TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CSV	*Comma separated values*
MVC	Modèle-Vue-Contrôleur
PdC	Point de Convergence
TC	Taux de Couverture
TE	Temps d'Exécution

INTRODUCTION

Le transistor a été inventé en 1947 par William Bradford Shockley, John Bardeen et Walter Houser Brattain. Pour cette découverte, ils ont obtenu le prix Nobel de physique de 1956. Ce semiconducteur est venu remplacer les tubes électroniques qui sont plus gros, plus énergivores et plus fragiles que le transistor. Avec le transistor, les ingénieurs des années 1960 ont pu créer des circuits de plus en plus complexes avec un nombre grandissant de composants. À cette époque, les branchements entre les composants étaient faits manuellement. Bientôt, la complexité des circuits a rendu leur fabrication très ardue à cause du nombre élevé de branchements requis. Ce phénomène se nomme la tyrannie des nombres. Pour permettre l'augmentation de la complexité des circuits, il y avait un réel besoin d'intégration et de miniaturisation. C'est en 1959 que Jack Kilby et, séparément, Robert Noyce ont créé les premiers circuits intégrés. La particularité de ces circuits est que toutes les pièces sont faites d'un même bloc de matière. Les composants sont fabriqués directement sur le bloc et sont ensuite branchés en utilisant des procédés de métallisation.

La création de ces premiers circuits intégrés a lancé une révolution. Les circuits contiennent de plus en plus de fonctions tout en utilisant des transistors de plus en plus petits. De telle sorte que Gordon E. Moore a émis, en 1965, la première version de la loi connue sous le nom de Loi de Moore. En 1965, Moore a dit (Moore (1998)) que le nombre de composants sur un circuit intégré abordable doublerait à chaque année. Suite à cette première hypothèse, Gordon E. Moore s'est corrigé en 1975 (Moore (1975)) pour émettre la version la plus connue de la loi qui dit que le nombre de composants va doubler tous les deux ans.

En observant les prévisions de la taille du canal du transistor du International Technology Roadmap for Semiconductors (ITRS (2013, tableau ORT1)), il est facile de voir que la densité de composant augmente d'un gain de 1,5 tous les deux ans. Cela correspond à la loi de Moore, adaptée par David House, qui stipule que la performance des circuits intégrés abordable va doubler tous les 18 mois. La croissance exponentielle de l'intégration permet aux circuits intégrés d'effectuer plus de fonctions en utilisant plus de transistors sur une surface qui ne croît pas exponentiellement. En effet, l'évolution des processeurs d'Intel (Intel Corporation (2012))

illustre bien cette croissance du nombre de transistors et de fonctions intégrées au sein d'une même puce. Par exemple, entre la première génération (2008) et la deuxième génération (2010) des processeurs *Core* d'Intel, il y a eu une augmentation du nombre de transistors d'un facteur de trois en plus d'une croissance d'un facteur de 1,6 de la fréquence d'opération. Cette intégration peut entraîner une croissance en terme de puissance dynamique. Normalement, plus il y a de composants qui commutent, plus la puissance dynamique sera grande.

Les processeurs d'usage personnel sont principalement utilisés dans les téléphones mobiles et autres périphériques mobiles (Cisco Systems (2013)). Ces appareils, n'étant pas alimentés par le secteur, sont alimentés par une pile. Dans le but de créer des périphériques et appareils qui offrent de meilleures performances tout en consommant moins d'énergie pour épargner la pile, les méthodes de conception de circuits intégrés à faible puissance sont de plus en plus pertinentes. Une de ces méthodes de conception implique un changement de paradigme qui permet de créer des circuits qui offrent, notamment, la même performance, mais qui consomment moins d'énergie. En effet, en n'utilisant pas d'horloge globale, la conception de circuits asynchrones devient une méthode permettant de réduire l'énergie requise par le circuit (Beerel *et al.* (2010)).

La méthode de conception traditionnelle synchrone implique plusieurs étapes de synthèse et de vérification (Weste et Harris (2010)). En effet, ces étapes sont toutes basées sur la gestion de la complexité par l'abstraction. La conception du circuit débute par la conceptualisation des requis. Ensuite, ces requis sont traduits en une description fonctionnelle. Cette description fonctionnelle est ensuite synthétisée en portes logiques et implantée dans une puce au moyen de la photolithographie. Pour garantir que le circuit implémente les requis établis, il y a un besoin de vérification fonctionnelle et d'analyse temporelle. L'analyse temporelle est réalisée par un outil qui permet de garantir l'intégrité des données stockées dans toutes les bascules du circuit. Une fois que le circuit est validé et implémenté dans une puce, il faut démontrer que le procédé de fabrication n'a pas introduit d'erreurs dans la puce. Le processus de test permet de détecter, en utilisant un équipement de test et des patrons, les puces qui fonctionnent correctement et les puces qui ne fonctionnent pas.

Étant donné que la majorité des circuits créés par photolithographie et synthèse sont des circuits synchrones, il y a un manque d'outils pour exploiter le paradigme asynchrone. Pour mieux utiliser et développer ce paradigme de conception, il est nécessaire de créer des outils permettant d'adapter les logiciels standards au paradigme asynchrone. Ceci a pour but d'inciter les concepteurs de puces synchrones à se tourner vers la conception asynchrone sans imposer de changements trop importants. L'outil développé dans ce mémoire tente de répondre à cette problématique en généralisant le concept d'analyse temporelle traditionnel tout en facilitant la création de patrons de test. Pour ce faire, les concepts de points de convergence et de trois structures ou configurations typiques des circuits asynchrones d'Octasic ont été élaborés.

Pour illustrer le développement de l'outil, le mémoire est divisé en quatre chapitres. Le premier sert d'introduction aux concepts de base utilisés pour l'élaboration de l'outil. Ces concepts sont la théorie des graphes, la méthode de conception synchrone et le test de circuits intégrés. De plus, le chapitre offre une revue de la littérature illustrant l'état de l'art de l'analyse temporelle et de la conception asynchrone. Ensuite, le deuxième chapitre introduit le paradigme de conception développé par Octasic. Cette méthode est basée sur la synchronisation des unités de traitement au moyen d'une nouvelle application du concept de jeton (Awad *et al.* (2013)). Une analyse des deux circuits typiques conçus par Octasic a permis d'élaborer trois configurations qui sont utilisées par l'outil pour généraliser le concept d'analyse temporelle. Le troisième chapitre présente le fonctionnement de l'outil, nommé qmi, en plus d'expliquer le concept du point de convergence et son application aux configurations de circuits typiques. Finalement, le dernier chapitre contient les résultats des analyses effectuées par qmi sur les deux circuits de tests.

Suite à l'analyse des résultats, le mémoire se termine par un résumé des points importants. La conclusion fait, notamment, état des limitations des résultats ainsi que des recommandations. Ce mémoire présente la principale contribution au domaine de la conception de circuits numériques. Cette contribution est l'outil qmi qui permet d'adapter les outils de conception traditionnels au paradigme asynchrone développé par Octasic.

CHAPITRE 1

CONCEPTS FONDAMENTAUX ET REVUE DE LA LITTÉRATURE

Le chapitre présent fait un survol des notions fondamentales utilisées lors de l'élaboration de l'outil. En effet, la théorie des graphes est présentée avant la méthode de conception synchrone traditionnelle. Ensuite, un survol des méthodes de conception asynchrone et leurs avantages sont exposés en plus d'une revue des méthodes de vérification temporelle des circuits. Finalement, le chapitre se conclut avec les concepts fondamentaux de la vérification des circuits intégrés.

1.1 Théorie des graphes

Les graphes sont un outil mathématique permettant de décrire et analyser les liens entre des éléments. Ces éléments, selon le contexte, peuvent avoir des significations très variées. Les sous-sections suivantes donnent un aperçu des notions utilisées lors du développement de l'outil.

1.1.1 Concepts de base

Comme l'illustre l'équation 1.1, un graphe G est défini par deux ensembles, soit un ensemble de nœuds E et un ensemble de sommets V Diestel (2005, page 2).

$$G = (V, E) \tag{1.1}$$

La Figure 1.1 illustre un graphe simple. Les ensembles V et E se trouvent aux équations 1.2 et 1.3 respectivement.

$$V = \{1, 2, \ldots, 7\} \tag{1.2}$$

$$E = \{\{1,5\},\{1,6\},\{1,7\},\{2,4\},\{2,3\},\{3,4\},\{4,7\},\{4,5\},\{7,1\}\} \tag{1.3}$$

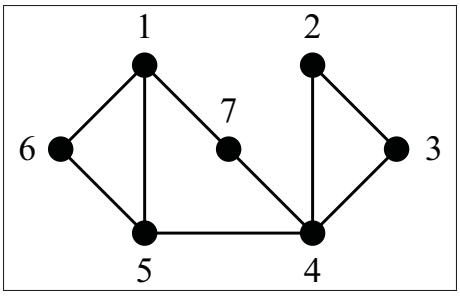

Figure 1.1 Graphe simple

Avec les définitions des ensembles représentant un graphe, il est possible de classifier et analyser ce graphe. En effet, il existe plusieurs types. Notamment, un graphe peut être cyclique, dirigé ou trivial. L'ordre d'un graphe, noté $|G|$, est le nombre de sommets présents dans le graphe. Un graphe où $G = (0,0)$ est trivial, ainsi que celui où l'ordre est égal à 0 ou 1, Diestel (2005, page 2). De plus, si les liens entre les sommets ont un sens, par exemple dans la Figure 1.2, l'ensemble E doit être interprété différemment. Effectivement, lorsque le graphe est dirigé, un lien $\{1,2\}$ n'implique pas un lien $\{2,1\}$. Contrairement à l'interprétation de l'ensemble E d'un graphe non-dirigé. Si l'élaboration de tous les chemins à partir d'un sommet vers tous les sommets permet de visiter le sommet de départ, le graphe est de type cyclique. Par exemple, le graphe de la Figure 1.2 est cyclique, car le chemin $1 \to 6 \to 5 \to 1$ est une boucle.

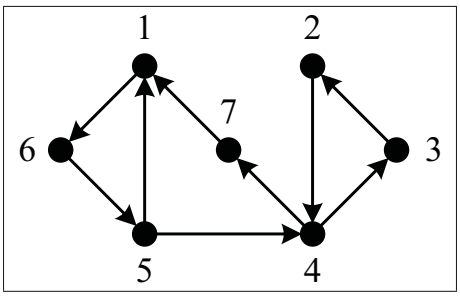

Figure 1.2 Graphe dirigé

1.1.2 Représentations logicielles des graphes

Comme l'indique Cormen *et al.* (2009, page 589), il existe, minimalement, deux structures de données pour représenter un graphe au sein d'une application. Soit la liste d'adjacence et la matrice d'adjacence. De plus, il peut y avoir des propriétés attachées aux sommets et aux nœuds. Ces propriétés sont qualitatives et sont nécessairement très variées.

La liste d'adjacence, Cormen *et al.* (2009, page 590), est en fait un tableau de liste chaînée. L'index dans le tableau représente le numéro identificateur du sommet et la liste chaînée contient les numéros identificateurs de tous les sommets vers lesquels il y a un lien. La direction du lien est encodée directement dans la liste chaînée. En effet, pour avoir un lien réciproque entre deux sommets, il doit y avoir le numéro identificateur d'un sommet dans la liste d'adjacence de l'autre et vice versa. L'équation 1.4 contient les listes d'adjacence du graphe représenté à la Figure 1.1.

$$\begin{aligned}
&\mathbf{1} : 5 \rightarrow 6 \rightarrow 7 \\
&\mathbf{2} : 3 \rightarrow 4 \\
&\mathbf{3} : 2 \rightarrow 4 \\
&\mathbf{4} : 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \\
&\mathbf{5} : 1 \rightarrow 4 \rightarrow 6 \\
&\mathbf{6} : 1 \rightarrow 5 \\
&\mathbf{7} : 1 \rightarrow 4
\end{aligned} \tag{1.4}$$

Une autre structure de données pour représenter un graphe est la matrice d'adjacence, Cormen *et al.* (2009, page 591). La matrice d'adjacence est carrée avec $n = |G|$. Chaque élément a_{ij} de la matrice suit la règle de l'équation 1.5.

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases} \tag{1.5}$$

La matrice d'adjacence de la Figure 1.1 est représentée dans l'équation 1.6.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{1.6}$$

1.1.3 Cas d'utilisation

Les graphes peuvent être utilisés pour représenter des données en plus d'être utiles pour résoudre des problèmes. En effet, il n'est pas étrange de représenter des structures de données comme les listes chaînées et les arbres de recherche en utilisant des graphes Cormen *et al.* (2009, section III). De plus, le design de circuit séquentiel nécessite souvent l'utilisation de machines à états finis. Ces machines peuvent être illustrées en utilisant un graphe où chaque sommet est un état et chaque nœud est une transition d'un état à un autre. Une propriété qualitative des nœuds serait la condition à respecter pour activer la transition. Finalement, les circuits numériques peuvent se représenter naturellement sous forme de graphe. Effectivement, l'outil développé représente les circuits comme étant un graphe où chaque sommet est une porte logique et que chaque nœud est un lien entre une entrée d'une porte logique et la sortie d'une autre porte logique.

1.2 Méthode de conception traditionnelle

Cette sous-section présente la méthodologie de design traditionnellement utilisée dans la conception de circuits numériques. Il est pertinent de revoir cette méthodologie, car l'outil développé est basé sur une évolution de cette méthode de conception.

1.2.1 Domaines d'abstraction

Pour mieux gérer l'élaboration d'un système numérique complexe, un concepteur utilise plusieurs domaines d'abstraction. En effet, comme à la Figure 1.3, il existe au moins trois domaines d'abstraction. Plus le domaine est bas, plus il est près de la réalisation matérielle. Ces domaines s'appliquent bien au design de circuits numériques complexes, Weste et Harris (2010, page 615). En effet, le domaine comportemental explique comment le circuit doit se comporter. Il décrit ainsi quelles fonctions le circuit doit réaliser. Le domaine structurel explicite comment les unités de traitement des données et de contrôle sont réalisées en utilisant des éléments de base. Ces éléments de base peuvent faire partie d'une bibliothèque de portes standard. Les portes peuvent être, par exemple, des fonctions logiques comme le ET et le OU ainsi que des éléments mémoire comme une bascule D ou un verrou. Le domaine d'abstraction le plus bas, le domaine physique, décrit comment doivent être placées et branchées les cellules standard.

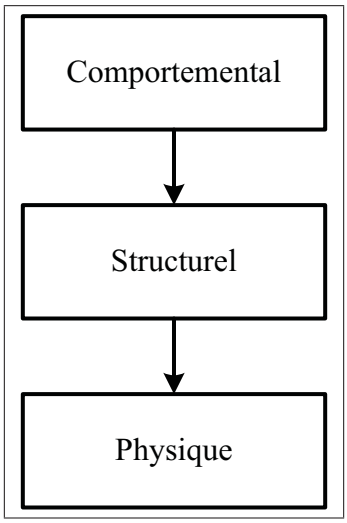

Figure 1.3 Domaines d'abstraction du design numérique

1.2.2 Design structurel

Une approche fréquemment utilisée dans le design traditionnel est le design structurel. Le but de cette approche est de faciliter la gestion de la complexité. Ceci est de plus en plus fréquent,

car la complexité et le niveau d'intégration des circuits numériques ne cessent d'augmenter. Le design structurel implique quatre avantages. Soit la hiérarchisation, la régularité, la modularité et la localité.

La hiérarchisation est un concept utilisé pour mettre à l'œuvre la devise *diviser pour régner*, Weste et Harris (2010, page 620). En effet, les concepteurs d'un circuit séparent le système en blocs de plus en plus petits tant que les blocs ne sont pas d'une complexité facilement gérable.

Cette hiérarchisation entraîne une autre propriété du design structurel, celle de la régularité, Weste et Harris (2010, page 623). Effectivement, en séparant le système en plus petits blocs, il est normal que certains blocs aient la même fonction. Ceci fait en sorte qu'un même module peut être utilisé plusieurs fois dans le système ainsi réduisant le nombre total de blocs à développer. Naturellement, un bloc utilisé plusieurs fois au sein d'un même design peut être utilisé dans un autre design. De plus, la taille d'un tel bloc générique permet de facilement valider son comportement.

La hiérarchisation et la modularité impliquent à leur tour les concepts de localité et de modularité, Weste et Harris (2010, page 626). Un bloc est dit modulaire si son interface et ses interactions avec les autres modules sont très bien définies. Cette particularité rend aisée la vérification fonctionnelle du module. Donc, la détection de disparités entre la description comportementale et structurelle sera plus facile. La localité signifie qu'un bloc peut utiliser un autre bloc sans se soucier de la façon dont l'autre module effectue sa tâche. Les modules deviennent donc des boîtes noires au niveau de leur utilisation et seulement la description des interactions est importante.

1.2.3 Méthodologie de design de circuits numériques

La méthodologie traditionnelle utilisée pour mettre en place le design structurel est illustrée à la Figure 1.4.

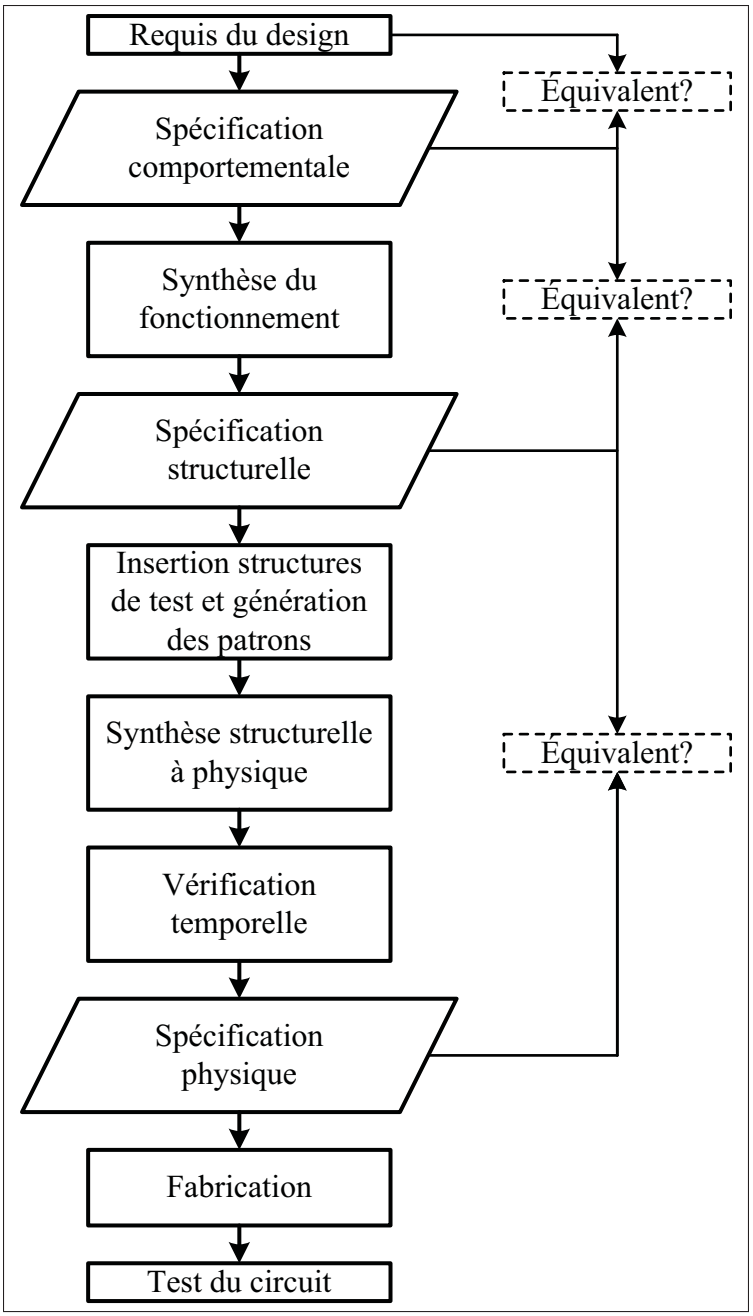

Figure 1.4 Flot de développement traditionnel. Avec le traitement (boîtes) et les résultats (parallélogrammes) Adaptée de Weste et Harris (2010, page 637)

Cette méthodologie s'applique une fois que le système à réaliser a été séparé en modules selon l'approche de design structurel. Le concepteur suit la méthodologie pour chaque module ainsi que pour l'intégration des modules dans le système complet. Comme l'indique la première

étape, tout débute par la spécification des requis. Cette étape cruciale explicite clairement ce que le module doit accomplir comme tâche. En utilisant les requis, le concepteur produit la spécification comportementale du module en utilisant soit un langage de description matériel ou des moyens plus classiques comme les tables de vérité ou les équations booléennes. Les langages de description matérielle souvent utilisés sont le *Verilog* et le *VHDL*.

La spécification comportementale est ensuite synthétisée en une description structurelle. La synthèse se fait au moyen d'un outil, souvent commercial. L'outil de synthèse analyse la spécification comportementale dans le but d'obtenir une description structurelle qui réalise le comportement requis. Cette description structurelle est souvent exprimée en termes de portes standard. La synthèse est commandée par des contraintes. Une de ces contraintes est l'espace disponible sur la puce pour réaliser le circuit. S'il y a peu d'espace, l'outil de synthèse tente de mieux optimiser le design pour réduire sa complexité et le nombre de portes requises.

La dernière étape pour obtenir la description physique du module est la synthèse matérielle. Cette étape est souvent entièrement automatisée et comporte deux phases. La première phase, le placement, consiste à placer dans l'espace disponible les ensembles de portes standard de telle sorte que la distance entre les portes soit minimisée. La dernière phase, le routage, sert à créer toutes les connexions nécessaires entre les cellules standard pour réaliser la fonction requise. Suite au placement et au routage, le concepteur peut extraire des informations concernant le délai de propagation de tous les chemins du design et la puissance consommée. Avec ces informations, il peut retourner à l'étape de spécification comportementale pour optimiser le design du module pour qu'il consomme moins d'énergie et augmenter la vitesse d'opération.

Il est important pour le concepteur de vérifier l'intégrité du design à chaque étape. Cette vérification est identifiée à la Figure 1.4 par les boîtes *Équivalent*. Aussitôt qu'une erreur est détectée, il est impératif de la corriger immédiatement, car une erreur dans le design peut être très coûteuse à réparer une fois que le design est réalisé.

Une des étapes de vérification du fonctionnement est l'analyse temporelle du circuit. Cette étape permet d'assurer, en partie, l'intégrité des données stockées dans toutes les bascules du

module lors de l'utilisation du circuit. D'autres vérifications temporelles au niveau du temps de transition et de la largeur d'impulsion sont nécessaires pour valider complètement l'intégrité des données. L'intégrité d'une bascule est validée lorsqu'il est possible de prouver que les courses entre les chemins de données et les chemins d'horloge des bascules font en sorte que le signal d'horloge est toujours activé une fois que tous les signaux de données sont stables. Cette analyse donne la fréquence d'opération maximale du module. Pour obtenir un module plus performant, il existe plusieurs méthodes d'optimisation. Une méthode simple est de séparer le traitement du module en plusieurs petites étapes en insérant des bascules entre les étapes dans le chemin de traitement des données. La séparation du traitement des données en plusieurs étapes donne un pipeline de traitement.

Finalement, une fois que le design est fabriqué sur une puce, il est d'autant plus important de s'assurer que le processus de fabrication n'a pas créé d'erreurs au niveau du fonctionnement.

1.3 Conception de circuits asynchrones

Avec le désir d'offrir des performances similaires aux circuits synchrones tout en réduisant l'énergie requise par le circuit, le paradigme asynchrone devient intéressant. Le but du design asynchrone est d'enlever le réseau de distribution de l'horloge globale Beerel *et al.* (2010, page 5). Ce réseau devient de plus en plus complexe à placer correctement en raison du nombre grandissant de modules et de fonctions réalisés par les circuits synchrones. Certains types de design, comme les microprocesseurs et les contrôleurs réseau de haute-performance, sont déjà réalisés avec des approches très personnalisées. En effet, ces méthodes réalisent une partie de la synthèse et du placement du design de façon manuelle. Ces méthodes se prêtent bien au développement asynchrone. Plusieurs compagnies, notamment Fulcrum, Handshake Solutions, Octasic et Tiempo ont produit des puces et des outils de qualité en utilisant le paradigme asynchrone.

1.3.1 Avantages du paradigme asynchrone

Le paradigme asynchrone offre quelques avantages par rapport au paradigme de conception synchrone Beerel *et al.* (2010). Certaines méthodes de conception asynchrones utilisent le concept de design structurel comme la méthodologie synchrone. Ceci fait en sorte que le design est séparé en plusieurs modules de complexité gérable. Normalement, dans la conception synchrone, le concepteur doit prendre en considération le pire cas pour déterminer la fréquence maximale d'opération. En effet, cette fréquence suppose le pire cas de délai de propagation dans le nuage combinatoire pilotant l'entrée des bascules en plus de supposer le pire cas de propagation dans le chemin d'horloge. Certaines méthodes de conception asynchrone utilisent des signaux de contrôle qui dépendent des données Beerel *et al.* (2010, page 8). Ces méthodes permettent d'utiliser le cas moyen de propagation, contrairement au pire cas, ainsi augmentant la performance moyenne du module.

Le fait que la majorité des circuits asynchrones n'ont pas d'horloge globale implique qu'ils sont moins sensibles aux variations du procédé de fabrication. En effet, dans les circuits synchrones, le réseau de l'horloge est le plus gros réseau de distribution présent sur la puce. Cela étant dit, une variation du délai dans une branche de l'arbre de distribution de l'horloge peut avoir des conséquences catastrophiques sur le fonctionnement global du circuit. Les circuits asynchrones, quant à eux, génèrent leurs signaux de contrôle localement. Donc, ils sont moins sensibles aux variations par rapport aux circuits synchrones, Beerel *et al.* (2010, page 9).

Sachant que dans un circuit synchrone le réseau de distribution de l'horloge est présent sur tout le circuit, il est normal d'affirmer que la capacité électrique de ce réseau est grande. De plus, les circuits synchrones commutent à des vitesses de plus en plus grandes. Ces deux faits font en sorte que le réseau de distribution de l'horloge consomme beaucoup d'énergie. Rappelons que selon le type de design de circuit asynchrone, ce réseau de distribution n'est pas présent. Donc, les circuits asynchrones peuvent consommer moins d'énergie que les circuits synchrones, Beerel *et al.* (2010, page 9). En outre, à moins d'utiliser des techniques avancées de conception, le réseau d'horloge d'un circuit synchrone commute tout le temps. Ceci peut

engendrer des transitions inutiles dans les modules de traitement ponctuellement inutilisés. Le paradigme asynchrone est basé sur les événements des signaux de données. Ainsi toutes les transitions sont significatives. Avec moins de transitions, un circuit asynchrone pourrait consommer moins qu'un circuit synchrone réalisant une fonction identique.

Finalement, selon la méthode de conception, les modules d'un circuit asynchrone possèdent des interfaces très bien définies. En effet, le besoin de synchronisation entre les modules force le concepteur à utiliser la même interface uniformément pour tous les modules. Ceci rend le design des modules plus facile à valider comparativement à l'approche *ad hoc* des interfaces des modules présents dans les circuits synchrones, Beerel *et al.* (2010, page 11).

1.3.2 Méthodes de synchronisation

Il existe plusieurs méthodes de synchronisation utilisées dans la conception de circuits asynchrones. Le besoin de synchronisation provient du fait que l'horloge globale est retirée du design. De plus, l'utilisation du design structurel, comme pour la conception de circuits synchrones, fait en sorte que le design est séparé en plus petits modules effectuant une partie du traitement complet. Ces modules doivent maintenant être synchronisés autrement que par l'horloge globale.

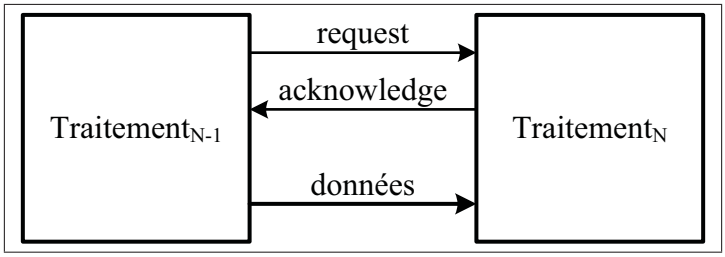

Figure 1.5 Liens de communication entre deux modules de traitement
Adaptée de Beerel *et al.* (2010)

La Figure 1.5 illustre l'interface entre deux modules de traitement. Règle générale, l'élément $Traitement_{N-1}$ indique au module $Traitement_N$ qu'il est prêt à transférer des données présentes

sur le signal *données* en activant le signal *request*. Le module $Traitement_N$ indique qu'il a sauvegardé les données en activant le signal *acknowledge*.

Il existe une multitude de protocoles pour gérer l'activation des signaux de contrôle *request* et *acknowledge*. Cependant, seuls deux protocoles de synchronisation les plus fréquents et intuitifs sont présentés. Il s'agit de la synchronisation à deux phases et la synchronisation à quatre phases Beerel *et al.* (2010, pp. 16-17 et 24-27).

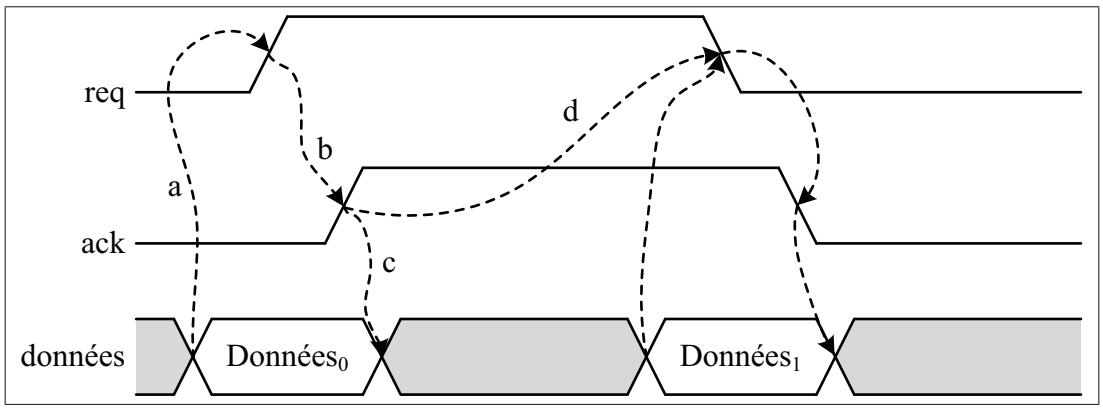

Figure 1.6 Synchronisation à deux phases
Adaptée de Beerel *et al.* (2010)

La synchronisation à deux phases est présentée à la Figure 1.6. La synchronisation se fait au niveau de la transition et ne fait pas la distinction entre le front montant et le front descendant. Après le délai identifié par *a*, le module $Traitement_{N-1}$ indique au suivant que les données sont prêtes. Le module $Traitement_N$ utilise cette transition pour rafraîchir ses bascules d'entrées et indique en utilisant le signal *ack*, après un délai *b*, que le transfert est terminé. Le délai *b*+*d* est utilisé par le module $Traitement_{N-1}$ pour cadencer sa fréquence d'opération. Tous ces délais dépendent des données et les modules doivent posséder des unités de détection de la fin du traitement pour piloter ces signaux.

La synchronisation à quatre phases, illustrée à la Figure 1.7, fait une distinction sur les transitions des signaux de contrôle. Les transitions descendantes arrivent nécessairement après le front montant. Le signal montant *req* est utilisé pour indiquer que d'autres données sont prêtes

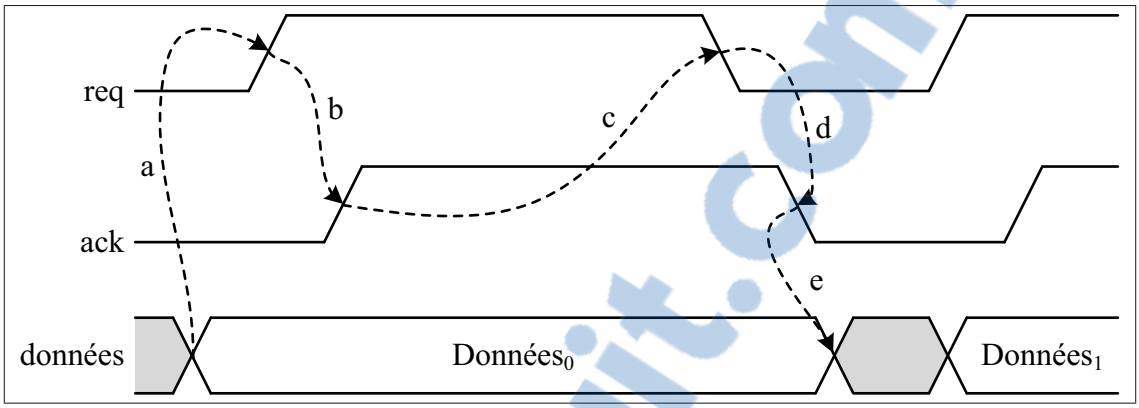

Figure 1.7 Synchronisation à quatre phases
Adaptée de Beerel *et al.* (2010)

et le signal montant *ack* signal que les données sont transférées. Après un délai de traitement *c*, le module transmetteur indique qu'il est prêt à transférer d'autres données. Le module récepteur indique quand il est prêt avec un front descendant sur le signal *ack*. Ce schéma de synchronisation est plus facile à vérifier que le schéma à deux phases, car la vérification se fait selon l'interaction bidirectionnelle des signaux de contrôle. Il est important de noter que ces modules peuvent être en mode *push* ou *pull*. En effet, en permutant la direction des signaux de contrôle *req* et *ack*, les données sont demandées en mode *pull* par le module $Traitement_N$. Les principes de synchronisation s'appliquent dans une direction comme dans l'autre. De plus, ces signaux de contrôle peuvent être utilisés pour transférer un ou plusieurs bits de données.

Une dernière méthode de synchronisation implique de générer le signal *ack* après un certain délai appliqué sur le signal *req*, Beerel *et al.* (2010, page 164). Ce délai dépend du traitement à réaliser sur les données. Ainsi, il n'est plus nécessaire de créer un circuit permettant la détection de la fin du traitement. Une méthode regroupant la gestion et la création des signaux de contrôle dans des modules externes aux modules de traitement a été développée par Octasic, Awad *et al.* (2013). Cette méthode est traitée plus en détail dans le chapitre suivant.

1.3.3 Outils d'aide au design

Il existe quelques flots de conception appliqués aux circuits asynchrones. La compagnie Theseus Logic développe un outil permettant la séparation des modules de traitement et l'insertion automatique de modules de synchronisations, Beerel *et al.* (2010, page 7). Cependant, les résultats de cette séparation automatique ne se comparent pas encore favorablement avec la séparation manuelle telle que proposée par Octasic. Le design conçu manuellement contient des endroits de synchronisation précis et connus. À moins que la séparation automatique utilise strictement le même algorithme que le designer, cette séparation ne sera pas optimale. Une autre compagnie, Handshake Solutions, propose un nouveau langage de développement et un outil de synthèse pour décrire des circuits asynchrones. Les résultats de cet outil ne sont pas optimaux. De plus, ce flot nécessite l'apprentissage par le concepteur d'un nouvel environnement et d'un nouveau langage.

La vérification fonctionnelle d'un circuit asynchrone lors de la conception nécessite un effort considérable de la part du concepteur. De plus, un effort comparable, sinon plus grand, est nécessaire pour vérifier le circuit une fois qu'il est fabriqué. Cette vérification fonctionnelle et la validation doivent être faites manuellement vu le manque d'outils. Donc, il y a un réel besoin pour un outil permettant de faciliter cette vérification fonctionnelle et la validation tout en utilisant les outils du flot de conception traditionnelle synchrone.

1.4 Vérification temporelle des circuits

Les bascules de type D sont les éléments mémoire les plus utilisés lors de la conception de circuits synchrones. La construction de ces bascules est souvent basée sur l'enchaînement de deux verrous, ou *latch*s. Soit un verrou sensible au niveau haut et l'autre sensible au niveau bas. L'enchaînement de ces verrous provoque deux contraintes d'utilisation pour assurer l'intégrité des données sauvegardées dans la bascule. Premièrement, la donnée présentée sur l'entrée donnée de la bascule doit être stable depuis un certain temps avant l'arrivée du front de l'horloge.

Deuxièmement, la donnée doit être stable pour un certain temps après l'arrivée du front. Ces contraintes se nomment *setup* et *hold*, respectivement.

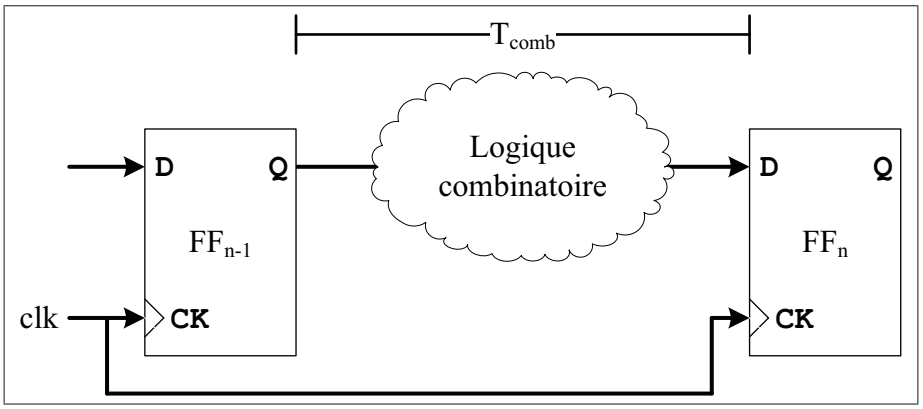

Figure 1.8 Unité d'analyse temporelle

Nécessairement, tout design numérique doit assurer l'intégrité des données. C'est pourquoi les concepteurs effectuent une analyse temporelle du design pour vérifier s'il peut fonctionner à la fréquence prévue. La Figure 1.8 illustre une unité d'analyse temporelle de base. En effet, tout le design peut être séparé en petites unités d'analyse. Chaque unité a pour entrées les entrées du circuit ou la sortie donnée d'une bascule, soit les bascules FF_{n-1} dans la figure. Il va de même pour les sorties de l'unité, soit les bascules FF_n de la Figure 1.8. Sachant quelles sont les entrées et les sorties, il devient facile d'évaluer chacune des unités. Le délai T_{comb} de la Figure 1.8 identifie le délai le plus long dans le nuage combinatoire situé entre les données d'entrées et les données de sorties.

Les deux contraintes d'utilisation des bascules sont mises en équations en utilisant la Figure 1.9 et la Figure 1.8. L'équation 1.7 représente la contrainte de type *setup* et l'équation 1.8 représente la contrainte de type *hold*, ces deux équations sont adaptées de Sapatnekar (2006, page 6-8).

$$T_{clk} \geq T_s + T_{comb} + \phi \tag{1.7}$$

$$T_{comb} \geq T_h + \phi \tag{1.8}$$

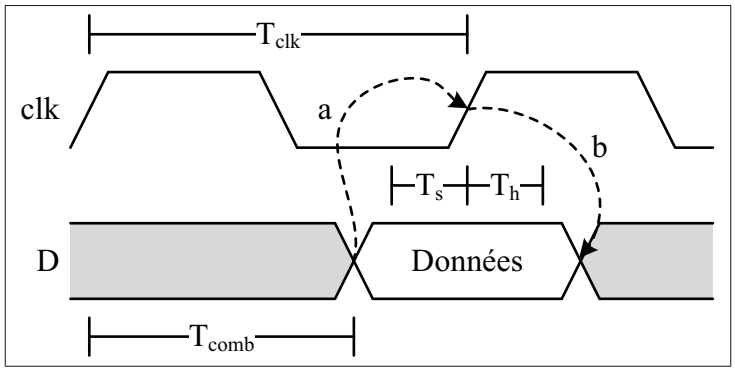

Figure 1.9 Relations temporelles de l'unité d'analyse

La contrainte de *setup* indique que la période de l'horloge ne peut pas être plus petite que la somme du délai le plus long du nuage combinatoire avec le temps de *setup* de la bascule. Le paramètre ϕ est utilisé pour modéliser le décalage de l'horloge dans cette unité d'analyse. Ce décalage est causé, entre autres, par le procédé de fabrication et la nature imparfaite des conducteurs. L'autre contrainte, de *hold*, stipule que les données ne peuvent changer trop rapidement. En effet, les prochaines données à être sauvegardées ne peuvent qu'arriver une fois que les précédentes ont été correctement stockées dans les bascules. Pour ce faire, le délai dans le nuage combinatoire doit être égal ou supérieur au délai de *hold*.

Il est important de bien mesurer le délai des nuages combinatoire et de le comparer aux inégalités pour s'assurer que les contraintes sont respectées. Sans quoi l'intégrité des données et l'état du circuit ne peuvent être garantis lors de son fonctionnement. Le délai combinatoire est composé de deux parties. Il y a le délai causé par les cellules, ou portes logiques, présentes dans le nuage et il y a le délai causé par les interconnexions entre ces cellules. Avant les années 1990s, les délais causés par les interconnexions étaient très petits par rapport aux délais des portes logiques, Kahng *et al.* (2011, page 221). Avec la miniaturisation des transistors et la modulation des tensions d'alimentation, les délais causés par les interconnexions ne sont plus négligeables. Minimalement, il existe deux méthodes pour mesurer le délai du nuage combinatoire. Premièrement, il est possible d'effectuer une simulation de tous les cas d'entrées du nuage combinatoire pour déterminer le délai le plus long. Cette simulation, même pour des circuits de petite taille, devient rapidement inefficace. Une autre méthode, dite statique, n'utilise pas la

simulation fonctionnelle, mais bien la structure du circuit, ignorant ainsi son fonctionnement. Cette méthode, beaucoup plus efficace, est utilisée par les outils d'analyse automatique. Les sous-sections suivantes présentent les méthodes automatiques d'analyse statique traditionnelle et d'analyse statistique-statique.

1.4.1 Analyse statique

Les outils d'analyse statique fonctionnent en créant une représentation sous forme de graphe du circuit à analyser. En effet, le circuit est d'abord séparé en unités d'analyse. Ces unités peuvent être aussi petites qu'un nuage combinatoire et aussi complexes que le circuit en entier. L'élaboration des liens hiérarchiques entre les modules du circuit permet d'obtenir toutes les unités d'analyse qui composent le circuit. Ces graphes sont composés, naturellement, de nœuds et de vertex. Les vertex représentent les portes dans le nuage combinatoire et les nœuds représentent les interconnexions. Ces portes logiques peuvent elles aussi être représentées comme étant des graphes, avec un vertex pour chaque entrée et chaque sortie et un nœud pour chaque lien entrée-sortie. Comme mentionné précédemment, les entrées des unités d'analyse sont des entrées principales du circuit ou des sorties données d'éléments mémoire. Les sorties sont soit des sorties principales du circuit ou des entrées données de bascules, Sapatnekar (2006, page 6-2). Il est intéressant de noter que le temps d'exécution des outils automatiques est beaucoup plus rapide que l'approche par simulation. En effet, le temps d'exécution est linéaire avec le nombre de nœuds et de vertex, tandis que pour la simulation, il est exponentiel.

Le but des outils d'analyse temporelle est de déterminer les délais et les chemins critiques du circuit pour ensuite vérifier les inégalités de l'équation 1.7 et de l'équation 1.8. Pour ce faire, l'outil résout les unités d'analyse récursivement, du niveau le plus bas au niveau le plus haut. Le niveau le plus haut étant celui qui contient les entrées et sorties du design. La résolution se fait par la méthode du chemin critique. Cette méthode se fait en deux temps. Premièrement, l'outil calcule tous les délais de tous les nœuds et les vertex des entrées vers les sorties. Il existe un délai pour la transition montante et un délai pour la transition descendante. Les délais des portes logiques sont obtenus dans la librairie de cellules standard. Les fabricants des librairies

assurent le calcul des délais internes. Les délais dus aux interconnexions sont calculés par l'outil d'analyse en utilisant les données qualitatives des cellules standard. Il existe plusieurs méthodes de calcul de précision et de temps d'exécution variables, Sapatnekar (2006, page 6-3). Toutes ces méthodes utilisent la capacité parasite calculée de l'interconnexion, le temps de transition à l'entrée de la cellule standard ainsi que sa taille. Deuxièmement, l'outil détermine le chemin critique en effectuant le chemin inverse, soit de la sortie vers les entrées. Le chemin critique est défini comme étant celui qui a le délai le plus long, tant pour une transition montante que descendante. Il peut donc y avoir deux chemins critiques pour chaque sortie de chaque unité d'analyse.

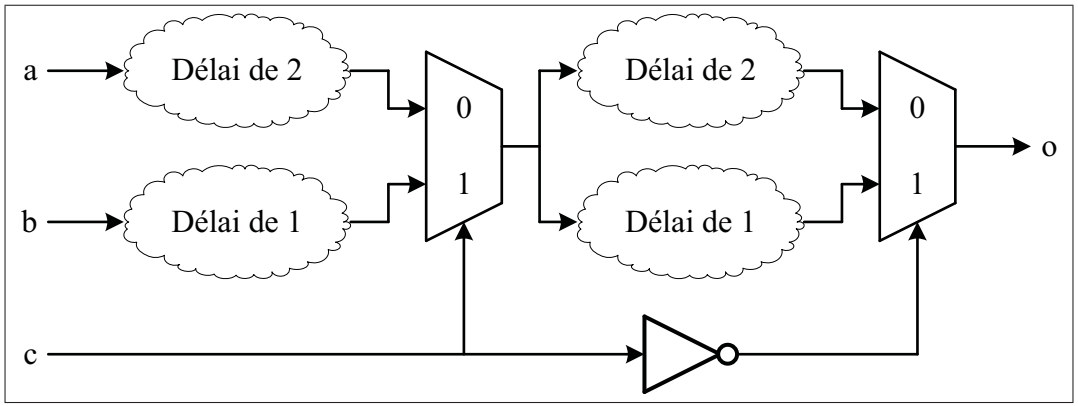

Figure 1.10 Exemple d'un circuit combinatoire contenant au moins un faux chemin critique
Adaptée de Sapatnekar (2006)

La nature structurelle de l'analyse fait en sorte que les outils d'analyse temporelle peuvent identifier des chemins critiques qui n'existent pas, Sapatnekar (2006, page 6-4). La Figure 1.10 illustre un tel faux chemin critique. En effet, un outil d'analyse statique trouverait un chemin critique avec un délai de 4 unités entre la sortie *o* et l'entrée *a*. Cependant, en observant le fonctionnement du circuit, il est évident qu'un tel chemin n'existe pas. Ceci est dû au fait que l'outil pose l'hypothèse que les éléments du circuit n'ont pas de conséquences sur les délais réels. Donc, les résultats fournis par les outils d'analyse sont souvent pessimistes. Ceci pourrait

causer une perte de temps au niveau du design si le concepteur tentait de résoudre un problème de vérification temporelle qui n'arrivera jamais.

Une fois que les chemins critiques ont été identifiés, l'outil calcule deux autres valeurs pour chacun des vertex du chemin critique. En utilisant la même méthode que le calcul des délais à la première phase, l'outil trouve le temps réel d'arrivée de la donnée pour chaque vertex. Ensuite, il procède de la sortie vers l'entrée pour calculer le temps requis d'arrivée en utilisant comme contrainte la période de l'horloge, Kahng *et al.* (2011, pages 226-227). Si, pour chaque chemin critique, le temps réel d'arrivée est plus petit ou égal au temps requis d'arrivée, la contrainte de *setup* est vérifiée. De plus, si la donnée sur le chemin critique reste stable pendant au moins le temps de *hold*, la contrainte de *hold* est alors vérifiée. La différence entre le temps réel d'arrivée et le temps requis d'arrivée permet d'obtenir la marge. Si la marge est positive, le design pourrait fonctionner à une fréquence plus élevée. Sinon, le design doit être ajusté pour fonctionner à la fréquence voulue ou la fréquence voulue doit être changée.

1.4.2 Analyse statistique-statique

L'analyse temporelle statique traditionnelle est basée sur l'hypothèse que tous les paramètres qui dépendent du procédé sont stables et identiques sur toutes les puces produites. Ces paramètres comprennent, notamment, la tension d'alimentation, la longueur de la grille et du canal du transistor, l'épaisseur de l'oxyde ainsi que la température d'opération. Cependant, les procédés produisant des puces avec des transistors de tailles d'ordre nanométrique introduisent des variances au niveau des paramètres physiques de la puce. Ces variances peuvent être observées d'une puce à l'autre et au sein d'une même puce. Nécessairement, les paramètres physiques de la puce ont un impact sur les délais des portes logiques et des interconnexions. Pour tenter de caractériser ces paramètres, plusieurs outils d'analyse permettent l'utilisation de fichiers décrivant une configuration de ces paramètres physiques. Pour bien cerner le cadre d'utilisation et les différentes combinaisons des paramètres, il est fréquent de devoir utiliser 12 fichiers de configuration différents, Blaauw *et al.* (2008). Certains concepteurs vont même utiliser jusqu'à une centaine de fichiers. Ceci rend l'étape de vérification temporelle beaucoup plus longue.

Pour pallier ce problème, plusieurs chercheurs se sont penchés sur comment modéliser adéquatement ces paramètres physiques, Blaauw *et al.* (2008).

Il y a trois catégories d'incertitudes, Blaauw *et al.* (2008). En effet, il y a les incertitudes causées par les erreurs de modélisation des portes et des liens. De plus, il a les incertitudes causées par le procédé de fabrication. Finalement, il a les incertitudes causées par l'environnement d'utilisation de la puce. Les incertitudes causées par le procédé de fabrication entraînent des variances au niveau des paramètres physiques de la puce. Rappelons que ces paramètres sont, notamment, la taille du transistor, la concentration de l'agent dopant, la largeur et l'épaisseur des fils et l'épaisseur de l'oxyde de grille. Ces paramètres physiques ont un impact direct sur les caractéristiques électriques des transistors et des cellules standard utilisées dans la puce. Évidemment, les caractéristiques électriques ont un effet sur les délais des portes et des fils. Il est important de noter que des caractéristiques électriques différentes et complémentaires peuvent dépendre des mêmes paramètres physiques.

Les outils d'analyse temporelle statistique posent l'hypothèse que les paramètres physiques à modéliser sont effectivement des variables aléatoires indépendantes. Si elles ont une corrélation, cette corrélation est bien définie est prise en charge par l'outil. Les variations du procédé causant les incertitudes des paramètres physiques peuvent être classifiées en plusieurs catégories, Blaauw *et al.* (2008). D'abord, les variances peuvent être déterministes ou non déterministes. Les variations déterministes sont causées par le placement et le routage du circuit. Le placement et le routage ont des effets connus sur le procédé et peuvent être modélisés. Les variations non déterministes peuvent causer des changements au niveau des paramètres au sein de la même puce ou bien d'un ensemble de puces à un autre. En effet, les changements au sein de la même puce sont dus, notamment, à la non-régularité de la lumière appliquée et à la distribution non uniforme des acides lors de l'exposition. De plus, les écarts des paramètres d'un ensemble de puces à un autre sont causés par les petites variations de la calibration ainsi que par les imprécisions de l'équipement de production.

Traditionnellement, les outils d'analyse statique utilisent un graphe pour représenter le circuit à analyser. Les nœuds de ce graphe représentent les interconnexions et les vertex représentent les cellules standard du circuit. Attachés à ces nœuds et vertex, sont les délais des éléments représentés. Ensuite, l'outil trouve tous les délais et le chemin critique. L'outil d'analyse statistique fonctionne selon la même méthode. Cependant, les paramètres physiques permettant de calculer les délais des portes et des liens sont des variables aléatoires. Ainsi, les délais maximaux des chemins critiques ont eux aussi des composantes aléatoires. Ces délais peuvent être caractérisés par une distribution normale, Blaauw *et al.* (2008). Différents problèmes viennent complexifier la modélisation et la caractérisation des chemins critiques. En effet, lors de la fabrication, certains chemins auront des variations physiques qui sont corrélées soit dans le temps ou dans l'espace. De plus, la distribution normale peut être moins efficace comme moyen de représentation si les variations physiques engendrées par le procédé sont des changements non linéaires. Finalement, un autre problème de modélisation survient lorsque la fonction de comparaison utilisée pour trouver le chemin critique entraîne un biais.

Les outils d'analyse statistique pallient ces problèmes et modélisent tout de même le chemin critique en utilisant des méthodes traditionnelles numériques ou des méthodes d'analyse statistique. En effet, certains outils utilisent une méthode d'échantillonnage de l'espace de recherche composé par toutes les possibilités des variables des caractéristiques physiques et une méthode de résolution numérique pour caractériser le chemin critique trouvé. D'autres outils utilisent la même méthode, mais avec une simulation de type Monte-Carlo pour mieux caractériser et échantillonner l'espace de recherche. Cependant, ces méthodes nécessitent beaucoup de temps de calcul vu leur nature numérique. Pour cette raison, les outils plus modernes d'analyse statistique utilisent des méthodes d'analyses statistiques. La section IV de l'article de revue des méthodes statistiques, Blaauw *et al.* (2008), contient plus de détails sur ces méthodes.

1.4.3 Méthodes d'analyse appliquées aux circuits asynchrones

L'application de l'analyse temporelle des circuits aux designs asynchrones utilisant un protocole de synchronisation à deux ou quatre phases peut entraîner certains problèmes, Yahya *et al.*

(2013). En effet, il est facile de remarquer qu'il y a au moins quatre catégories de problèmes auxquelles les concepteurs font face lors de la vérification temporelle de circuits asynchrones. Premièrement, il y a le modèle de délai. Traditionnellement, le délai dans les portes logiques et les interconnexions est présumé être le pire dans le but d'obtenir une fréquence d'opération qui garantit le bon fonctionnement du circuit dans toutes les conditions. Cependant, les circuits asynchrones ne possèdent pas d'horloge globale et les délais sont considérés comme étant dans la moyenne. Donc, ces délais doivent être modélisés statistiquement. Deuxièmement, la méthode de résolution traditionnelle ne peut pas être appliquée normalement, car la nature asynchrone des circuits ne donne pas une structure comme celle présente dans les circuits synchrones. Mais, il existe plusieurs méthodes pour modéliser correctement les interactions d'un design asynchrone. Donc, il est possible de caractériser correctement les délais en développant une nouvelle méthode de résolution. Ensuite, il y a la question du type de métrique ou du critère de performance à analyser. En effet, la nature asynchrone n'impose pas le pire cas de délai, mais bien le cas moyen. Ceci est causé par l'absence d'une horloge globale. Ainsi, les outils d'analyse pour circuits asynchrones doivent trouver une méthode pour mesurer le cas moyen et évaluer la distribution statistique des délais. Finalement, il existe plusieurs types de méthodologies de conception de circuits asynchrones. Ces méthodes génèrent une structure de circuit qui leur est propre. Les outils de vérification temporelle, en plus de répondre aux besoins mentionnés précédemment, doivent être le plus génériques possible. Sinon, leur utilisation sera limitée à un type de méthodologie et a un type de structure. Pour pallier ces problèmes, plusieurs outils ont été développés. Les paragraphes qui suivent présentent trois outils typiques.

L'outil développé dans Davies et Woods (1996) analyse les contraintes pour les circuits asynchrones à base de micropipelines avec des modules de synchronisation qui implémentent le protocole de synchronisation à 4-phases vu précédemment. Ces contraintes sont élaborées selon les liens de dépendance entre l'élément transmetteur et l'élément récepteur et sont exprimées en fonction des signaux *ack* et *req*. La nature des contraintes à vérifier fait en sorte qu'elles ne se valident pas facilement avec les outils d'analyse traditionnels. Pour les vérifier, les auteurs de l'outil ont donc recours à la simulation. Ce nouvel outil est utile seulement pour

un type de design de circuit asynchrones, celui basé sur les micropipelines. Il n'est pas généralisé et ne s'applique pas aux autres méthodologies. De plus, l'utilisation de la simulation engendre des temps de vérification plus longs que les outils de vérification des circuits synchrones. Finalement, la création d'un nouvel outil implique souvent la création d'un nouveau format de fichier de configuration.

Une autre suite logicielle, Karlsen et Røine (1999), est inspirée de Davies et Woods (1996). Celle-ci nécessite la simulation exhaustive du circuit pour la vérification des contraintes. La suite prend comme paramètres d'entrées le circuit sous sa forme *netlist* ainsi qu'une description sous forme de graphe de l'interaction entre les modules. Un outil de la suite permet de vérifier la concordance entre la *netlist* et le graphe d'interactions. Pour ce faire, il a besoin des résultats de simulation. Cet outil valide ultimement les contraintes de *setup* et de *hold*. Le même problème que l'outil développé dans Davies et Woods (1996) survient, l'outil est basé sur la simulation et nécessite l'apprentissage d'un nouveau langage.

L'outil de Yahya *et al.* (2013) est plus récent et plus avancé. En effet, il effectue une analyse statistique des délais. Il répond aux problèmes mentionnés au début de cette sous-section en développant un outil basé sur la simulation et sur l'utilisation de modèles. Effectivement, ces modèles sont les éléments de base de construction de circuits asynchrones au sein de la méthodologie imposée par l'outil. Ces éléments de base sont très bien caractérisés en terme de performance et leurs délais sont aussi bien connus. Les éléments sont liés en utilisant une description *XML* et une méthode de synchronisation basée sur le protocole à 4-phases. Comme pour les deux autres outils, cet outil comporte des problèmes au niveau de l'acceptation générale. En effet, l'outil nécessite au concepteur d'apprendre une nouvelle méthodologie de design et nouveau langage.

Certaines compagnies ont développé des outils et des flots de conception pour faciliter la création et la vérification de circuits asynchrones. En effet, les compagnies Handshake Solutions, Fulcrum et Theseus Logic ont tous un flot unique, Beerel *et al.* (2010). Cependant, ces flots et

outils de conception imposent au concepteur d'apprendre un autre flot, comme pour Fulcrum, et parfois même un autre langage comme dans le cas de Handshake Solutions.

En observant brièvement ce qui se fait dans l'industrie et dans la littérature, il est facile de constater que ces méthodes et outils sont inadéquats. En effet, ils nécessitent l'apprentissage d'un nouveau langage ou d'un flot de conception. De plus, si le design est déjà existant, il faut recoder le design, sans fautes, en utilisant le nouveau langage de conception. Ceci pourrait être considéré comme étant inacceptable pour les concepteurs de circuits traditionnels. Donc, pour que le paradigme asynchrone soit plus accepté, les outils devraient tenter d'adapter le flot traditionnel aux circuits asynchrones et non pas forcer les concepteurs à s'adapter à un nouvel outil.

1.5 Test des circuits fabriqués

Le flot traditionnel de conception de circuits synchrones, illustré à la Figure 1.4, prévoit l'implantation de la spécification physique dans un circuit intégré. Cette implantation se fait au moyen d'un processus basé sur la photolithographie. Naturellement, le procédé n'est pas parfait. En effet, certaines puces produites ne seront pas fonctionnelles. Dans le but d'obtenir une production de qualité, il est impératif de s'assurer de ne pas fournir des puces non fonctionnelles, ou pire, partiellement fonctionnelles, aux clients. Le test, Bushnell et Agrawal (2002), permet d'effectuer deux choses. Premièrement, le test permet de s'assurer que la puce produite fonctionne selon la spécification. Deuxièmement, le test permet de valider le fonctionnement du procédé. Il est important de noter que la vérification du fonctionnement de la puce une fois produite est pertinente seulement si le circuit conçu a été validé après toutes les étapes de conception. Cela doit être fait pour garantir que la tâche réalisée par la puce est bien celle qui était planifiée initialement.

Les procédés de fabrication peuvent introduire de légères défectuosités au niveau physique de la puce. Une défectuosité peut entraîner des erreurs au niveau du fonctionnement de la puce, Bushnell et Agrawal (2002, page 58). Il existe plusieurs types de défectuosités, mais

les plus fréquentes sont les circuits ouverts et les courts-circuits. Les erreurs causées par les défectuosités sont souvent formalisées par des modèles de pannes. Ces derniers sont utilisés par les outils d'aide au test pour permettre à l'ingénieur de test de vérifier la fabrication de la puce.

1.5.1 Modèles de pannes

Selon le domaine d'abstraction, il existe plusieurs modèles de pannes. En effet, il existe des modèles pour représenter de manière abstraite l'effet des défectuosités induites dans le circuit par le procédé de fabrication. Le même principe s'applique pour modéliser les pannes d'un design synthétisé. Le niveau d'intérêt visé par la méthodologie de test, le niveau physique, possède aussi plusieurs modèles de pannes. Cependant, trois types de pannes sont plus pertinents que les autres, Bushnell et Agrawal (2002, pages 60-70). Les modèles de pannes les plus utilisés sont le *bridging*, le *delay* et le *stuck-at*.

Le modèle *bridging* tente de représenter une défectuosité au niveau de la fabrication matérielle qui causerait un court-circuit entre deux groupes de signaux. Le groupe ainsi lié peut être dominant haut ou dominant bas Bushnell et Agrawal (2002, page 61). Un groupe dominant haut réalise la fonction OU et un groupe dominant bas réalise la fonction ET.

La panne *stuck-at* pose l'hypothèse qu'un signal du circuit soit fixé au niveau haut, dit *stuck-at 1*, ou au niveau bas, dit *stuck-at 0*. Les tests sont faits en supposant qu'il y aura seulement un nœud du circuit qui comportera une panne de type *stuck-at*. Cette hypothèse n'est pas fausse, car l'espace de recherche pour les pannes de types *stuck-at* simples recouvre presque tout le temps l'espace de recherche des pannes *stuck-at* multiples. En effet, si le nombre de sorties est suffisamment grand et que le circuit est suffisamment complexe, l'hypothèse tient, Bushnell et Agrawal (2002, page 63).

Il est important de noter que ces pannes sont de type structurel. En effet, ces modèles ne se soucient pas du fonctionnement global du circuit. Ils utilisent la structure interne du circuit, soit les branchements entre les cellules standard. Donc, le modèle *stuck-at* s'applique effectivement

aux entrées et aux sorties de toutes les cellules standard instanciées dans le circuit. Il va de soi pour le modèle de pont, il s'applique aux nœuds entre les éléments de base.

Finalement, la panne de type *delay* est particulièrement intéressante lorsqu'elle s'applique aux circuits asynchrones. En effet, les circuits développés en utilisant une méthode comme dans Awad *et al.* (2013) ont une contrainte serrée sur le chemin d'horloge. Une erreur de délai rendrait le circuit non fonctionnel. Il existe, généralement, cinq sous-classes de pannes de type *délai*. Soit, les pannes de type chemin, segment, ligne ainsi que les pannes de type transitions et de délai par les cellules standard Bushnell et Agrawal (2002, page 61). Deux de ces sous-classes, soit la panne de transition et la panne de délai sur un chemin, sont très pertinentes pour les circuits synchrones et asynchrones. Une panne de transition implique qu'une cellule standard a un délai de propagation différent des autres cellules identiques. De cette façon, une porte peut être lente à monter, ou lente à descendre, Bushnell et Agrawal (2002, page 70). Dans les deux cas, le délai est trop long pour que la donnée présentée aux cellules branchées sur la sortie soit valide. Une panne de délai sur un chemin modélise une accumulation de délai dans un chemin de données. Un chemin de données est compris entre deux éléments mémoire. Entre ces deux éléments mémoire, il peut y avoir plusieurs chemins de données distincts. Chacun de ces chemins peut avoir une panne lorsqu'il est lent à descendre ou lent à monter.

1.5.2 Méthodes de test

Il y a deux méthodes pour tester les designs réalisés sur une puce. En effet, il y a le test fonctionnel et le test structurel, Bushnell et Agrawal (2002, page 156). Le test fonctionnel implique de tester exhaustivement toutes les combinaisons d'entrées du circuit et de vérifier la sortie pour s'assurer qu'elle correspond au modèle simulé. Cette méthode devient rapidement inefficace. Effectivement, pour tester exhaustivement un additionneur 32 bits par 32 bits, il faut tester les 2^{64} combinaisons d'entrées différentes. En supposant que l'équipement de test automatisé fonctionne à 1 GHz, il lui faudrait 2^{64} ns. Donc, l'équipement de test aurait besoin de 584,56 années pour vérifier tous les patrons. Visiblement, ceci est beaucoup trop long et inefficace. Le test structurel, au lieu de se baser sur le fonctionnement de la puce, est basé sur les connexions

entre les cellules standard qui forment le design. Il est très pertinent d'utiliser cette méthode par rapport à la méthode de test fonctionnelle, car en utilisant la méthode de test fonctionnelle, les mêmes chemins sont vérifiés plusieurs fois.

Les tests structurels se font en utilisant des patrons de tests. En supposant qu'il y a seulement une panne à la fois, un même vecteur de test peut être utilisé pour tester plusieurs pannes à la fois. Il existe des outils de génération automatique des patrons de tests qui utilisent des techniques pour réduire le nombre de patrons à utiliser pour vérifier un maximum de pannes possible. En effet, ces outils minimisent le nombre de patrons tout en conservant un haut taux de couverture. Le taux de couverture est défini comme étant le nombre de pannes que les patrons peuvent détecter divisé par le nombre de pannes totales. Il n'est pas rare d'obtenir des taux de couverture au-delà de 98%, Bushnell et Agrawal (2002, page 157).

Les patrons visent une panne en particulier dans le design. Soit, par exemple, une panne de type *stuck-at 1* ou *stuck-at 0*. Pour ce faire, ils utilisent les entrées primaires du module pour fixer l'endroit de la panne au niveau inverse de la panne testée. Par exemple, pour vérifier s'il y a une panne de type *stuck-at 1*, le patron doit assigner ce nœud à la valeur 0. Ensuite, le patron stimule le reste du circuit pour propager la valeur à l'endroit de la panne vers les sorties primaires du circuit. Finalement, les sorties primaires sont comparées aux sorties primaires d'un modèle sans panne utilisant le même patron d'entrées. Si les deux sont identiques, il n'y a pas de panne.

Naturellement, les circuits de nature asynchrone ne sont pas supportés par les outils du flot traditionnel. Pour permettre à ces outils de générer des patrons de tests pour des circuits asynchrones, il faut générer de multiples fichiers de configuration. L'outil qmi a pour but de faciliter la création de ces fichiers de configuration.

1.5.3 Tessent de Mentor Graphics

La suite logicielle Tessent de Mentor Graphics est utilisée pour générer des patrons de tests et pour ajouter automatiquement les circuits nécessaires pour permettre le test des puces pro-

duites. Cette suite permet l'utilisation des fichiers de configuration mentionnés dans la sous-section précédente. En effet, ces fichiers sont des scripts Tcl qui permettent de paramétrer l'outil de génération des patrons de test. L'outil développé dans ce mémoire, qmi, permet de générer cesdits fichiers. Ainsi, il est possible de manipuler un outil du flot traditionnel de conception pour qu'il puisse traiter des circuits asynchrones.

1.6 Conclusion

En résumé, le chapitre explique les notions théoriques de base nécessaires au développement de l'outil répondant à la problématique. En effet, il a été question de la théorie des graphes, utilisée par l'outil ainsi que par les outils de vérification temporelle statiques et statistiques. Ensuite, le flot de conception traditionnel des circuits synchrones basés sur le design structurel a été survolé. Les avantages et les types de conception de circuits asynchrones ont été explicités. Finalement, la pertinence du test et de l'analyse temporelle des circuits ont été mises de l'avant ainsi que les liens avec la conception de circuits asynchrones.

CHAPITRE 2

MÉTHODOLOGIE DE DESIGN D'OCTASIC

Le chapitre qui suit présente d'abord la méthodologie de design utilisée par Octasic. Ensuite, le fonctionnement général des deux circuits de tests analysés, soit tok_test et pico_alu, est expliqué. Finalement, le chapitre se conclut avec une description des trois types de configuration généralisant les structures mises à l'œuvre dans l'implémentation des circuits développés par Octasic.

2.1 Architecture typique des circuits d'Octasic

La nature asynchrone des designs produits par Octasic s'inspire du modèle traditionnel de conception de circuits synchrones. L'architecture de conception synchrone haute-vitesse est illustrée à la Figure 2.1.

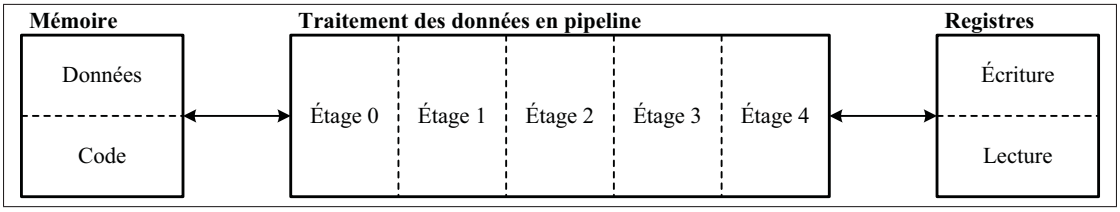

Figure 2.1 Pipeline de traitement des données traditionnel

Le circuit de démonstration illustré dans la figure précédente contient deux ressources, soit un banc de registre et une mémoire. De plus, le traitement des données est réalisé par une unité de traitement très performante. La performance est assurée par la séparation du traitement en cinq étages. Une représentation de chaque étage typique est disponible à la Figure 2.2. Il y a un banc de bascules entre les étages et chaque étage effectue une petite partie du traitement. Donc, chaque banc de bascules est à la fois la sortie de l'étage N et l'entrée de l'étage $N+1$. Les étages sont cadencés par un signal d'horloge global au circuit. La période de cette horloge est ajustée pour être légèrement plus grande que le temps de propagation le plus long

de l'étape de traitement la plus lente. L'horloge doit être distribuée uniformément dans tout le circuit. Ceci pose des problèmes au niveau du routage, notamment par la latence causée par les délais de propagation des branches de l'arbre de distribution. De plus, cet arbre de distribution consomme beaucoup d'énergie due à la grande vitesse à laquelle il commute.

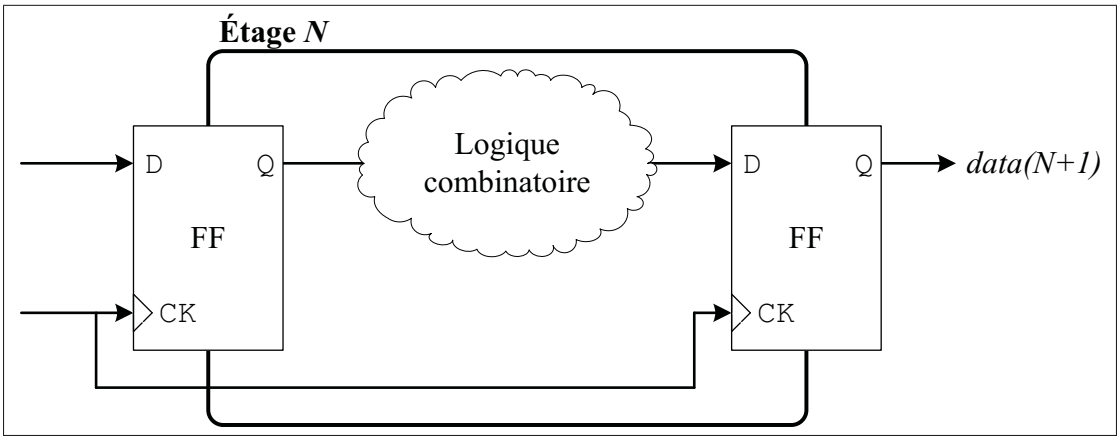

Figure 2.2 Étage d'un pipeline traditionnel

Dans le but d'éliminer le réseau de distribution et de faciliter le design des modules traitant les données, Octasic a en quelque sorte déconstruit le modèle traditionnel de conception. En effet, au lieu d'avoir une seule unité de traitement très performante, les designs d'Octasic sont réalisés en utilisant plusieurs unités de traitement non-pipelinées, ou monocycle, fonctionnant en parallèle.

Cette déconstruction engendre deux problèmes, celui de l'ordonnancement des opérations et celui de la gestion de l'accès aux ressources. L'ordonnancement des opérations se faisait naturellement avec une seule unité de traitement. Cependant, maintenant que les unités fonctionnent en parallèle, il faut indiquer aux unités quelles étapes elles doivent accomplir. La Figure 2.3 rend évident le problème de gestion des ressources. Effectivement, plusieurs unités de traitement peuvent accéder aux mêmes ressources. Il y a donc un besoin pour un moyen de synchronisation des accès aux ressources du circuit.

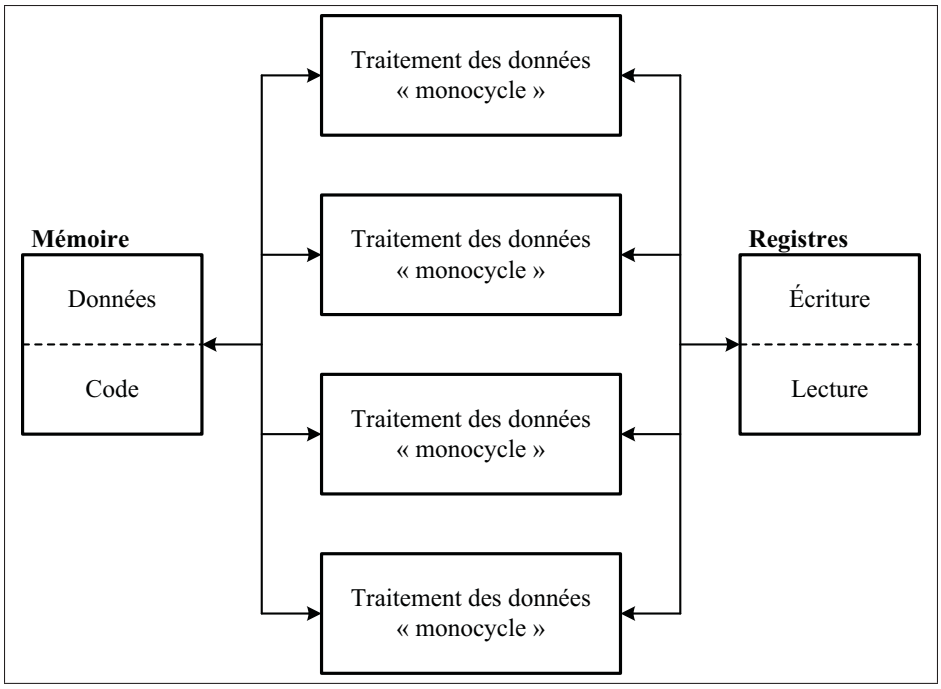

Figure 2.3 Pipeline de traitement des données deconstruit

2.2 Synchronisation par jetons

Pour répondre au besoin de synchronisation, Octasic a développé une technique utilisant le principe du *jeton*, voir Awad *et al.* (2013). Il doit y avoir, notamment, un jeton par type d'accès aux ressources. Un jeton peut aussi exister pour représenter le lancement du traitement des données et un autre pour indiquer que le traitement des données est terminé. Dans le cas illustré dans la Figure 2.4, il y existe effectivement au moins quatre jetons. Un pour l'accès en lecture de la mémoire, nommé *memrd*, et un autre pour l'écriture appelé *memwr*. Pour le banc de registres, il y a un jeton pour la lecture, *regrd*, et un jeton pour l'écriture, *regwr*.

Les jetons sont asservis par des unités de traitement des jetons. Il doit y avoir une unité de traitement des jetons pour chaque unité de traitement des données. Les modules de traitement des jetons sont branchés en boucle pour assurer l'ordre de passage des jetons. Le chemin en pointillé de la Figure 2.4 contient tous les jetons. Dans une implémentation réelle, il y a une ligne distincte par jeton.

En plus de synchroniser les accès aux ressources, les jetons permettent aussi de gérer l'ordre de traitement. En effet, dans l'exemple, les unités de traitement des données doivent nécessairement lire la mémoire contenant le code dans le but de savoir quelle est l'opération qu'elles doivent effectuer. Ensuite, ces unités déterminent quelles sont les ressources dont elles ont besoin pour effectuer leurs tâches.

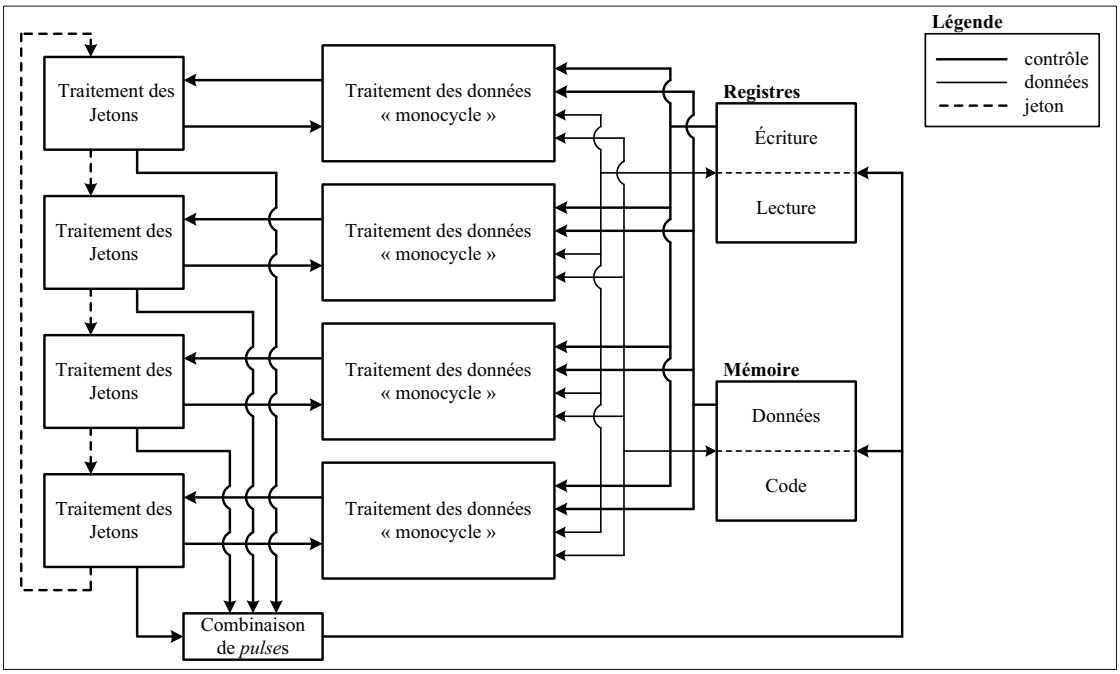

Figure 2.4 Synchronisation par jetons. Montre les chemins de données en point simple, de contrôle en gras et des jetons en pointillé

Toute la synchronisation du circuit repose sur le fonctionnement des unités de traitement des jetons. La possession d'un jeton par l'unité de traitement des données se fait par son unité de traitement des jetons. Une fois que le jeton est à l'intérieur de l'unité, cette dernière doit décider si elle utilise le jeton ou non. Si l'unité de traitement des données attachée à l'unité de traitement des jetons n'a pas besoin de la ressource associée au jeton, le module de traitement des jetons passe le jeton à l'unité de traitement suivante. Cependant, si l'unité de traitement des données a besoin du jeton, le module de traitement des jetons va générer les signaux de contrôle nécessaires pour permettre l'accès à la ressource. Les signaux de contrôles sont marqués par

les lignes grasses et les signaux de données sont tracés par des lignes minces à la Figure 2.4. Une fois l'accès terminé, le module de traitement des jetons passe le jeton au module suivant. Le passage du jeton se fait donc avec délai ou sans délai. En effet, si le jeton n'est pas utilisé par l'unité, le passage se fait sans délai. S'il est utilisé, le passage se fait avec délai. Ce délai est prédéterminé et dépend du temps d'accès à la ressource. Donc, chaque jeton a une vitesse maximale de passage déterminée par le temps d'accès à la ressource qu'il gère. Dans les conditions de passage ou d'utilisation d'un jeton, il y a aussi la dépendance de l'ordre d'utilisation des jetons. Par exemple, la dernière opération effectuée par une unité de traitement des données est l'écriture dans le banc de registre et la première chose à effectuer est la lecture en mémoire. Donc, l'unité de traitement des jetons ne peut pas passer le jeton d'écriture du registre sans avoir passé le jeton de lecture en mémoire.

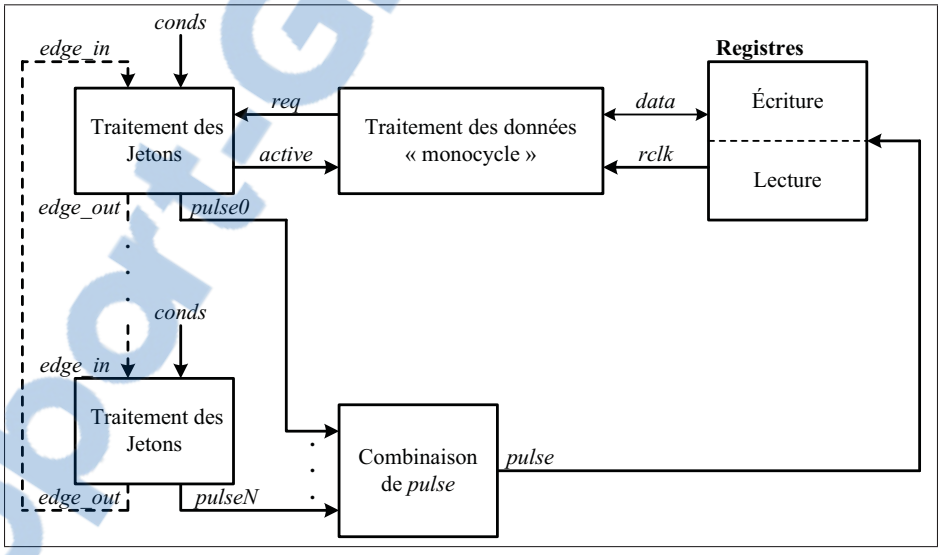

Figure 2.5 Détails de l'utilisation d'un jeton et de sa ressource

Les signaux concrètement utilisés pour réaliser les fonctions des jetons sont représentés à la Figure 2.5. Chaque module de gestion de jetons possède une entrée *edge_in* et une sortie *edge_out* pour l'entrée et la sortie de chaque jeton. De plus, les signaux représentant les conditions de passage sont identifiés par *conds*.

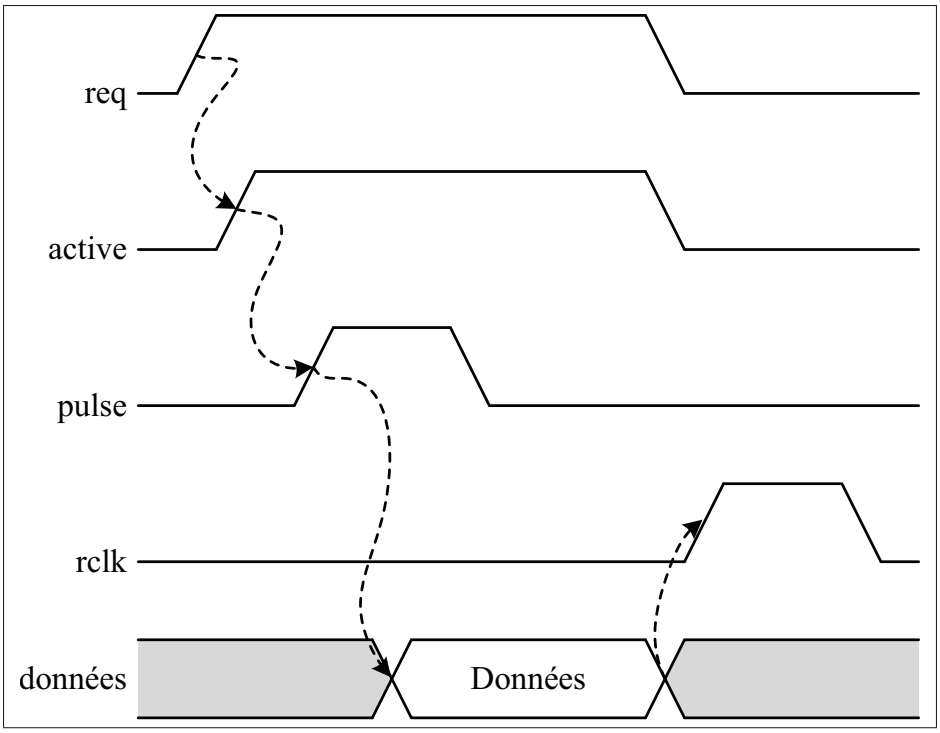

Figure 2.6 Chronogramme d'utilisation d'un jeton

La Figure 2.6 illustre les liens entre les signaux de contrôle d'un jeton. Premièrement, pour indiquer au module de traitement des jetons que l'unité de traitement des données à besoin d'une ressource, elle active la ligne *req* correspondant à la ressource en question. Ensuite, lorsque le module de traitement des jetons possède le jeton et qu'il détermine qu'il y a un besoin, il active les signaux *pulseN* et *active*. Le signal *active* indique à l'unité de traitement des données que sa requête d'accès est en train de se dérouler. Étant donné qu'il y a plusieurs modules de traitement des jetons et seulement une ressource, il existe une unité de combinaison des signaux *pulseN* permettant de générer une seule impulsion pour piloter l'entrée *pulse* de la ressource. La ressource utilise l'impulsion *pulse* pour traiter les données du signal *data*. Dans ce cas-ci, le traitement peut être soit une écriture dans le banc de registre avec le jeton *regwr* ou une lecture avec le jeton *regrd*. Pour alléger la Figure 2.5, elle ne contient pas les paramètres supplémentaires que l'unité de traitement pourrait fournir à la ressource. Finalement, lorsque le transfert de données entre la ressource et l'unité de traitement est complété, la ressource active le signal *rclk*.

2.3 Circuits de test illustrant la méthodologie

Des circuits simplifiés ont été développés par une ressource interne chez Octasic dans le but d'illustrer la méthodologie utilisée par les concepteurs de circuits intégrés chez Octasic. Il y a deux circuits de tests, soit tok_test (Leclerc (2013a)) et pico_alu (Leclerc (2013b)). Le fonctionnement général et le partitionnement entre les signaux de données et les signaux contrôle de ces circuits sont expliqués dans les sous-sections suivantes.

2.3.1 tok_test

tok_test est un circuit de test très simplifié. En effet, il est composé uniquement d'une ressource, d'un jeton et d'une unité de traitement. Une vue d'ensemble de tok_test est disponible à la Figure 2.7.

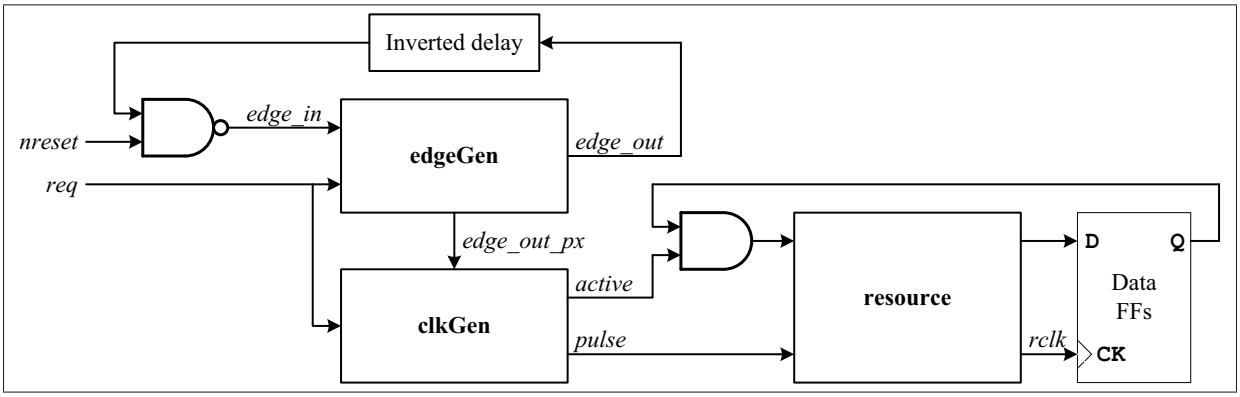

Figure 2.7 Vue d'ensemble du circuit tok_test
Adaptée de Leclerc (2013a)

L'architecture du circuit suit la méthodologie établie en séparant le traitement du jeton, la ressource et l'exécution. Le traitement du jeton est réalisé par les modules *edgeGen* et *clkGen*. L'implémentation de ces modules est détaillée à la Figure 2.8 et à la Figure 2.9 respectivement. L'unité de traitement présentée à la Figure 2.10 est simpliste, car elle effectue uniquement un

décalage à gauche d'une position. De plus, l'unité de traitement est présente dans le module comprenant la ressource qui est en réalité un banc de bascules.

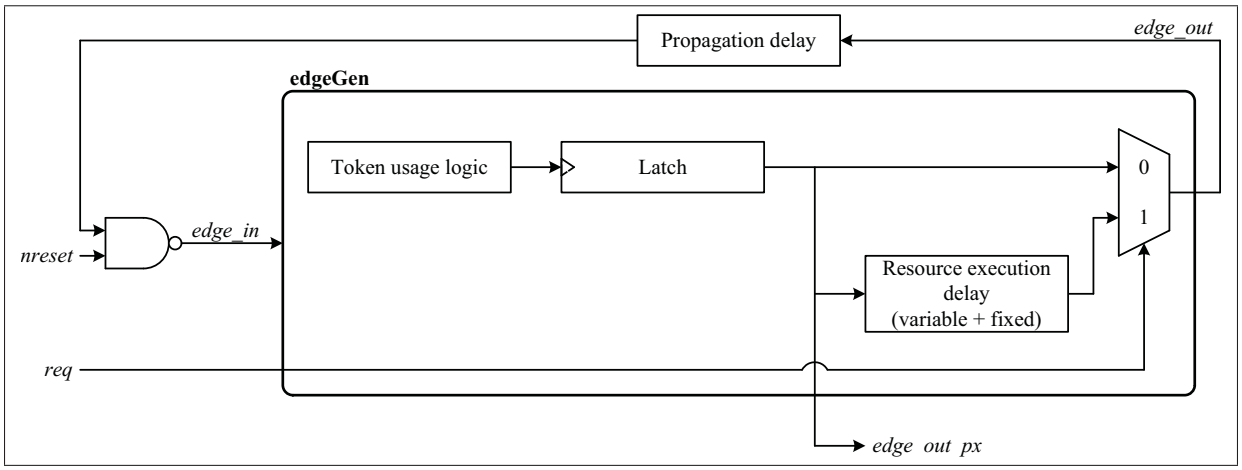

Figure 2.8 Générateur de fronts de tok_test
Adaptée de Leclerc (2013a)

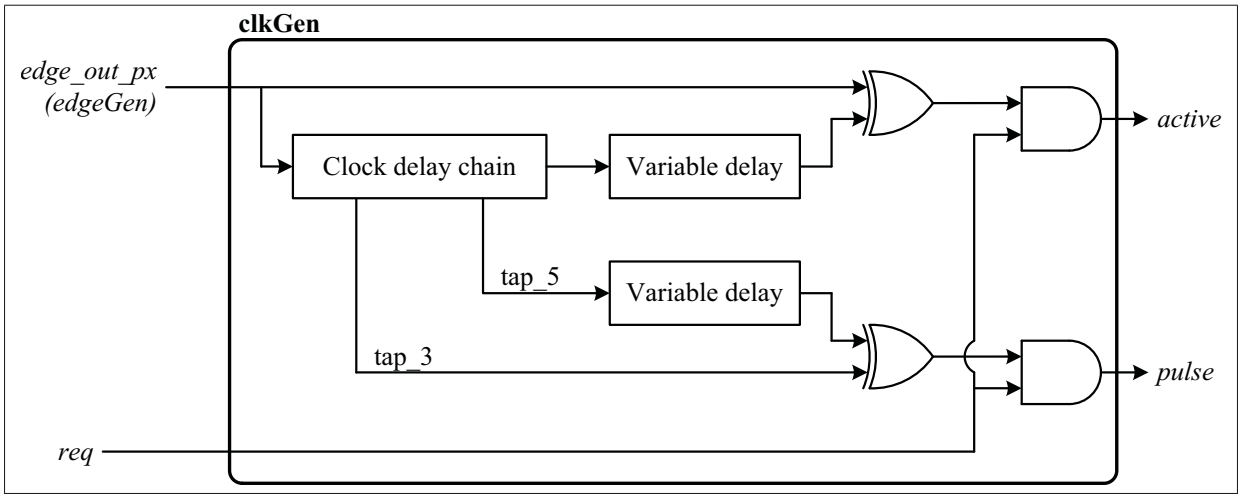

Figure 2.9 Générateur des signaux de contrôle de tok_test
Adaptée de Leclerc (2013a)

Avec la façon dont le jeton et l'accès à la ressource sont séparés du traitement des données, le partitionnement entre les signaux de données et les signaux contrôle devient évident. En effet,

les modules *edgeGen* et *clkGen* comprennent les signaux de contrôle et le module *resource* contient les signaux de données.

Le fonctionnement de tok_test peut être expliqué à partir du jeton. En effet, le jeton évolue dans la boucle créée par les signaux *edge_in* et *edge_out* en passant par le module *edgeGen*. Selon le cas, le jeton peut être utilisé ou non. L'utilisation du jeton provoque un délai supplémentaire entre les signaux *edge_in* et *edge_out* ainsi limitant la vitesse maximale d'opération du circuit. La durée du délai n'est pas variable dans ce cas-ci et est fixée à un temps qui correspond au temps de propagation le plus long parmi les signaux de données du module *resource* en plus du temps de traitement de données. Le jeton cause l'activation du module *clkGen* lorsque la ligne *req* est au niveau logique actif. Rappelons que cette ligne indique au contrôleur d'accès à la ressource qu'une unité de traitement désire utiliser la ressource qu'il gère. Lorsque le contrôleur détermine qu'il peut donner accès à l'unité, les signaux *pulse* et *active* sont pilotés. Toutes les bascules du système sont ensuite mises à jour dans l'ordre prescrit par leurs rôles. Comme l'indique la Figure 2.10, les bascules d'entrées de la ressource, identifiées par *Input FFs* sont rafraîchies en premier. Ensuite, le signal *pulse* se propage au travers du module de délai *Data delay* et cause l'actualisation des bascules de sorties nommées *Output FFs*. Le délai est ajusté afin que le signal d'horloge arrive aux bascules après la donnée la plus lente des chemins de données du nuage combinatoire *Data processing*.

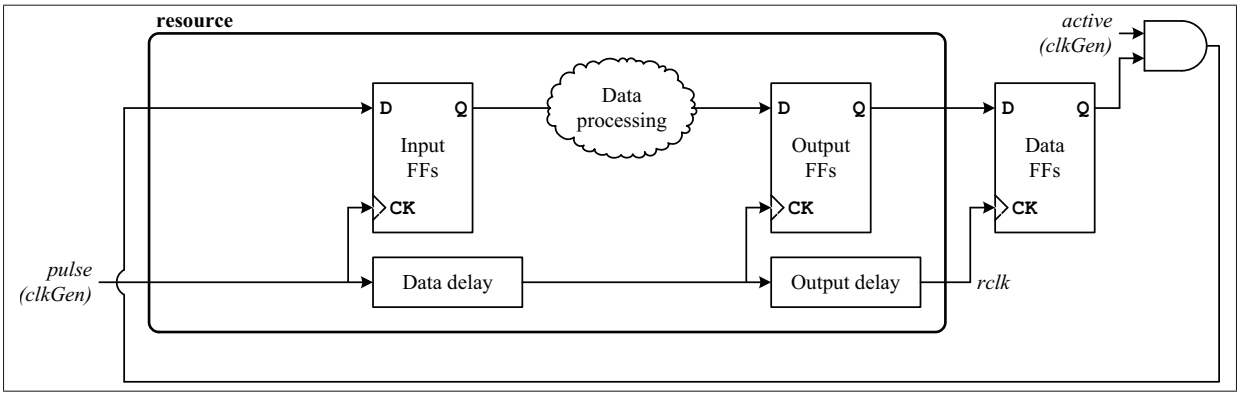

Figure 2.10 Chemin de données de tok_test
Adaptée de Leclerc (2013a)

2.3.2 pico_alu

Le deuxième circuit de test, nommé pico_alu, est beaucoup plus complet que le circuit tok_-test. Effectivement, la Figure 2.11 montre que le circuit est formé de quatre unités de traitement nommées *pico_alu0*, *pico_alu1*, *pico_alu2* et *pico_alu3*. Le système possède aussi trois ressources qui ne sont pas représentées dans la figure. La Figure 2.11 est grandement simplifiée pour mieux illustrer le lien entre les unités de traitement et une ressource. Il y a quatre ensembles de modules, numérotés de 0 à 3. Un pour chaque unité de traitement. Pour alléger la représentation visuelle, seulement les signaux de l'ensemble 0 sont indiqués. Il faut se rappeler que les ensembles 1, 2 et 3 ont les mêmes signaux sauf *pulseN* et *rclkN* qui sont propres à chaque ensemble. De plus, la Figure 2.11 illustre l'interaction entre les modules de traitement des jetons et les unités de traitement pour seulement un jeton et une ressource. Il a un montage similaire pour les deux autres jetons qui commandent des ressources. Seuls les jetons de début et de fin n'ont pas de ressource à proprement parler. Ces jetons sont utilisés seulement dans un but de synchronisation. Deux des ressources permettent de générer une opérande chacune et l'autre indique aux unités de traitement qu'elle est l'opération à effectuer sur les données.

2.3.2.1 Traitement des jetons

Comme pour le circuit de test précédent, le fonctionnement de pico_alu peut être expliqué en utilisant la gestion des jetons. Cinq jetons composent la gestion du système, soit *istart*, *opcode*, *gen_async*, *gen_sync* et *iend*. Il est important de noter qu'il n'y a qu'un seul jeton par ressource et que chaque unité de traitement utilise optionnellement ce jeton. Le jeton est ensuite passé au module suivant. De plus, les unités peuvent utiliser qu'un jeton à la fois. Donc, les jetons peuvent être tous utilisés en même temps dans plusieurs unités différentes. Cependant, un jeton ne pourra jamais devancer un autre, car l'ordre de passage est imposé par les modules de traitement des jetons. À raison d'un module de traitement, ou contrôleur de jeton, par unité de traitement.

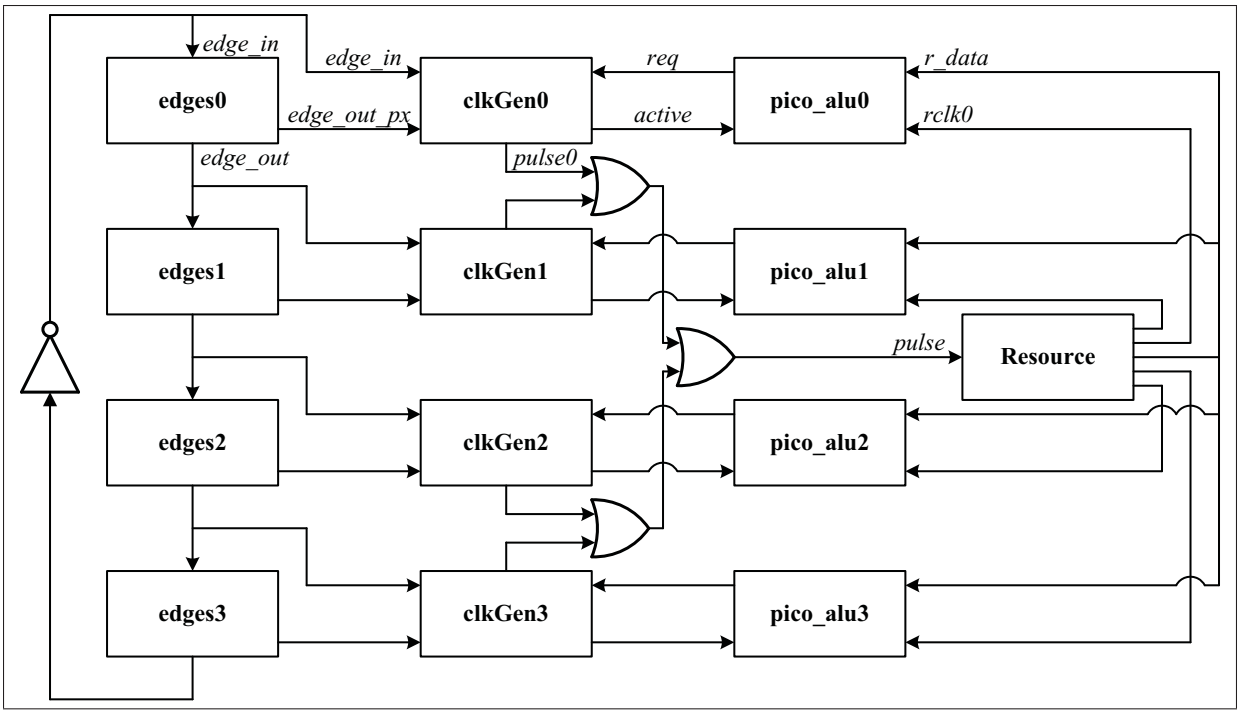

Figure 2.11 Utilisation du circuit pico_alu dans un groupe de quatre unités de traitement

Le jeton *istart* indique le début et le jeton *iend* indique la fin d'une opération. Les jetons *op-code*, *gen_async* et *gen_sync* sont utilisés pour générer une opération, un opérande asynchrone et un opérande synchrone. La ressource qui génère des opérandes de façon synchrone le fait en utilisant une horloge interne au circuit de test. Le circuit pico_alu est un circuit de démonstration, il n'effectue pas de travail efficace. Donc, les opérandes et les opérations à effectuer sur ces opérandes sont générés par des registres à décalage à rétroaction linéaire. Étant donné la nature aléatoire des opérandes et des opérations, la synchronisation entre les domaines synchrone du module et asynchrone du système n'est pas significative. Les liens de dépendance entre les jetons du circuit sont représentés à la Figure 2.12. Seulement deux des trois ressources, soit les opérandes, ont un signal *req* commandé par les unités de traitement. Le générateur d'opérations, quant à lui, a son signal *req* toujours actif.

Le contrôle des accès aux ressources est fait en utilisant la même stratégie que le circuit de test tok_test. Les unités de traitement des jetons sont composées de deux modules, soit *edges* et *clkGen*. Le module *edges* est utilisé pour implémenter le diagramme de dépendance des jetons.

En effet, selon l'utilisation du circuit et l'état des jetons, le module détermine s'il faut passer le jeton ou le garder. Si le module doit garder un jeton, il le fait jusqu'à ce que les conditions de passage soient validées. Le module *clkGen* est utilisé pour générer les signaux de contrôle des ressources si le module *edges* détermine que la ressource peut être utilisée. Les unités de traitement de données indiquent, selon l'opération à réaliser, quelles sont les ressources dont elles ont besoin avec le signal *req*. Une fois que l'accès aux ressources est accordé par les modules composant le contrôleur de jeton, le signal *active* est actif. La ressource est activée avec le signal *pulse* et indique à l'unité de traitement de données qu'elle a terminé de générer les données en activant le signal *rclk*.

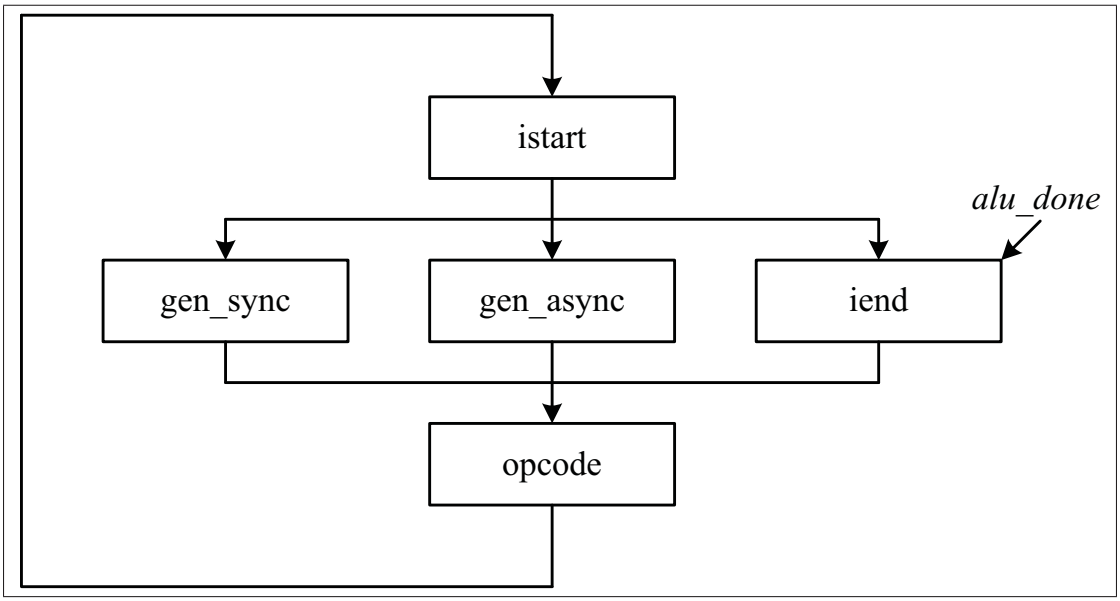

Figure 2.12 Liens de dépendances entre les jetons du circuit pico_alu
Adaptée de Leclerc (2013b)

Bref, une unité de traitement de données indique qu'elle a besoin d'une ressource avec le signal *req*. Ensuite, l'unité de traitement des jetons donne accès à la ressource selon l'état du circuit et des jetons en activant les signaux *active* et *pulse*. Une fois l'accès terminé, la ressource indique que la donnée est prête avec le signal *rclk*.

En observant la Figure 2.11, il est facile d'affirmer que la séparation entre les signaux de contrôle et les signaux de données est claire. Effectivement, les modules de traitement des jetons génèrent les signaux de contrôle et les unités de traitement de données composent les signaux de données.

2.3.2.2 Unité de traitement de données

L'unité de traitement est instanciée quatre fois dans le circuit de test et elle peut elle aussi être séparée selon le chemin de contrôle et le chemin des données. La Figure 2.13 offre une vue d'ensemble du l'unité de traitement.

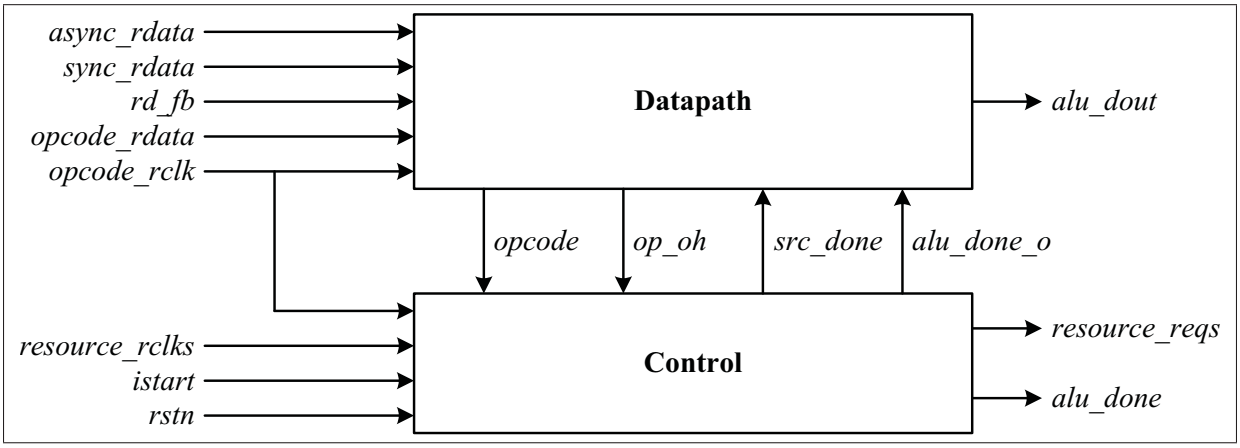

Figure 2.13 Vue d'ensemble de l'unité de traitement du circuit pico_alu

Le fonctionnement de la section de traitement des données, illustrée à la Figure 2.14, se fait en utilisant trois signaux d'horloge. Une première horloge, nommée *opcode_rclk*, est utilisée pour mettre à jour les bascules contenant l'opération à effectuer. Ces bascules sont appelées *opcode FFs*. La deuxième horloge, identifiée par le signal *src_done* actualise les bascules contenant les opérandes. Il existe quatre sources pour les opérandes, soit le générateur sychrone, le générateur asynchrone, une autre unité de traitement ou la constante 0. Le dernier signal de contrôle, *alu_done_o* rafraîchit les bascules de sorties nommées *output FFs*. Ces bascules contiennent le résultat de l'opération effectuée sur les opérandes choisis. Trois opérations sont implémentées

par l'unité de traitement de données, soit l'addition, le décalage vers la gauche et le ou exclusif. L'unité de traitement contient aussi des décodeurs *one-hot* pour la source des opérandes et le type d'opération à effectuer. Le type d'opération et les sources des opérandes sont transmis au module de génération de signaux de contrôle.

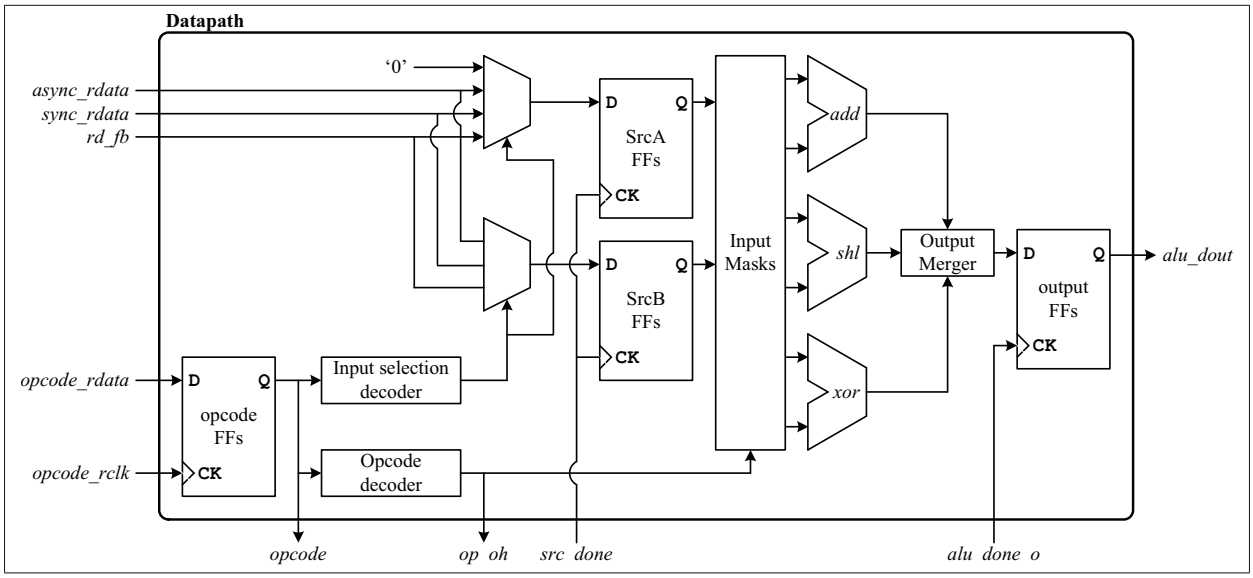

Figure 2.14 Section de traitement des données de l'unité de traitement du circuit pico_alu
Adaptée de Crépeau (2013)

Le module de génération des signaux de contrôle est représenté à la Figure 2.15. L'exécution du module se fait une fois que le jeton *istart* est passé. Le module *Control* fonctionne en ordonnant l'activation des trois signaux d'horloges utilisées par le chemin de données et l'unité de traitement. Premièrement, si l'unité a besoin d'accéder aux générateurs d'opérandes, le module de gestion active la sortie *resource_reqs* branchée au module de traitement des jetons. Bien entendu, il a un signal *req* par générateur d'opérandes. Deuxièmement, lorsque les ressources ont été accédées et que les opérandes sont prêts pour être transférés à l'unité de traitement de données, le générateur active le signal *src_done*. Finalement, en utilisant les informations concernant le type d'opération à effectuer, le contrôleur de signaux choisi quel délai appliquer au signal *src_done* pour générer l'horloge *alu_done_o*. Ce délai permet d'assurer que le signal d'horloge se rendra aux bascules contenant le résultat de l'opération seulement une fois que

l'opération est complétée. La durée dépend de la nature de la fonction à appliquer sur les opérandes. Le dernier signal d'horloge, *alu_done* est en fait un signal qui indique que le traitement est complété et que la prochaine opération peut être entamée.

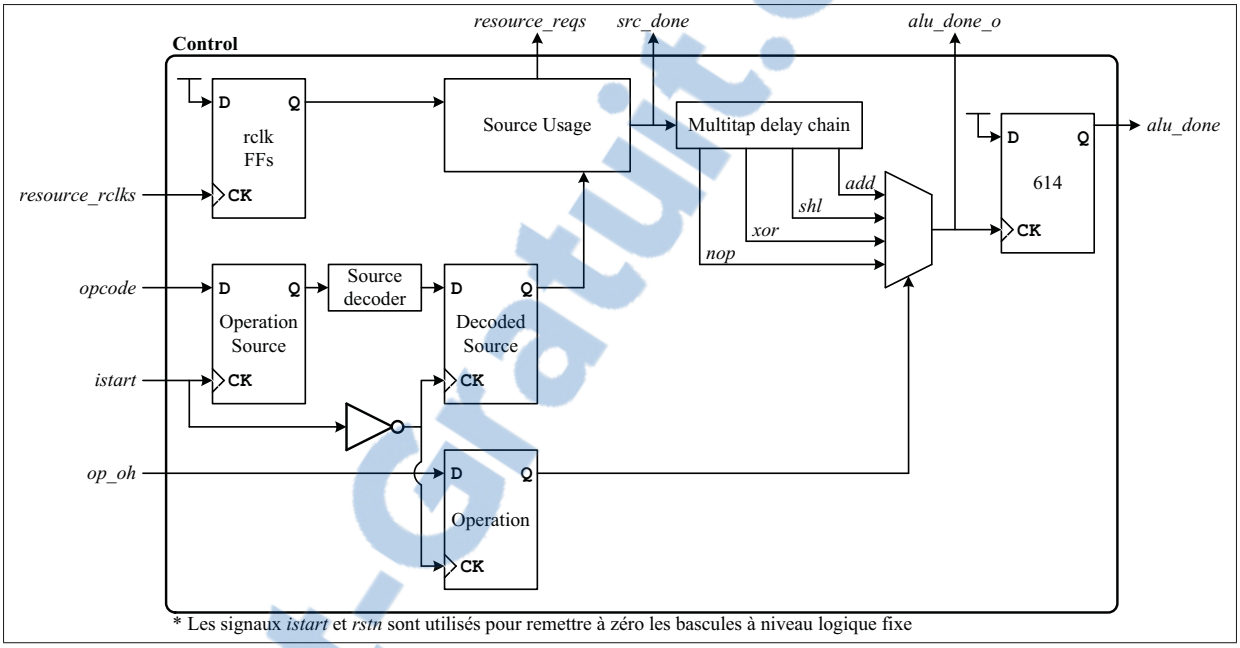

Figure 2.15 Section contrôle de l'unité de traitement de données du circuit pico_alu
Adaptée de Crépeau (2013)

2.4 Structures fréquemment utilisées dans les designs d'Octasic

Dans le but de valider un circuit quelconque, un concepteur doit s'assurer de deux choses. Premièrement, le circuit doit accomplir ce qu'il est censé faire et deuxièmement, le circuit doit le faire en respectant certaines contraintes. Au moins une de ces contraintes est de nature temporelle. En effet, le concepteur doit s'assurer que les chemins d'horloges sont les derniers chemins activés pour une bascule donnée. Sinon, l'intégrité des données stockées dans la bascule n'est pas garantie. Dans le cas des circuits représentant la méthodologie de design utilisée par Octasic, une analyse manuelle des liens entre les chemins de données et les chemins d'horloge a été exécutée.

Avec cette analyse manuelle, certaines structures de conception ont été révélées. Ces structures sont souvent utilisées dans les circuits de tests. Étant donné que ces circuits reflètent bien le type de circuit développé chez Octasic, il est supposé que ces structures se retrouvent aussi dans les circuits de production. Trois structures fréquemment utilisées ont été décelées. Elles se nomment *Configuration A*, *Configuration B* et *Configuration C*.

Ces structures seront exploitées par l'outil de découverte automatique des points de convergence dans le but de classifier ces points. Les points correctement classifiés seront ensuite utilisés par le concepteur du circuit et l'ingénieur de test pour faciliter la conception et la validation. Les sous-sections suivantes présentent les trois configurations.

2.4.1 Configuration A

La configuration A, illustrée à la Figure 2.16, montre un exemple typique d'un étage de pipeline dans un circuit asynchrone. En effet, l'entrée horloge de la bascule source est pilotée par une horloge. Ce même signal d'horloge est retardé puis ensuite utilisé pour piloter l'entrée de la bascule destination. Contrairement à un circuit synchrone normal, la même impulsion d'horloge est utilisée pour lancer le traitement se faisant dans le nuage *Logique comb. et séq. (sauf DFF)* et pour effectuer la sauvegarde du résultat du traitement. Le délai dans le nuage présent sur le chemin d'horloge est ajusté pour être plus long que le délai de propagation le plus long du nuage combinatoire présent sur le chemin des données.

La configuration A est la configuration la plus utilisée dans les circuits de test. Par exemple, les bascules de sorties, identifiées par *Output FFs*, et de données, marquées par la mention *Data FFs*, de la Figure 2.10 sont toutes reconnues comme étant de type A. De plus, les bascules de sorties, nommées *output FFs*, à la Figure 2.14 sont aussi classifiées comme étant de type A.

2.4.2 Configuration B

Une généralisation de la configuration A permet de créer la Configuration B. Cette configuration a la particularité de voir une sortie donnée d'une bascule source piloter une entrée horloge

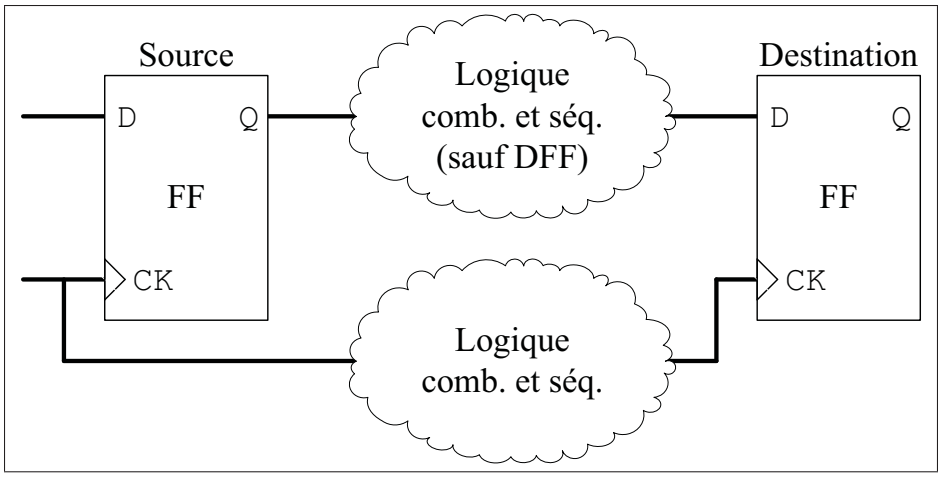

Figure 2.16 Configuration A

d'une bascule de destination. Comme à l'habitude, le signal d'horloge est retardé en fonction du temps de propagation des chemins de données comme il est montré à la Figure 2.17

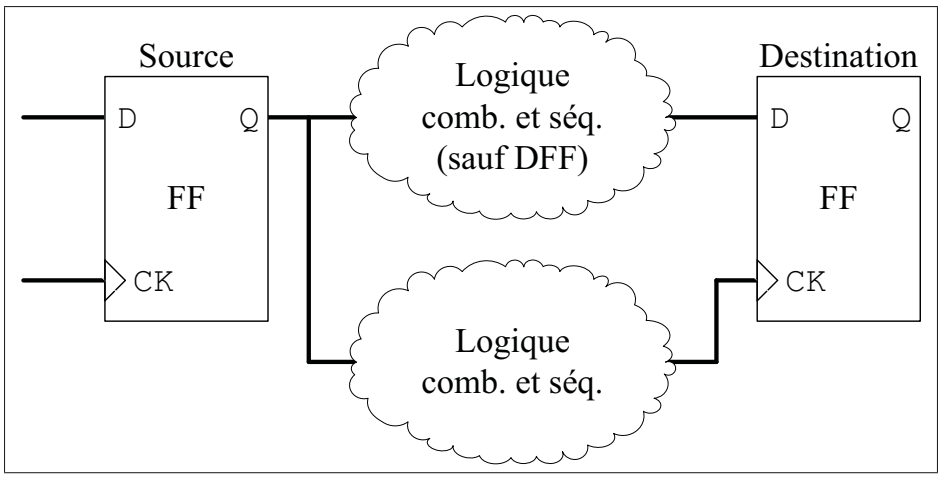

Figure 2.17 Configuration B

Un cas d'utilisation serait qu'un bit d'un banc de bascules source est réservé pour un signal de contrôle de type *rclk* et que ce dernier doit être synchronisé avec d'autres avant de permettre aux bascules de destination d'être rafraîchies. Cette configuration est seulement une généralisation selon la configuration A et ne se retrouve pas dans les deux circuits de tests.

2.4.3 Configuration C

La configuration C est un autre cas fréquent où l'entrée horloge d'une bascule de destination est pilotée par une combinaison de signaux de conditions. Comme l'illustre la Figure 2.18, la configuration C est utilisée pour combiner plusieurs conditions différentes avant de produire un seul signal d'horloge. Il est important de noter que la bascule qui pilote l'entrée horloge de la bascule de destination a son entrée donnée à un niveau logique fixe. L'entrée donnée est maintenue à un niveau fixe par une cellule de type *TIE*. Il existe une cellule *TIEHI* pour le niveau logique haut et *TIELO* pour le niveau logique bas. Cette particularité est utilisée par l'outil de découverte.

Des exemples réels de points d'intérêt de configuration C sont les bascules de sorties de la Figure 2.14 si le signal *alu_done_o* était remplacé par le signal *alu_done*. Ce remplacement ne causerait pas de changement au niveau du fonctionnement du circuit pico_alu. La combinaison est mise à l'œuvre dans le gestionnaire des signaux de contrôle dépeint à la Figure 2.15. Effectivement, le signal *alu_done* est généré une fois que tous les signaux dont il dépend ont été reçus par le module *Control*. Le signal *alu_done* dépend du jeton *istart* en plus de dépendre de tous les signaux *rclk*. La combinaison s'illustre avec les éléments *rclk FFs*, *Source Usage* et *Multitap delay chain* ainsi que les bascules actualisées par le signal d'entrée *istart*. Une autre façon d'illustrer la configuration C est de voir plusieurs configurations A en parallèle où la bascule de destination utilise la combinaison des impulsions d'entrées pour se rafraîchir.

2.5 Conclusion

Dans ce chapitre, il a été question de la méthodologie de design utilisée chez Octasic et de son origine. En effet, une explication du principe de haut niveau des jetons, la méthode de synchronisation développée par Octasic, a été présentée. De plus, une étude du fonctionnement des circuits de tests a été détaillée. Ces études ont permis de comprendre l'application de la méthodologie dans un contexte simple, mais représentatif. Le chapitre se termine par la définition des configurations découvertes suite à l'analyse manuelle des circuits de tests. Les configurations

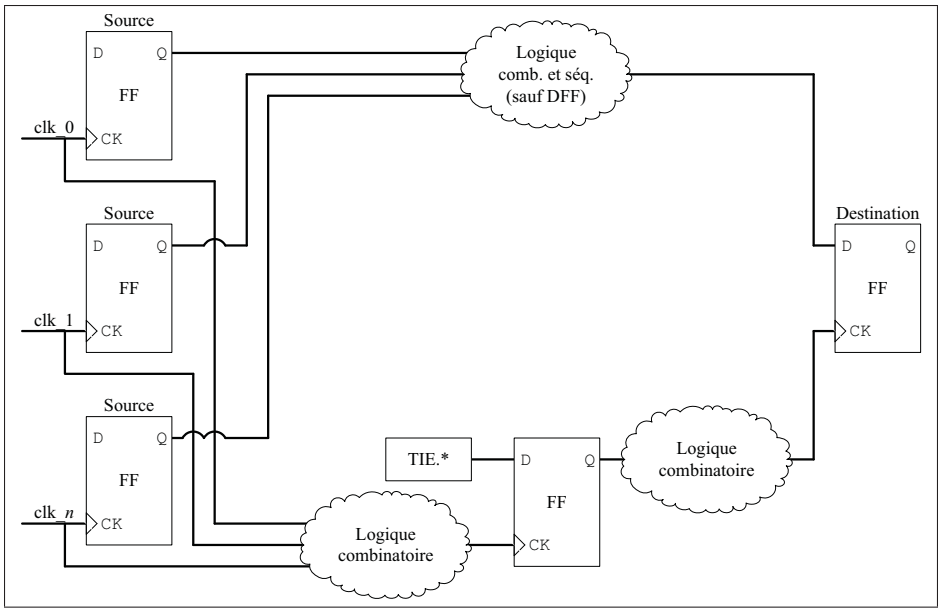

Figure 2.18 Configuration C

sont les structures fréquemment retrouvées dans les circuits développés par Octasic et sont les structures reconnues par l'outil de découverte automatique des points de convergence.

CHAPITRE 3

QMI : OUTIL DE DÉCOUVERTE DES POINTS DE CONVERGENCE

Ce chapitre présente le développement et l'utilisation de l'outil de découverte des points de convergence (PdC). D'abord, les points de convergence sont définis. Ensuite, une explication du fonctionnement de haut-niveau de l'application est fournie. Finalement, le chapitre se termine par une présentation de la configuration de l'outil et des résultats générés.

3.1 Point de convergence

Pour comprendre la notion du point de convergence, il est nécessaire de définir les concepts du point d'intérêt, du chemin de données ainsi que du chemin d'horloge :

Point d'intérêt : élément de type mémoire du circuit ;

Chemin de données : chemin, exploré à rebours, qui débute à l'entrée de type donnée d'un point d'intérêt et qui se termine à la sortie de type donnée d'un autre élément mémoire. Ce chemin peut contenir des *latchs* transparents ;

Chemin d'horloge : chemin, exploré à rebours, débutant à l'entrée de type horloge d'un point d'intérêt et se terminant à la sortie de type donnée d'un élément mémoire dont l'entrée donnée n'est pas à un niveau logique fixe.

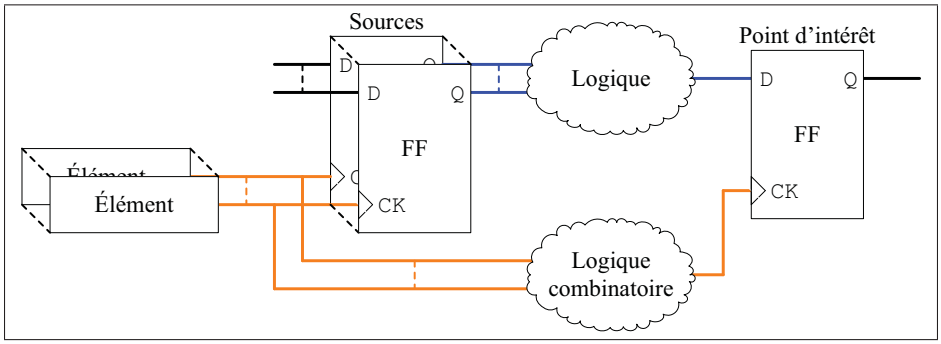

Figure 3.1 Chemins d'horloge (en orange) et de données (en bleu)

Par exemple, les chemins bleus (haut de la figure) de la Figure 3.1 sont les chemins de données. Les chemins orange (bas de la figure) sont les chemins d'horloge.

Un point de convergence, dénoté PdC, est défini comme étant l'endroit dans le circuit où les chemins de données d'un point d'intérêt convergent avec les chemins d'horloge du même point d'intérêt. Ce dernier est trouvé par l'exploration à rebours des chemins de données et des chemins d'horloge à partir d'un point d'intérêt. Il est important de noter que dans un design très uniforme, tous les chemins de données convergeraient avec le même chemin d'horloge au même point.

3.2 Utilisation de qmi

L'outil qmi est développé de toutes pièces en C++. L'élaboration de cet outil a nécessité environ 10000 lignes de codes uniques en excluant les lignes vides et les commentaires. Le flot de conception traditionnel démontré dans la Figure 1.4 peut être adapté et utilisé dans un contexte de conception asynchrone. La Figure 3.2 illustre l'intégration de qmi dans ce flot adapté.

Comme pour la conception synchrone, la conception asynchrone commence par la spécification des requis. Ces requis explicitent le fonctionnement précis de la puce. La réalisation de ces requis génère des fichiers VHDL. Ces derniers sont ensuite synthétisés en une netlist Verilog par un outil interne chez Octasic, soit octgtech. Pour utiliser qmi dans le but d'adapter les outils traditionnels au paradigme asynchrone, il est nécessaire de générer un graphe représentant le circuit. Ce graphe, de format G♭ , est obtenu par l'outil gflat. Avec cette représentation, qmi peut analyser le circuit et générer des fichiers d'aides à l'analyse. Les fichiers PdC peuvent être utilisés par un outil d'analyse temporelle traditionnel ou par l'outil interne d'Octasic, soit octimize. Finalement, le fichier Tcl peut être utilisé par la suite Tessent de Mentor Graphics pour faciliter la génération de patrons de test.

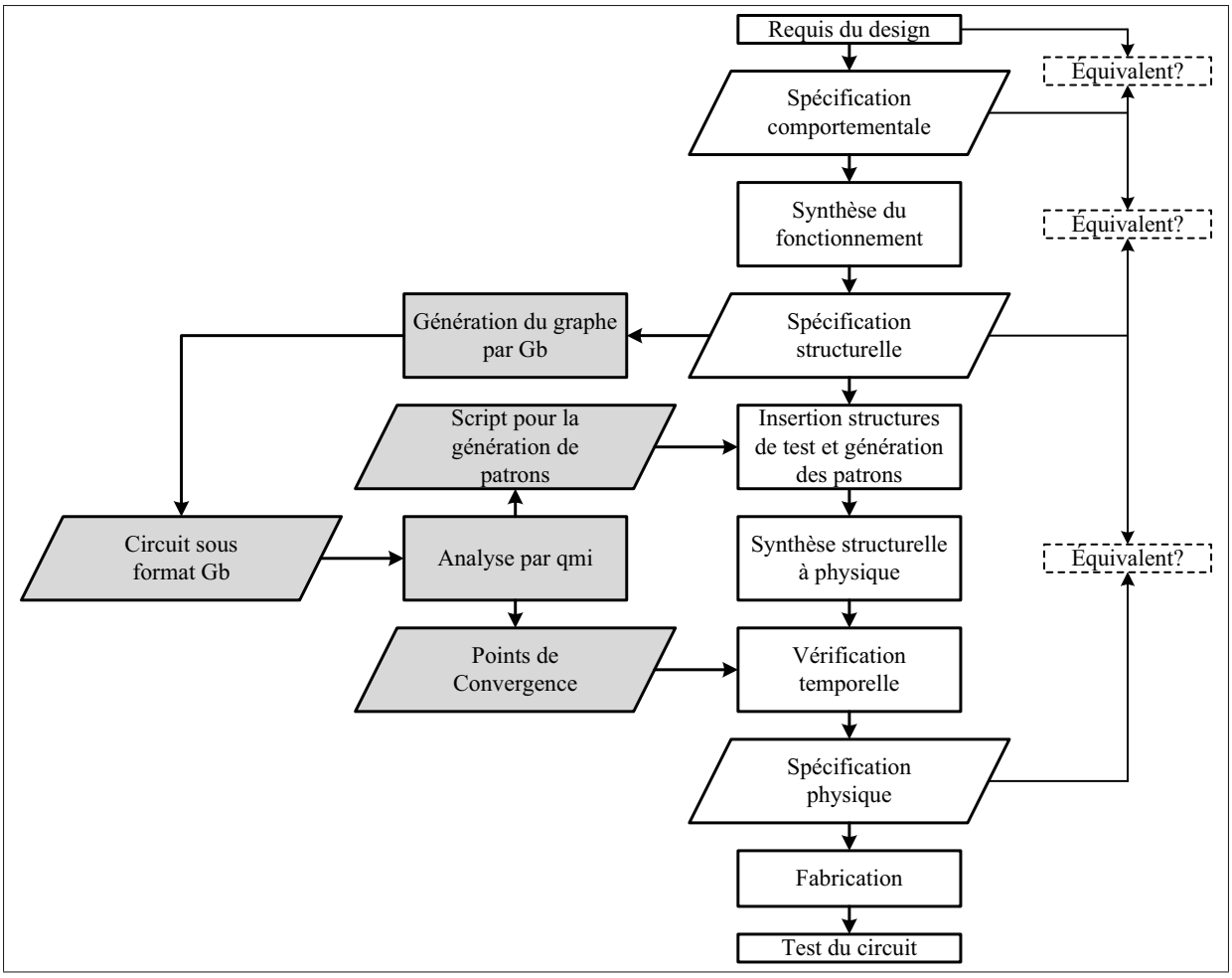

Figure 3.2 Flot de conception avec l'utilisation de qmi noté par le fond gris

3.3 Fonctionnement de haut-niveau

La découverte des PdC permet d'identifier quels sont les chemins de données et les chemins d'horloge pertinents d'un point d'intérêt donné. Cette identification permet aussi de classifier le point d'intérêt selon une des trois configurations établies dans la section 2.4. Ces chemins peuvent ensuite être traités par un outil d'analyse temporel ou utilisé par un outil externe.

Comme schématisé à la Figure 3.3, qmi prend deux types de données en entrée. Soit, les fichiers de configuration et la base de données du circuit à analyser. Les fichiers de configuration offrent la possibilité à l'utilisateur de changer le comportement de l'analyseur de PdC ainsi que d'influencer la génération des résultats. La base de données du circuit est en fait une représen-

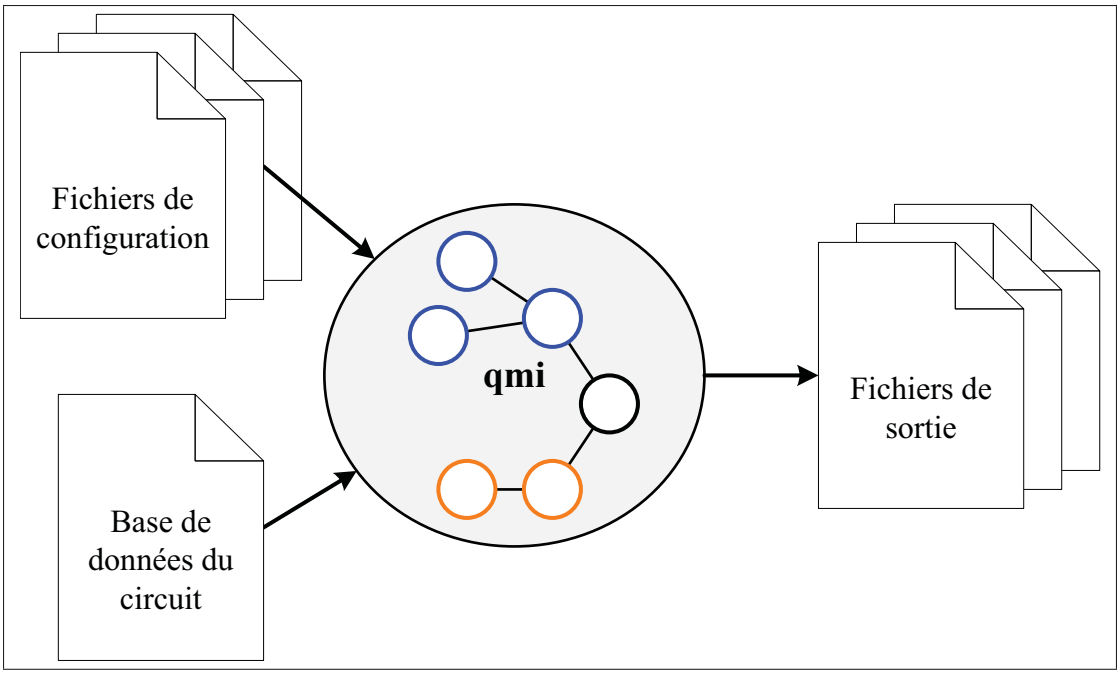

Figure 3.3 Intrants et extrants de qmi

tation sous forme de graphe du circuit où chaque nœud est un élément du circuit et chaque arête est une connexion entre une entrée d'un élément et une sortie d'un autre élément. Une description plus exhaustive du format de graphe utilisé est disponible à l'annexe I. L'outil réalise sa tâche en trois phases. Il y a la phase de configuration, de découverte des PdC et de génération des résultats. La phase de configuration sera traitée dans la section Configuration de l'outil.

Ces trois étapes peuvent être expliquées à l'aide du patron de conception logiciel Modèle-Vue-Contrôleur (MVC). En effet, la Figure 3.4 rend évidents les liens entre les parties de l'application. La phase découverte est réalisée par les modules *analyseur* et *détecteurs*. La phase génération des résultats est, quant à elle, implémentée par les éléments *afficheurs*.

3.3.1 Découverte des points de convergence

Une fois l'outil correctement configuré, il démarre la découverte des points de convergence en établissant une liste de points d'intérêt. Cette liste peut être générée automatiquement par qmi ou bien manuellement par l'utilisateur. Ensuite, pour chaque point d'intérêt, une liste des

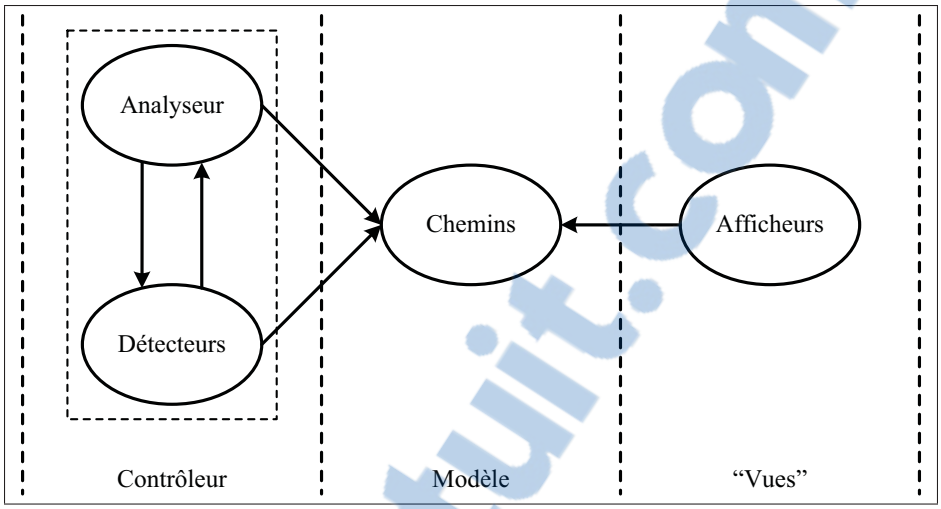

Figure 3.4 Le patron MVC appliqué à qmi

chemins de données est créée à partir des entrées données du point d'intérêt. Conjointement, une liste des chemins d'horloge est élaborée à partir des entrées horloge du point d'intérêt. Ces listes forment le modèle tel qu'identifié par la Figure 3.4. La formation des chemins de données se fait selon l'algorithme 3.1 à la page 60. La liste des chemins d'horloge est créée en utilisant le même algorithme, mais les chemins sont de type *CLOCK* et les conditions d'arrêt sont celles de la sous-section Conditions d'arrêt B. L'algorithme est basé sur la gestion de deux listes. En effet, il y a une liste qui contient les chemins incomplets et une autre qui contient les chemins complets. Au départ, la liste de chemins incomplets est initialisée avec des chemins contenant une seule instance, celle représentant le point d'intérêt. L'exécution se poursuit, itération par itération, tant que la liste de chemins incomplets n'est pas vide. À chaque itération, tous les membres de la liste de chemins incomplets se font ajouter les instances qui pilotent les entrées de l'instance la plus récemment jointe au chemin. Si l'instance annexée correspond à une condition d'arrêt, le chemin est enlevé de la liste de chemins incomplets et ajouté dans la liste de chemins complets.

Ensuite, lorsque les listes des chemins sont complétées, la découverte du PdC et la classification du point d'intérêt peuvent se faire. Avec chaque chemin de données de chaque point d'intérêt, l'outil génère un triplet composé du chemin de données, d'un chemin d'horloge et d'un type.

Le type fait référence aux trois configurations établies soit A, B ou C. Pour ce faire, qmi fait appel aux détecteurs. Si le détecteur de la configuration *N* (où *N = A, B ou C*) identifie un point de convergence entre un chemin de données et un chemin d'horloge, l'outil détermine que ce chemin de données est de type *N*. Sachant qu'il existe trois détecteurs, un pour chaque configuration identifiée, le type du triplet peut être *A*, *B* ou *C*. Cependant, si aucun détecteur n'établit de lien, le type est *uncovered*. De plus, il existe aussi un type spécial, *ignored*, pour le cas où le chemin de données est à un niveau logique fixe et ne nécessite donc pas d'analyse.

La détection du point de convergence et de la configuration se fait selon certaines règles. Rappelons que le chemin de données est dit de type A si l'entrée horloge de l'élément qui débute le chemin est piloté par une instance présente dans un des chemins d'horloge. L'élément du chemin d'horloge devient alors le point de convergence. De plus, une règle similaire est utilisée pour le détecteur de configuration C. Cependant, pour être de type C, le chemin d'horloge qui converge avec le chemin de données doit avoir une cellule mémoire branchée à un niveau logique fixe. Finalement, une configuration B est identifiée si le chemin de données commence par le même élément qu'un des chemins d'horloge.

Après qu'un triplet a été créé pour chaque chemin de données de chaque point d'intérêt, la phase de génération des résultats est exécutée.

3.3.1.1 Conditions d'arrêt A

L'élaboration d'un chemin de donnée est terminée si l'instance à ajouter au chemin est soit :

- un élément mémoire ;
- un élément ayant un niveau logique fixe ;
- le module racine (le *top*) ; ou
- déjà dans le chemin.

3.3.1.2 Conditions d'arrêt B

L'élaboration d'un chemin d'horloge est terminée si l'instance à ajouter au chemin est soit :

- un élément mémoire avec ses entrées données branchées à un niveau logique fixe ;
- un élément ayant un niveau logique fixe ;
- le module racine (le *top*) ; ou
- déjà dans le chemin.

3.3.2 Génération des résultats

L'étape de génération des résultats est implémentée par les éléments *afficheurs* de la Figure 3.4. Les afficheurs spécifiés par l'utilisateur utilisent le modèle et les triplets créés dans la phase découverte pour générer les résultats. Afin d'assurer une grande flexibilité au niveau des formats de sortie, qmi possède cinq afficheurs différents. Ces afficheurs peuvent être classés en deux catégories, soit les afficheurs produisant des résultats pour analyse manuelle et ceux qui génèrent des fichiers utilisables par un autre outil. La structure de qmi permet d'ajouter facilement d'autres générateurs de résultats.

Les trois formats lisibles offrent divers niveaux de détails sur les points de convergence. En effet, le format *Liste* établit une liste exhaustive de tous les nœuds de tous les chemins de données en plus d'indiquer le type de configuration. Le format *Table* offre les mêmes informations, mais sans la liste exhaustive des nœuds. Finalement, le format *Stats* permet à l'utilisateur de savoir le nombre de chemins de données classifiés selon la configuration A, B et C. Il y a effectivement une configuration par chemin de données, donc il y a parfois plus d'une configuration par point d'intérêt. Il est intéressant de noter que qmi peut aussi générer ces résultats dans un format *Comma Seperated Values* (CSV).

Les deux derniers afficheurs permettent à qmi de générer des fichiers pour d'autres outils utilisés dans l'industrie. Le premier afficheur crée un script qui contient toutes les pannes pour tous les points d'intérêt du circuit analysé. Ce fichier indique aussi quelles sont les horloges de test

Algorithme 3.1 Élaboration des chemins de données

```
Début aux entrées données du point d'intérêt;
pour Toutes les entrées données faire
    Créer un chemin de type DATA;
    Ajouter le numéro d'identification du point d'intérêt à la liste d'instance dans le
    chemin;
    Ajouter le chemin à la liste de chemins incomplets;
fin pour
tant que la liste de chemins incomplets n'est pas vide faire
    début Avec l'instance qui source le chemin
        Ajouter un nœud représentant la première entrée au chemin;
        Ajouter le numréo d'indentification de l'instance à la liste d'instance dans le
        chemin;
        si Conditions d'arrêt A alors
            Ajouter le chemin à la liste de chemins complets;
            Enlever le chemin de la liste de chemin incomplets;
        fin si
        si l'instance qui source le chemin est une cellule à niveau fixe alors
            Enlever le chemin de la liste de chemin incomplets;
        fin si
        si l'instance qui source le chemin a plus d'une entrée alors
            pour les entrées autre que la première faire
                Créer une copie du chemin;
                Ajouter un nœud représentant l'entrée au chemin;
                Ajouter le numréo d'indentification de l'instance source à la liste
                d'instance dans la copie;
                Ajouter la copie du chemin de la liste de chemin incomplets;
                si Conditions d'arrêt A alors
                    Ajouter la copie du chemin à la liste de chemins complets;
                    Enlever la copie du chemin à la liste de chemins incomplets;
                fin si
            fin pour
        fin si
    fin
fin tq
Copier la liste de chemins complets vers le modèle;
Remettre à zéro les données d'élaboration;
```

pour la capture et le lancement. Le script peut ensuite être importé dans l'outil Tessent de Mentor Graphics pour former automatiquement des patrons de tests. Le deuxième afficheur produit

un fichier de format DOT. Le fichier est ensuite utilisé par Graphviz pour créer une représentation graphique des points d'intérêt, des chemins de données, des chemins d'horloge ainsi que des PdC. Les chemins de données sont tracés en bleu et les chemins d'horloge en orange. Il existe un code de couleur pour les types de configurations. En effet, le vert chartreuse est utilisé pour marquer les PdC de configuration A, le magenta pour les points de configuration B et le turquoise pour les configuration C. Dans le cas ou la configuration est *unconvered*, les points sont peints en rouge. Par défaut, Graphviz génère un fichier de type *Scalable Vector Graphics*. Ce type offre le choix à l'utilisateur de transformer facilement ce fichier dans un format rastérisé comme *Portable Network Graphics* ou JPG ou vectoriel comme *Post Script* ou PDF.

3.4 Configuration de l'outil

L'application qmi est configurable en utilisant des fichiers textes. Un seul paramètre est passé lors de l'invocation de qmi, soit le chemin vers le fichier principal de configuration. Ce fichier contient des chemins menant à d'autres fichiers qui permettent de paramétrer les parties spécifiques de l'outil. Les sous-sections suivantes détaillent le contenu de tous ces fichiers de configuration.

3.4.1 Fichier de configuration principal

Le fichier de configuration principal, dont le format est représenté à la Figure 3.5, permet à l'utilisateur de spécifier les informations de base.

La liste suivante explicite les paramètres de base ainsi que leur signification :

***graph_file* :** chemin vers le fichier contenant la base de données G♭ du circuit à analyser ;

***gate_file* :** chemin vers le fichier qui décrit les cellules de base ;

***targets_file* :** chemin vers le fichier des points d'intérêt. Si le chemin est *auto*, qmi générera automatiquement la liste de points d'intérêt ;

```
graph_file = <filename>.gflat
gate_file = <filename>.qg
targets_file = auto | <filename>.qt
output_file = cout | <filename>
printer = <printer1>
          .
          .
          .
printer = <printerN>
print_config = <filename>.qp
```

Figure 3.5 Format du fichier de configuration de qmi (extension .conf)

***output_file* :** nom du fichier de sortie. Si le nom est *cout*, la sortie standard de la console est utilisée ;

***printer* :** avec ce paramètre, l'utilisateur peut spécifier quels sont les afficheurs qui doivent être exécutés après la découverte des PdC. Il peut y avoir plus d'un *printer* ;

***printer_config* :** fichier contenant les paramètres des afficheurs.

3.4.2 Description des cellules de base

Étant donné que qmi peut supporter plusieurs ensembles de cellules de base, il faut un moyen pour exprimer le type de ces cellules. L'outil utilise le paramètre *stdlib* pour charger la base de données Gb des cellules de base. Par défaut, toutes les cellules de base sont considérées comme étant du type combinatoire. Le fichier de description permet de changer ce type pour un type spécialisé. L'outil qmi fonctionne avec quatre types spécialisés, soit *mémoire* pour les éléments séquentiels, *délai* pour les unités d'une chaîne à délai, *niveau fixe* pour les cellules branchées à un niveau fixe et *test* pour les éléments de test.

La Figure 3.6 illustre le format du fichier. Les mots clefs *mem*, *de*, *tie* et *ct* sont utilisés pour spécifier le type de la cellule. L'utilisateur fait appel au mot clef *mem* pour spécifier que la cellule est un élément mémoire, *de* pour un élément de délai, *tie* pour une cellule à niveau fixe et *ct* pour une cellule de test. Il existe aussi des mots clefs applicables aux broches des éléments

```
stdlib = <stdlib>.gstd
mem : gateName1
    clk : pinName
    data : pinName
    ignore : pinName
mem : gateName2
    clk : pinName
    data : pinName
    ignore : pinName
        .
        .
        .
mem : gateNameN
de : delayElement1
         .
         .
         .
de : delayElementN
tie : tieElement1
       .
       .
       .
tie : tieElementN
ct : ctGateName1
    ignore : pinName
        .
        .
        .
ct : ctGateNameN
```

Figure 3.6 Format du fichier de description des portes (extension .qg)

de base, soit *clk*, *data* et *ignore*. Certaines broches, comme les broches utilisées seulement en mode test, doivent être ignorées par l'outil d'analyse. Pour les ignorer, le mot clef *ignore* suivi du nom de la broche est utilisé. Le même concept est mis à l'œuvre pour identifier les broches de type donnée avec le mot clef *data* et les broches de type horloge avec le mot clef *clk*.

3.4.3 Points d'intérêt

La Figure 3.7 met en lumière le format du fichier utilisé pour indiquer les points d'intérêt. Le fichier contient une ligne par point d'intérêt et cette dernière contient le nom complet de l'instance à analyser.

```
InstanceName1
InstanceName2
InstanceName3
      .
      .
      .
InstanceNameN
```

Figure 3.7 Format du fichier de la liste des points d'intérêts (extension .qt)

3.4.4 Configuration des afficheurs

Le format du fichier de configuration des afficheurs, représenté à la Figure 3.8, est de forme $clef = valeur$.

```
key1 = value1
key2 = value2
key3 = value3
      .
      .
      .
keyN = valueN
```

Figure 3.8 Format du fichier de configuration des afficheurs (extension .qp)

Les différents afficheurs utilisent le même fichier, mais avec des clefs qui leur sont propres pour se configurer. Par exemple, les trois afficheurs créant des résultats lisibles ont tous une option sous le nom de <*afficheur*>*_type* qui permet de changer la sortie pour un format CSV.

L'afficheur pour Graphviz a une option *graphviz_style* permettant de générer soit un fichier par point d'intérêt ou un seul fichier pour tous les points. Le générateur de script pour Tessent possède une option *tessent_output* qui lui indique de créer un fichier de sortie qui n'est pas celui spécifié dans la configuration principale.

3.5 Conclusion

Ce chapitre a permis de mieux définir les points de convergence en plus de fournir une explication de haut-niveau de qmi, l'analyseur de PdC. En effet, le fonctionnement général en trois phases, soit la configuration, la découverte et la génération des résultats, a été présenté selon le modèle MVC. Ensuite, il a été question de la méthode de configuration simple à base de fichiers texte. Finalement, la nature et l'utilité des résultats générés par les cinq afficheurs ont été explicitées. Une explication détaillée de la réalisation de l'outil est disponible dans l'annexe II. Dans le chapitre suivant, les résultats des analyses des deux circuits de test seront présentés et analysés.

CHAPITRE 4

COUVERTURE ET POINTS DE CONVERGENCE

Ce chapitre traite de la couverture de qmi en plus de donner les résultats détaillés et les analyses faites sur les deux circuits de tests décrits au chapitre 2. Suite au dévoilement et à la validation des résultats, une analyse portant sur le taux de couverture et les représentations graphiques des PdC est présentée. En dernier lieu, une discussion sur la limitation des résultats termine ce chapitre.

Cette section présente dans un premier temps les résultats qui ont servi à valider l'outil développé. Par la suite, on retrouve les résultats les plus probants de l'analyse des deux circuits de tests. Notons qu'une version plus complète des résultats est disponible dans l'annexe III.

4.1 Validation de qmi

La validation de l'outil s'est faite en deux phases. La première phase de validation a été faite par échantillonnage. L'échantillonnage a été exécuté sur l'analyse des chemins de données des bascules de sortie du circuit de test pico_alu. Ce circuit, plus complet, est plus représentatif pour confirmer ou infirmer la sortie de qmi. La méthode de validation utilisée ici est manuelle et laborieuse. En effet, certains chemins de données ont été choisis aléatoirement et leurs points de convergence ont été déterminés manuellement. Ces points de convergence ont ensuite été comparés aux points trouvés par l'outil afin d'assurer la validité des résultats générés. Sachant qu'il y a un total de 1536 chemins dont 51 chemins de type *A* et 102 chemins de type *uncovered* analysés, ceci correspond à un taux d'échantillonnage de 10%. Il est important de noter qu'étant donné que la distribution des types est connue, l'échantillonnage a été ajusté pour obtenir 10% de tous les chemins de type *A* ainsi que 10% de tous les chemins de type *uncovered*.

Un problème que peut poser l'utilisation de ces circuits de tests est qu'ils manquent la version hypothétique de la configuration A, soit la configuration B. En effet, les structures les plus fréquentes se classifient en configuration A et il y a quelques configurations C. Pour assurer

une détection adéquate de tous les types de configurations, une seconde phase de validation a été jugée nécessaire. Le circuit de génération de parité représenté à la Figure 4.1 a été analysé par qmi. En utilisant les bascules *ConfA*, *ConfB* et *ConfC* comme points d'intérêt, l'outil a su identifier correctement la configuration des chemins de données provenant de ces bascules.

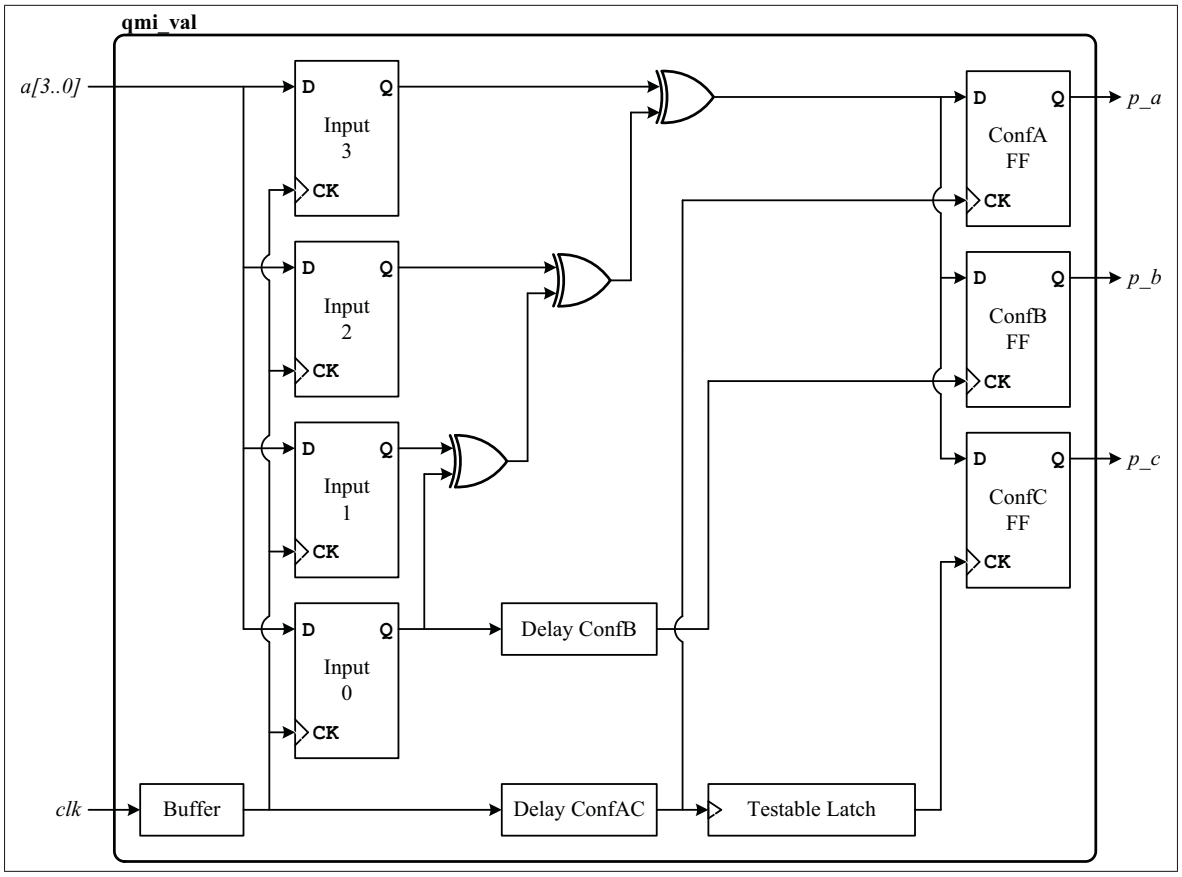

Figure 4.1 Circuit de génération de parité

Les résultats obtenus dans les deux phases de validation étaient probants, l'outil qmi a été utilisé pour analyser les deux circuits de test.

4.2 Résultats et analyses

Les résultats ont été obtenus avec l'outil de découverte automatique des points de convergence, qmi. Pour faciliter la mesure du temps d'exécution, l'outil a été exécuté de telle sorte qu'il

analyse seulement un groupe de points d'intérêt à la fois. Sachant que les points d'intérêt sont effectivement des bascules, les groupes de points d'intérêt sont définis selon le contenu des bascules. L'ordinateur utilisé pour exécuter l'outil qmi possède un processeur Xeon E7-8870 d'Intel cadencé à 2,40 GHz ainsi que 1 To de mémoire vive. Le système d'exploitation est Scientific Linux 5.8 avec le noyau 2.6.18-371.1.2.el5. L'outil qmi est compilé avec gcc 4.1.2 en utilisant le paramètre d'optimisation *-O0*. L'utilisation d'un paramètre supérieur à *-O0* permettrait d'obtenir de meilleurs performances. Cependant, le paramètre *-O0* a été utilisé pour établir les performances de références.

Les circuits utilisés pour illustrer le fonctionnement de qmi sont très simples par rapport aux circuits réellement implémentés. Certes, ces circuits sont petits, mais ils contiennent toutes les structures utilisées dans les circuits de production. Ainsi, ces circuits sont représentatifs et le fonctionnement de l'outil sera le même sur un circuit plus complet. L'analyse des résultats tente de démontrer que qmi répond aux besoins explicités dans la section Introduction. La première partie couvre les résultats et leur analyse pour le circuit tok_test et la deuxième traite le circuit pico_alu.

Pour mesurer la classification des chemins de données des points d'intérêt par l'outil, le taux de couverture (TC) défini par l'équation 4.1 est utilisé.

$$TC = \frac{\textit{Nombre de chemins de données classifiés}}{\textit{Nombre de chemins de données}} \tag{4.1}$$

4.2.1 Circuit tok_test

Rappelons que le circuit de test tok_test est illustré aux figures 2.8, 2.9 et 2.10. Son architecture permet une analyse manuelle aisée tout en conservant une assez bonne représentativité. Les résultats sont séparés selon la nature des points d'intérêts. En effet, il y a trois catégories de points d'intérêt et la nature des points d'intérêt dépend de la nature des données que contiennent les bascules. Donc, il existe un groupe de points d'intérêt pour les bascules conte-

nant les données, un groupe pour les bascules contenant les entrées et un dernier groupe pour les bascules contenant les sorties.

La Figure 4.2 illustre le point de convergence identifié pour les quatre bascules de la catégorie données. La correspondance entre les figures 2.8, 2.9 et 2.10 et les figures suivantes se fait en utilisant le nom des instances. En effet, toutes les instances qui débutent par */u1/* proviennent du module de génération des signaux de contrôle de la Figure 2.9. Les instances dont le nom débute par */u0/* proviennent du module de génération de front de la Figure 2.8. Finalement, les instances débutant par */u2/* se trouvent dans le chemin de donnée de la Figure 2.10. Étant donné la complexité non négligeable du circuit relié aux entrées des bascules d'entrées, la représentation graphique des PdC est scindée en quatre figures. Soit, la Figure 4.3, la Figure 4.4, la Figure 4.5 et la Figure 4.6. Finalement, la Figure 4.7 donne le point de convergence des bascules de sorties.

4.2.1.1 Représentation graphique

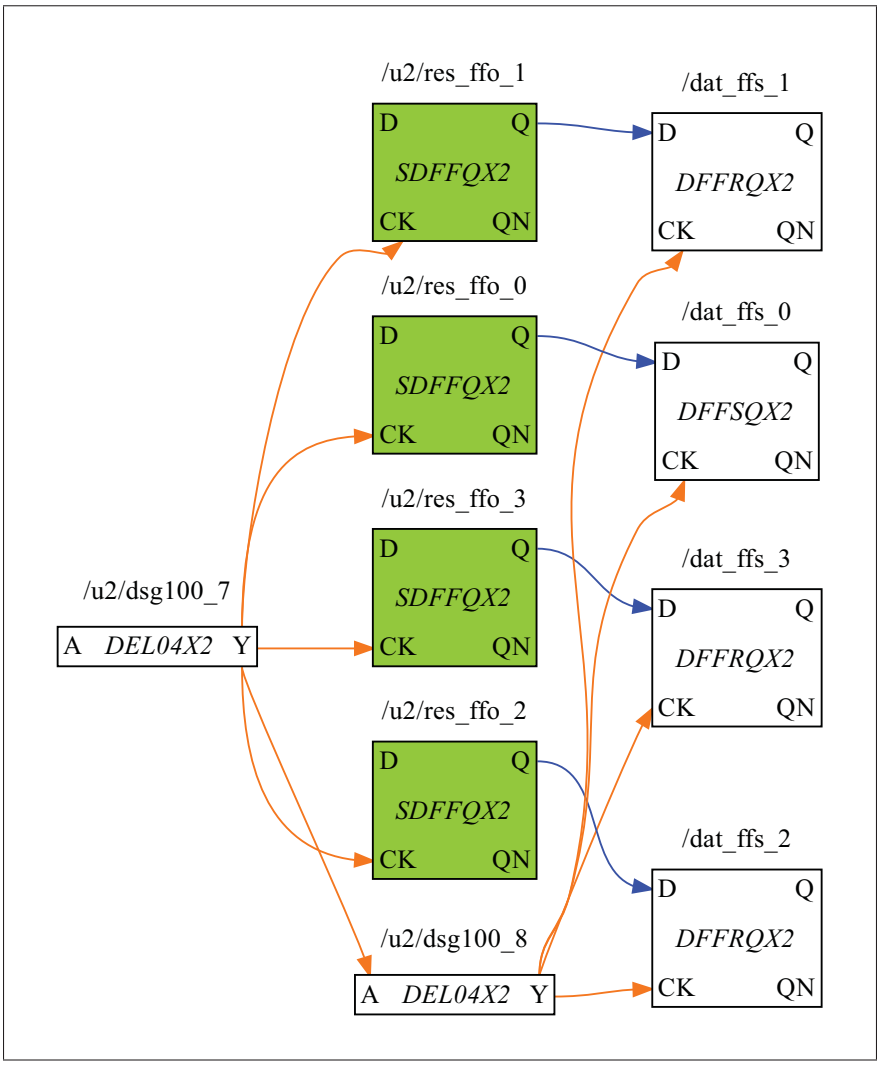

Figure 4.2 Point de convergence (l'élément /u2/dsg100_7) des bascules données du circuit tok_test

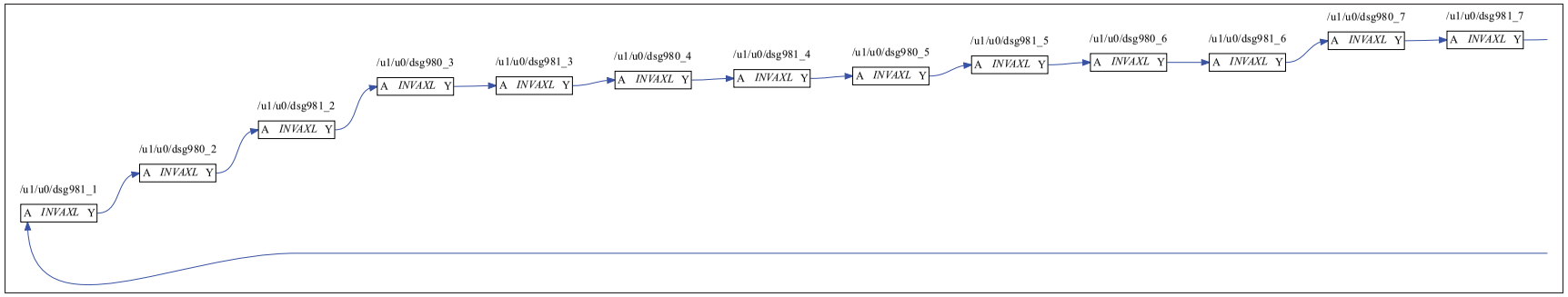

Figure 4.3 Point de convergence (point identifié sur (4)) des bascules d'entrées du circuit tok_test (1)

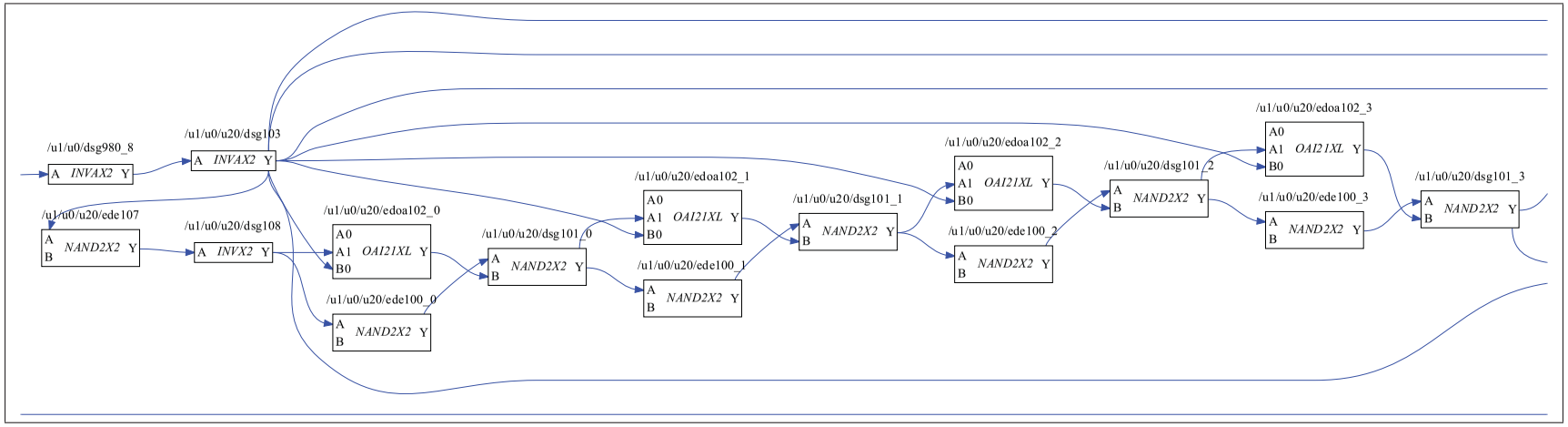

Figure 4.4 Point de convergence (point identifié sur (4)) des bascules d'entrées du circuit tok_test (2)

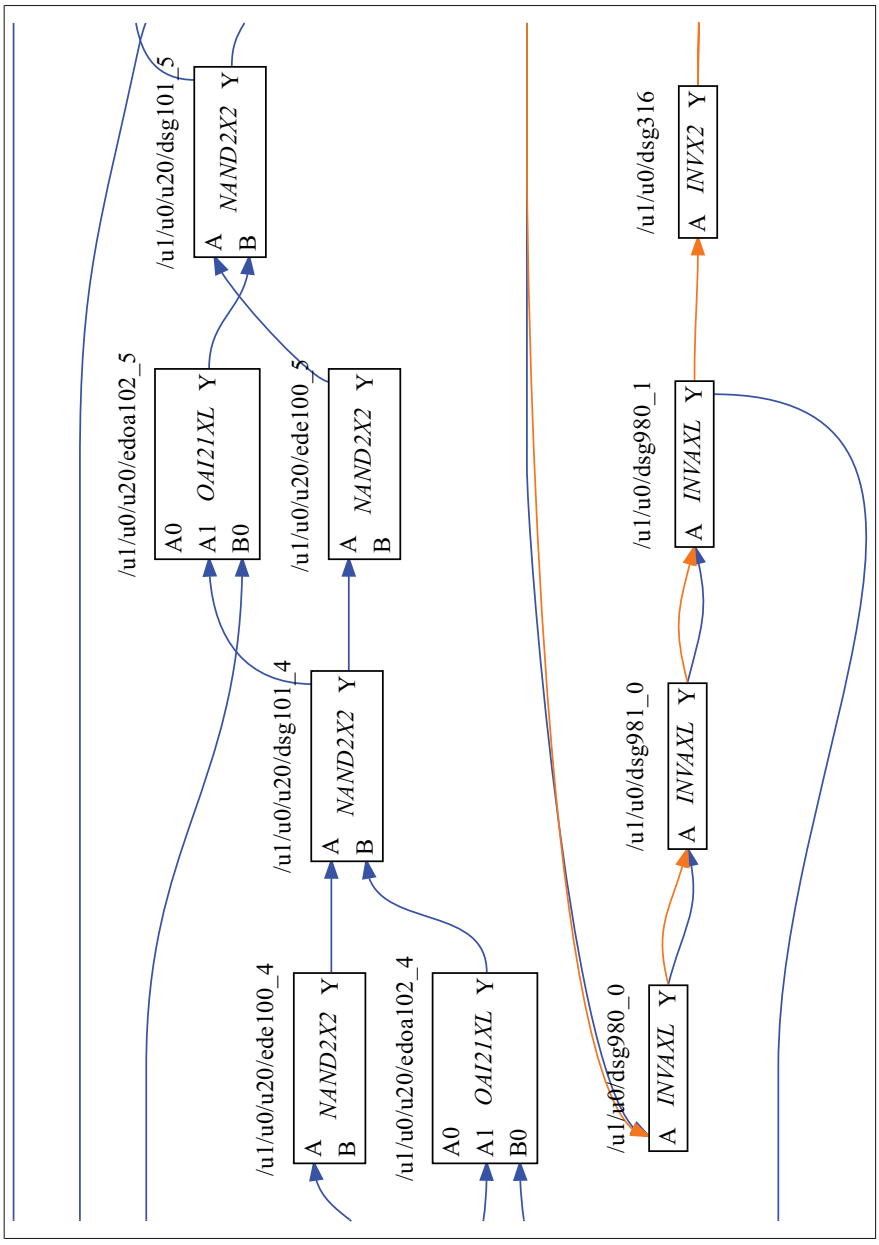

Figure 4.5 Point de convergence (point identifié sur (4)) des bascules d'entrées du circuit tok_test (3)

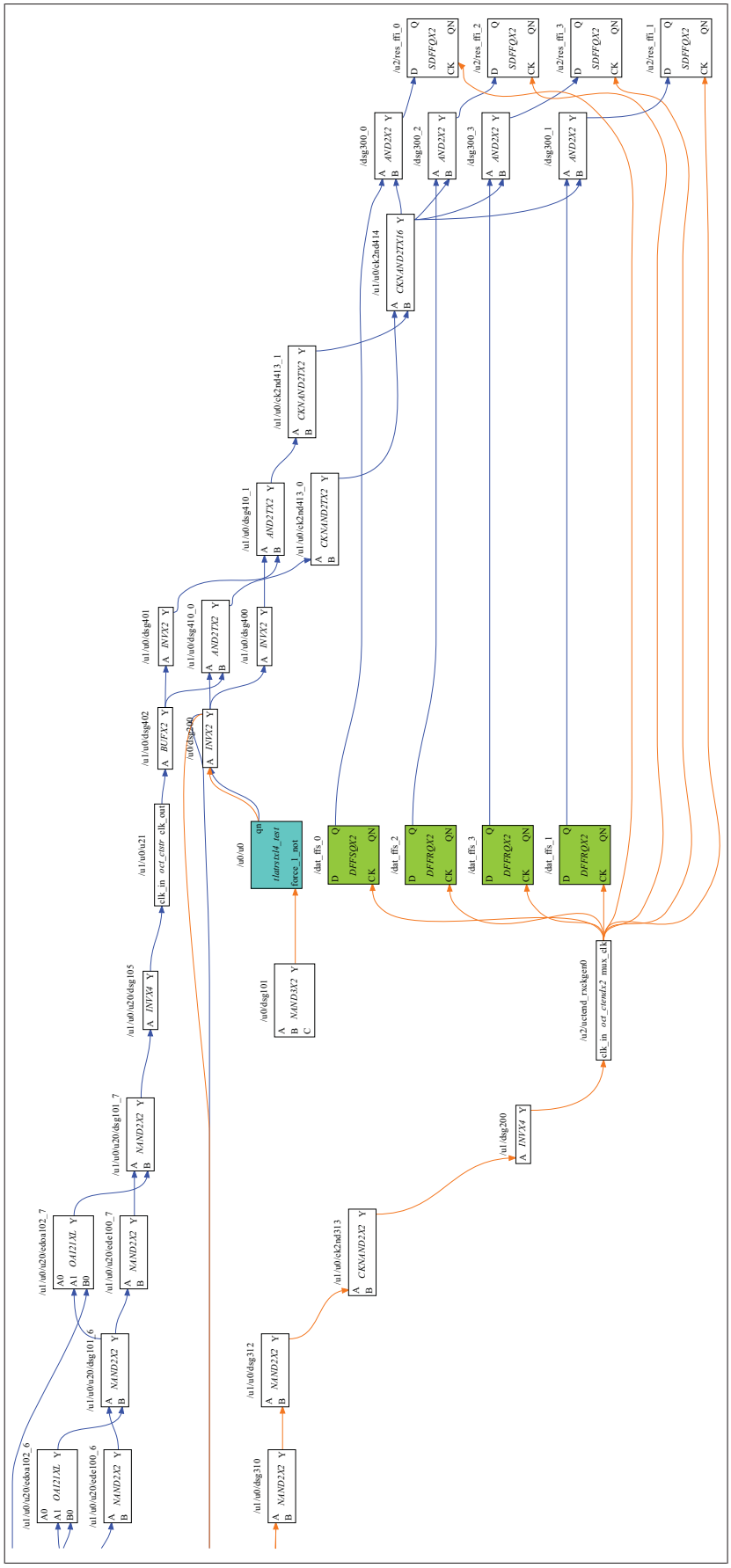

Figure 4.6 Point de convergence (l'élément /u2/uctend_rxckgen0) des bascules d'entrées du circuit tok_test (4)

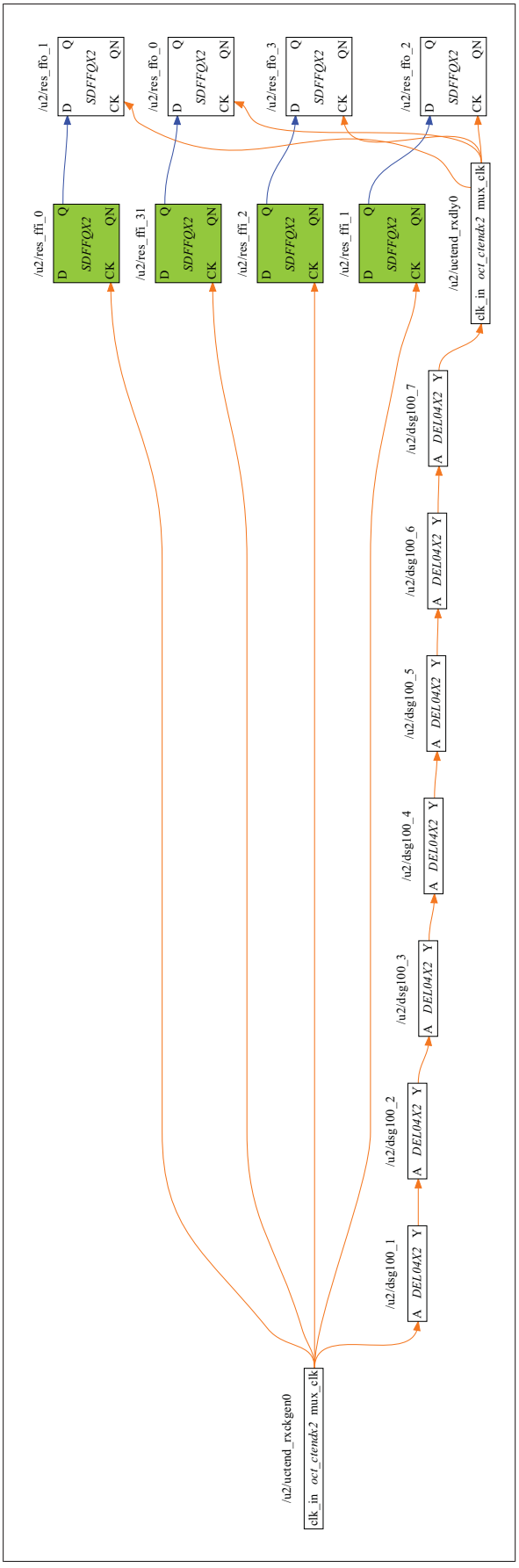

Figure 4.7 Point de convergence (l'élément /u2/uctend_rxckgen0) des bascules de sorties du circuit tok_test

4.2.1.2 Couverture et temps d'exécution

La présentation de la couverture se fait comme la représentation graphique, soit séparée en trois catégories. Les tableaux 4.1, 4.2 et 4.3 contiennent les résultats pour tous les points d'intérêts. Étant donné qu'un point d'intérêt peut avoir plusieurs chemins de données distincts menant à son entrée donnée, les configurations sont associées aux chemins de données et non au point d'intérêt lui-même. Les temps d'exécutions (TE) sont mesurés par GNU time 1.7 et le résultat rapporté est le temps d'exécution réel.

Pour le groupe des bascules contenant les données, qmi a mis 3,3 heures de temps de calcul pour obtenir la classification des 32 chemins de données.

Tableau 4.1 Configurations identifiées pour les bascules contenant les données du circuit tok_test

Points d'intérêt	32
Configuration A	32
Configuration B	0
Configuration C	0
Uncovered	0
Total	32

Le groupe des bascules contenant les sorties possède 32800 chemins de données différents et qmi a nécessité 12 heures de calculs pour les classifier.

Tableau 4.2 Configurations identifiées pour les bascules contenant les entrées du circuit tok_test

Points d'intérêt	32
Configuration A	32
Configuration B	0
Configuration C	32768
Uncovered	0
Total	32800

Le groupe des bascules contenant les valeurs de sorties a une structure très simple et a seulement nécessité 0,05 seconde de calcul.

Tableau 4.3 Configurations identifiées pour les bascules contenant les sorties du circuit tok_test

Points d'intérêt	32
Configuration A	32
Configuration B	0
Configuration C	0
Uncovered	0
Total	32

4.2.1.3 Analyse des résultats

En observant le Tableau 4.1 pour le premier groupe de points d'intérêt composé des bascules données, il est facile de déterminer un TC de 100%. En effet, l'outil a réussi à classifier tous les chemins de données de tous les points d'intérêts, il n'y a pas de chemins dans la catégorie *Uncovered*. La Figure 4.2, qui contient quatre points d'intérêts sur 32, illustre clairement les liens entre les chemins de données, le chemin d'horloge et le PdC. Effectivement, il est évident que tous les chemins de données, en bleu, convergent avec le même chemin d'horloge, en orange, au même endroit. Ce point de convergence est la sortie d'un élément délai qui pilote à la fois les bascules vert chartreuse et une ligne à délai composée d'un seul élément. La sortie de cette chaîne est utilisée pour piloter l'entrée horloge des points d'intérêts. Étant donné la structure reconnue par l'outil, les bascules sont représentées en vert chartreuse pour indiquer que ces points sont de type A. La comparaison du troisième groupe, représenté par le Tableau 4.3 et la Figure 4.7, avec le premier permet d'établir le même constat. La seule différence dans ce cas est la longueur de la ligne à délai composée cette fois-ci de sept éléments.

Le deuxième groupe, composé des bascules d'entrées, est plus particulier que les deux autres groupes. En effet, il existe une relation spéciale entre les chemins de données et les chemins d'horloge. La Figure 2.10 illustre bien les liens entre les bascules contenant les données, iden-

tifiées par le nom *Data FFs* et les bascules contenant les entrées, identifiées par *Input FFs*. Les entrées données des bascules d'entrées sont clairement branchées aux sorties des bascules contenant les données. Cependant, la course entre les données et l'horloge n'est pas au sein d'un même front, mais bien entre deux fronts. En effet, le signal *pulse* ne peut pas changer d'état plus rapidement que le temps de propagation causé par les éléments *Data delay* et *Output delay*. L'ensemble des figures pour le groupe des bascules d'entrées, soit les figures 4.3, 4.4, 4.5 et 4.6, ont une allure peu commune due aux liens entre les bascules nommées *Data* et les bascules nommées *Input*. Il y a d'abord le chemin de données qui débute par l'instance nommée /u0/u0, peint en turquoise, qui est identifié comme étant de configuration C. Cette identification est causée par le fait que les chemins de données des points d'intérêts sont masqués par le signal *active* qui indique à l'utilisateur d'une ressource que ce dernier peut l'utiliser. L'observateur des figures peut s'apercevoir que la course entre les chemins de données et les chemins d'horloge représentée à la Figure 4.6 est en fait une course de nature synchrone, car il n'y a pas de logique combinatoire dans le chemin d'horloge. L'entrée *pulse* du module représenté à la Figure 2.10 est le point de convergence identifié par l'outil. Le lien entre les entrées de type données des bascules *Input FFs* et la sortie donnée des bascules *Data FFs* est à sens inverse par rapport au lien du chemin d'horloge. La constatation de la course synchrone et l'observation du lien entre les bascules *Data* et *Input* indiquent que cette course doit être validée par le concepteur. Même si la course n'est pas asynchrone, l'outil trouve tout de même un point pour chaque chemin de données. Ces points sont effectivement des faux points au niveau de l'analyse faite par qmi, car la dépendance temporelle entre les chemins de données et d'horloge se fait entre deux fronts. Donc, ils nécessitent une attention particulière de la part du concepteur et doivent être analysés par un outil d'analyse temporelle traditionnel.

4.2.2 Circuit pico_alu

L'architecture du circuit pico_alu est représentée aux figures 2.13, 2.14 et 2.15. Contrairement au circuit précédent, le niveau de complexité ne permet pas une analyse manuelle simple. Comme pour le circuit tok_test, la présentation des résultats se fait selon la nature des points

d'intérêt. Effectivement, les bascules de pico_alu se catégorisent bien en cinq types, encore une fois selon la nature des données qu'elles contiennent. Il y a une classe de résultats pour les entrées données, une pour les entrées instruction, une pour les sorties et une dernière pour les bascules contenant le décodage du type d'instruction. Il existe aussi un type pour les bascules contenant le décodage des sources de l'opération. La représentation graphique des points de convergence présentée dans cette section se fait sur un sous-ensemble d'un point d'intérêt sur huit.

4.2.2.1 Représentation graphique

Il est important de noter qu'au moment de la rédaction, l'outil n'avait pas terminé son exécution pour trouver les PdC des deux premiers types. Soit, le groupe des bascules contenant les entrées données et le groupe des bascules contenant l'instruction. Donc, seuls les résultats pour le groupe des bascules contenant les sorties, le type d'instruction et le décodage de la source sont présentés.

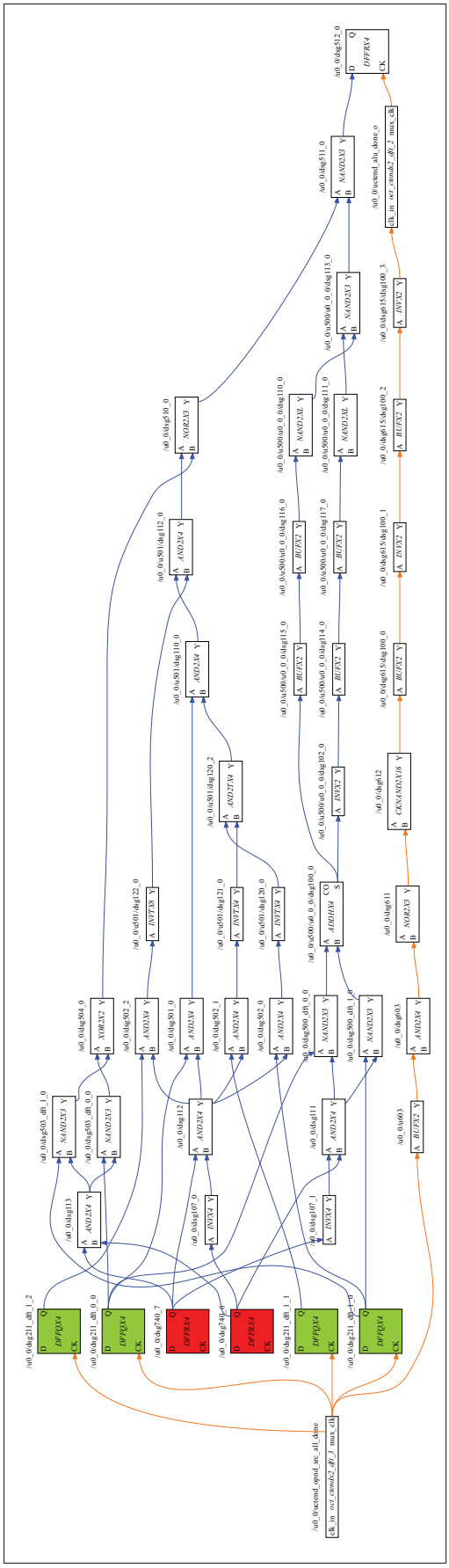

Figure 4.8 Analyse et PdC (/u0_0/uctend_opnd_src_all_done) des bascules de sorties du circuit pico_alu

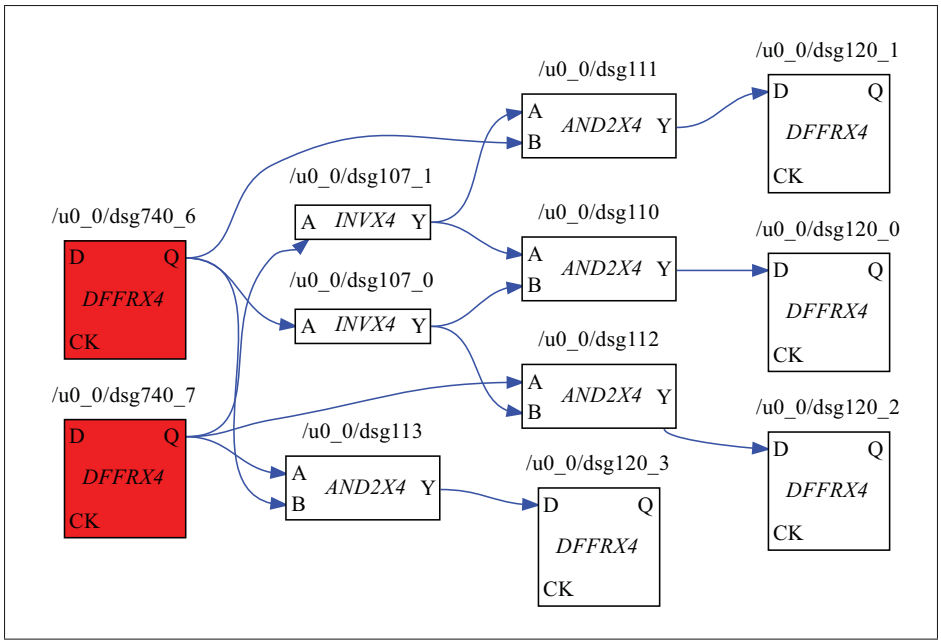

Figure 4.9 Analyse des bascules du type d'instruction du circuit pico_alu

4.2.2.2 Couverture et temps d'exécution

Comme pour le premier circuit de test, les tableaux de couverture sont calculés en fonction de tous les points d'intérêt et séparés par groupe de bascules. Les résultats présentés dans le Tableau 4.4 et le Tableau 4.5 vont exiger au moins 40 jours de calcul chacun. Des circuits équivalents synchrones traités par un outil traditionnel nécessiteraient quelques minutes de traitement. Effectivement, au moment de la rédaction, l'outil n'a pas terminé son exécution. Selon une application manuelle des règles d'élaboration des chemins et de comparaison des chemins, l'outil devrait éventuellement trouver les résultats présentés dans le Tableau 4.4 et le Tableau 4.5. Le groupe des bascules contenant les sorties a, quant à lui, eu besoin de 0,68 seconde de calculs. L'écart entre les temps d'exécution de l'outil est causé par l'algorithme qu'il utilise pour élaborer les chemins d'horloge. Les deux autres groupes, soit ceux des bascules contenant le type d'opération et le décodage des sources des opérandes, ont requis 0,03 et 0,05 seconde de traitement, respectivement.

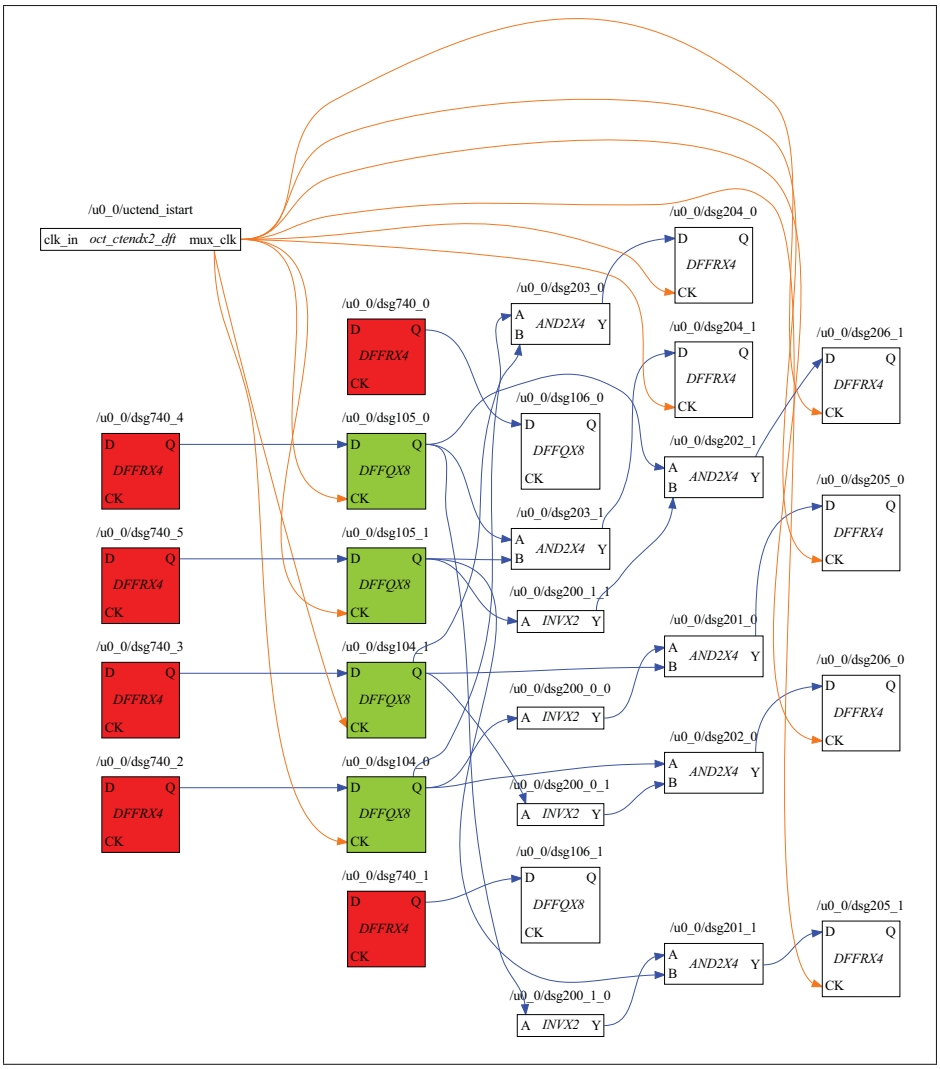

Figure 4.10 Analyse et PdC (/u0_0/uctend_istart) des bascules du décodage de la source du circuit pico_alu

4.2.2.3 Analyse des résultats

Il est important de noter que l'analyse des résultats pour le groupe de bascules contenant les entrées données est basée sur des résultats obtenus par l'application manuelle des règles d'élaboration de l'outil. Donc, ces résultats ne sont pas disponible graphiquement. Ce groupe de bascules contient huit bascules et 20 chemins de données distincts par bascule pour un total de 160 chemins distincts. Comme l'illustre le Tableau 4.4, qmi trouverait effectivement 96 points de convergence sur une possibilité de 160, soit un TC de 60%. Donc, qmi réussirait a trouver 12

Tableau 4.4 Configurations obtenues manuellement des chemins de données des bascules d'entrées du circuit pico_alu

Points d'intérêt	8
Configuration A	96
Configuration B	0
Configuration C	0
Uncovered	64
Total	160

Tableau 4.5 Configurations obtenues manuellement des chemins des bascules d'instruction du circuit pico_alu

Points d'intérêt	8
Configuration A	8
Configuration B	0
Configuration C	0
Uncovered	0
Total	8

points de convergence, par bascule. Huit de ces chemins sont pilotés par les bascules contenant la source des opérandes. Par la boucle créée avec le signal de sélection, la bascule de données d'entrées est ultimement pilotée par la bascule contenant la source des opérandes. L'outil identifie ces chemins comme étant des configuration A. Les règles de recherche empêchent qmi de trouver des points de convergence pour huit des 20 chemins. Le chemin provenant de la source d'opérandes aléatoires asynchrone converge en une configuration A. Les trois derniers

Tableau 4.6 Configurations identifiées pour les bascules de sorties du circuit pico_alu

Points d'intérêt	8
Configuration A	512
Configuration B	0
Configuration C	0
Uncovered	1024
Total	1536

Tableau 4.7 Configurations des chemins des bascules contenant le type d'opération selon l'instruction du circuit pico_alu

Points d'intérêt	4
Configuration A	0
Configuration B	0
Configuration C	0
Uncovered	8
Total	8

Tableau 4.8 Configurations identifiées pour les bascules du décodage des sources des opérandes du circuit pico_alu

Points d'intérêt	12
Configuration A	12
Configuration B	0
Configuration C	0
Uncovered	6
Total	18

chemins sont effectivement des faux chemins comme ceux identifiés pour les bascules d'entrées de la ressource du circuit tok_test. En effet, le front qui rafraîchit les bascules d'entrées est lui-même retardé avant de se rendre aux bascules qui pilotent le chemin. Donc, il va de soi que les bascules qui pilotent ces trois chemins auront une sortie stable. L'outil identifie tout de même ces chemins comme étant des configuration A.

Comme pour le groupe de bascules contenant les entrées données, la revue des résultats pour les bascules contenant l'instruction est basée sur des résultats obtenus manuellement. Donc, ces résultats ne sont pas disponible graphiquement. Les règles d'élaboration trop génériques et moins axées sur la nature du circuit font en sorte que l'outil n'a effectivement pas terminé son exécution. S'il avait terminé, il trouverait que ces chemins sont de type A. En effet, elle représente le cas le plus simple où la bascule source, située dans le générateur d'instruction, est rafraîchie par un front qui est retardé en utilisant un simple élément de délai avant de mettre

à jour les bascules contenant le type d'instruction. Donc, comme l'indique le Tableau 4.5, le taux de couverture est de 100%.

L'outil qmi n'a pas réussi à trouver un point de convergence pour aucun des chemins de données du groupe des bascules contenant le décodage du type d'opération. En effet, le Tableau 4.7 et la Figure 4.9 indiquent bien qu'il y a huit chemins de données et aucun point de convergence. Le rouge dans la Figure 4.9 indique que le chemin de donnée est de type *uncovered*. L'outil ne converge pas, car les règles d'élaboration des chemins d'horloge empêchent l'outil de chercher un point de convergence dans un niveau hiérarchique plus haut que celui du chemin de donnée analysé. Cette règle existe pour réduire le temps de calcul. Elle est appuyée par le fait que généralement les horloges proviennent du même niveau hiérarchique que les données. La numérotation des niveaux hiérarchiques suit le patron suivant : le niveau racine est 0 et les niveaux plus bas sont identifiés par des nombres croissants. Étant donné que le chemin de données est au niveau hiérarchique N et que les chemins d'horloge proviennent d'un niveau plus petit que N, l'outil cesse la recherche et indique que le chemin de donnée est de type *uncovered*.

Le groupe de bascules de sorties offre un exemple typique de chemins de données de configuration A. L'illustration à la Figure 4.8 contient qu'une seule bascule de sortie dans le but de simplifier la présentation, mais le Tableau 4.6 contient un résumé de la classification de tous les chemins de données de tous les points d'intérêt de ce groupe. Les chemins de données débutent par deux types de bascules. En effet, les bascules vertes sont des éléments mémoire contenant les opérandes et les bascules rouges contiennent l'opération à être fait sur les opérandes. Tous les chemins qui débutent par les opérandes convergent avec le même chemin d'horloge. Tandis que les chemins qui débutent par le type d'opération ne convergent pas pour la raison expliquée dans l'analyse des points d'intérêts du décodage de l'opération. Le chemin d'horloge avec lequel les chemins de données convergent contient visiblement une ligne à délai. Donc, ces PdC identifient bien des courses asynchrones. Pour le groupe des bascules contenant les sorties, le TC est seulement de 33%. Ceci est causé par le lien entre le résultat et le masquage engendré par le type d'opération.

Le résultat de l'analyse du groupe contenant les bascules du décodage des sources des opérandes est illustré à la Figure 4.10. Ce groupe contient deux niveaux de bascules, soit les bascules d'entrées et les bascules de sorties. La classification des chemins de données se fait en un temps pour les deux niveaux de bascules et est représentée dans le Tableau 4.8. Les bascules d'entrées, identifiées par les nombres 104, 105 et 106 ont un lien direct avec les bascules contenant l'instruction à exécuter elles-mêmes identifiées par le nombre 740. Cependant, pour la même raison que les bascules du décodage du type d'opération, il est impossible pour qmi de trouver un PdC pour les bascules d'entrées. Les bascules identifiées par les nombres 204, 205 et 206 sont les bascules de sorties. Certains chemins de données des bascules de sorties ont convergé avec un chemin d'horloge selon la configuration A. Les autres chemins de données débutent par les bascules contenant l'instruction et sont donc *uncovered* comme pour les chemins des bascules contenant les entrées. Le point de convergence est un élément de test indiquant la fin du signal *istart*. Comme pour les chemins de données des bascules *Input* du circuit tok_test il est évident que ces courses sont synchrones, car il n'y a pas chaîne à délai ou de logique combinatoire dans le chemin d'horloge. La dépendance des chemins de données aux bascules contenant l'instruction fait en sorte que le TC est seulement de 66,67% pour le groupe des bascules contenant le décodage des sources des opérandes.

4.2.3 Résumé des analyses

L'outil qmi en est à sa première version et contient visiblement des lacunes. Malgré tout, en effectuant le ratio entre le nombre de chemins correctement classifiés et le nombre de chemins de données total pour les deux circuits, l'outil obtient un TC combiné de 96,81%. Soit un TC de 36,30% pour le circuit pico_alu et 100% pour le circuit tok_test. Ce taux de couverture permet d'affirmer que l'outil réalise bien sa tâche. En effet, en répondant à la problématique, qmi devient un outil intéressant qui peut aider les concepteurs à vérifier leurs circuits et qui permet aux ingénieurs de tests d'élaborer plus facilement des patrons de test pour les structures de circuit utilisées. Le Tableau 4.9 et Tableau 4.10 offrent un résumé des TC et des TE présentés par groupe de bascules. Le temps d'exécution d'au moins 30 jours pour les groupes d'entrées

et d'instruction provient de l'algorithme d'élaboration des chemins d'horloge. Malgré le fait que le nombre de chemins de données de ces groupes est plus petit que le nombre de chemins de données du groupe des bascules de sorties, la nature des chemins d'horloge provoque une explosion combinatoire du nombre de chemins à explorer.

Tableau 4.9 Tableau récapitulatif des résultats du circuit tok_test

Groupe	**Tableau**	**Chemins total**	**Chemins classifiés**	**TC(%)**	**TE**
Données	4.1	32	32	100	3,31 heures
Entrées	4.2	32800	32800	100	12,06 heures
Sorties	4.3	32	32	100	0,05 s
	Total	32864	32864	100	15,37 heures

Tableau 4.10 Tableau récapitulatif des résultats du circuit pico_alu

Groupe	**Tableau**	**Chemins total**	**Chemins classifiés**	**TC(%)**	**TE**
Entrées	4.4	160	96	*	>30 jours
Instruction	4.5	8	8	*	>30 jours
Sorties	4.6	1536	512	33,33	0,68 s
Opération	4.7	8	0	0	0,03 s
Sources	4.8	18	12	66,67	0,05 s
	Total	1730	628	36,30	>60 jours

4.3 Limitations des résultats

La sous-section précédente explique une cause pour laquelle l'outil n'obtient pas un taux de couverture de 100%. Effectivement, l'algorithme d'élaboration des chemins d'horloge a été conçu pour limiter le temps de calcul, se faisant, son efficacité a été affectée. Il existe aussi d'autres raisons pour justifier les limitations des résultats obtenus. Par exemple, qmi ne cherche que pour trois types de configurations. Une étude plus élaborée des chemins catégorisés comme étant *uncovered* pourrait révéler d'autres types de configurations. De plus, la nature des circuits oblige une analyse plus complexe par rapport à un circuit synchrone. Ceci est notamment causé par le type de masquage utilisé qui rend la distinction entre les signaux de données

et les signaux d'horloge plus floue. Le développement de l'outil étant axé sur la fonctionnalité et non sur la performance cause un temps d'exécution beaucoup trop long pour les cas les plus fréquents. Cela étant dit, l'outil est à sa première version et les versions futures auront assurément un meilleur TC ainsi qu'une vitesse d'exécution plus grande.

4.4 Conclusion

En bref, le chapitre explique comment les résultats obtenus ont été validés statistiquement par une méthode manuelle. Suite à la validation, les points de convergences obtenus pour les deux circuits de test, soit tok_test et pico_alu, ont été présentés graphiquement. De plus, il y a eu la révélation des tableaux résumés contenant la classification des chemins de données pour chaque groupe d'étude. Après la présentation des résultats suit l'analyse qui fait état du taux de couverture obtenu par l'outil. Un taux de couverture de 96,81% a été obtenu et ce nombre est justifié dans la section 4.3. Finalement, il est pertinent de mentionner que l'outil, malgré certaines lacunes, répond aux besoins mentionnés dans la section Introduction.

CONCLUSION

Les premiers chapitres du mémoire offrent les bases nécessaires pour comprendre la pertinence et le développement de l'outil d'analyse qmi. Le premier chapitre fait un survol de la théorie des graphes et de la conception asynchrone. De plus, les explications de la méthodologie de conception synchrone et de l'analyse temporelle des designs synchrones permettent d'illustrer le manque à gagner des outils de conception traditionnels. Les survols des méthodes de test et outils d'aide au test des circuits fabriqués permettent de constater la même lacune. C'est à dire, les outils traditionnels ne sont pas adaptés pour les circuits asynchrones. Le chapitre suivant fait état de la méthodologie de conception de circuits asynchrones par Octasic. Suite à l'analyse de deux circuits de tests illustrant cette méthodologie, il a été possible d'élaborer trois configurations de circuits qui se retrouvent souvent dans les designs développés par la compagnie. Avec la notion des configurations s'attache le concept du point de convergence. L'idée du point de convergence provient de la généralisation de l'analyse temporelle statique. En effet, les chemins de données et les chemins d'horloge peuvent tous deux contenir des nuages de logique combinatoire. Contrairement aux chemins d'horloge traditionnels qui ne sont, pratiquement, que des arbres de distribution composés de fils et de tampons.

Les concepts des configurations et du point de convergence sont utilisés par qmi pour classifier les bascules des deux circuits de tests. Le taux de couverture de cet outil est défini comme étant le ratio entre le nombre de chemins à classifier et les chemins correctement classifiés. La section résultats du mémoire illustre que l'outil qmi réussit à obtenir un taux de couverture acceptable de 96,81% sur tous les chemins de données provenant des deux circuits.

Rappelons que la méthodologie asynchrone est pertinente lorsqu'il est nécessaire de construire des circuits performants et peu énergivores. Effectivement, les circuits développés contiennent de plus en plus de portes logiques. Ceci permet aux circuits de réaliser une multitude de fonctions avec des performances très intéressantes mais qui consomment plus d'énergie. Il est évident que les appareils mobiles sont de plus en plus populaires et qu'en fonctionnant avec des piles, ils ne peuvent se permettre de contenir des processeurs énergivores. C'est pourquoi il est pertinent d'utiliser des méthodes de conception à faible puissance telle que la méthode de

conception asynchrone. Mais, étant donné que la majorité des circuits développés n'utilisent pas la méthodologie de conception asynchrone, il y a un réel manque d'outils. Ce manque provient du fait que le paradigme asynchrone n'est pas encore généralement accepté par tous les concepteurs. En effet, si le paradigme n'est pas accepté, les fabricants d'outils ne seront pas enclins à en développer pour cette méthodologie. De plus, le paradigme ne sera pas accepté s'il n'y a pas d'outils d'aide à la conception.

Avec un taux de couverture de 96,81% sur le nombre total des chemins de données et la possibilité de générer de scripts permettant l'utilisation de logiciels traditionnels dans un contexte asynchrone, qmi réussit à combler quelque peu le manque d'outil. En effet, il devient possible d'utiliser des outils commerciaux à d'autres fins. Ceci pourrait inciter les concepteurs à utiliser l'approche asynchrone sans changer le flot de conception généralement utilisé.

RECOMMANDATIONS

Malgré le fait que qmi répond bien au problème créé par le manque d'outil, qmi possède certaines limitations.

En effet, le temps d'exécution est inadéquat pour un contexte d'utilisation réel. Pour réduire le temps d'exécution de qmi, il serait pertinent de changer le format de graphe utilisé pour un format plus concis, qui nécessite moins de ressource mémoire et qui permet une gestion plus directe des attributs et des paramètres des nœuds et vertex. De plus, la nature individuelle des détecteurs de configurations fait en sorte que certaines parties des détecteurs effectuent le même travail. Pour couper le temps d'exécution, il serait intéressant de combiner ces fonctions redondantes dans un détecteur de base. Finalement, la hiérarchisation et la modularisation du design pourraient être exploitées pour permettre à l'outil de réaliser son travail en moins de temps. En effet, il existe souvent plusieurs instances des mêmes modules au sein d'un même design. La nature écrasée du graphe empêche l'outil de découvrir que certains modules sont identiques. Donc, l'outil effectue le même traitement sur plusieurs instances d'un même module. Un format de graphe hiérarchisé pourrait ainsi réduire le temps d'exécution.

Finalement, le taux de couverture de l'outil devrait être de 100%. Cependant, il n'est que de 96,81%. Pour augmenter ce taux, il serait approprié d'analyser tous les chemins qui n'ont pas été classifiés correctement dans le but de trouver des similitudes dans leurs structures. Comme pour l'analyse des circuits de tests effectuée dans le deuxième chapitre, une analyse des chemins non classifiés pourrait entraîner la création d'autres configurations. De plus, les règles d'élaboration des chemins de données et des chemins d'horloge pourraient être revues et améliorées. Présentement, certaines règles sont ajustées pour compenser le long temps d'exécution. Ces compensations pourraient être retirées une fois que l'outil sera plus performant dans le but d'augmenter le taux de couverture.

ANNEXE I

DÉVELOPPEMENT ET VALIDATION DE G♭

L'outil gflat permet de créer un graphe modélisant un circuit à partir de sa représentation *net-list*. En effet, gflat extrait les informations des interconnexions entre les cellules standard. Le format G♭ est un format intuitif dans lequel le circuit est représenté comme étant un graphe où les sommets sont les cellules standard et les liens entre ces sommets sont les connexions entre les entrées des cellules standard et les sorties d'autres éléments de base. Les cellules standard proviennent d'une librairie. Cette librairie contient des éléments combinatoires réalisant les fonctions logiques principales. Elle possède aussi des cellules représentant des éléments mémoire comme des verrous et des bascules. Ces cellules sont parfois définies par la compagnie de fabrication de puces et sont utilisées par tous les outils de conception traditionnels.

1. Définition du format G♭

Il existe plusieurs formats de graphe privés. Effectivement, ces formats sont souvent peu documentés et à l'usage exclusifs des compagnies de conception et des compagnies créatrices d'outil. Dans le but d'offrir un format plus libre et moins contraignant, l'outil développé dans ce mémoire utilise le format de graphe défini dans cette annexe.

Le format G♭ est basé sur l'utilisation de numéros identificateurs et sur l'utilisation de représentations simples des données du circuit. En effet, les données du circuit sont représentées par trois structures, soit les instances, les branches et les nœuds. Ces structures possèdent toutes un numéro identificateur qui leur est propre. Il existe des numéros pour identifier les instances, les branches et les nœuds. Chaque instance représente une instanciation d'une cellule standard dans le design. Chaque port d'entrée et chaque port de sortie de ces instances sont associés à une branche. Les ports sont identifiés par un index et par un nom de nœud. Les index sont attribués en ordre croissant aux ports d'entrées en premier et ensuite aux ports de sorties, et ce en ordre alphabétique selon le nom du port. Les branches contiennent tous les nœuds entre une sortie d'une instance et l'entrée d'une autre instance. La branche est une structure de don-

nées qui contient tous les alias d'un même fil, ou connexion physique, entre deux instances. En terminologie de graphe, les instances sont les vertex et les branches sont les nœuds. Il est important de noter que les branches contiennent les numéros identificateurs des nœuds et l'encodage de ce numéro permet d'identifier si le nœud représente un fil du circuit ou un port d'une instance.

Les sous-sections suivantes expliquent l'implémentation de ces structures de données.

1.1 Données de base

Le format G♭ a été créé pour être écrit dans un tampon mémoire. Ce tampon est ensuite compressé en utilisant la librairie zlib et le résultat est sauvegardé dans un fichier. Donc, toutes les données du graphe doivent être facilement sérialisables. Pour ce faire, seules une petite quantité de données de bases sont utilisées. En effet, le Tableau-A I-1 contient tous les types de données de base utilisés par le format G♭ .

Tableau-A I-1 Description des types de base utilisés dans le format G♭

Type	Description
uint$N$_t	Un entier non-signé de N bits où $N = 8, 16, 32$ *ou* 64.
c_str	Une chaîne de caractères incluant le caractère de terminaison de chaîne '0x00'.
NetID	Un uint64_t représentant le numéro d'identification d'un nœud. Si le bit 63 est à 1, les bits 62 jusqu'à 32 contiennent un InstanceID et les bits 31 jusqu'à 0 contiennent l'index du port de l'instance en question.
BranchID	Un entier de type uint64_t contenant le numéro d'identification d'une branche.
InstanceID	Un entier de type uint32_t contenant le numéro d'identification d'une instance.

1.2 Types composés de données de base

Les descriptions qui suivent expliquent les formats des données composées. Chaque ligne d'un format est exprimée sous la forme suivante : *[nom : type]*. Le paramètre *nom* indique le nom

du champ et le paramètre *type* indique son type. Optionnellement, le type peut être multiplié par la variable contenent le nombre de fois que ce type est présent pour donner une idée de l'espace requise par le champ. Le type peut être soit un type de données de base ou un type composé.

La Figure-A I-1 illustre les données qui composent la structure string_table. Cette structure permet de facilement sérialiser dans un tampon mémoire une table de chaînes de caractères. Chaque chaîne possède un index et est sauvegardée dans le tableau *data*. Dans le cadre du format G♭ , il y a une string_table pour les noms des nœuds et une string_table pour le nom des instances. Pour retrouver le nom d'un nœud ou d'une instance, on utilise son numéro identificateur comme index dans sa string_table.

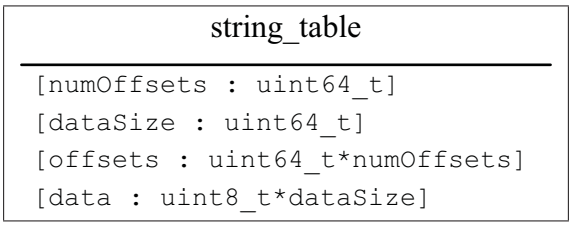

Figure-A I-1 Type composé string_table

Toutes les branches du graphe sont enregistrées dans une structure de données. Cette structure est représentée à la Figure-A I-2. Selon le même principe que la string_table, la branch_table s'utilise avec un index. Cet index est effectivement le numéro identificateur de la branche. La branch_table contient le nombre de branches et une représentation sérialisée de chaque branche.

Le format d'une branche est disponible à la Figure-A I-3. Étant donné qu'une branche n'est qu'une liste de tous les nœuds entre la sortie d'une instance et l'entrée d'une autre, la structure branch contient que le nombre de nœuds et lesdits nœuds.

La dernière structure qui contient les données du graphe est la instance_table. Comme son nom l'indique, elle possède toutes les instances du circuit. Pour ce faire, la structure contient le

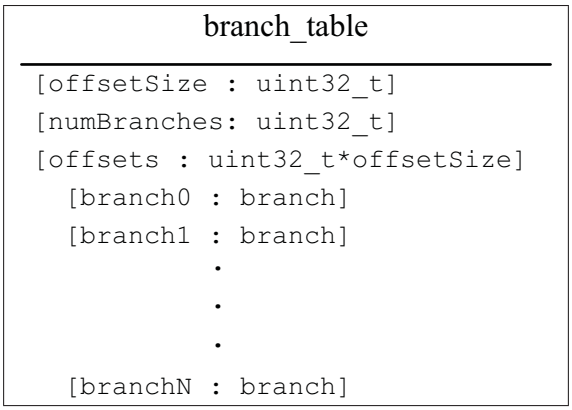

Figure-A I-2 Type composé branch_table

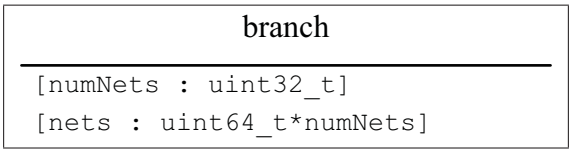

Figure-A I-3 Type composé branch

nombre d'instances ainsi que toutes instances sérialisées les unes à la suite des autres. Chaque instance est représentée sous le format de la Figure-A I-5.

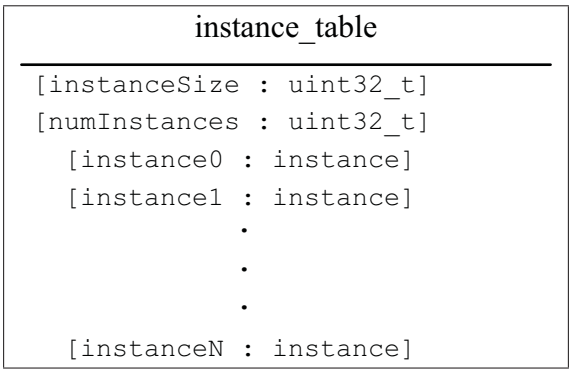

Figure-A I-4 Type composé instance_table

Une instance contient toutes les données nécessaires pour l'identifier correctement. Elle a un nom qui représente sa fonction, un numéro unique, le nombre de ports qu'elle possède ainsi que l'index du premier port de sortie. L'index du premier port de sortie est pertinent dû à l'ordre

d'assignation des numéros de ports expliqué précédemment. Une instance peut être un élément combinatoire ou un élément mémoire, la structure instance contient un champ pour illustrer cette propriété.

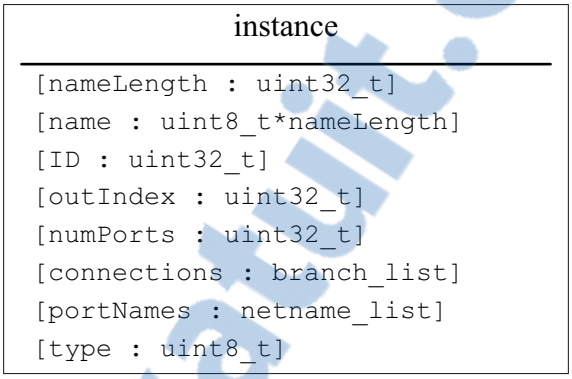

Figure-A I-5 Type composé instance

Finalement, chaque port de l'instance possède une branche et un nom. Il existe une liste de branches et une liste de noms de nœuds. Ces listes sont adressées selon l'index du port. Leurs formats respectifs se trouvent dans la Figure-A I-6 et dans la Figure-A I-7. Ces deux listes sont effectivement des tableaux qui contiennent les numéros identificateurs des branches ou des nœuds.

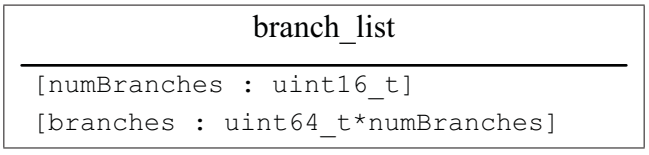

Figure-A I-6 Type composé branch_list

La dernière structure, la map, permet de sérialiser une structure associative. Cette structure permet d'obtenir le numéro identificateur d'une instance en utilisant son nom.

1.3 Formats des fichiers générés par l'outil gflat

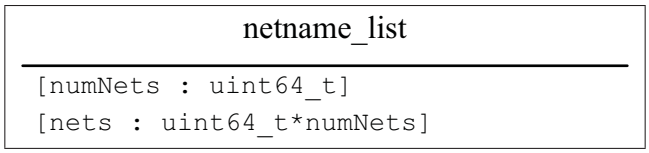

Figure-A I-7 Type composé netname_list

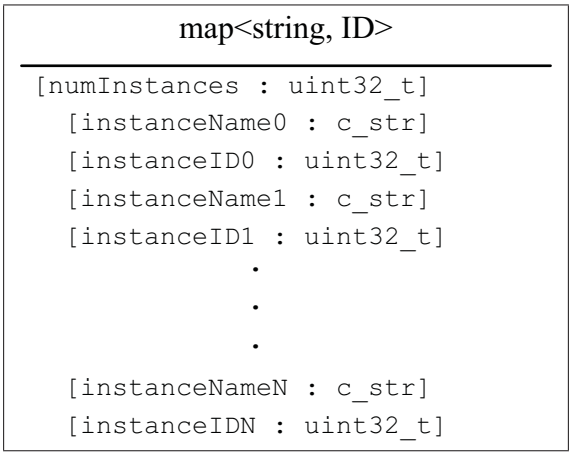

Figure-A I-8 Type composé map<string, ID>

1.3.1 Format G♭

Le premier fichier généré par gflat contient une sérialisation du graphe. Donc, il contient toutes les structures de données nécessaires pour représenter le circuit. Ces structures sont celles expliquées dans la section précédente. D'après la Figure-A I-9, le fichier G♭ contient une string_-table pour le nom des instances et une autre pour le nom des nœuds. De plus, elle contient la liste de toutes les branches et la liste de toutes les instances. Finalement, elle contient la structure associative qui permet d'obtenir le numéro identificateur d'une instance à partir de son nom.

Toutes ces structures sont sérialisées dans un tampon mémoire avant d'être compressées puis sauvegardées dans un fichier portant l'extension *.gflat*.

1.3.2 Format gstd

<filename>.gflat

```
[netTable : string_table]
[branchTable : branch_table]
[instanceNameTable : string_table]
[instanceTable: instance_table]
[instanceNameToID : map<string, ID>
```

Figure-A I-9 Format du fichier G♭ basé sur les types composés

Le deuxième fichier généré par gflat contient la librairie de cellules de base utilisée pour la création du graphe. Toutes les instances du graphe font référence à une porte définie dans ce fichier.

Le format de la sérialisation de la librairie standard est illustré à la Figure-A I-10. Le premier champ du fichier contient le nombre de cellules standard que le fichier possède. Les cellules standard sont caractérisées par un nom, qui reflète leur fonction, ainsi qu'une liste de ports. La liste de ports est sérialisée en commençant par le nombre de ports qu'elle contient suivie de la liste de ports à proprement parler. Chaque port comprend un numéro identificateur, soit son index, et un nom.

2. Élaboration de l'outil

L'architecture générale de l'outil gflat est présentée à la Figure-A I-11.

Le développement de l'outil est centré sur un *plugin* pour un logiciel nommé VRQ. VRQ est un outil permettant d'effectuer des manipulations sur des fichiers Verilog. En effet, VRQ transforme la description Verilog d'une *netlist* en un format interne qui lui est propre. Ensuite, VRQ exécute une liste de *plugin*, un à la suite de l'autre, pour modifier la *netlist*. Donc, la sortie du premier *plugin* devient l'entrée du deuxième et ainsi de suite. Pour faciliter la conception de l'outil gflat, VRQ a été retenu comme analyseur syntaxique pour le langage Verilog.

gflat est un *plugin* qui se fait exécuter au sein de VRQ. L'outil a besoin de trois paramètres d'entrées. Soit, le nom de l'entité principale, le fichier Verilog représentant la librairie standard

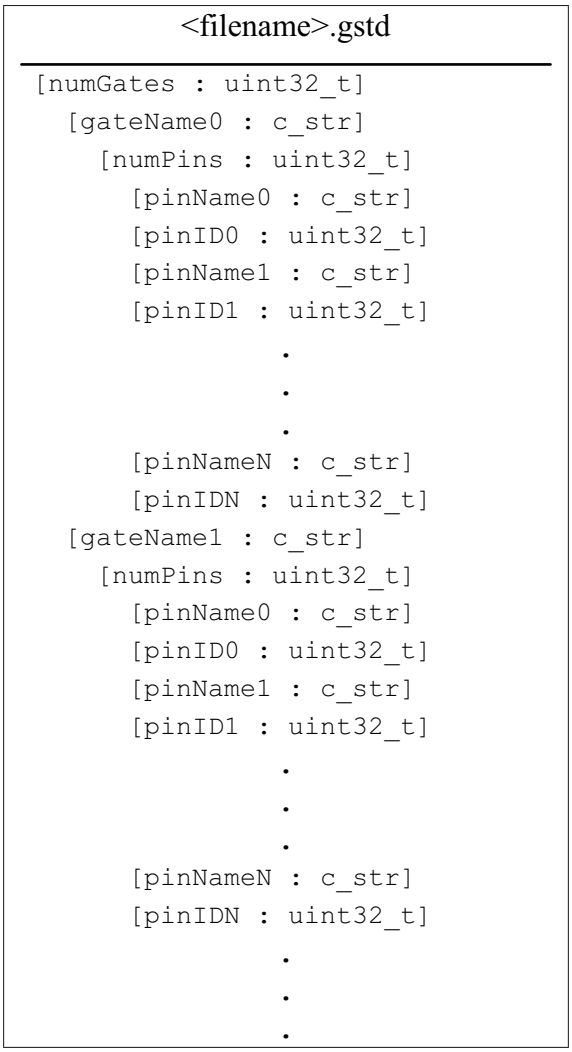

Figure-A I-10 Format du fichier gstd basé sur les types de base

et la *netlist* du design. Le but de gflat est de générer la liste de toutes les instances du circuit et la liste de toutes les branches du circuit. L'outil parcourt le graphe du format interne de VRQ pour visiter chacune des instances des modules correspondants à la librairie standard. En se faisant, gflat élabore en parallèle la liste des instances, la liste des branches et les tableaux de noms. Pour sauvegarder une instance dans la table d'instances, tous ses ports doivent être assignés à une branche complète. Une branche est considérée complète lorsqu'elle débute et se termine par un port d'une autre instance. La gestion de la création des branches se fait au moyen d'un système de sources et de récepteurs.

Une fois toutes les instances complétées, gflat procède à la création du fichier au format G♭ .

Dans cette annexe, seulement le fonctionnement de haut niveau a été expliqué, car gflat est un logiciel de moyenne envergure. Le code de gflat est dans un dépôt *git* hébergé sur les serveurs de Bitbucket et est disponible en contactant l'auteur de ce document.

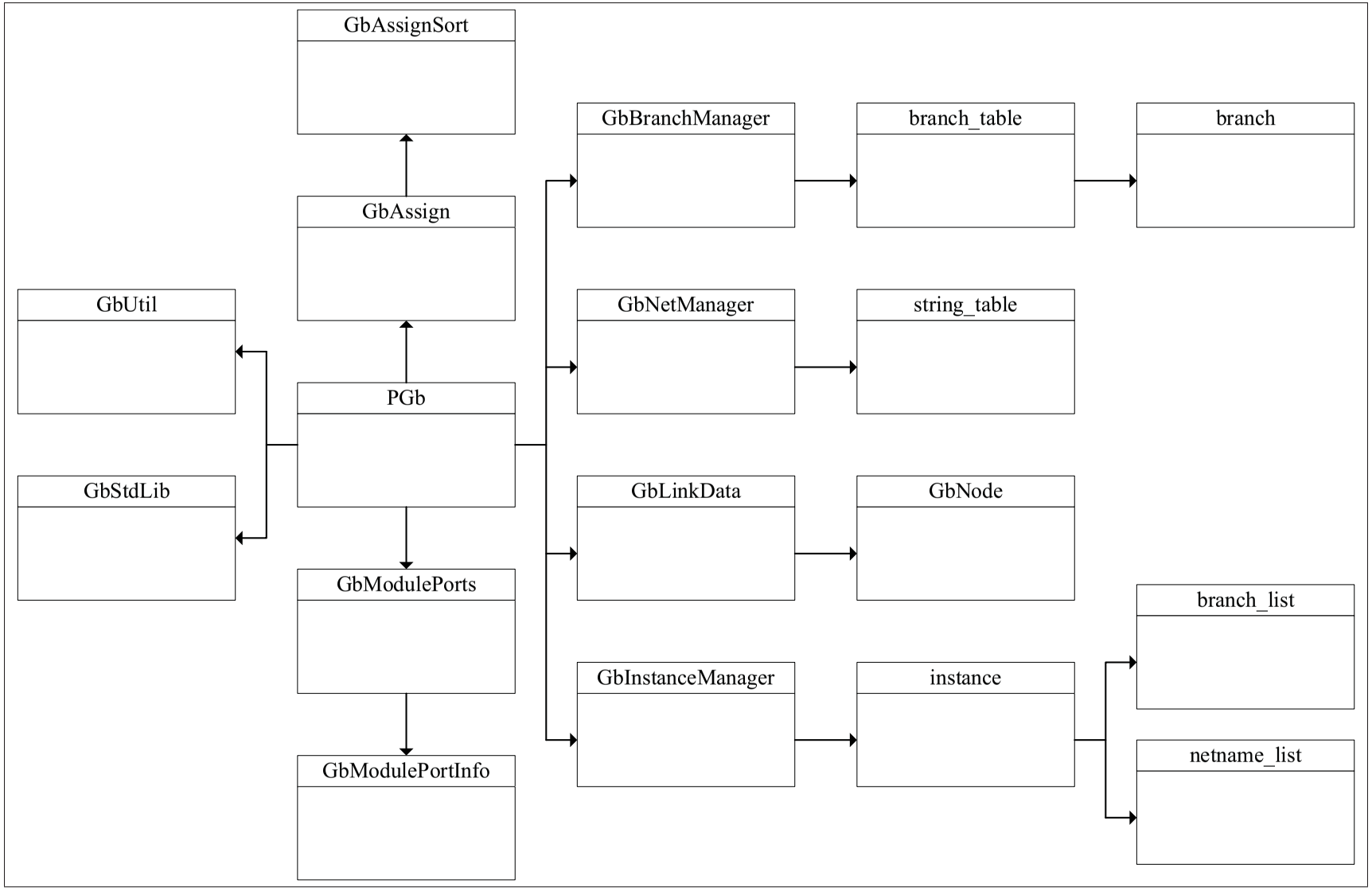

Figure-A I-11 Interactions entre les principaux composants de gflat

3. Validation de l'outil

La validité de l'outil a été assurée par une méthode simple. En effet, en utilisant le circuit représentatif tok_test, une liste d'instances et une liste de branches ont été élaborées en suivant la même méthode que gflat, mais manuellement. Ces listes ont été sauvegardées dans un format facilement comparable à la sortie de déverminage de gflat. Ensuite, les listes générées manuellement et automatiquement par gflat ont été comparées. Suite à cette comparaison, tout indique que la sortie de gflat est conforme et suit l'implémentation attendue.

ANNEXE II

ÉLABORATION DE L'OUTIL QMI

Cette annexe décrit le fonctionnement de haut-niveau du code de qmi. Le chapitre 3 contient une description du fonctionnement. Comme pour gflat, le code de qmi est dans un dépôt *git* et l'accès peut être autorisé par l'auteur du mémoire.

L'outil qmi, comme expliqué dans le chapitre 3, prend plusieurs fichiers de configuration et paramètres en entrée. Deux de ces paramètres font référence aux fichiers produits par gflat. Effectivement, qmi utilise le ficher gflat pour obtenir la description sous forme de graphe du circuit à analyser. De plus, il utilise le fichier gstd pour charger la librairie de cellules standard. Il y a des fichiers de configurations pour configurer l'analyseur et les afficheurs. Finalement, il y a un fichier de configuration qui est utilisé pour caractériser la librairie standard. C'est avec ce fichier qu'un concepteur indique à qmi quelles cellules standard sont des éléments mémoire, des éléments de test, des éléments de délai et des cellules à niveau fixe.

Avec ces données, l'outil qmi trouve tous les points de convergence (PdC) pour tous les points d'intérêt. Un point de convergence, dénoté PdC, est défini comme étant l'endroit dans le circuit où les chemins de données d'un point d'intérêt convergent avec les chemins d'horloge du même point d'intérêt. Le chapitre 3 décrit comment l'outil trouve ces points. Ensuite, il peut générer plusieurs fichiers de sorties selon les afficheurs utilisés. Un concepteur peut demander à qmi d'exécuter plusieurs afficheurs sur les mêmes résultats. En effet, comme mentionné dans le chapitre 3, l'outil peut créer une représentation graphique sous forme de fichier Graphviz. De plus, qmi peut générer des scripts pour aider la génération de patrons de tests avec la suite logicielle Tessent de Mentor Graphics. Finalement, il est possible d'obtenir un listing complet des PdC, un tableau récapitulatif des PdC ou des statistiques sur les PdC.

Pour réaliser sa tâche, qmi est réalisé en plusieurs classes. La section suivante regroupe ces classes en ensembles et explique les liens entre les groupes.

1. Description de haut-niveau du code de qmi

Le développement de l'outil qmi se base sur une application du patron de conception Modèle-Vue-Contrôleur (MVC). En effet, l'analyse de graphes pour la génération de sous-graphes se prête très bien au patron MVC. La Figure-A II-1 illustre les liens entre les ensembles de classes de l'outil. Chaque ensemble se catégorise dans une des trois facettes du patron MVC.

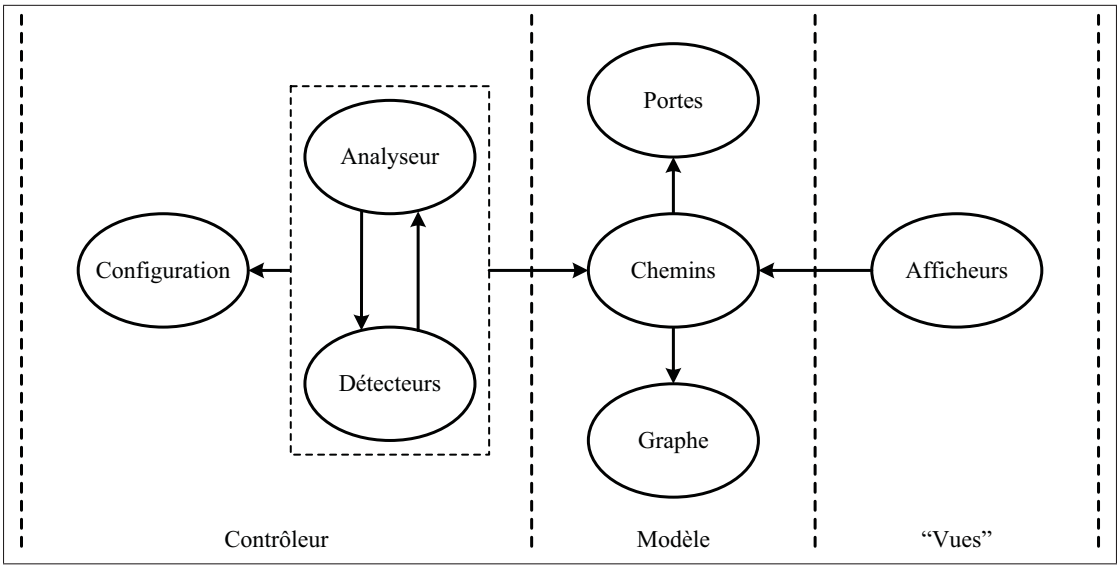

Figure-A II-1 Liens entre les modules selon le modèle MVC. Chaque ensemble comprend plusieurs classes

Le Tableau-A II-1 explicite les classes principales de chaque groupe.

Tableau-A II-1 Classes membres des groupes réalisant qmi

Groupe	**Classes**
Configuration	QmiConfig, QmiKeyValueLoader, QmiTargets
Analyseur	QmiAnalyser
Détecteurs	QmiDetectorInterface, QmiDetectorConfA, QmiDetectorConfB, QmiDetectorConfC
Portes	QmiGate, QmiGateDescription
Chemins	QmiPath, QmiPathBuilder, QmiPathModel, QmiSink, QmiSource, QmiPoint, QmiPoints, QmiNode, QmiNodeSet
Graphe	QmiGbGraph, QmiInstance, QmiBranch, QmiSource, QmiSink
Afficheurs	QmiPrinterInterface, QmiPrinterFactory, QmiPrinterGraphviz, QmiPrinterListing, QmiPrinterStats, QmiPrinterTable, QmiPrinterTessent

Le patron MVC est un patron de conception logiciel pour la programmation orientée objet. Il permet de découpler, ou de séparer, les concepts du contrôleur, du modèle et de la vue. Le découplage entre les concepts est illustré par les lignes pointillées de la Figure-A II-1. Cette séparation permet de bien définir le comportement et l'interaction entre les concepts. Un exemple de l'application du patron MVC pourrait être un chiffrier. En effet, dans ce contexte, le modèle serait les données à traiter. Le contrôleur serait les fonctions exécutées sur le modèle et les vues seraient les représentations sous forme de tableaux et de graphiques. Dans le développement de qmi, le modèle est composé du graphe à traiter, la librairie de cellules standard et les sous-graphes créés. Le contrôleur de qmi permet de générer les sous-graphes en utilisant les détecteurs de configuration et la configuration. Les vues, complètement séparées du contrôleur, n'utilisent que les sous-graphes créés. Il existe une vue par type de fichier de sortie.

La description de la fonctionnalité de chaque ensemble se fait bien en la jumelant au patron MVC. En effet, les trois premiers ensembles du Tableau-A II-1 font partie du contrôleur. L'ensemble nommé configuration est responsable de la gestion de la configuration de l'application. Ses classes membres font l'analyse syntaxique de tous les fichiers de configuration utilisés par qmi. De plus, le groupe configuration s'occupe de la création de la liste des points d'intérêt. Cette liste peut être spécifiée manuellement par l'utilisateur ou créée automatiquement. L'ensemble nommé analyseur regroupe seulement une classe. Cette classe utilise la configuration générée par l'ensemble configuration pour créer des sous-graphes à partir du graphe qui représente le circuit. Le fonctionnement de l'analyseur est décrit dans le chapitre 3 et ne sera pas répété ici. Finalement, les détecteurs sont un ensemble de classes toutes basées sur la même interface. Cette interface permet d'utiliser facilement plusieurs détecteurs différents et permet aussi l'intégration aisée de nouveaux détecteurs. Les détecteurs sont utilisés par l'analyseur pour identifier les PdC et leurs configurations. Rappelons qu'il y a trois configurations différentes. Soit, la configuration A, la configuration B et la configuration C.

Les trois ensembles suivants du Tableau-A II-1 font partie du modèle. Le modèle des données utilisé par qmi est séparé en trois parties. Il y a le graphe qui représente le circuit, la librairie de cellules standard et les sous-graphes. Il est important de noter que qmi fait abstraction du format

de fichier utilisé pour représenter le graphe. Un autre format de graphe pourrait facilement être utilisé à condition de créer une classe qui permet d'obtenir les instances et les branches entre ces instances dans un format neutre et propre à qmi. Ce format est en fait des classes qui adaptent la notion que qmi possède d'une instance et d'une branche au format de fichier. Les sous-graphes contiennent tous les PdC de tous les points d'intérêt. Le modèle est le point central de l'application. En effet, la configuration et la découverte des PdC créent le modèle et les afficheurs utilisent ce modèle.

Le dernier ensemble est celui qui correspond au concept de vue du patron MVC. L'architecture des afficheurs utilise aussi une interface commune et plusieurs classes qui réalisent cette interface. Il est donc facile d'ajouter un autre afficheur. Ces afficheurs utilisent le modèle pour produire différentes vues des PdC.

Comme mentionné dans le chapitre 3, le fonctionnement de qmi se fait en trois étapes. La première étape, la configuration, est réalisée par le groupe configuration. Les données générées par ce groupe sont utilisées par l'étape suivante, l'analyse. L'analyse met à l'œuvre le contrôleur et le modèle du patron MVC. Donc, à cette étape seuls les ensembles de configuration, analyseur et détecteurs sont utilisés. Ces ensembles créent les sous-graphes représentant les PdC de tous les points d'intérêt. La dernière étape, l'affichage, est réalisée par les classes du groupe afficheurs. En effet, selon la configuration, l'utilisateur peut choisir de créer plusieurs fichiers de sortie qui représentent de façon différente les résultats trouvés par l'analyse du circuit.

ANNEXE III

RÉSULTATS COMPLETS DES ANALYSES DES CIRCUITS DE TEST

Pour alléger la représentation graphique des résultats des analyses de qmi sur les deux circuits de tests, un nombre réduit de bascules de chaque groupe était illustré. Cette section contient les représentations graphiques complètes. Les tableaux des statistiques des résultats présentés graphiquement dans cette section sont disponibles dans le chapitre 4.

Étant donné la nature informatisée des représentations graphiques, ces représentations doivent être vues dans la version numérique, soit PDF, du mémoire. Les figures des résultats sont à la base des images en format SVG, elles ont été traduites en format PDF et intégrées dans ce document. Donc, elles sont exprimées dans un format vectoriel et peuvent subir un zoom quasi infini tout en conservant la qualité d'affichage.

Les deux sous-sections suivantes contiennent les représentations graphiques des analyses des circuits de test tok_test et pico_alu respectivement. De plus, la sous-section pour le circuit de test pico_alu contient un exemple du fichier de configuration utilisé par Tessent pour aider la génération des patrons de tests. Ce fichier n'est pas nécessairement facile à lire pour un humain, car à la base, il est créé pour une utilisation automatisée. Le fichier est fourni uniquement pour être rigoureux et pour démontrer que l'outil peut effectivement aider à générer des patrons de tests.

1. tok_test

Les résultats complets sont présentés dans le même ordre que lors de la discussion des résultats. En effet, la Figure-A III-1 (page 110) contient la représentation graphique des PdC pour le groupe formé des bascules contenant les données. Ensuite, la Figure-A III-2 (page 111) illustre la conceptualisation graphique des PdC du groupe des bascules d'entrées. Finalement, la Figure-A III-3 (page 119) montre la schématisation des PdC du groupe des bascules de sorties.

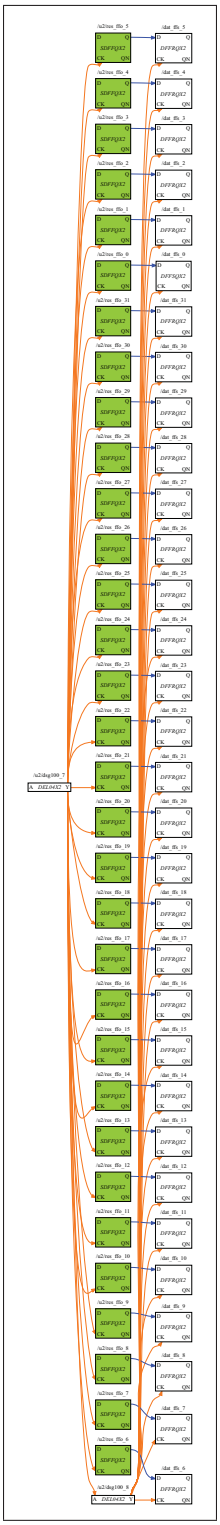

Figure-A III-1 Point de convergence (/u2/dsg100_7) pour toutes les bascules du groupe contenant les données du circuit tok_test

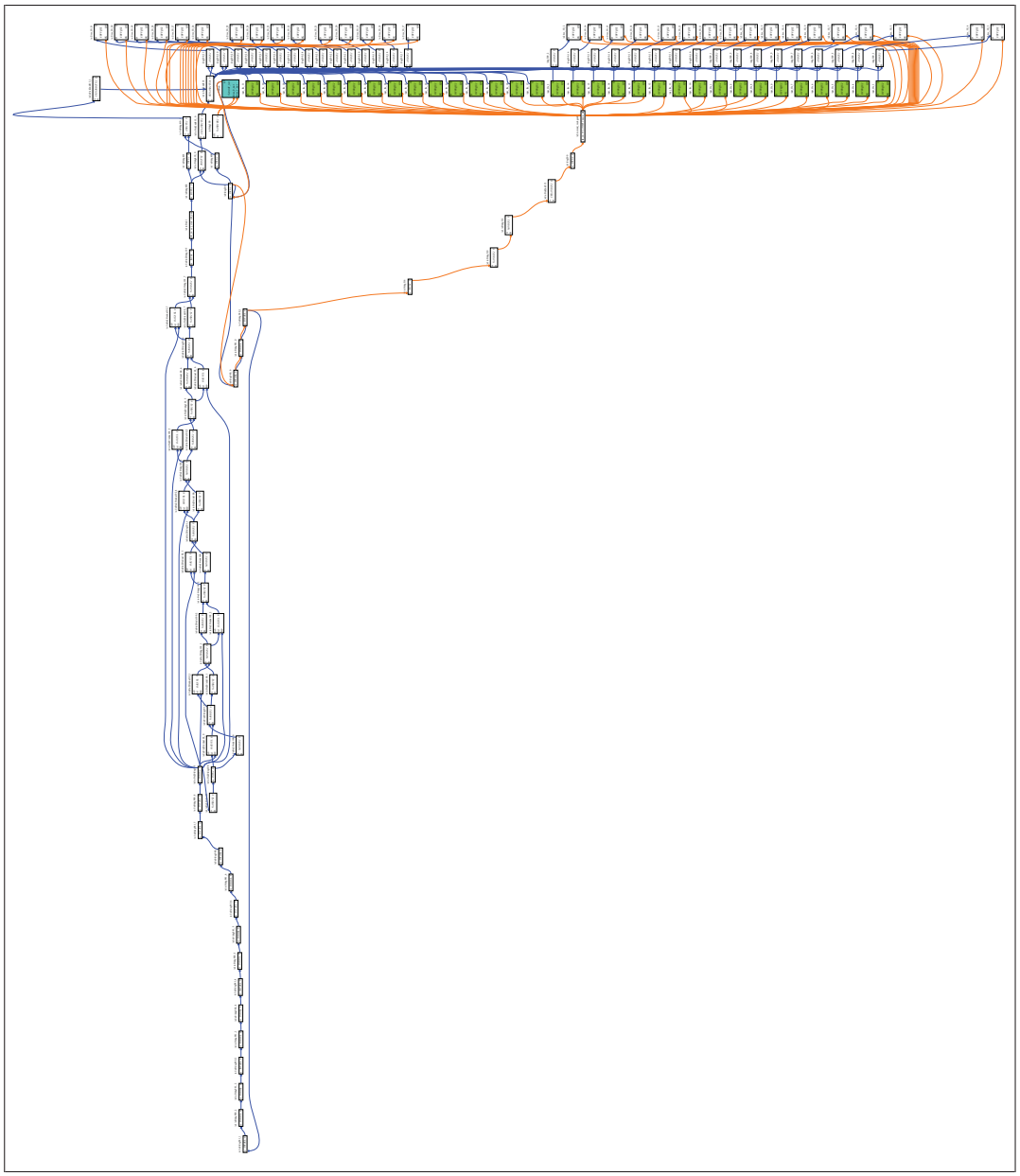

Figure-A III-2 Point de convergence (/u2/uctend_rxckgen0) pour toutes les bascules du groupe contenant les entrées du circuit tok_test

2. pico_alu

Au moment de l'écriture, l'outil n'avait pas terminé l'exécution des analyses pour le groupe de bascules contenant les données d'entrées et pour le groupe de bascules contenant l'instruction. De plus, les résultats pour le groupe de bascules du décodage de la source et pour le groupe de bascules contenant le type d'opération sont déjà présentés dans leur totalité dans le chapitre 4. Donc, seulement la représentation graphique du groupe de bascules contenant les sorties est illustrée à la Figure-A III-4 (page 120). Finalement, le fichier permettant d'aider la génération de patrons de tests pour le même groupe que la Figure-A III-4 est disponible dans l'Extrait-A III-1 (page 112).

```
##############################################################################
# CAUTION: AUTO-GENERATED FILE
#
# Mon August 04 21:31:02 EDT 2014
##############################################################################
# Format:
#   This script defines 2N+1 variables.
#
#   The "root" variable is groups_list. It's a list of N elements where N is the
#   number of groups.
#
#   A group is defined has being the "base name" of an ensemble of targets.
#   For example, the targets "dsg512_0", "dsg512_1", "dsg512_2" and "dsg512_3"
#   have the the same base name of "dsg512".
#
#   For each group, there is a list named groupName_faults which contains all
#   the possible faults for the data paths of that group.
#   There is also an array called groupNames_clocks which has two members:
#   launch that contains the launch clock name and capture that contains
#   the name of the capture clock.
#
#   In the preceeding example, if qmi was ran using only the specified targets,
#   this script would contain something like:
#       set groups_list [list dsg512]
#       set dsg512_faults [list node1 node2 node2]
#       array set dsg512_clocks {launch clk_L capture clk_C}
##############################################################################
set groups_list [list /u0_0/dsg512]
```

```
array set /u0_0/dsg512_clocks {launch /u0_0/opnd_src_all_done_poCt capture \
                               /u0_0/alu_done_o_poCt}
set /u0_0/dsg512_faults [list /u0_0/dsg107_0/A /u0_0/dsg107_0/Y /u0_0/dsg107_1/A \
    /u0_0/dsg107_1/Y /u0_0/dsg111/A /u0_0/dsg111/B \
    /u0_0/dsg111/Y /u0_0/dsg112/A /u0_0/dsg112/B \
    /u0_0/dsg112/Y /u0_0/dsg113/A /u0_0/dsg113/B \
    /u0_0/dsg113/Y /u0_0/dsg740_6/CK /u0_0/dsg740_6/D \
    /u0_0/dsg740_6/Q /u0_0/dsg740_7/CK /u0_0/dsg740_7/D \
    /u0_0/dsg740_7/Q /u0_0/dsg211_dft_0_0/CK /u0_0/dsg211_dft_0_0/D \
    /u0_0/dsg211_dft_0_0/Q /u0_0/dsg211_dft_0_1/CK /u0_0/dsg211_dft_0_1/D \
    /u0_0/dsg211_dft_0_1/Q /u0_0/dsg211_dft_0_2/CK /u0_0/dsg211_dft_0_2/D \
    /u0_0/dsg211_dft_0_2/Q /u0_0/dsg211_dft_0_3/CK /u0_0/dsg211_dft_0_3/D \
    /u0_0/dsg211_dft_0_3/Q /u0_0/dsg211_dft_0_4/CK /u0_0/dsg211_dft_0_4/D \
    /u0_0/dsg211_dft_0_4/Q /u0_0/dsg211_dft_0_5/CK /u0_0/dsg211_dft_0_5/D \
    /u0_0/dsg211_dft_0_5/Q /u0_0/dsg211_dft_0_6/CK /u0_0/dsg211_dft_0_6/D \
    /u0_0/dsg211_dft_0_6/Q /u0_0/dsg211_dft_0_7/CK /u0_0/dsg211_dft_0_7/D \
    /u0_0/dsg211_dft_0_7/Q /u0_0/dsg211_dft_1_0/CK /u0_0/dsg211_dft_1_0/D \
    /u0_0/dsg211_dft_1_0/Q /u0_0/dsg211_dft_1_1/CK /u0_0/dsg211_dft_1_1/D \
    /u0_0/dsg211_dft_1_1/Q /u0_0/dsg211_dft_1_2/CK /u0_0/dsg211_dft_1_2/D \
    /u0_0/dsg211_dft_1_2/Q /u0_0/dsg211_dft_1_3/CK /u0_0/dsg211_dft_1_3/D \
    /u0_0/dsg211_dft_1_3/Q /u0_0/dsg211_dft_1_4/CK /u0_0/dsg211_dft_1_4/D \
    /u0_0/dsg211_dft_1_4/Q /u0_0/dsg211_dft_1_5/CK /u0_0/dsg211_dft_1_5/D \
    /u0_0/dsg211_dft_1_5/Q /u0_0/dsg211_dft_1_6/CK /u0_0/dsg211_dft_1_6/D \
    /u0_0/dsg211_dft_1_6/Q /u0_0/dsg211_dft_1_7/CK /u0_0/dsg211_dft_1_7/D \
    /u0_0/dsg211_dft_1_7/Q /u0_0/dsg500_dft_0_0/A /u0_0/dsg500_dft_0_0/B \
    /u0_0/dsg500_dft_0_0/Y /u0_0/dsg500_dft_0_1/A /u0_0/dsg500_dft_0_1/B \
    /u0_0/dsg500_dft_0_1/Y /u0_0/dsg500_dft_0_2/A /u0_0/dsg500_dft_0_2/B \
    /u0_0/dsg500_dft_0_2/Y /u0_0/dsg500_dft_0_3/A /u0_0/dsg500_dft_0_3/B \
    /u0_0/dsg500_dft_0_3/Y /u0_0/dsg500_dft_0_4/A /u0_0/dsg500_dft_0_4/B \
    /u0_0/dsg500_dft_0_4/Y /u0_0/dsg500_dft_0_5/A /u0_0/dsg500_dft_0_5/B \
    /u0_0/dsg500_dft_0_5/Y /u0_0/dsg500_dft_0_6/A /u0_0/dsg500_dft_0_6/B \
    /u0_0/dsg500_dft_0_6/Y /u0_0/dsg500_dft_0_7/A /u0_0/dsg500_dft_0_7/B \
    /u0_0/dsg500_dft_0_7/Y /u0_0/dsg500_dft_1_0/A /u0_0/dsg500_dft_1_0/B \
    /u0_0/dsg500_dft_1_0/Y /u0_0/dsg500_dft_1_1/A /u0_0/dsg500_dft_1_1/B \
    /u0_0/dsg500_dft_1_1/Y /u0_0/dsg500_dft_1_2/A /u0_0/dsg500_dft_1_2/B \
    /u0_0/dsg500_dft_1_2/Y /u0_0/dsg500_dft_1_3/A /u0_0/dsg500_dft_1_3/B \
    /u0_0/dsg500_dft_1_3/Y /u0_0/dsg500_dft_1_4/A /u0_0/dsg500_dft_1_4/B \
    /u0_0/dsg500_dft_1_4/Y /u0_0/dsg500_dft_1_5/A /u0_0/dsg500_dft_1_5/B \
    /u0_0/dsg500_dft_1_5/Y /u0_0/dsg500_dft_1_6/A /u0_0/dsg500_dft_1_6/B \
    /u0_0/dsg500_dft_1_6/Y /u0_0/dsg500_dft_1_7/A /u0_0/dsg500_dft_1_7/B \
    /u0_0/dsg500_dft_1_7/Y /u0_0/u500/u0_0_0/u0/dsg121_0/A0 /u0_0/u500/u0_0_0/u0/dsg121_0/A1 \
    /u0_0/u500/u0_0_0/u0/dsg121_0/B0 /u0_0/u500/u0_0_0/u0/dsg121_0/Y \
    /u0_0/u500/u0_0_0/u0/dsg121_1/A0 /u0_0/u500/u0_0_0/u0/dsg121_1/A1 \
    /u0_0/u500/u0_0_0/u0/dsg121_1/B0 /u0_0/u500/u0_0_0/u0/dsg121_1/Y \
```

```
/u0_0/u500/u0_0_0/u1/dsg121/A0 /u0_0/u500/u0_0_0/u1/dsg121/A1 \
/u0_0/u500/u0_0_0/u1/dsg121/B0 /u0_0/u500/u0_0_0/u1/dsg121/Y \
/u0_0/u500/u0_0_0/dsg100_0/A /u0_0/u500/u0_0_0/dsg100_0/B \
/u0_0/u500/u0_0_0/dsg100_0/CO /u0_0/u500/u0_0_0/dsg100_0/S /u0_0/u500/u0_0_0/dsg102_0/A \
/u0_0/u500/u0_0_0/dsg102_0/Y /u0_0/u500/u0_0_0/dsg100_1/A /u0_0/u500/u0_0_0/dsg100_1/B \
/u0_0/u500/u0_0_0/dsg100_1/CI /u0_0/u500/u0_0_0/dsg100_1/CO /u0_0/u500/u0_0_0/dsg100_1/S \
/u0_0/u500/u0_0_0/dsg102_1/A /u0_0/u500/u0_0_0/dsg102_1/B /u0_0/u500/u0_0_0/dsg102_1/CO \
/u0_0/u500/u0_0_0/dsg102_1/S /u0_0/u500/u0_0_0/dsg104/A /u0_0/u500/u0_0_0/dsg104/Y \
/u0_0/u500/u0_0_0/dsg100_2/A /u0_0/u500/u0_0_0/dsg100_2/B /u0_0/u500/u0_0_0/dsg100_2/CO \
/u0_0/u500/u0_0_0/dsg100_2/S /u0_0/u500/u0_0_0/dsg102_2/A /u0_0/u500/u0_0_0/dsg102_2/Y \
/u0_0/u500/u0_0_0/dsg100_3/A /u0_0/u500/u0_0_0/dsg100_3/B /u0_0/u500/u0_0_0/dsg100_3/CI \
/u0_0/u500/u0_0_0/dsg100_3/CO /u0_0/u500/u0_0_0/dsg100_3/S /u0_0/u500/u0_0_0/dsg102_3/A \
/u0_0/u500/u0_0_0/dsg102_3/B /u0_0/u500/u0_0_0/dsg102_3/CO /u0_0/u500/u0_0_0/dsg102_3/S \
/u0_0/u500/u0_0_0/dsg115_0/A /u0_0/u500/u0_0_0/dsg115_0/Y /u0_0/u500/u0_0_0/dsg114_0/A \
/u0_0/u500/u0_0_0/dsg114_0/Y /u0_0/u500/u0_0_0/dsg116_0/A /u0_0/u500/u0_0_0/dsg116_0/Y \
/u0_0/u500/u0_0_0/dsg117_0/A /u0_0/u500/u0_0_0/dsg117_0/Y /u0_0/u500/u0_0_0/dsg110_0/A \
/u0_0/u500/u0_0_0/dsg110_0/B /u0_0/u500/u0_0_0/dsg110_0/Y /u0_0/u500/u0_0_0/dsg111_0/A \
/u0_0/u500/u0_0_0/dsg111_0/B /u0_0/u500/u0_0_0/dsg111_0/Y /u0_0/u500/u0_0_0/dsg113_0/A \
/u0_0/u500/u0_0_0/dsg113_0/B /u0_0/u500/u0_0_0/dsg113_0/Y /u0_0/u500/u0_0_0/dsg115_1/A \
/u0_0/u500/u0_0_0/dsg115_1/Y /u0_0/u500/u0_0_0/dsg114_1/A /u0_0/u500/u0_0_0/dsg114_1/Y \
/u0_0/u500/u0_0_0/dsg116_1/A /u0_0/u500/u0_0_0/dsg116_1/Y /u0_0/u500/u0_0_0/dsg117_1/A \
/u0_0/u500/u0_0_0/dsg117_1/Y /u0_0/u500/u0_0_0/dsg110_1/A /u0_0/u500/u0_0_0/dsg110_1/B \
/u0_0/u500/u0_0_0/dsg110_1/Y /u0_0/u500/u0_0_0/dsg111_1/A /u0_0/u500/u0_0_0/dsg111_1/B \
/u0_0/u500/u0_0_0/dsg111_1/Y /u0_0/u500/u0_0_0/dsg113_1/A /u0_0/u500/u0_0_0/dsg113_1/B \
/u0_0/u500/u0_0_0/dsg113_1/Y /u0_0/u500/u0_0_0/dsg152/A /u0_0/u500/u0_0_0/dsg152/Y \
/u0_0/u500/u0_0_0/dsg115_2/A /u0_0/u500/u0_0_0/dsg115_2/Y /u0_0/u500/u0_0_0/dsg114_2/A \
/u0_0/u500/u0_0_0/dsg114_2/Y /u0_0/u500/u0_0_0/dsg116_2/A /u0_0/u500/u0_0_0/dsg116_2/Y \
/u0_0/u500/u0_0_0/dsg117_2/A /u0_0/u500/u0_0_0/dsg117_2/Y /u0_0/u500/u0_0_0/dsg110_2/A \
/u0_0/u500/u0_0_0/dsg110_2/B /u0_0/u500/u0_0_0/dsg110_2/Y /u0_0/u500/u0_0_0/dsg111_2/A \
/u0_0/u500/u0_0_0/dsg111_2/B /u0_0/u500/u0_0_0/dsg111_2/Y /u0_0/u500/u0_0_0/dsg113_2/A \
/u0_0/u500/u0_0_0/dsg113_2/B /u0_0/u500/u0_0_0/dsg113_2/Y /u0_0/u500/u0_0_0/dsg115_3/A \
/u0_0/u500/u0_0_0/dsg115_3/Y /u0_0/u500/u0_0_0/dsg114_3/A /u0_0/u500/u0_0_0/dsg114_3/Y \
/u0_0/u500/u0_0_0/dsg116_3/A /u0_0/u500/u0_0_0/dsg116_3/Y /u0_0/u500/u0_0_0/dsg117_3/A \
/u0_0/u500/u0_0_0/dsg117_3/Y /u0_0/u500/u0_0_0/dsg110_3/A /u0_0/u500/u0_0_0/dsg110_3/B \
/u0_0/u500/u0_0_0/dsg110_3/Y /u0_0/u500/u0_0_0/dsg111_3/A /u0_0/u500/u0_0_0/dsg111_3/B \
/u0_0/u500/u0_0_0/dsg111_3/Y /u0_0/u500/u0_0_0/dsg113_3/A /u0_0/u500/u0_0_0/dsg113_3/B \
/u0_0/u500/u0_0_0/dsg113_3/Y /u0_0/u500/u0_1_0/u1/dsg121/A0 \
/u0_0/u500/u0_1_0/u1/dsg121/A1 /u0_0/u500/u0_1_0/u1/dsg121/B0 \
/u0_0/u500/u0_1_0/u1/dsg121/Y /u0_0/u500/u0_1_0/dsg100_0/A /u0_0/u500/u0_1_0/dsg100_0/B \
/u0_0/u500/u0_1_0/dsg100_0/CO /u0_0/u500/u0_1_0/dsg100_0/S \
/u0_0/u500/u0_1_0/dsg102_0/A /u0_0/u500/u0_1_0/dsg102_0/Y /u0_0/u500/u0_1_0/dsg100_1/A \
/u0_0/u500/u0_1_0/dsg100_1/B /u0_0/u500/u0_1_0/dsg100_1/CI /u0_0/u500/u0_1_0/dsg100_1/CO \
/u0_0/u500/u0_1_0/dsg100_1/S /u0_0/u500/u0_1_0/dsg102_1/A /u0_0/u500/u0_1_0/dsg102_1/B \
/u0_0/u500/u0_1_0/dsg102_1/CO /u0_0/u500/u0_1_0/dsg102_1/S /u0_0/u500/u0_1_0/dsg104/A \
```

```
/u0_0/u500/u0_1_0/dsg104/Y /u0_0/u500/u0_1_0/dsg100_2/A /u0_0/u500/u0_1_0/dsg100_2/B \
/u0_0/u500/u0_1_0/dsg100_2/CO /u0_0/u500/u0_1_0/dsg100_2/S /u0_0/u500/u0_1_0/dsg102_2/A \
/u0_0/u500/u0_1_0/dsg102_2/Y /u0_0/u500/u0_1_0/dsg100_3/A /u0_0/u500/u0_1_0/dsg100_3/B \
/u0_0/u500/u0_1_0/dsg100_3/CI /u0_0/u500/u0_1_0/dsg100_3/CO /u0_0/u500/u0_1_0/dsg100_3/S \
/u0_0/u500/u0_1_0/dsg102_3/A /u0_0/u500/u0_1_0/dsg102_3/B /u0_0/u500/u0_1_0/dsg102_3/CO \
/u0_0/u500/u0_1_0/dsg102_3/S /u0_0/u500/u0_1_0/dsg132/A /u0_0/u500/u0_1_0/dsg132/Y \
/u0_0/u500/u0_1_0/dsg133/A /u0_0/u500/u0_1_0/dsg133/Y /u0_0/u500/u0_1_0/dsg115_0/A \
/u0_0/u500/u0_1_0/dsg115_0/Y /u0_0/u500/u0_1_0/dsg114_0/A /u0_0/u500/u0_1_0/dsg114_0/Y \
/u0_0/u500/u0_1_0/dsg116_0/A /u0_0/u500/u0_1_0/dsg116_0/Y /u0_0/u500/u0_1_0/dsg117_0/A \
/u0_0/u500/u0_1_0/dsg117_0/Y /u0_0/u500/u0_1_0/dsg110_0/A /u0_0/u500/u0_1_0/dsg110_0/B \
/u0_0/u500/u0_1_0/dsg110_0/Y /u0_0/u500/u0_1_0/dsg111_0/A /u0_0/u500/u0_1_0/dsg111_0/B \
/u0_0/u500/u0_1_0/dsg111_0/Y /u0_0/u500/u0_1_0/dsg113_0/A /u0_0/u500/u0_1_0/dsg113_0/B \
/u0_0/u500/u0_1_0/dsg113_0/Y /u0_0/u500/u0_1_0/dsg115_1/A /u0_0/u500/u0_1_0/dsg115_1/Y \
/u0_0/u500/u0_1_0/dsg114_1/A /u0_0/u500/u0_1_0/dsg114_1/Y /u0_0/u500/u0_1_0/dsg116_1/A \
/u0_0/u500/u0_1_0/dsg116_1/Y /u0_0/u500/u0_1_0/dsg117_1/A /u0_0/u500/u0_1_0/dsg117_1/Y \
/u0_0/u500/u0_1_0/dsg110_1/A /u0_0/u500/u0_1_0/dsg110_1/B /u0_0/u500/u0_1_0/dsg110_1/Y \
/u0_0/u500/u0_1_0/dsg111_1/A /u0_0/u500/u0_1_0/dsg111_1/B /u0_0/u500/u0_1_0/dsg111_1/Y \
/u0_0/u500/u0_1_0/dsg113_1/A /u0_0/u500/u0_1_0/dsg113_1/B /u0_0/u500/u0_1_0/dsg113_1/Y \
/u0_0/u500/u0_1_0/dsg152/A /u0_0/u500/u0_1_0/dsg152/Y /u0_0/u500/u0_1_0/dsg115_2/A \
/u0_0/u500/u0_1_0/dsg115_2/Y /u0_0/u500/u0_1_0/dsg114_2/A /u0_0/u500/u0_1_0/dsg114_2/Y \
/u0_0/u500/u0_1_0/dsg116_2/A /u0_0/u500/u0_1_0/dsg116_2/Y /u0_0/u500/u0_1_0/dsg117_2/A \
/u0_0/u500/u0_1_0/dsg117_2/Y /u0_0/u500/u0_1_0/dsg110_2/A /u0_0/u500/u0_1_0/dsg110_2/B \
/u0_0/u500/u0_1_0/dsg110_2/Y /u0_0/u500/u0_1_0/dsg111_2/A /u0_0/u500/u0_1_0/dsg111_2/B \
/u0_0/u500/u0_1_0/dsg111_2/Y /u0_0/u500/u0_1_0/dsg113_2/A /u0_0/u500/u0_1_0/dsg113_2/B \
/u0_0/u500/u0_1_0/dsg113_2/Y /u0_0/u500/u0_1_0/dsg115_3/A /u0_0/u500/u0_1_0/dsg115_3/Y \
/u0_0/u500/u0_1_0/dsg114_3/A /u0_0/u500/u0_1_0/dsg114_3/Y /u0_0/u500/u0_1_0/dsg116_3/A \
/u0_0/u500/u0_1_0/dsg116_3/Y /u0_0/u500/u0_1_0/dsg117_3/A /u0_0/u500/u0_1_0/dsg117_3/Y \
/u0_0/u500/u0_1_0/dsg110_3/A /u0_0/u500/u0_1_0/dsg110_3/B /u0_0/u500/u0_1_0/dsg110_3/Y \
/u0_0/u500/u0_1_0/dsg111_3/A /u0_0/u500/u0_1_0/dsg111_3/B /u0_0/u500/u0_1_0/dsg111_3/Y \
/u0_0/u500/u0_1_0/dsg113_3/A /u0_0/u500/u0_1_0/dsg113_3/B /u0_0/u500/u0_1_0/dsg113_3/Y \
/u0_0/u500/u2_0_0/dsg121/A0 /u0_0/u500/u2_0_0/dsg121/A1 /u0_0/u500/u2_0_0/dsg121/B0 \
/u0_0/u500/u2_0_0/dsg121/Y /u0_0/dsg501_0/A /u0_0/dsg501_0/B \
/u0_0/dsg501_0/Y /u0_0/dsg501_1/A /u0_0/dsg501_1/B \
/u0_0/dsg501_1/Y /u0_0/dsg501_2/A /u0_0/dsg501_2/B \
/u0_0/dsg501_2/Y /u0_0/dsg501_3/A /u0_0/dsg501_3/B \
/u0_0/dsg501_3/Y /u0_0/dsg501_4/A /u0_0/dsg501_4/B \
/u0_0/dsg501_4/Y /u0_0/dsg501_5/A /u0_0/dsg501_5/B \
/u0_0/dsg501_5/Y /u0_0/dsg501_6/A /u0_0/dsg501_6/B \
/u0_0/dsg501_6/Y /u0_0/dsg501_7/A /u0_0/dsg501_7/B \
/u0_0/dsg501_7/Y /u0_0/dsg502_0/A /u0_0/dsg502_0/B \
/u0_0/dsg502_0/Y /u0_0/dsg502_1/A /u0_0/dsg502_1/B \
/u0_0/dsg502_1/Y /u0_0/dsg502_2/A /u0_0/dsg502_2/B \
/u0_0/dsg502_2/Y /u0_0/u501/dsg120_0/A /u0_0/u501/dsg120_0/Y \
/u0_0/u501/dsg121_0/A /u0_0/u501/dsg121_0/Y /u0_0/u501/dsg120_2/A \
```

```
/u0_0/u501/dsg120_2/B /u0_0/u501/dsg120_2/Y /u0_0/u501/dsg120_3/A \
/u0_0/u501/dsg120_3/B /u0_0/u501/dsg120_3/Y /u0_0/u501/dsg120_4/A \
/u0_0/u501/dsg120_4/B /u0_0/u501/dsg120_4/Y /u0_0/u501/dsg120_5/A \
/u0_0/u501/dsg120_5/B /u0_0/u501/dsg120_5/Y /u0_0/u501/dsg122_0/A \
/u0_0/u501/dsg122_0/Y /u0_0/u501/dsg122_1/A /u0_0/u501/dsg122_1/Y \
/u0_0/u501/dsg110_0/A /u0_0/u501/dsg110_0/B /u0_0/u501/dsg110_0/Y \
/u0_0/u501/dsg110_1/A0 /u0_0/u501/dsg110_1/A1 /u0_0/u501/dsg110_1/B0 \
/u0_0/u501/dsg110_1/B1 /u0_0/u501/dsg110_1/Y /u0_0/u501/dsg110_2/A0 \
/u0_0/u501/dsg110_2/A1 /u0_0/u501/dsg110_2/B0 /u0_0/u501/dsg110_2/B1 \
/u0_0/u501/dsg110_2/Y /u0_0/u501/dsg130_2/A /u0_0/u501/dsg130_2/B \
/u0_0/u501/dsg130_2/Y /u0_0/u501/dsg140_2/A /u0_0/u501/dsg140_2/B \
/u0_0/u501/dsg140_2/Y /u0_0/u501/dsg110_3/A0 /u0_0/u501/dsg110_3/A1 \
/u0_0/u501/dsg110_3/B0 /u0_0/u501/dsg110_3/B1 /u0_0/u501/dsg110_3/Y \
/u0_0/u501/dsg130_3/A0 /u0_0/u501/dsg130_3/A1 /u0_0/u501/dsg130_3/B0 \
/u0_0/u501/dsg130_3/B1 /u0_0/u501/dsg130_3/Y /u0_0/u501/dsg140_3/A \
/u0_0/u501/dsg140_3/B /u0_0/u501/dsg140_3/Y /u0_0/u501/dsg110_4/A0 \
/u0_0/u501/dsg110_4/A1 /u0_0/u501/dsg110_4/B0 /u0_0/u501/dsg110_4/B1 \
/u0_0/u501/dsg110_4/Y /u0_0/u501/dsg130_4/A0 /u0_0/u501/dsg130_4/A1 \
/u0_0/u501/dsg130_4/B0 /u0_0/u501/dsg130_4/B1 /u0_0/u501/dsg130_4/Y \
/u0_0/u501/dsg140_4/A /u0_0/u501/dsg140_4/B /u0_0/u501/dsg140_4/Y \
/u0_0/u501/dsg110_5/A0 /u0_0/u501/dsg110_5/A1 /u0_0/u501/dsg110_5/B0 \
/u0_0/u501/dsg110_5/B1 /u0_0/u501/dsg110_5/Y /u0_0/u501/dsg130_5/A0 \
/u0_0/u501/dsg130_5/A1 /u0_0/u501/dsg130_5/B0 /u0_0/u501/dsg130_5/B1 \
/u0_0/u501/dsg130_5/Y /u0_0/u501/dsg140_5/A /u0_0/u501/dsg140_5/B \
/u0_0/u501/dsg140_5/Y /u0_0/u501/dsg110_6/A0 /u0_0/u501/dsg110_6/A1 \
/u0_0/u501/dsg110_6/B0 /u0_0/u501/dsg110_6/B1 /u0_0/u501/dsg110_6/Y \
/u0_0/u501/dsg130_6/A0 /u0_0/u501/dsg130_6/A1 /u0_0/u501/dsg130_6/B0 \
/u0_0/u501/dsg130_6/B1 /u0_0/u501/dsg130_6/Y /u0_0/u501/dsg140_6/A \
/u0_0/u501/dsg140_6/B /u0_0/u501/dsg140_6/Y /u0_0/u501/dsg110_7/A0 \
/u0_0/u501/dsg110_7/A1 /u0_0/u501/dsg110_7/B0 /u0_0/u501/dsg110_7/B1 \
/u0_0/u501/dsg110_7/Y /u0_0/u501/dsg130_7/A0 /u0_0/u501/dsg130_7/A1 \
/u0_0/u501/dsg130_7/B0 /u0_0/u501/dsg130_7/B1 /u0_0/u501/dsg130_7/Y \
/u0_0/u501/dsg140_7/A /u0_0/u501/dsg140_7/B /u0_0/u501/dsg140_7/Y \
/u0_0/u501/dsg112_0/A /u0_0/u501/dsg112_0/B /u0_0/u501/dsg112_0/Y \
/u0_0/u501/dsg112_1/A /u0_0/u501/dsg112_1/B /u0_0/u501/dsg112_1/Y \
/u0_0/u501/dsg112_2/A /u0_0/u501/dsg112_2/B /u0_0/u501/dsg112_2/Y \
/u0_0/u501/dsg112_3/A /u0_0/u501/dsg112_3/B /u0_0/u501/dsg112_3/Y \
/u0_0/u501/dsg112_4/A0 /u0_0/u501/dsg112_4/A1 /u0_0/u501/dsg112_4/B0 \
/u0_0/u501/dsg112_4/B1 /u0_0/u501/dsg112_4/Y /u0_0/u501/dsg112_5/A0 \
/u0_0/u501/dsg112_5/A1 /u0_0/u501/dsg112_5/B0 /u0_0/u501/dsg112_5/B1 \
/u0_0/u501/dsg112_5/Y /u0_0/u501/dsg112_6/A0 /u0_0/u501/dsg112_6/A1 \
/u0_0/u501/dsg112_6/B0 /u0_0/u501/dsg112_6/B1 /u0_0/u501/dsg112_6/Y \
/u0_0/u501/dsg112_7/A0 /u0_0/u501/dsg112_7/A1 /u0_0/u501/dsg112_7/B0 \
/u0_0/u501/dsg112_7/B1 /u0_0/u501/dsg112_7/Y /u0_0/dsg503_dft_0_0/A \
```

```
/u0_0/dsg503_dft_0_0/B /u0_0/dsg503_dft_0_0/Y /u0_0/dsg503_dft_0_1/A \
/u0_0/dsg503_dft_0_1/B /u0_0/dsg503_dft_0_1/Y /u0_0/dsg503_dft_0_2/A \
/u0_0/dsg503_dft_0_2/B /u0_0/dsg503_dft_0_2/Y /u0_0/dsg503_dft_0_3/A \
/u0_0/dsg503_dft_0_3/B /u0_0/dsg503_dft_0_3/Y /u0_0/dsg503_dft_0_4/A \
/u0_0/dsg503_dft_0_4/B /u0_0/dsg503_dft_0_4/Y /u0_0/dsg503_dft_0_5/A \
/u0_0/dsg503_dft_0_5/B /u0_0/dsg503_dft_0_5/Y /u0_0/dsg503_dft_0_6/A \
/u0_0/dsg503_dft_0_6/B /u0_0/dsg503_dft_0_6/Y /u0_0/dsg503_dft_0_7/A \
/u0_0/dsg503_dft_0_7/B /u0_0/dsg503_dft_0_7/Y /u0_0/dsg503_dft_1_0/A \
/u0_0/dsg503_dft_1_0/B /u0_0/dsg503_dft_1_0/Y /u0_0/dsg503_dft_1_1/A \
/u0_0/dsg503_dft_1_1/B /u0_0/dsg503_dft_1_1/Y /u0_0/dsg503_dft_1_2/A \
/u0_0/dsg503_dft_1_2/B /u0_0/dsg503_dft_1_2/Y /u0_0/dsg503_dft_1_3/A \
/u0_0/dsg503_dft_1_3/B /u0_0/dsg503_dft_1_3/Y /u0_0/dsg503_dft_1_4/A \
/u0_0/dsg503_dft_1_4/B /u0_0/dsg503_dft_1_4/Y /u0_0/dsg503_dft_1_5/A \
/u0_0/dsg503_dft_1_5/B /u0_0/dsg503_dft_1_5/Y /u0_0/dsg503_dft_1_6/A \
/u0_0/dsg503_dft_1_6/B /u0_0/dsg503_dft_1_6/Y /u0_0/dsg503_dft_1_7/A \
/u0_0/dsg503_dft_1_7/B /u0_0/dsg503_dft_1_7/Y /u0_0/dsg504_0/A \
/u0_0/dsg504_0/B /u0_0/dsg504_0/Y /u0_0/dsg504_1/A \
/u0_0/dsg504_1/B /u0_0/dsg504_1/Y /u0_0/dsg504_2/A \
/u0_0/dsg504_2/B /u0_0/dsg504_2/Y /u0_0/dsg504_3/A \
/u0_0/dsg504_3/B /u0_0/dsg504_3/Y /u0_0/dsg504_4/A \
/u0_0/dsg504_4/B /u0_0/dsg504_4/Y /u0_0/dsg504_5/A \
/u0_0/dsg504_5/B /u0_0/dsg504_5/Y /u0_0/dsg504_6/A \
/u0_0/dsg504_6/B /u0_0/dsg504_6/Y /u0_0/dsg504_7/A \
/u0_0/dsg504_7/B /u0_0/dsg504_7/Y /u0_0/dsg510_0/A \
/u0_0/dsg510_0/B /u0_0/dsg510_0/Y /u0_0/dsg511_0/A \
/u0_0/dsg511_0/B /u0_0/dsg511_0/Y /u0_0/dsg512_0/CK \
/u0_0/dsg512_0/D /u0_0/dsg510_1/A /u0_0/dsg510_1/B \
/u0_0/dsg510_1/Y /u0_0/dsg511_1/A /u0_0/dsg511_1/B \
/u0_0/dsg511_1/Y /u0_0/dsg512_1/CK /u0_0/dsg512_1/D \
/u0_0/dsg510_2/A /u0_0/dsg510_2/B /u0_0/dsg510_2/Y \
/u0_0/dsg511_2/A /u0_0/dsg511_2/B /u0_0/dsg511_2/Y \
/u0_0/dsg512_2/CK /u0_0/dsg512_2/D /u0_0/dsg510_3/A \
/u0_0/dsg510_3/B /u0_0/dsg510_3/Y /u0_0/dsg511_3/A \
/u0_0/dsg511_3/B /u0_0/dsg511_3/Y /u0_0/dsg512_3/CK \
/u0_0/dsg512_3/D /u0_0/dsg510_4/A /u0_0/dsg510_4/B \
/u0_0/dsg510_4/Y /u0_0/dsg511_4/A /u0_0/dsg511_4/B \
/u0_0/dsg511_4/Y /u0_0/dsg512_4/CK /u0_0/dsg512_4/D \
/u0_0/dsg510_5/A /u0_0/dsg510_5/B /u0_0/dsg510_5/Y \
/u0_0/dsg511_5/A /u0_0/dsg511_5/B /u0_0/dsg511_5/Y \
/u0_0/dsg512_5/CK /u0_0/dsg512_5/D /u0_0/dsg510_6/A \
/u0_0/dsg510_6/B /u0_0/dsg510_6/Y /u0_0/dsg511_6/A \
/u0_0/dsg511_6/B /u0_0/dsg511_6/Y /u0_0/dsg512_6/CK \
/u0_0/dsg512_6/D /u0_0/dsg510_7/A /u0_0/dsg510_7/B \
/u0_0/dsg510_7/Y /u0_0/dsg511_7/A /u0_0/dsg511_7/B \
```

```
/u0_0/dsg511_7/Y /u0_0/dsg512_7/CK /u0_0/dsg512_7/D]
```

Extrait-A III-1 Exemple du fichier de configuration pour le groupe de bascules contenant les sorties

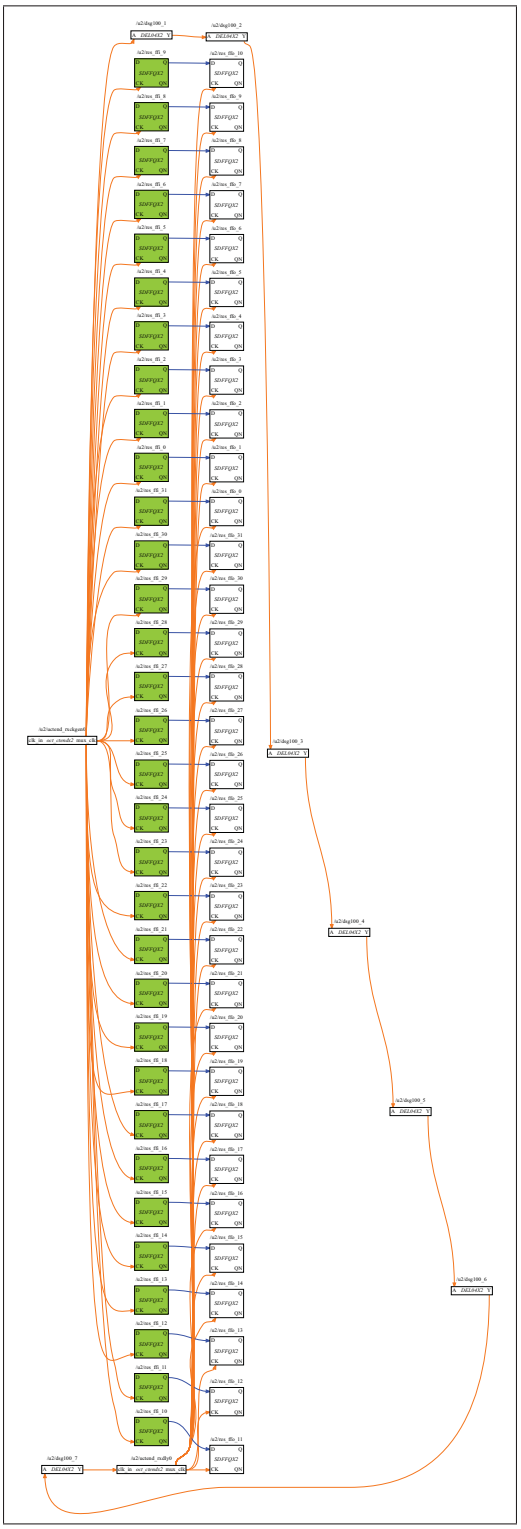

Figure-A III-3 Point de convergence (/u2/dsg100_7) pour toutes les bascules du groupe contenant les sorties du circuit tok_test

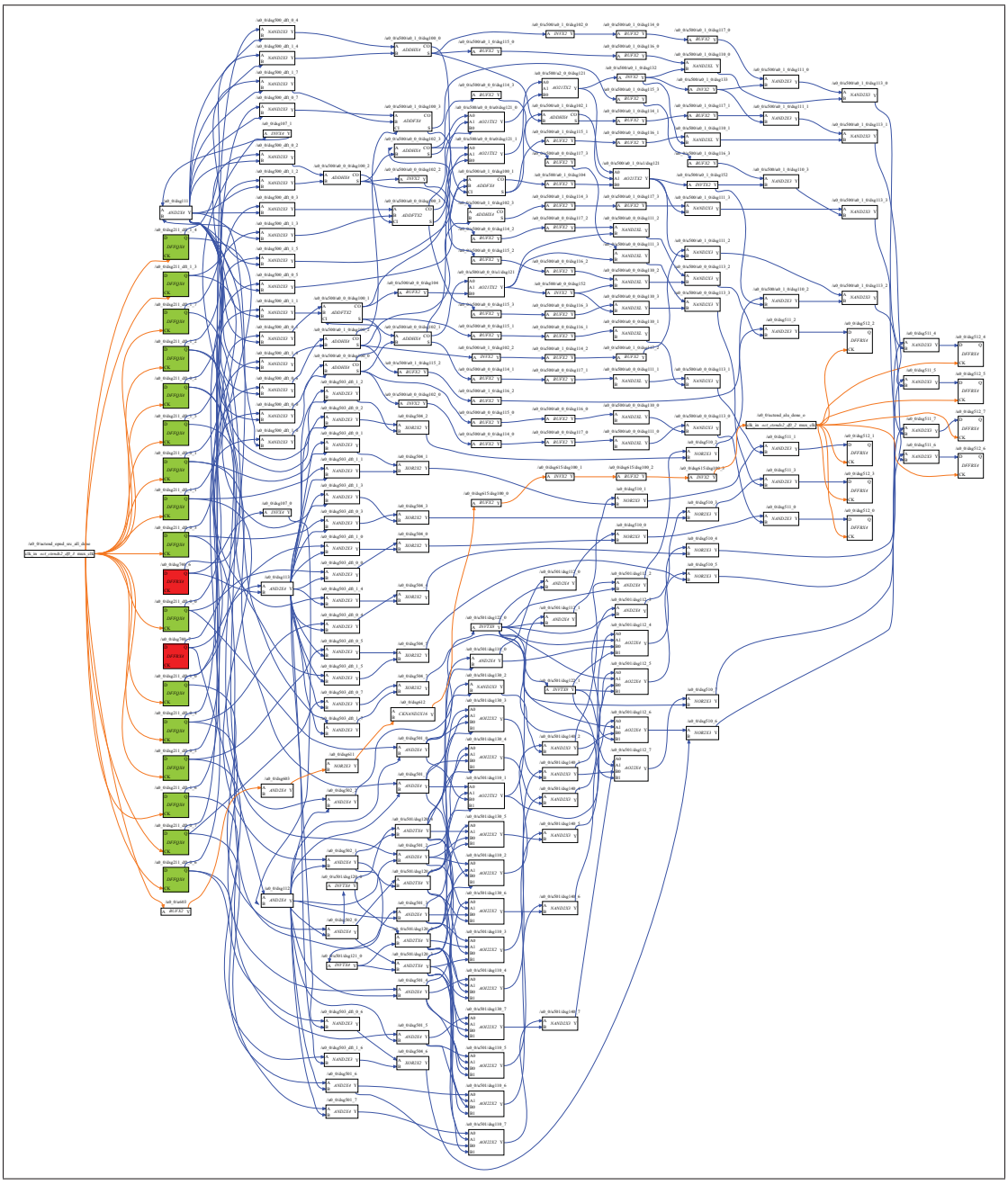

Figure-A III-4 Point de convergence (/u0_0/uctend_opnd_src_all_done) pour le groupe de bascules contenant les sorties du circuit pico_alu

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Awad, Thomas J., Michel Laurence, Martin Filteau, Pascal M. Gervais, et Douglas Morrissey. 14 Novembre 2013. « Method for sharing a resource and circuit making use of same ». <http://www.google.com/patents/US20130305077>. US Patent App. 13/948,685.

Beerel, Peter A., Recep O. Ozdag, et Marcos Ferretti, 2010. *A designer's guide to asynchronous VLSI*. 1re édition. New York : Cambridge University Press, 352 p.

Blaauw, David, Kaviraj Chopra, Ashish Srivastava, et Lou Scheffer. Avril 2008. « Statistical Timing Analysis : From Basic Principles to State of the Art ». *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, n° 4, p. 589-607.

Bushnell, Michael L. et Vishwani D. Agrawal, 2002. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. 1re édition. New York : Kluwer Academic Publishers.

Cisco Systems, Inc. Février 2013. « Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update, 2013–2018 ». En ligne. p. 40. <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf>. Consulté le 1 août 2014.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, et Clifford Stein, Juin 2009. *Introduction to Algorithms*. 3e édition. The MIT Press, 1312 p.

Crépeau, Daniel. 2013. « token test2 DFT strategy Vr 1.0.2.pdf - Représentation schématique du circuit pico_alu ». Document interne.

Davies, Rhodri M. et John V. Woods. 1996. « Timing Verification for Asynchronous Design ». In *Proceedings of the Conference on European Design Automation*. (Genève, Suisse 1996), p. 78–83. IEEE Computer Society Press.

Diestel, Reinhard, 2005. *Graph Theory*. Graduate Texts in Mathematics. 3e édition. Springer, 422 p.

Intel Corporation. Juillet 2012. « Intel Chips Timeline ». En ligne. p. 1. <http://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/history-intel-chips-timeline-poster.pdf>. Consulté le 1 août 2014.

ITRS. 2013. « International Technology Roadmap for Semiconductors - 2013 Edition : IRC Overview ». En ligne. p. 34. <http://www.itrs.net/Links/2013ITRS/2013Chapters/2013Overview.pdf>. Consulté le 1 août 2014.

Kahng, Andrew B., Jens Lienig, Igor L. Markov, et Jin Hu, 2011. *VLSI Physical Design : From Graph Partitioning to Timing Closure*. 1re édition. Springer.

Karlsen, Per Arne et Per Torstein Røine. 1999. « A Timing Verifier and Timing Profiler for Asynchronous Circuits ». In *Proceedings of the Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*. (Barcelone, Espagne 1999), p. 13-23.

Leclerc, Jean-Michel. 2013a. « token_circuit_v2.vsd - Représentation schématique du circuit tok_test ». Document interne.

Leclerc, Jean-Michel. 2013b. « token_diagram.svg - Diagramme de dépendance des jetons du circuit pico_alu ». Document interne.

Moore, Gordon E. 1975. « Progress in Digital Integrated Electronics ». In *1975 International Electron Devices Meeting*. p. 11-13.

Moore, Gordon E. Janvier 1998. « Cramming More Components Onto Integrated Circuits ». *Proceedings of the IEEE*, vol. 86, n° 1, p. 82-85.

Sapatnekar, Sachin S. Mars 2006. Static timing analysis. Lavagno, L., Louis Scheffer, et Grant Martin, editors, *EDA for IC Implementation, Circuit Design, and Process Technology*, chapter 6. CRC Press, 1re édition.

Weste, Neil et David Harris, Mars 2010. *CMOS VLSI Design : A Circuits and Systems Perspective*. 4e édition. Addison-Wesley, 864 p.

Yahya, Eslam, Laurent Fesquet, Yehea Ismail, et Marc Renaudin. 2013. « Statistical Static Timing Analysis of Conditional Asynchronous Circuits Using Model-Based Simulation ». In *Proceedings of the 19th IEEE International Symposium on Asynchronous Circuits and Systems*. (Santa Monica, Californie, É-U 2013), p. 67-74. IEEE Computer Society.