres technologiques des FPGAs ainsi que la taille de leurs bitstreams statiques. Nous remarquons d'après les Figures 6.8.a et 6.8.b que l'évolution du temps d'exécution est linéaire puisqu'il s'agit de l'élaboration des tâches logicielles relatives à l'architecture reconfigurable avec l'outil PlanAhead. D'autre part, d'après la Figure 6.8.c, le temps d'exécution nécessaire pour construire un système Multi-RPRs reste une caractéristique spécifique pour chaque FPGA puisque les paramètres technologiques peuvent varier d'un FPGA à un autre.

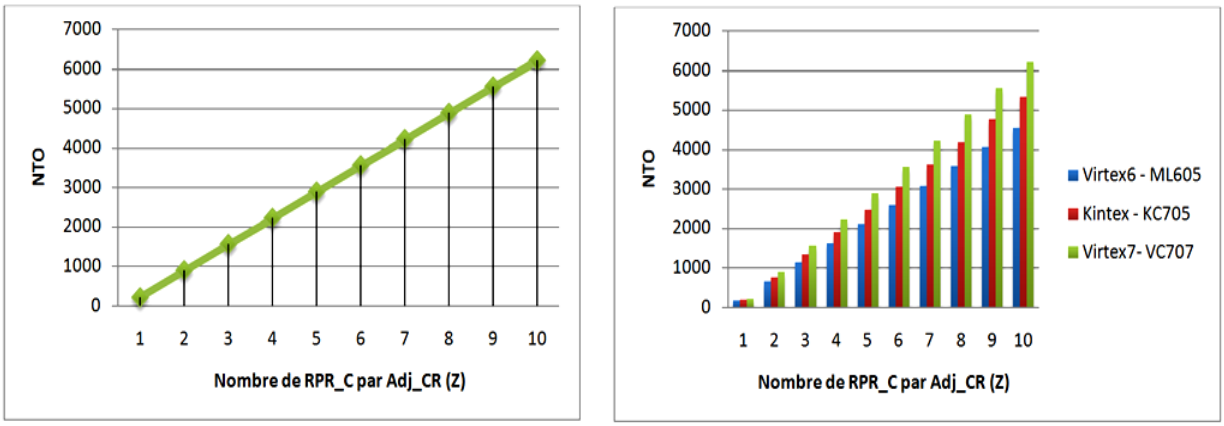
6.4.1.2 Evaluation en termes de temps d'exécution du mécanisme AFLORA

Afin d'évaluer les performances d'AFLORA, nous mettons en œuvre différentes configurations de différentes tailles sur FPGAs Xilinx Virtex7 XC7VX485T. Les expériences sont menées à l'aide du processeur Intel Xeon CPU E31270 de 3.4GHZ et 16 Go de RAM pour permettre le calcul du temps d'exécution réel. La Figure 6.9.a montre les évaluations théoriques du Nombre Total d'Opérations (NTO) en fonction de la taille du système selon l'équation décrite dans la Section 3.3 du Chapitre 3 :

$$\mathbf{NTO = NTO = 3.Z.X\_max - 2.X\_max}$$

La Figure 6.9.b montre l'évolution théorique du nombre d'opération en fonction du FPGA utilisé et plus précisément en fonction de la largeur X_max du FPGA.

L'évolution du NTO en fonction de la taille du système en termes de nombre de RPRs est proportionnelle avec la distribution équitable dans les régions d'horloges adjacentes (Z). En effet, plus que Z augmente le nombre d'opérations augmente à cause de la nécessité des recherches supplémentaires de l'occurrence de la séquence SEQ sur la largeur des régions d'horloges adjacentes de référence.
D'autre part, le nombre d'opérations varie en fonction du paramètre technologique X_max comme illustré dans la Figure 6.9.b.



a- Evolution du NTO en fonction de la taille du système

b- Evolution du NTO en fonction du paramètre technologique du FPGA utilisé

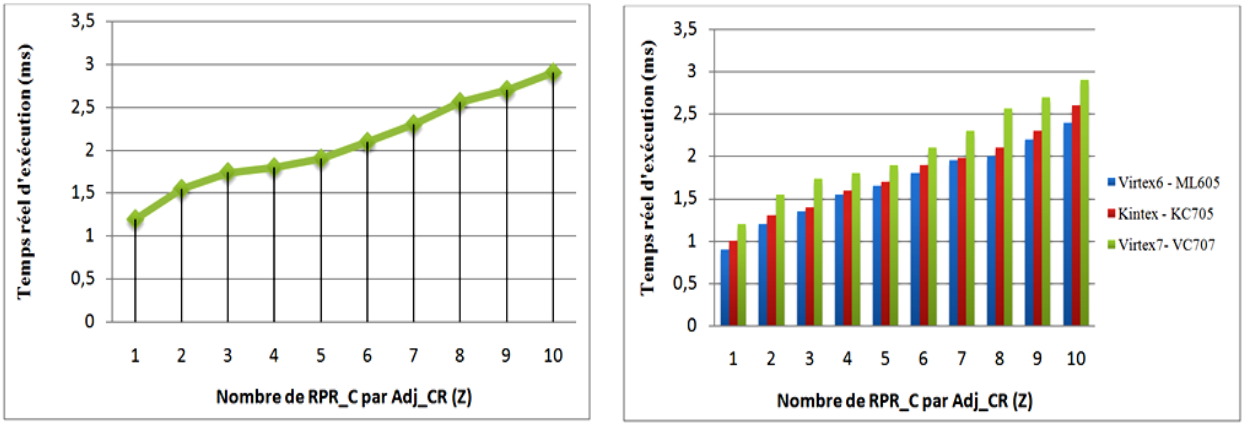
Figure 6.9 : Evaluation du nombre d'opérations du mécanisme AFLORA

En effet, en se référant aux paramètres X_max des FPGAs utilisés dans le tableau 6.5, nous remarquons que le FPGA Kintex ayant la largeur X_max = 190 consomme moins d'opérations que le FPGA Virtex7 ayant une largeur X_max = 222. De plus, vu le nombre d'opérations mesuré sur le FPGA virtex6 (X_max =162), le nombre d'opérations mesuré en utilisant le FPGA Kintex consomme plus ou moins d'opérations vu que pour un système Multi-RPRs de taille fixe, la valeur du paramètre Z est différents suivant l'exemple présenté dans le tableau 6.5. Suivant l'équation permettant de calculer le paramètre Z, nous remarquons que le nombre des régions d'horloges a un impact sur le nombre d'opérations.

Nous confirmons ainsi l'efficacité d'application de notre algorithme de floorplanning automatique AFLORA sur différents FPGAs pour des systèmes reconfigurables de différentes dimensions.

La Figure 6.10 illustre le temps réel d'exécution sur le processeur de l'algorithme AFLORA. Nous remarquons l'accordance du comportement entre les résultats théoriques obtenus dans la Figure 6.9 et les résultats expérimentaux.

Nous remarquons d'après ces résultats que pour un système Multi-RPRs implémenté sur le FPGA Virtex7 XC7VX485T et de taille 64 (Z = 10) en considèrant que ce FPGA dispose des ressources nécessaires et d'une géométrie adéquate pour mettre en œuvre ce système, que le NTO théorique est égal à 6216 comparaisons qui correspond à 2.8 ms de temps d'exécution sur le processeur utilisé. Ainsi, notre algorithme automatique conduit à une productivité de conception élevée par rapport au floorplanning manuel qui nécessite plusieurs heures.



a- Evolution du temps réel d'exécution en fonction de la taille du système

b- Evolution du temps réel d'exécution en fonction du paramètre technologique du FPGA utilisé

Figure 6.10 : Evaluation du temps d'exécution réel du mécanisme AFLORA sur processeur Intel Xeon CPU E31270

6.4.2 L'impacte de favoriser la reconfigurabilité sur la performance d'un système Multi-RPRs

6.4.2.1 Le surcoût en termes de ressources

Il est évident que l'utilisation des interfaces statiques au niveau de chaque RPR nécessité un surcoût en termes de ressources logiques (LUTs). Etant donné que chaque RPR dispose au maximum de 3 ports d'entrée et 3 ports de sortie dont chacune est spécifiée sur 32 bits et chaque broche requière un élément LUT pour être placée au sein de l'interface, le coût total est donc de 192 LUTs (48 Slices) au maximum pour une seule RPR. La Figure 6.11 illustre les différentes combinaisons d'E/S possibles qu'un système Multi-RPRs peut nécessiter en fonction de sa taille (9, 25, et 64) et en fonction de ce qu'une RPR peut réquisitionner.

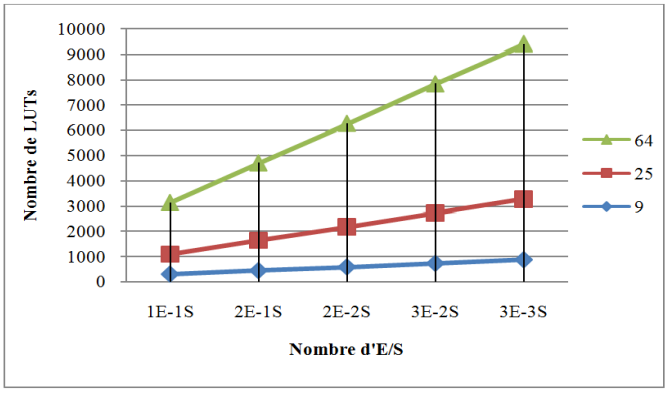


Figure 6.11 : Evolution du surcoût en termes de LUTs en fonction de la taille du système Multi-RPRs et les réquisitions en termes de ports d'E/S par RPR

La Figure 6.11 montre une évolution linéaire du surcoût en termes de LUTs. Un système Multi-RPRs de taille 64 nécessite 3072 Slices pour interfacer 6 ports d'E/S, ce qui correspond à peu près 1% des ressources logiques disponibles sur le FPGA Virtex7 XC7VX485T. Ce surcoût est négligeable pour les FPGAs récents riches en ressources pour permettre plus de flexibilité d'un système Multi-RPRs.

6.4.2.2 Evaluation de la fréquence

Soit deux systèmes multiprocesseurs de même taille en termes d'unités de calcul, reposant sur la même stratégie de synthèse et d'implémentation et synchronisés avec la même fréquence de consigne. Le premier est un système reconfigurable Multiprocesseurs (Multi-RPR) dont chaque RPR comporte un seul module partiellement reconfigurable bien spécifique (une seule configuration du système). Suite à l'étape de synthèse, translate, MAP et P&R en utilisant les outils Xilinx, la fréquence globale du système est récupérée à partir du rapport « *Static Timing Report* » généré par PlanAhead 14.7. Le deuxième système Multiprocesseurs est totalement statiqueoù le module utilisé dans le premier système est un module statique. La fréquence est récupérée à partir du même rapport mais sur l'outil ISE 14.7.

A la fin des processus d'implémentation et la récupération des fréquences efficaces de chaque système, nous remarquons que le système reconfigurable est légèrement moins rapide que le système statique. L'évolution de la fréquence de ces deux systèmes en fonction de leur taille, implémentés sur le FPGA Virtex7 XC7VX485T avec une fréquence de consigne de 200 Mhz est illustré dans la Figure 6.12. Cette Figure montre la diminution de la fréquence moyenne de 3% du système dynamiquement reconfigurable par rapport au système statique. Ceci s'explique par le fait que le placement et routage dans le système dynamiquement reconfigurable à subit un forçage dû au contraintes de P&R que nous venons de mettre en œuvre pour favoriser la relocalisation de bitstreams. Cet aspect exerce une influence sur le chemin critique de ce système, d'où la baisse de la fréquence de fonctionnement. Cependant, cette réduction est négligeable par rapport à la flexibilité du système grâce à la relocalisation de bitstream ainsi qu'un temps de conception plus rapide.

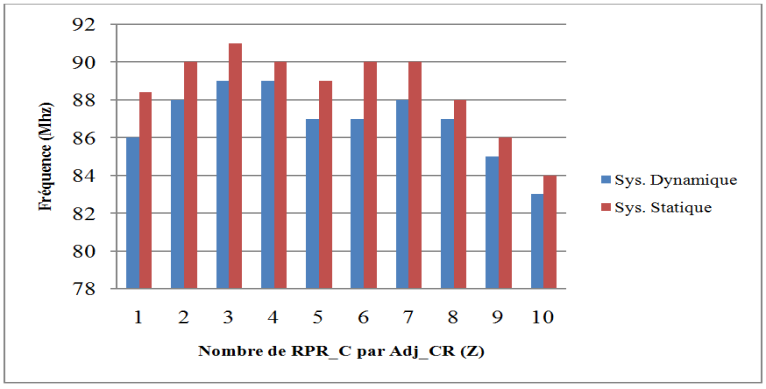


Figure 6.12 : Comparaison des fréquences réalisées entre un système statique et un système dynamique

6.4.3 Etude comparative

Notre étude comparative porte en premier temps sur la comparaison du temps d'exécution du mécanisme AFLORA par rapport aux algorithmes existants. Le Tableau 6.6 montre les temps d'exécution de floorplanning des travaux [38], [40], [45] et [36] ainsi que le temps d'exécution d'AFLORA en fonction du nombre des RPRs. Ces travaux cités permettent de garantir un floorplanning qui vise à satisfaire des contraintes temporelles relatives au temps de propagation entre l'ensemble des modules du système. Contrairement au floorplanning assuré par AFLORA, ces travaux ne permettent pas d'effectuer un floorplanning régulier visant la relocalisation de bitstream.

#RPR	(HO) [38]	(O) [38]	[45]	[40]	[36]	AFLORA
3	-	-	2.5 min	2 min	-	1.2 ms
5	17.2 sec	2.3 min	-	2 min	10.9 sec	1.2 ms
8	-	-	3.33 min	2 min	-	1.6 ms
10	1.9 min	31.9 min	-	3 min	23.9 sec	1.6 ms
15	2.3 min	32.07 min	6.5 min	3.5 min	40.6 sec	1.7 ms
20		32.38 min	-	4 min	65 sec	1.7 ms
24	-	-	8.8 min	5 min	-	1.7 ms
25	2.97 min	32.97 min	-	5.5 min	93 sec	1.9 ms
32	-	-	-	9.7 min	-	2 ms

Tableau 6.6 : Comparaison du temps d'exécution entre l'algorithme AFLORA et les travaux existants

Tous les résultats ont été obtenus en utilisant le processeur Intel Core Duo T6600 - 2.2 GHz avec un système d'exploitation Linux. Le Tableau 6.6 confirme la rapidité de notre algorithme de floorplanning régulier AFLORA par rapport aux travaux existants. Cependant, ceci s'explique par la nature heuristique de notre algorithme contrairement à ces travaux qui se basent sur des algorithmes de résolution des problèmes d'optimisation linéaire qui sont généralement gourmands en termes de temps d'exécution.

Le deuxième point de discussion est la complexité. Nous remarquons que la complexité de l'algorithme AFLORA est la même (O(n)) comparée à celles des travaux cités dans le Tableau 6.6. Ceci prouve l'efficacité de notre algorithme puisqu'il permet le floorplanning rapide avec un nombre d'opérations réduit.

Ces deux points remarquables de notre algorithme de floorplanning AFLORA permettent d'augmenter la productivité du concepteur qui utilise les FPGAs Xilinx.

6.5 Conclusion

Dans ce chapitre, nous avons démontré le déroulement des flots d'automatisation proposés. Ensuite, nous avons détaillé les différentes fonctionnalités et les étapes du mode opératoire sur différentes familles FPGA-Xilinx avec leurs diversités technologiques pour différents grains de parallélisme. Ceci nous a permis de prouver l'efficacité des flots proposés ainsi que la flexibilité de notre algorithme AFLORA puisque son exécution s'applique sur les familles récentes des FPGA Xilinx et pouvant s'adapter rapidement aux exigences applicatives grâce au paramétrage technologique et architectural. Les résultats expérimentaux ont montré l'efficacité des deux mécanismes d'automatisation et leurs intérêts dans la conception des systèmes multi-RPRs dynamiquement reconfigurables.

CHAPITRE 7 : Conclusion et Perspectives

7.1 Conclusion Générale

Dans cette thèse nous avons présenté deux mécanismes d'automatisation de deux flots de conception complémentaires visant à favoriser la reconfiguration dynamique partielle parallèle.

Le premier flot proposé, ADForMe, vise à améliorer le flot de la RDP traditionnel de Xilinx. Il permet d'établir dans un système à multi-RPRs, un floorplanning automatique régulier et compatible à la relocalisation de bitstream. Ce flot se base d'une part sur l'exploration des plateformes FPGA-Xilinx tout en extrayant des paramètres technologiques et d'une autre part sur l'analyse de l'architecture Multi-RPRs et d'en extraire des paramètres architecturaux dans le but de favoriser l'exécution de l'algorithme AFLORA que nous avons développé. La complexité de l'algorithme AFLORA est linéaire en fonction de O(n). En effet AFLORA permet d'effectuer le floorplanning des RPRs d'une manière rapide.

Le deuxième flot proposé vise à favoriser la technique de relocalisation 1D et 2D afin de permettre le broadcast d'un bitstream partiel (fonctionnalité) vers un ensemble de RPRs pour une configuration du système. En effet, la relocalisation s'effectue suite à l'uniformisation des P&Rs entre les interfaces des RPRs et la partie statique à partir d'un ensemble de RPRs de référence. Ce flot permet donc l'optimisation de la taille de la mémoire de bitstream partiels.

Ces deux flots permettent d'assurer la flexibilité et la réutilisabilité des composants IPs intégrés dans les architectures à Multi-RPRs afin de réduire la complexité en termes de temps de conception et d'améliorer productivité des concepteurs.

Dans le but permettre l'application de ces deux flots de conception proposés, nous avons proposé une architecture matérielle adéquate. Cette architecture est capable d'effectuer le broadcast d'un bitstream partiel vers un ensemble de RPRs grâce à deux mécanismes essentiels qui sont : le control d'écriture/lecture des bitstreams partiels dans/depuis la mémoire externe, le déclanchement et le control de la RDPP permettant de reconfigurer dynamiquement les la mémoire reconfigurable du FPGA.

7.2 Perspectives

De nouvelles pistes peuvent être explorées dans la continuité des travaux menés dans cette thèse, ainsi que différentes améliorations pouvant être appliquées dans le but de renforcer

l'efficacité des implémentations des systèmes multi-RPRs. Nous pouvons citer dans ce contexte :

Amélioration de l'algorithme de recherche des RPRs AFLORA

Dans le but d'améliorer la qualité des résultats de notre algorithme de recherche des RPRs dans les systèmes Multi-RPRs, une contrainte importante peut être prise en considération. Cette contrainte est la longueur des signaux d'interconnexion qui relient directement les RPRs avec les modules statiques. Les longueurs des ces signaux doivent être courtes dans le but de réduire te temps de propagation et diminuer le chemin critique. Pour cela, les modules statiques doivent être situés sur le voisinage des RPRs suivant un floorplanning bien étudié. Dans [45], les auteurs proposent une heuristique simple basée sur le mécanisme Nonlinear Integer Programming. Les résultats expérimentaux ont montré l'efficacité de leur algorithme PRFloor pour implémenter des systèmes dynamiquement reconfigurables avec une diversité de configurations allant jusqu'à 130 modules dont 24 sont des RPRs et consommant 85% des ressources FPGA Virtex-6 XC6VLX240T.

Dans ce cadre, une amélioration de notre algorithme AFLORA peut être effectuée pour améliorer la qualité des résultats QoR. Il s'agit de :

- Identifier automatiquement les modules statiques communiquant directement avec chaque RPRs.
- Extraire les connexions directes et leurs temps de propagation.
- Estimer l'allocation et le placement de chacun de ces modules statiques en fonction des temps de propagations des signaux de connexion
- Constituer un bloc élémentaire formé par une RPR et ces modules statiques directs.
- Effectuer le floorplanning automatique de ces blocs à l'aide du mécanisme de réplication.

Intégration de la dynamicité dans les architectures massivement parallèles

L'un des défis pour 2019 identifié dans le rapport d'ITRS[2] est de fournir une capacité d'adaptation plus rapide, de réutilisation et de reconfiguration. Cependant, jusqu'à ce jour, aucuns des projets existants ne proposent un modèle d'exécution parallèle reconfigurable dynamiquement. L'aspect dynamique dans un modèle d'exécution massivement parallèles

[2] http://www.itrs.net/Links/2012ITRS/Home2012.htm

consiste à concevoir un système avec un ensemble d'IPs dynamiques qui sont instanciées et remplacées à l'exécution. Le concepteur peut présélectionner l'ensemble des IPs à mettre en œuvre dans le système selon les besoins de l'application.

Les architectures parallèles de HoMade [92] [93] [94] et SCAC « Synchronous Communication Asynchronous Computation » [95] [96] présentent des plateformes parfaitement adéquates permettant l'application de notre flot de conception dédié pour les systèmes Multi-RPRs. L'architecture parallèle de HoMade est construite à base d'un ensemble de processeurs esclaves gérés par un processeur maître. Les processeurs maître/esclaves sont identiques et implantés sur un réseau sur puce régulier de type Torus. HoMade est un processeur RISC et à pile. Il est construit à base d'une unité de contrôle, une pile de donnée et d'un ensemble d'IPs et peut être instancié sur mesure pour l'application que nous voulons exécuter en mode parallèle. La figure 7.1 illustre la composition globale de ce processeur.

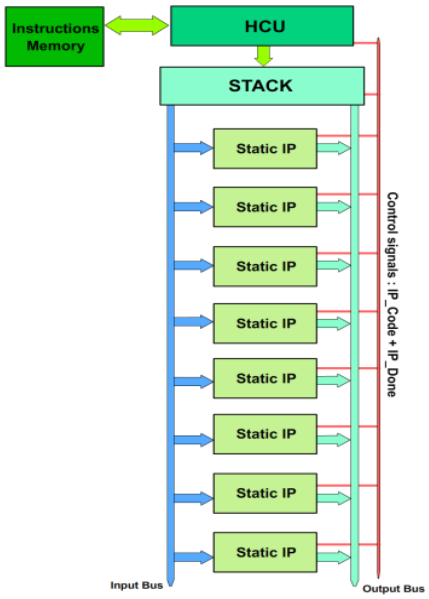


Figure 7.1 : Architecture globale du processeur HoMade

Les IPs instanciés au sein de ce processeur sont statiques et procèdent au maximum 3 entrées et 3 sorties dont chacune est codée sur 32 bits. Ceci représente un aspect remarquable pour permettre l'intégration d'un IP dynamique (voir la Figure 4.15) au sein de chaque processeur esclave. La gestion de la RDPP peut être assurée par un IP dédié intégré dans le processeur HoMade maître disposant de la même interface processeur illustrée dans la Figure 5.2. Le déclanchement de la RDPP est effectuée grâce à l'exécution d'une instruction dédiée.

Cette instruction doit être définie à partir du guide du processeur HoMade et permettant la sollicitation de l'IP de la RDPP (IP long).

Pour intégrer la dynamicité au modèle SCAC, deux points doivent être étudiés : la structure de l'IP dynamique qui permet une simple et rapide commutation d'un IP à un autre et le mécanisme de gestion de la reconfiguration dynamique. Dans l'exécution en mode SCAC, il est possible d'instancier des HW-CEs pour une fonction spécifique dans une partie du programme. Ces HW-CEs effectuent cette fonction puis restent inactifs durant le reste de l'exécution. Pour chaque nouvelle fonction, il faut allouer un nouveau espace pour un nouveau HW-CE ; ce qui représente un gâchis de la surface occupée et une perte de performance (limiter l'extensibilité du système et augmenter la consommation d'énergie).

La modularité de l'architecture SCAC et le principe d'exécution selon plusieurs niveaux facilite la reconfiguration dynamique dans le troisième niveau d'exécution du modèle SCAC. En effet, cette reconfiguration dynamique nécessite l'utilisation des CEs dynamiques avec une interface externe générique. Vu que l'implémentation proposée du modèle SCAC intègre des accélérateurs HW-CEs respectant cette caractéristique et en tirant profit de la reconfiguration dynamique des nouvelles technologies FPGAs, l'ensemble des HW-CEs d'une configuration SCAC peuvent évoluer au cours de l'exécution du programme : certains seront instanciés, d'autres seront désactivés ou encore remplacés.
Pour contrôler la reconfiguration dynamique, il faut définir une instruction qui permet de lancer une reconfiguration dynamique sur un ensemble de CEs actifs selon les caractéristiques technologiques [97] du FPGA. Quand une implémentation de SCAC est générée, nous pouvons effectuer une instanciation dynamique des CEs lors de l'exécution de l'algorithme, en utilisant la reconfiguration dynamique partielle (Dynamic Partial Reconfiguration, DPR) sur les régions reconfigurables de façon prédéfinie

Automatisation du floorplanning dans les systèmes Multi-FPGAs

En se référant à l'architecture multi-FPGAs présentée dans [32], chaque FPGA contient un ensemble de RPRs. Ces RPRs occupent les mêmes emplacements au sein de chaque FPGA. En effet, les FPGAs utilisées doivent impérativement être identiques pour permettre la reconfiguration parallèle des RPRs de même emplacement ayant une même fonction (bitstream).

Vu que les FPGAs utilisés sont identiques, alors il suffit d'exécuter l'algorithme AFLORA une fois pour permettre le floorplanning identique dans chaque FPGA. Ceci représente un

point fort de notre flot de conception ADForME. De plus, si nous considérons que les RPRs se trouvant dans le même composant FPGA exécutent la même fonction (bitstream) à la fois, alors l'application du flot d'automatisation de la relocalisation de bitstream suite à l'application de celui d'ADForME permet le broadcast d'un bitstream unique vers l'ensemble des RPRs. En effet, ceci revient à en déduire que ce bitstream peut être diffusé en même temps dans chacune des FPGAs si nos deux flots y sont appliqués.

L'application de nos deux flots de conception sur ce type d'architecture permet clairement la réduction de la complexité de conception en termes de temps de développement ce qui permet l'amélioration de la productivité des concepteurs. Ceci grâce à la flexibilité et la réutilisabilité des composants IPs intégrés dans les architectures à Multi-RPRs que nos deux flots garantissent.

BIBLIOGRAPHIE

[1] https://www.xilinx.com/products/design-tools/microblaze.html

[2] AXI Reference Guide - Xilinx: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[3] https://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[4] ICAP User Guide Xilinx: https://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ug702.pdf

[5] https://openpowerfoundation.org/

[6] K. Paulsson, M. Hübner, and J. Becker. On-line optimization of fpga power-dissipation by exploiting run-time adaption of communication primitives. In Proceedings of the 19th annual symposium on Integrated circuits and systems design, SBCCI '06, pages 173–178, New York, NY, USA, 2006. ACM.

[7] K. Paulsson, M. Hübner, and J. Becker. Dynamic power optimization by exploiting self-reconfiguration in xilinx spartan 3-based systems. Microprocessors and Microsystems, 33(1): 46–52, Feb. 2009.

[8] Altera. Increasing design functionality with partial and dynamic reconfiguration in 28-nm fpgas. Technical Report WP-01137-1.0, 2010.

[9] P. R. U. Guide. Ug702 (v14.3). Xilinx Inc., October 16, 2012.

[10] Xilinx. Two Flows for Partial Reconfiguration : Module Based or Difference Based, 2003.

[11] Xilinx. Two Flows for Partial Reconfiguration : Module Based or Difference Based, 2004.

[12] Xilinx. Early Access Partial Reconfigurable Flow, 2010.

[13] Xilinx. Synthesis and simulation design guide. Technical Report UG626 (v 11.4), 2009.

[14] E. Horta and J. W. Lockwood. Parbit : A tool to transform bifitles to implement partial reconguration of field programmable gate arrays (fpgas). Technical report wucs-01-13, July 2001.

[15] V.-. F. Configuration. Ug360 (v3.5). Xilinx Inc., September 11, 2012.

[16] B. Blodget, S. McMillan, and P. Lysaght. A lightweight approach for embedded reconfiguration of fpgas. In Design, Automation & Test in Europe, DATE'03, 2003.

[17] H. Tan, R. F. DeMara, A. J. Thakkar, A. Ejnioui, and J. Sattler. Complexity and performance evaluation of two partial reconfiguration interfaces on fpgas : A case study. In ERSA'06, pages 253–256, 2006.

[18] A. Cuoccio, P. R. Grassi, V. Rana, M. D. Santambrogio, and D. Sciuto. A generation flow for self-reconfiguration controllers customization. IEEE International Workshop on Electronic Design, Test and Applications, 0 :279–284, 2008.

[19] Xilinx. Logicore ip axi hwicap (v2.02.a) product specification. Technical Report DS817, 2012.

[20] C. Claus, F. H. Müller, J. Zeppenfeld, and W. Stechele. A new framework to accelerate virtex-ii pro dynamic partial self-reconfiguration. In Parallel and Distributed Processing Symposium, IPDPS'07, pages 1–7, 2007.

[21] Xilinx. Virtex-5 fpga configuration user guide. Technical Report UG191 (v3.11), 2012.

[22] M. Hübner, C. Schuck, and J. Becker. Elementary block based 2Dimensional dynamic and partial reconfiguration for virtex-ii fpgas. In Proceedings of the 20th international conference on Parallel and distributed processing, IPDPS'06, pages 196–196, 2006.

[23] M. Hübner, C. Schuck, M. Kuhnle, and J. Becker. New 2Dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits. In Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, ISVLSI '06, pages 97–, 2006.

[24] Xilinx. Opb hwicap product specification. Technical Report DS 280 (v1.3), 2004.

[25] …, PowerPC 405 Processor Block Reference Guide, 2004.

[26] Xilinx corporation, "Reconfiguring User Logic Using Custom ICAP Processor and Monitoring ICAP Signals Using ChipScope Core Lab", in http://forums.xilinx.com/xlnx/attachments/xlnx/EDK/23008/1/lab04.pdf

[27] Xilinx corporation, "Driving ICAP Resource", in http://home.mit.bme.hu/~feher/Reconf_Comp/10_Driving_ICAP.pdf

[28] F.Duhem, F.Muller, and P.Lorenzini, "FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA, "Reconfigurable Computing: Architectures, Tools and Applications, pp. 253-260, 2011.

[29] Robin Bonamy, Hung-Manh Pham, Sebastien Pillement, and Daniel Chillet, "UPaRC—Ultra-fast power-aware reconfiguration controller", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012

[30] Hung-Manh Pham, Van-Cuong Nguyen, Trong-Tuan Nguyen,"DDR2/DDR3-based ultra-rapid reconfiguration controller", Communications and Electronics (ICCE), 2012 Fourth International Conference on DOI: 10.1109/CCE.2012.6315949 -2012.

[31] Xilinx, Inc, \Logicore ip multi-port memory controller (mpmc)(ds643 v6.03.a),"March 2011.

[32] Venkatasubramanian Viswanathan; Rabie Ben Atitallah; Jean-Luc Dekeyser, "Massively Parallel Dynamically Reconfigurable Multi-FPGA Computing System", 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines Year: 2015

[33] A. Cevrero, P. Athanasopoulos, H. Parandeh-Afshar, P. Brisk, Y. Lebebici, P. Ienne, and M. Skerlj. 3d configuration caching for 2d fpgas. In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '09, pages 286–286, New York, NY, USA, 2009. ACM.

[34] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong. Performance benefits of monolithically stacked 3d-fpga. In Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA '06, pages 113–122, New York, NY, USA, 2006. ACM.

[35] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in Proc. Intl. Conf. on Reconfigurable Computing: architectures, tools and applications (ARC), 2012, pp. 13–25.

[36] C. Bolchini, A. Miele, and C. Sandionigi, "Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems," in Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL), 2011, pp. 532–538.

[37] Marco Rabozzi; Antonio Miele; Marco D. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGAs via Feasible Placements Detection", 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Year: 2015

[38] M. Rabozzi, J. Lillis, and M. D. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming," in Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2014, pp. 186–193.

[39] M. Rabozzi; R. Cattaneo; T. Becker; W. Luk; M. D. Santambrogio, "Relocation-Aware Floorplanning for Partially-Reconfigurable FPGA-Based Systems" in Proc. Intl. Conf. on Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015, pp. 97–104.

[40] Kevin E. Murray; Vaughn Betz, "HETRIS: Adaptive floorplanning for heterogeneous FPGAs", 2015 International Conference on Field Programmable Technology (FPT),Year: 2015

[41] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in FPL, 1997, pp. 213–222.

[42] L. Singhal and E. Bozorgzadeh, "Multi-layer Floorplanning on a Sequence of Reconfigurable Designs," in FPL, 2006, pp. 1–8.

[43] P. Banerjee et al., "Floorplanning for Partial Reconfiguration in FPGAs," in Inter. Conf. on VLSI Design, 2009, pp. 125–130.

[44] K. Vipin and S. Fahmy, "Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration," in ARC, 2012, pp. 13–25.

[45] Tuan D. A. Nguyenand Akash Kumar, "PRFloor: An Automatic Floorplanner for Partially Reconfigurable FPGA Systems", FPGA'16, February 21-23, 2016, Monterey, CA, USA

[46] D. Li and X. Sun. Nonlinear integer programming, volume 84. Springer Science & Business Media, 2006.

[47] …, 7 Series FPGAs Configuration User Guide (UG470) – Xilinx: https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[48] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert, "REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems,"in Proc. IPDPS Workshops, 2005.

[49] F. Ferrandi, M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto, "Dynamic reconfiguration: Core relocation via partial bitstreams filtering with minimal overhead," in Proc. Intl. Symp. on System-on-Chip (SOC), 2006, pp. 1–4.

[50] S. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, D. Sciuto, and P. Spoletini, "Internal and external bitstream relocation for partial dynamic reconfiguration," IEEE Trans. on VLSI Systems, vol. 17, no. 11, pp. 1650–1654, 2009.

[51] https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf , Xilinx Partial Reconfiguration User Guide (UG702)

[52] https://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/fpga_editor.htm#html/fe_b_overview.htm : (FPGA Editor Overview)

[53] H. Kalte and M. Porrmann, "REPLICA2Pro: task relocation by bitstream manipulation in virtex-II/Pro FPGAs," in Proc. Conf. on Computing Frontiers, 2006, pp. 403–412.

[54] Ichinomiya Y., Amagasaki M., Iida M., Kuga M., Sueyoshi T. (2012) A Bitstream Relocation Technique to Improve Flexibility of Partial Reconfiguration. In: Xiang Y., Stojmenovic I., Apduhan B.O., Wang G., Nakano K., Zomaya A. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2012. Lecture Notes in Computer Science, vol 7439. Springer, Berlin, Heidelberg

[55] Tomáš Drahonovský, Martin Rozkovec and Ondrej Novák, "A highly flexible reconfigurable system on a Xilinx FPGA", 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), Year: 2014,Pages: 1 - 6, DOI: 10.1109/ReConFig.2014.7032531

[56] Tomáš Drahonovský, Martin Rozkovec and Ondrej Novák, "Relocation of reconfigurable modules on Xilinx FPGA", 2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Year: 2013, Pages: 175 - 180, DOI: 10.1109/DDECS.2013.6549812

[57] Dynamic Reconfiguration of Xilinx FPGAs: https://www.xilinx.com/univ/FPL06_Invited_Presentation_PLysaght.pdf

[58] Dirk Koch, Christian Beckhoff, and Jim Torresen, "Zero Logic Overhead Integration of Partially Reconfigurable Modules", SBCCI 2010.

[59] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon. Image processing on a SIMD/SPMD architecture : OPSILA. In 9th International conference Patern Recognition, pages 430 – 433, Nov. 1988.

[60] https://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_c_stw_generating_software_libraries.htm

[61] http://lslwww.epfl.ch/pages/teaching/cours_lsl/lmi/xc5200.pdf

[62] Jean-Pierre Deschamps, Gery J.A. Bioul, Gustavo D. Sutter - 2006 - Technology & Engineering FPGA, ASIC and Embedded Systems Jean-Pierre Deschamps, Gery J.A. Bioul, Gustavo ... RoutingAdjacent to each CLB stands a general routing matrix (GRM).

[63] https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/03/10-nm-technology-fact-sheet.pdf

[64] Xilinx Vivado Design Suite Tcl Command Reference Guide (UG835): https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug835-vivado-tcl-commands.pdf

[65] Xilinx Vivado Design Suite User Guide: Using Tcl Scripting (UG894): https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug894-vivado-tcl-scripting.pdf

[66] http://www.businesswire.com/news/home/20150528005395/en/SRC-Computers-Launches-Saturn-1-Server-Reconfigurable

[67] Stratix IV GX FPGA Development Kit User Guide : https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_sivgx_fpga_dev_kit.pdf

[68] http://www.nvidia.fr/page/home.html

[69] https://www.mellanox.com/

[70] https://fr.wikipedia.org/wiki/IBM_POWER

[71] https://www-304.ibm.com/webapp/set2/sas/f/capi/home.html

[72] https://www.ibm.com/blogs/systems/ibm-nvidia-present-nvlink-server-youve-waiting/

[73] http://www-03.ibm.com/systems/fr/power/

[74] https://www.ece.cmu.edu/~calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf

[75] https://fr.wikipedia.org/wiki/Recuit_simulé

[76] Marco Rabozzi Gianluca Carlo Durelli Antonio Miele John Lillis Marco Domenico Santambrogio, "Floorplanning Automation for Partial-Reconfigurable FPGAs via Feasible Placements Generation", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Year: 2017, Volume: 251.

[77] L. Cheng and M. D. F. Wong, “Floorplan design for multimillion gate FPGAs,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 25, no. 12, pp. 2795–2805, Dec. 2006.

[78] Xilinx Inc. PlanAhead Design and Analysis Tool, accessed on May 17, 2016. [Online]. Available: http://www.xilinx.com/products/design-tools/planahead.html

[79] C. Beckhoff, D. Koch, and J. Torreson, “Automatic floorplanning and interface synthesis of island style reconfigurable systems with GoAhead,” in Proc. 26th Int. Conf. Archit. Comput. Syst. (ARCS), 2013, pp. 303–316.

[80] S. Yousuf and A. Gordon-Ross, “DAPR: Design automation for partially reconfigurable FPGAs,” in Proc. Int. Conf. Eng. Reconfigurable Syst.Algorithms (ERSA), 2010, pp. 1–7.

[81] http://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/whnjs.htm

[82] https://fr.wikipedia.org/wiki/X86

[83] https://pdfs.semanticscholar.org/4479/4724f7e65db747da6f68876ecb7f4a1d7cc5.pdf

[84] https://ark.intel.com/fr/products/39721/Intel-Xeon-Processor-W3565-8M-Cache-3_20-GHz-4_80-GTs-Intel-QPI

[85] https://www.xilinx.com/support/documentation/user_guides/ug369.pdf

[86] https://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf

[87] https://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf

[88] https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf

[89] https://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf
[90] http://www.pressreader.com/france/electronique-s/20170628/281535111002924
[91] https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_r_source_types.htm
[92] https://sites.google.com/site/aeom1jld/courses/spmd-sur-homade
[93] https://sites.google.com/site/homadeguide/home
[94] Jean Perier, Wissem Chouchene, Jean-Luc Dekeyser. Circuit Merging versus Dynamic Partial Reconfiguration -The HoMade Implementation. i-manager's Journal on Embedded Systems(JES), 2016.
[95] Hana Krichene, Mouna Baklouti, Philippe Marquet, Jean-Luc Dekeyser, Mohamed Abid: SCAC-Net: Reconfigurable Interconnection Network in SCAC Massively Parallel SoC. PDP2016: 759-762.
[96] Hana Krichene, Mouna Baklouti, Mohamed Abid, Philippe Marquet, Jean-Luc Dekeyser. G-MPSoC: Generic Massively Parallel Architecture on FPGA. WSEAS Transactions on Circuits and Systems, World Scientific and Engineering Academy and Society, 2015, 14
[97] R. B. Atitallah. Dynamic reconfiguration and low power design: towards self-adaptive massively parallel embedded systems. Habilitation à diriger des recherches en informatique, Universté de Valenciennes et du Hainaut-Cambrésis, 2014.
[98] https://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf
[99] https://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_d_bmm.htm
[100] https://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_d_bitinit.htm
[101] https://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf

Glossaires

A

ADForMe:	Automatic DPPR Flow For Multi-RPRs Architecture
AFLORA:	Automatic Floorplanning For Multi-RPRs Architectures
AR :	Ressources Disponibles
Adj_CR:	Région d'Horloge Adjacentes

B

BLACKBOX :	Boite vide
BLANK :	Vide

C

C :	Nombre de colonnes du FPGA
CR :	Région d'Horloge
CR_Height	Hauteur de la Région d'Horloge

D

DAA :	Différence d'Adresses Absolues
DR :	Ressources Dynamiques

F

FAR:	Frame Address Register
FPGA :	Field Programmable Gate Arrays
Frame_B:	Frame BRAM
Frame_C:	Frame CLB
Frame_D:	Frame DSP

I

ICAP :	Internal Configuration Access Port
IP:	Intellectual Propriety
IP_{Dyn} :	IP Dynamique
IPIF :	Interface Statique
IPreconfig :	IP de Reconfiguration

L

LUT :	Look Up Table

M

MIG :	Memory Interface Generator
MPR_C :	Module Partiellement Reconfigurable Commun

N

nbCR:	nombre de Régions d'Horloges
nbIT :	nombre d'Itérations
NET_D:	Dimension du réseau de RPRs

NCD : Native Circuit Description
NTO : Nombre Total d'Opérations

P

PBlock : Bloc Programmable
P&R : Place and Route
PCF : Physical Constraint File

R

RPR_C : Région Partiellement Reconfigurable Commune
RPR_Cref : Région Partiellement Reconfigurable Commune de référence
RDP : Reconfiguration Dynamique Partielle
RDPP : Reconfiguration Dynamique Partielle Parallèle

S

SR : Ressources Statiques

U

UC : Unité de Control
UCF : User Constraint File
Unsed_Range : Région non utilisable par le concepteur

V

$\mathbf{V_{trans}}$ **:** Vecteur de translation

X

X_max : Largeur du FPGA

Z

Z : Taux de répartition équitable des RPR_Cs dans chaque Adj_CR

Vers une Reconfiguration Dynamique Partielle Parallèle Par Prise En Compte De La Régularité Des Architectures FPGA-Xilinx

Résumé : Ce travail propose deux flots de conception complémentaires permettant le broadcast d'un bitstream partiel vers un ensemble de Régions Partiellement Reconfigurables (RPRs) identiques. Ces deux flots de conception sont applicables avec les FPGAs – Xilinx. Le premier appelé ADForMe (Automatic DPPR Flow For Multi-RPRs Architecture) permet l'automatisation du flot traditionnel de la RDP de Xilinx grâce à l'automatisation de la phase de floorplanning. Ce floorplanning est assuré par l'algorithme AFLORA (Automatic Floorplanning For Multi-RPRs Architectures) que nous avons conçu qui permet l'allocation identique de ces RPRs en termes de forme géométrique en tenant compte des paramètres technologiques du FPGA et des paramètres architecturaux de la conception dans le but de permettre la relocalisation de bitstream. Le deuxième flot proposé vise à favoriser la technique de relocalisation 1D et 2D afin de permettre le broadcast d'un bitstream partiel (fonctionnalité) vers un ensemble de RPRs pour une configuration du système. Ce flot permet donc l'optimisation de la taille de la mémoire de bitstream. Nous avons également proposé une architecture matérielle adéquate capable d'effectuer ce broadcast. Les résultats expérimentaux ont été effectués sur les FPGAs-Xilinx récents et ont prouvé la rapidité d'exécution de notre algorithme AFLORA ainsi que l'efficacité des résultats obtenus suite à l'application du flot d'automatisation de la relocalisation de bitstream. Ces deux flots permettent d'assurer la flexibilité et la réutilisabilité des composants IPs intégrés dans les architectures à Multi-RPRs afin de réduire la complexité en termes de temps de conception et d'améliorer productivité des concepteurs.

Mots-clés : Reconfiguration dynamique partielle parallèle, Floorplanning, Relocalisation de bitstream, Broadcast de bitstream, FPGA-Xilinx, ADForME, AFLORA.

Towards a Parallel Partial Dynamic Reconfiguration by Taking into Account the Regularity of FPGA-Xilinx Architectures

Abstruct : This work proposes two complementary design flows allowing the broadcast of a partial bitstream to a set of identical Partially Reconfigurable Regions (PRRs). These two design flows are applicable with FPGAs - Xilinx. The first one called ADForMe (Automatic DPPR Flow For Multi-RPRs Architecture) allows the automation of the traditional flow of Xilinx RDP through the automation of the floorplanning phase. This floorplanning is carried out by the AFLORA (Automatic Floorplanning For Multi-RPRs Architectures) algorithm which we have designed that allows the same allocation of these RPRs in terms of geometric shape taking into account the technological parameters of the FPGA and the architectural parameters of the design in order to allow the relocation of bitstream. The second proposed flow aims to promote the 1D and 2D relocation technique in order to allow the broadcast of a partial bitstream (functionality) to a set of RPRs for a system configuration. Therefore, this flow allows optimizing the size of the bitstream memory. We have also proposed suitable hardware architecture capable of performing this broadcast. The experimental results have been performed on the recent Xilinx FPGAs and have proved the speed of execution of our AFLORA algorithm as well as the efficiency of the results obtained by the application of the automation of the bitstream relocation technique flow. These two flows allow flexibility and reusability of IP components embedded in Multi-RPRs architectures to reduce complexity in design time and improve design productivity.

Keywords: Dynamic parallel partial reconfiguration, Floorplanning, AFLORA, Bitstream relocation, Bitstream Broadcasting, FPGA-Xilinx, ADForME.