Synthèse de contrôleurs séquentiels QDI faible consommation prouvés corrects

K. Alsayeg

UNIVERSITE DE GRENOBLE
INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
978-2-84813-155-9

T H E S E

pour obtenir le grade de

DOCTEUR DE L'Université de Grenoble
délivré par l'Institut polytechnique de Grenoble

Spécialité : « MICRO ET NANO ELECTRONIQUE »

préparée au laboratoire TIMA

dans le cadre de l'Ecole Doctorale EEATS

présentée et soutenue publiquement

par

Khaled ALSAYEG

le 01/09/2010

TITRE

« Synthèse de contrôleurs séquentiels QDI faible consommation prouvés corrects »

DIRECTEUR DE THESE
Marc RENAUDIN
CO-DIRECTEURS DE THESE
Gilles SICARD
Laurent FESQUET

JURY

Pr. Ian O'CONNOR	, Président
Pr. Bruno ROUZEYRE	, Rapporteur
Pr. Nathalie JULIEN	, Rapporteur
Pr. Marc RENAUDIN	, Directeur de thèse
Dr. Gilles SICARD	, Co-encadrant
Dr. Laurent FESQUET	, Co-encadrant
Dr. Katell MORIN-ALLORY	, Examinateur

UNIVERSITE DE GRENOBLE
INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
978-2-84813-155-9

THESE

pour obtenir le grade de

DOCTEUR DE L'Université de Grenoble
délivré par l'Institut polytechnique de Grenoble

Spécialité : « MICRO ET NANO ELECTRONIQUE »

préparée au laboratoire TIMA

dans le cadre de l'Ecole Doctorale EEATS

présentée et soutenue publiquement
par

Khaled ALSAYEG

le 01/09/2010

TITRE

« Synthèse de contrôleurs séquentiels QDI faible consommation prouvés corrects »

DIRECTEUR DE THESE
Marc RENAUDIN
CO-DIRECTEURS DE THESE
Gilles SICARD
Laurent FESQUET

JURY

Pr. Ian O'CONNOR	, Président
Pr. Bruno ROUZEYRE	, Rapporteur
Pr. Nathalie JULIEN	, Rapporteur
Pr. Marc RENAUDIN	, Directeur de thèse
Dr. Gilles SICARD	, Co-encadrant
Dr. Laurent FESQUET	, Co-encadrant
Dr. Katell MORIN-ALLORY	, Examinateur

Résumé

L'étude des circuits asynchrones est un secteur dans lequel de nombreuses recherches ont été effectuées ces dernières années. Les circuits asynchrones ont démontré plusieurs caractéristiques intéressantes comme la robustesse, l'extensibilité, la faible consommation ou le faible rayonnement électromagnétique. Parmi les différentes classes de circuits asynchrones, les circuits quasi-insensibles aux délais (QDI) ont montré des caractéristiques extrêmement intéressantes en termes de faible consommation et de robustesse aux variations PVT (Process, Voltage, Temperature). L'usage de ces circuits est notamment bien adapté aux applications fonctionnant dans un environnement sévère et pour lesquelles la consommation est un critère primordial. Les travaux de cette thèse s'inscrivent dans ce cadre et visent la conception et la synthèse de machines à états asynchrones (QDI) faiblement consommantes. Une méthode de synthèse dédiée à des contrôleurs asynchrones à faible consommation a donc été développée. Cette technique s'est montrée particulièrement efficace pour synthétiser les contrôleurs de grande taille. La méthode s'appuie sur une modélisation appropriée des contrôleurs et une technique de synthèse dirigée par la syntaxe utilisant des composants spécifiques appelés séquenceurs. Les circuits obtenus ont été vérifiés formellement afin de s'assurer de leurs propriétés en termes de robustesse et de correction fonctionnelle. A cette occasion, une méthode de vérification formelle a été mise en place pour valider les contrôleurs d'une part, et plus généralement, n'importe quel circuit asynchrone d'autre part. Cette technique fait appel à une modélisation hiérarchique des circuits asynchrones en PSL et à un outil de vérification formelle (RAT).

Abstract

The study of asynchronous circuits is an area where much research has been conducted in recent years. Asynchronous circuits have shown several interesting features like robustness, scalability, low consumption or low electromagnetic radiation. Among the different classes of asynchronous circuits, Quasi Delay Insensitive circuits (QDI) showed very interesting characteristics in terms of low power consumption and robustness to variations of PVT (Process, Voltage, and Temperature). The use of these circuits is particularly well suited for applications operating in a critical environment and for which consumption is paramount. In this framework, the work of this thesis aims the low power consumption design and synthesis of asynchronous state machines (QDI). A method for synthesizing low-consumption asynchronous sequential controllers has been developed. The method relies on an adequate modeling of controllers and a direct mapping synthesis technique using specific components called sequencers. This technique is suitable for synthesizing large controllers. The circuits obtained are formally verified to ensure their properties in terms of robustness and are proved functionally correct. Thereby, a formal verification method has been implemented to validate the sequential controllers on the one hand, and more generally, any other asynchronous circuit. This technique uses a hierarchical model of asynchronous circuits in PSL and a formal verification tool called RAT.

Je veux connaître les pensées de Dieu ; tout le reste n'est qu'un détail

Albert Einstein

A mes parant où tout a commencé

A ma famille pour tout leur soutien

A mes amis de m'avoir donné un coup de main

A mes animées de m'avoir rendu plus fort

Remerciement

Les travaux de cette thèse ont été réalisés au sein du laboratoire TIMA, sur le site Viallet de l'Institut National Polytechnique de Grenoble. Je remercie la directrice du laboratoire Mme. Dominique Borrione pour son accueil.

Je remercie Marc RENAUDIN, mon directeur de thèse, Professeur des universités, Directeur Technique de l'entreprise TIEMPO, pour m'avoir fait confiance et m'avoir intégré au sein de son équipe CIS, pour son encadrement et ses conseils très précieux.

Je souhaite remercier Gilles SICARD, HDR, Maitre de conférence à l'université Joseph Fourrier, pour son encadrement, son encouragement, sa manière directe et efficace, la confiance qu'il m'a accordée ainsi que ses conseils très précieux dont j'avais besoin et pour son amitié.

Je souhaite remercier Laurent FESQUET, HDR, Maitre de conférence à l'Institut Polytechnique de Grenoble, pour son encadrement, son engagement dans ce projet, ses corrections, son encouragement, ses conseils très précieux et pour son amitié.

Je tiens à remercier chaleureusement Katell MORIN-ALLORY, Maitre de conférence à l'Institut Polytechnique de Grenoble, pour son engagement dans ce projet et pour son aide à me faire comprendre un domaine loin de mes connaissances : « la preuve formelle ». Je la remercie pour tous ses conseils, encouragement, corrections et pour son amitié.

Je souhaite remercier les membres de mon jury de thèse :

Pr. Bruno ROUZEYRE, professeur à l'université de Montpellier, pour m'avoir fait l'honneur de participer à mon jury de thèse en tant que rapporteur.

Pr. Nathalie JULIEN, professeur à l'université de Bretagne Sud, pour m'avoir fait l'honneur de participer à mon jury de thèse en tant que rapporteur.

Pr. Ian O'CONNOR, professeur à l'Ecole Centrale de Lyon, pour m'avoir fait l'honneur de présider mon jury de thèse.

J'adresse mes remerciements aux personnes de TIMA qui m'ont aidé dans une multitude de taches administratives pendant ces années. Merci en particulier à Isabelle, Chantal, Anne-Laure, Joëlle, et Sophie M. pour leur gentillesse.

Un grand merci à Alexandre CHAGOYA, le soldat inconnu derrière tous nos projets de recherche, pour son amitié, sa gentillesse et son soutien.

Merci à Stéphane MANCINI maitre de conférence à l'INPG, Olivier ROUSETTO maitre de conférence à l'université Joseph Fourrier, le personnel du service électrique du laboratoire LPSC et en particulier Joël, pour leurs soutiens pendant mes stages de Masters.

Je remercie également Pr. Dominique DALLET et Lilian BOSSUET, HDR à l'IMS-Bordeaux pour leurs soutiens et compréhension pendant les six mois de mon poste d'ATER à Bordeaux.

Merci Eslam pour tout, d'être aussi honnête et sincère dans le moindre de tes gestes, tu défends avec un tel enthousiasme tes valeurs morales, d'être aussi intelligent, cultivé et ouvert. Tu es quelqu'un de bien et une personne rare.

Merci Jérémie pour tout, d'être mon camarade de bureau, de m'avoir supporté lorsque j'étais insupportable, d'être aussi perfectionniste et sérieux dans ton travail, de croire aux bonnes valeurs, d'avoir partagé autant de musique et de bons moments avec moi.

Merci Amr pour ton encouragement, ton amitié et ton soutien, je te dois beaucoup de choses.

Merci Hakim et Oussama pour votre amitié, vos encouragements, nos discussions et toutes les activités que nous avons partagées, pour vos gentillesses et vos simplicités.

Merci Franck pour ton savoir, tes connaissances très larges sur la langue française, nos discussions culturelles et tes corrections.

Merci à l'équipe CIS : David, Grégory, Livier, Cédric, Florin, Alexandre, Hatem, Hawraa, Taha, Christophe, Olivier pour les bons moments et les mauvais qu'on a passés ensemble.

Merci Simonne, Guillermo, Maria-Elena, Kathy, Cécilia et tous les autres danseurs de tango pour leur amitié, pour tant de soirées dansantes et pour m'avoir appris beaucoup de choses sur la vie, l'amour, la danse et l'amitié.

Merci Cyril et Maria pour votre amitié et soutien dès ma première semaine à Grenoble, pour tant de soirées et de sorties ensemble, Noël, nouvel an, les montagnes, la neige, les repas, Barcelone, tant de souvenirs que je n'oublierai pas.

Merci à Jorge et Ghalia d'être en harmonie avec la vie, avec autant de magie, merci pour les soirées, les fêtes, les concerts et les pique-niques.

Merci Adeline la « 3yshtar » pour beaucoup de choses, ton soutien, ta gentillesse, ta générosité, ton amitié, tes encouragements, la simplicité, l'amour et pleins de corrections du manuscrit et pleins de discussions, soirées, sorties qu'on a partagées. Tu es quelqu'un d'adorable qui ne mérite que des bonnes choses.

Merci Xavier pour tout, pour les montagnes, les balades, les soirées, le cinéma, les spectacles, les pique-niques, les bières, les mauvais et les bons moments, les corrections, ton soutien, ton amitié, ton optimisme, ta simplicité, Nice, Lozère et plein de choses.

Merci Emilie, Laurent, Steven, Ronan, Marie, Viridianna, Arnaud pour l'amitié et toutes les soirées et les sorties ensemble.

Merci Alvaro, Isabelle et Cécile pour vos soutiens et l'amitié et les bons repas et les pique-niques.

Merci Hussam, Riad, Oula pour ce chemin qu'on a partagé chacun à sa manière, nous avons eu des hauts et des bas mais tout s'est arrangé enfin.

Merci Christophe

Merci Sébastien

Merci pour tous ceux qui ont croisé mon chemin durant ces 7 dernières années, le 1 septembre 2003 ce projet a commencé et depuis un siècle a passé. Il y a eu des hauts et des bas, certains m'ont partagés des hauts et certains des bas et d'autres les deux. Ce que je sais, c'est que grâce à tout ces gens-là j'en suis là ou je suis, merci ainsi à vous tous.

SOMMAIRE

TABLE DES FIGURES

CHAPITRE 1 : INTRODUCTION

CHAPITRE 2 : ETAT DE L'ART

TABLE DES TABLEAUX

CHAPITRE 1: INTRODUCTION

With fame I become more and more stupid, which of course is a very common phenomenon.

La seule chose absolue dans un monde comme le nôtre, c'est l'humour

Albert Einstein

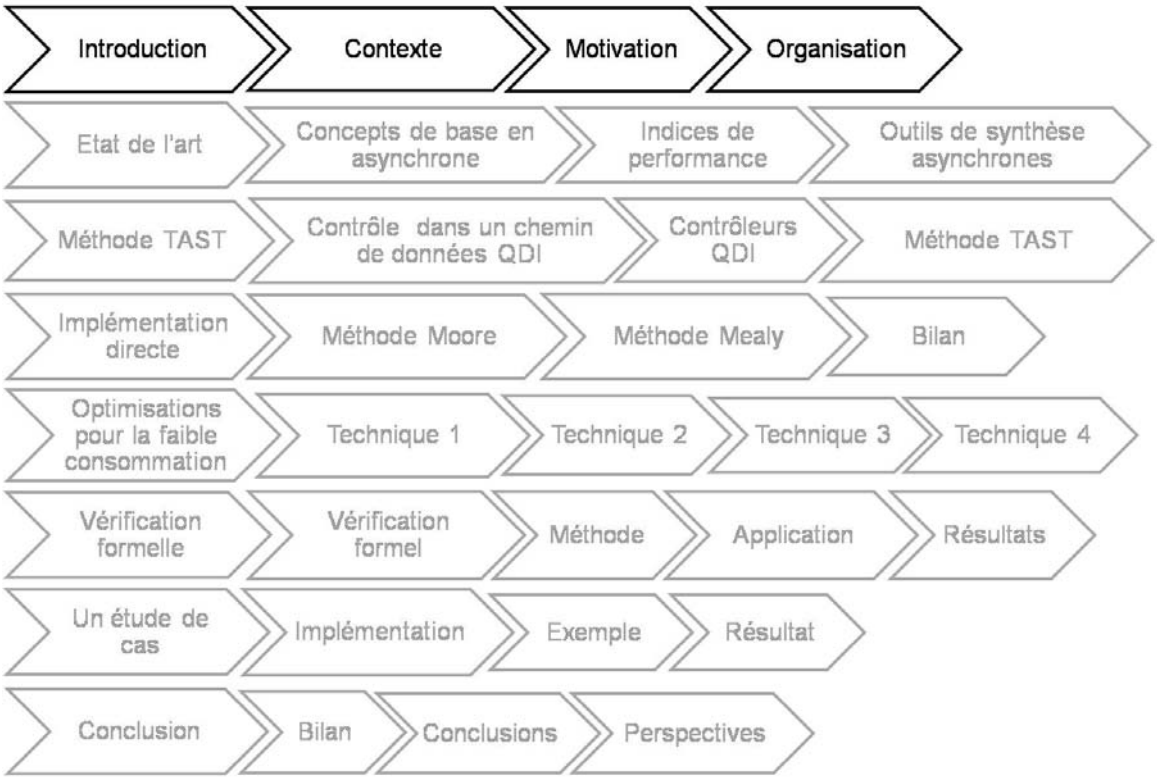


Figure 1: Plan de thèse

Sommaire

Table des figures :

1. Contexte général :

L’asynchronisme dans les circuits électroniques est un concept intéressant, et cela pas uniquement pour ses avantages en termes de réduction de la consommation, l’élimination de l’arbre d’horloge ou les problèmes liés à la conception synchrone (« skew », synchronisation, .etc.) ; mais également en raison des méthodes de conception des circuits asynchrones qui se distinguent des méthodes de conception synchrone habituelles. Le circuit n'échantillonne pas seulement ses données sur un front montant ou descendant de l’horloge, mais est susceptible de le faire à tout instant. Le circuit est actif dès qu'un événement se produit. On parle ici de circuits *event driven* ou *data driven*. En conséquence, il est impératif que la détection des évènements ne soient pas perturbée par du bruit ou des aléas. Les aléas ne sont donc pas autorisés et le concepteur doit, pour réaliser un circuit fonctionnel, concevoir des circuits ne produisant pas d'aléa. L’absence d'aléa et d’arbre de l’horloge constitue un avantage indéniable en terme de consommation électrique. De plus avec les nouvelles technologies, où la variabilité des procédés de fabrication augmente les risques de produire des aléas (dûs à la dispersion accrue des temps de propagation) et les efforts pour réduire les risques de *timing closure*. De plus, l’arbre d’horloge peut représenter une part non négligeable de la consommation d'un circuit (jusqu'à 30% à 40% de la consommation) et une surface également importante. L'impact de l'arbre d'horloge ne peut donc pas être minimisé et les contraintes liées à son usage dans les technologies à venir risquent fort d'être augmentées.

Avec la réduction de la taille des composants dans les technologies modernes, il paraît important de s'affranchir autant que possible des limitations introduites par la variabilité des procédés de fabrication, mais aussi d'offrir une robustesse suffisante aux variations de tension et température. En effet, la course aux dimensions nanométriques a introduit une sensibilité relative plus importante à la tension du fait de la réduction des tensions nominales dans les technologies avancées (environ 1 V). L’alternative asynchrone parait donc très intéressante car les circuits asynchrones, de part leur synchronisation locale, offrent plus de robustesse vis-à-vis de ces multiples paramètres variables grâce à leurs hypothèses temporelles moins fortes que celles des circuits synchrones.

En revanche, un circuit asynchrone est probablement plus complexe à modéliser mais sa conception demeure simple grâce notamment à sa synchronisation locale et à sa modularité intrinsèque. Il est ainsi très facile d’ajouter un composant à un système existant sans que cela n’altère le fonctionnement global du système. Il n’y a en effet pas d’hypothèse temporelle

générale (chemin critique). Un circuit asynchrone se construit de façon naturelle par l'assemblage de plusieurs blocs tel un LEGO. Pour augmenter le débit d'un circuit, il suffit d'ajouter des *buffers* sur les chemins de données sans pour autant se poser les questions épineuses de synchronisation (chemins multicyles par exemple).

Enfin un circuit asynchrone fonctionne de manière concurrente, c'est-à-dire que plusieurs événements peuvent être traités en parallèle. Il n'y a donc pas de besoin d'attendre un signal de synchronisation pour effectuer une opération. En fait, tout dépend de la disponibilité du composant et des données. La conception asynchrone n'est donc pas difficile en elle-même, mais peut-être de prime abord déconcertant au vu du nombre d'approches possibles. Le concepteur débutant aura donc de la peine à choisir l'approche la plus pertinente [1].

Il existe par conséquent de nombreuses façons de réaliser un circuit et de donc nombreuses familles. L'absence de formation à cette logique et manque d'outils CAO efficaces, permettant à la fois de modéliser les comportements de tels systèmes et de réaliser des implémentations matérielles efficaces en termes de vitesse, surface, consommation reste un handicap majeur pour l'exploitation de ces circuits dans l'industrie de la microélectronique . Ce manque d'outils pour la conception de circuits asynchrone se justifie par une communauté asynchrone dans le monde relativement petite pour pouvoir relever tous ces défis simultanément.

Contexte particulier :

La famille des circuits asynchrones QDI ou quasi insensibles au délai est particulièrement intéressante pour plusieurs raisons. D'abord ces circuits sont très robustes aux variations de procédé de fabrication, de tension et de température grâce à des hypothèses temporelles très relâchée. Dans les circuits QDI, certaines fourches doivent être isochrones, en d'autres termes les délais des branches de ces fourches sont considérés identiques [2-4]. Cette hypothèse est facile à mettre en œuvre en prenant soin pendant l'étape de routage d'équilibrer les longueurs des fils des fourches isochrones. Ces circuits ont démontré également des caractéristiques intéressantes de robustesse. Ils peuvent notamment fonctionner sur une large gamme de tensions qui facilite l'usage de technique DVS (Dynamic Voltage Scaling)[5] dans les systèmes. Cette technique est très utile pour les applications à faible consommation d'énergie. Ce type de circuits montre également une puissance moyenne et un rayonnement électromagnétique faible par rapport à un circuit synchrone, notamment grâce à une activité électrique dont la consommation est repartie dans le temps [6]. La conception des circuits QDI est directe et facile à mettre en œuvre pour

réaliser des circuits complexes comme des microprocesseurs [5]. La synthèse de ces circuits est automatisable car leurs règles de production sont très directes [2].

2. Motivation de cette thèse :

Pour répondre au manque d'outil de conception, le groupe CIS travaille depuis quelques années sur un outil de conception asynchrone QDI : TAST (TIMA Asynchronous Synthesis Tool). Cette thèse constitue une pierre de la conception de ces circuits asynchrones QDI, en particulier ceux dédiés au contrôle séquentiel. Bien que TAST se soit montré efficace pour générer des circuits de chemins de données QDI bien optimisés, la conception de la partie du contrôle n'a pas été encore étudiée. De plus, à notre connaissance, la conception de contrôleurs séquentiels QDI n'a encore pas été traitée ; cette thèse intervient pour répondre donc à plusieurs questions :

1- Comment se représente le contrôle séquentiel dans un circuit asynchrone QDI ?

2- Comment nous pourrions modéliser le comportement d'un contrôleur séquentiel QDI ?

3- Quelle est la méthode la plus efficace pour implémenter des circuits de contrôle séquentiel asynchrone QDI en prenant en compte la robustesse, la consommation, la surface et la vitesse du circuit.

4- Quelles sont les méthodes de synthèse les plus efficaces pour la conception des contrôleurs complexes ?

5- Est-ce que le circuit produit par la méthode de synthèse satisfait la spécification ? Est-ce qu'il est QDI ? Comment pourrions-nous en être sûrs ?

Pour répondre à toutes ces questions, nous avons travaillé sur trois axes ; d'abord nous avons proposé une méthode de synthèse qui pourrait s'intégrer dans le flot TAST et qui permet de modéliser et d'implémenter des contrôleurs séquentiels QDI de manière efficace.

Ensuite, nous avons travaillé sur les composants de la bibliothèque cible afin de trouver des combinaisons logiques optimisées en termes de performance ; finalement nous avons proposé une méthode de vérification formelle qui valide les circuits synthétisés et qui nous permet de vérifier formellement la justesse et la robustesse de ces circuits. La Figure 2 montre le flot de conception TAST avec les trois axes de travail de cette thèse.

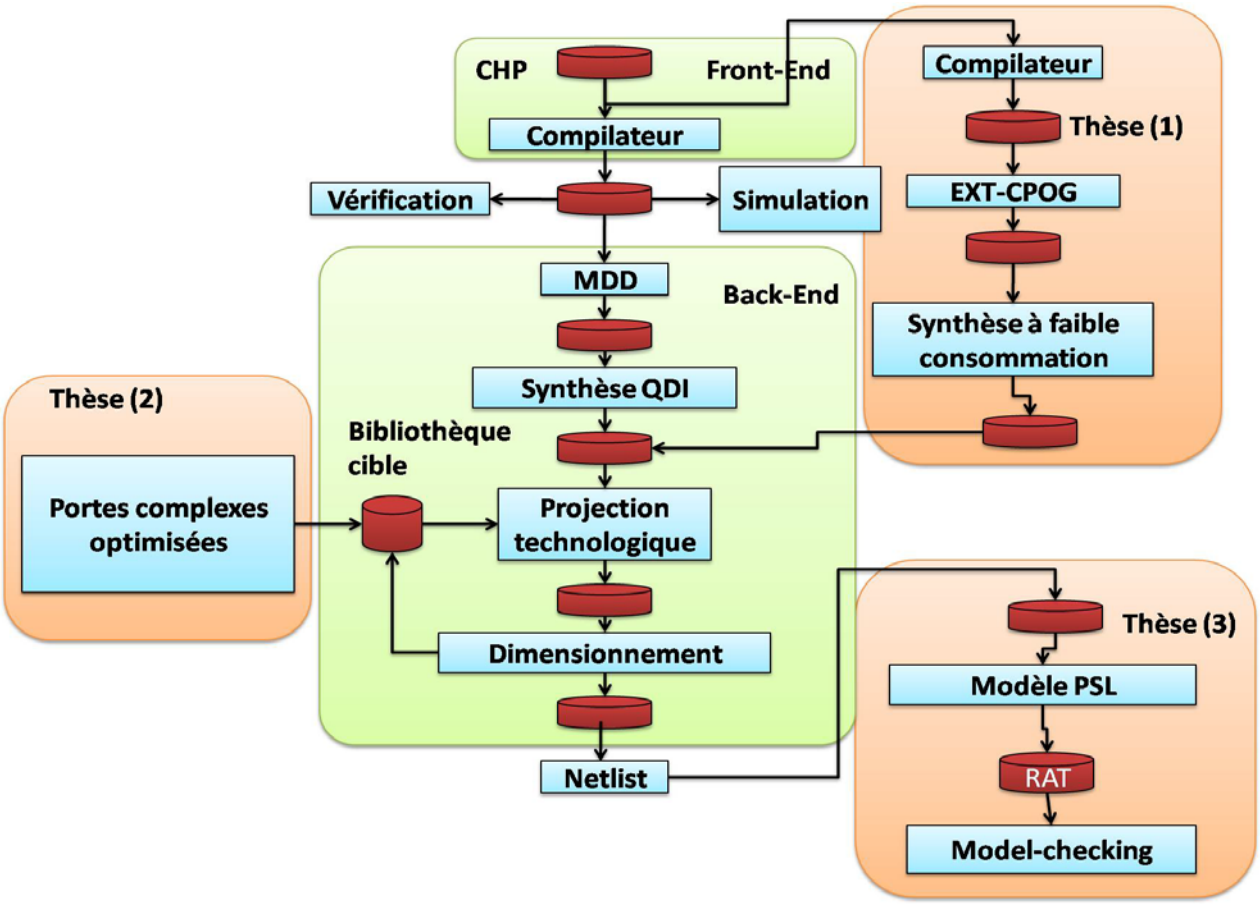


Figure 2: l'intégration du travail de thèse dans le flot de conception TAST

3. Organisation du manuscrit :

Ce manuscrit est un essai pour répondre à toutes ces questions à travers ses 8 chapitres qui sont organisés de la manière suivante.

Dans le chapitre 2, nous présentons les concepts de base des circuits asynchrones, comme l'architecture des circuits, le protocole de synchronisation, les familles asynchrones et les notions qui sont utiles pour comprendre la méthode proposée. Nous présentons aussi un groupe de paramètres nécessaires par la suite pour évaluer l'efficacité des méthodes proposées et pour les comparer les unes aux autres afin de tirer une conclusion sur la méthode la plus efficace. Nous présentons ensuite plusieurs approches de synthèse asynchrone en se focalisant sur la méthode qui concerne la partie du contrôle séquentiel ou les machines à états asynchrones.

Le chapitre 3 détaille la première méthode étudiée. La méthode est inspirée du flot de conception TAST, elle consiste en plusieurs étapes : une modélisation HDL par un langage de haut niveau dédié à la conception asynchrone ; une traduction en format intermédiaire (MDD) un diagramme à décisions multiples, spécialement conçue pour modéliser les circuits

asynchrones QDI ; une synthèse par des portes logiques à deux entrées et une projection technologique sur une bibliothèque de composants. Nous traitons également une problématique importante : la gestion des acquittements internes et externes au contrôleur.

Dans le chapitre 4, nous proposons deux autres méthodes de synthèse. Nous profitons des études trouvées dans l'état de l'art sur des contrôleurs similaires, nous exploitons une de ces méthodes de modélisation afin de décrire le comportement des contrôleurs séquentiels QDI. Nous appliquons ensuite des techniques de synthèse simples à mettre en œuvre pour implémenter les circuits cibles ; l'algorithme de synthèse est en effet une implémentation directe de la modélisation qui est sous forme de graphe. Deux méthodologies sont possibles. Nous étudions en détail les deux approches, puis nous présentons les algorithmes nécessaires pour optimiser la description et pour traduire le modèle en circuit en exploitant un composant séquentiel appelé séquenceur. Dans ce chapitre, nous présentons notre conclusion sur la méthode la plus efficace parmi les trois méthodes présentées.

Chapitre 5 : Après avoir choisi la bonne méthode, nous présentons plusieurs techniques pour la réduction de la consommation. La première méthode consiste à remplacer certaines combinaisons logiques par des fonctions logiques équivalentes plus avantageuses en termes de consommation et de surface. La deuxième technique est une exploitation des portes dissymétriques efficace dans certains cas pour réduire la consommation et la surface. La troisième méthode est une réalisation des implémentations des structures proposées sous forme de portes complexes, ce qui est intéressant du point de vue surface, vitesse et consommation. La quatrième approche est une évaluation de la technique DVS sur le type de circuits présenté dans cette thèse.

Le chapitre 6 est consacré à la vérification formelle des circuits asynchrones, en particulier les circuits ciblés par ce travail de thèse. Nous présentons un rapide état de l'art sur la vérification formelle des circuits asynchrones et quelques concepts de base utiles pour comprendre la méthode adaptée ici. La méthode consiste en une vérification hiérarchique du circuit par un outil de vérification de Modèle et par la modélisation du circuit dans un langage de description de propriétés. La méthode se compose de plusieurs étapes qui sont présentées en détails.

Chapitre 7 : Afin de montrer l'intérêt des méthodes proposées, nous implémentons un contrôleur séquentiel qui met en évidence leurs avantages pour des exemples plus larges et plus complexes. L'implémentation est un circuit de contrôle pour un registre à décalage configurable. Les techniques de réduction de l'énergie sont également évaluées à travers cet exemple. La

méthode de synthèse est aussi vérifiée formellement en appliquant la méthode proposée au chapitre 6 sur un exemple de circuit synthétisé.

Chapitre 8 : Enfin, la conclusion dresse le bilan du travail effectué et propose des perspectives à ce travail.

4. Bibliographie :

1. Hauck, S., *Asynchronous Design Methodologies : An Overview.* Proceeding of the IEEE, 1995. **83**: p. 66-93.
2. Bregier, V., *Automatic synthesis of Asynchronous Proven Quasi Delay Insensitive Circuits.* PHD thesis, Institut National Polytechnique Grenoble, 2007.
3. Folco, B., *Contribution à la synthèse des circuits asynchrones Quasi Insensibles aux Délais, application aux systèmes sécurisés.* These, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2007.
4. J. Fragoso, G.S.a.M.R., *Automatic generation of 1-of-M QDI asynchronous adders.* SBCCI 2003 Proceedings, 16th Symposium on Integrated Circuits and Systems Design., 2003: p. 149-154.
5. Rios, D., *Systeme a microprocesseur asynchrone basse consommation* These Docteur-Ingénieur, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2008.
6. Slimani, K., *Une Méthodologie de Conception de Circuits Asynchrones à Faible Consommation d'Energie: Application au Microprocesseur MIPS* These, Institut National Polytechnique de Grenoble - INPG, 2004.

CHAPITRE 2: ETAT DE L'ART

The crippling of individuals I consider the worst evil of capitalism. Our whole educational system suffers from this evil. An exaggerated competitive attitude is inculcated into the student, who is trained to worship material success as a preparation for his future career.

Although I am a typical loner in my daily life, my awareness of belonging to the invisible community of those who strive for truth, beauty, and justice has prevented me from feelings of isolation.

There is only one road to human greatness: through the schools of hard knocks.

Albert Einstein

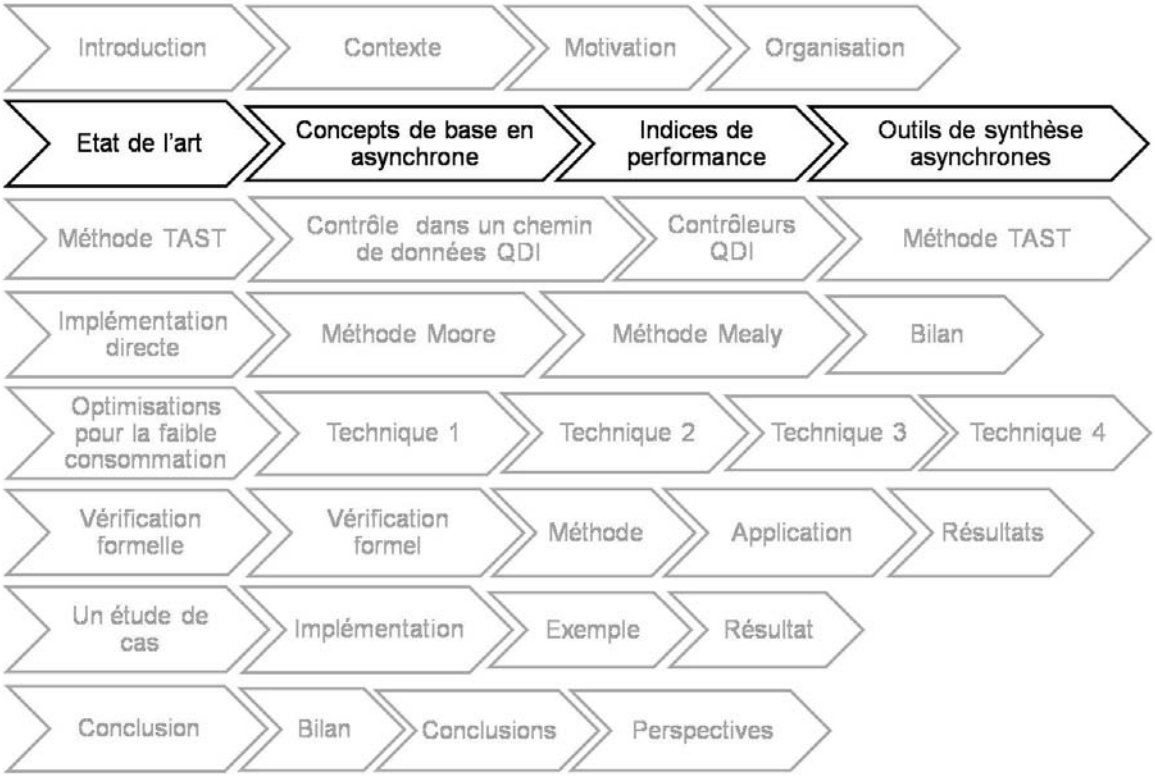


Figure 1: Plan de thèse

Sommaire

Table des figures :

Table des tableaux

1. Introduction :

Avec l'utilisation des technologies récentes, les concepteurs sont confrontés à de nouveaux problèmes tels que les interférences électromagnétiques, la variabilité des procédés de fabrication, qui s'ajoutent aux contraintes habituelles telles que la consommation électrique, ou les paramètres environnement aux (température et tension d'alimentation). Les caractéristiques intrinsèques des circuits asynchrones en font une alternative intéressante [1-2]. De ce fait, de nombreux travaux sont menés dans la recherche et l'industrie pour évaluer ce type de technologie [3-4]: (1) faible consommation d'énergie, (2) pas de problème de distribution du signal d'horloge (3), grande robustesse (4), faible émission électromagnétique, (5), meilleures modularité et tolérance contre la variabilité des procédés de fabrication, tension (Voltage) et température (PVT).

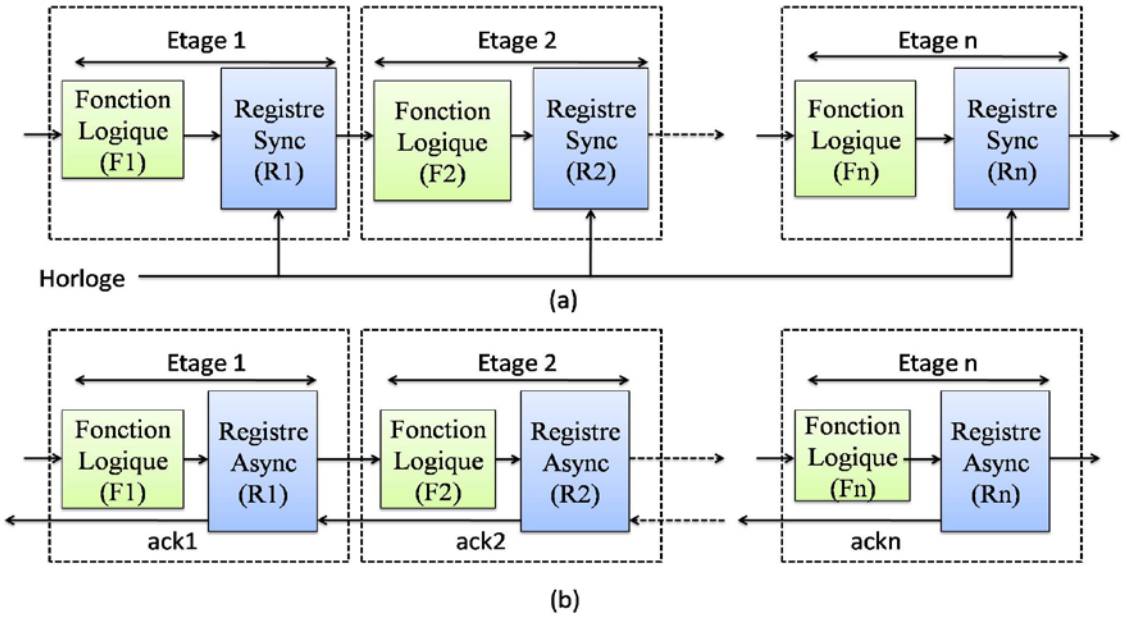


Figure 2: Conception synchrone (a) vs asynchrone (b)

Le modèle de conception synchrone est montré à la Figure 2-a. D'une part, la fonctionnalité du circuit est réalisée par des blocs combinatoires, d'autre part, les registres synchrones échantillonnent les résultats de ces blocs. Un signal d'horloge global contrôle l'échantillonnage de ces registres. La période de l'horloge est fixée de telle sorte que tous les blocs fonctionnels finissent leurs activités et stabilisent leurs données de sorties qui sont ensuite échantillonnées. Cela implique une hypothèse temporelle globale de type pire cas qui est appliquée à l'ensemble du circuit.

En revanche, la synchronisation dans les circuits asynchrones (Figure 2-b) est mise en œuvre par un protocole dit de poignée de main. Ce protocole contrôle la communication entre des blocs

fonctionnels adjacents. Cette synchronisation locale évite l'hypothèse temporelle générale; cette synchronisation locale est la raison principale des avantages des circuits asynchrones.

Les circuits asynchrones sont souvent décomposés en blocs fonctionnels entre lesquels communiquent des données (dits jetons) via des canaux de communication. Un canal de communication définit un moyen d'échange de données point à point entre modules asynchrones. Il se compose d'un paquet de données et d'un protocole de communication (Figure 3). Les échanges de données à travers un canal sont directionnels : de l'émetteur au récepteur. L'émetteur (ou le récepteur) est défini actif s'il est l'initiateur d'un transfert d'information. Il est passif s'il répond à une demande de transfert. Un canal de communication sert aussi à synchroniser les différents modules entre eux. Ce type de canal ne contient aucune donnée, il est également défini comme un canal de contrôle.

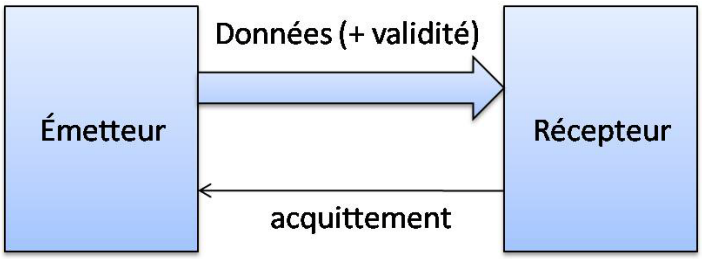


Figure 3: Canal de communication

Les circuits asynchrones peuvent être classés en fonction de leurs hypothèses temporelles, de leur protocole de communication et de leur architecture [3, 5].

Les portes et les fils causent un retard dans un circuit électronique. Un circuit insensible aux délais "DI" est conçu pour fonctionner correctement avec des retards non connus et bornés. Certains circuits contiennent des fourches dans leur connexion, lorsque le retard dans les fils des branches de ces fourches est supposé égal, celles-ci sont appelés fourches isochrones. Le circuit est, dans ce cas, dit Quasi Insensible aux délais ou "QDI". Dans les circuits indépendants à la vitesse, les portes sont supposées avoir des retards non bornés. Toutefois, les fils sont supposés sans retard. Dans les circuits de Huffman ou les circuits micro-pipeline, les délais des portes ou des fils sont présumés connus et bornés.

Les protocoles asynchrones de type poignée de main peuvent être classés en deux grandes catégories [3, 5] : le protocole 2-Phases (Figure 4) et le protocole 4-phases (Figure 5).

Le protocole 2-phases (Figure 4) constitue la séquence d'échange d'information minimale nécessaire à une communication : les données sont représentées par des fronts (montants et descendants).

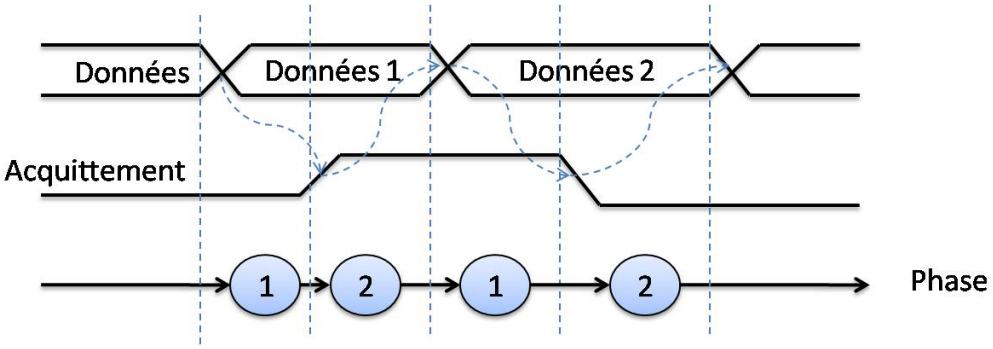


Figure 4: Protocole deux phases

Les deux phases sont les suivantes :

– Phase 1 : c'est la phase active du récepteur qui détecte la présence d'une donnée, effectue le traitement et génère le signal d'acquittement.

– Phase 2 : c'est la phase active de l'émetteur qui détecte le signal d'acquittement et émet une nouvelle donnée si elle est disponible. Ce protocole, malgré son apparente efficacité, n'est que peu utilisé en raison des difficultés rencontrées dans la logique nécessaire à son implantation. L'asymétrie entre les deux phases successives (montée ou descente de l'acquittement) se révèle un obstacle important à la réalisation de ce protocole.

Les différentes phases du protocole quatre phases sont les suivantes (Figure 5) :

Phase 1 : le récepteur détecte une nouvelle donnée, effectue le traitement et génère le signal d'acquittement.

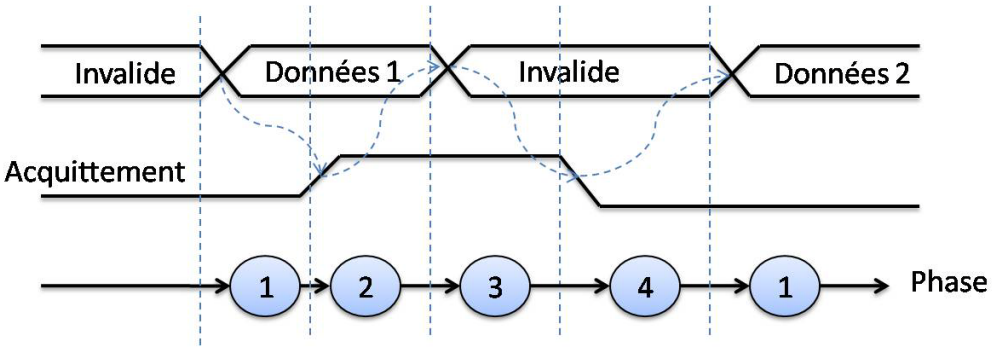


Figure 5: Protocole quatre phases.

– Phase 2 : l'émetteur détecte le signal d'acquittement et invalide les données.

– Phase 3 : le récepteur détecte l'état invalide et désactive le signal d'acquittement.

– Phase 4 : l'émetteur détecte l'état invalide du signal d'acquittement et émet une nouvelle donnée si elle est disponible. A l'opposé du protocole deux phases, le protocole quatre phases est le plus utilisé en raison de la symétrie de ses phases. En doublant leur nombre, l'état du système est invariablement le même au début et à la fin d'une communication.

Comme nous venons de le voir, les protocoles de communication doivent détecter la présence d'une donnée sur leurs entrées et leurs disponibilités. Pour résoudre ce problème, il est nécessaire d'adopter un codage particulier pour les données. Le codage 3 états, appelé souvent codage double rails en raison de l'utilisation de 2 fils par bit de données, est un cas particulier du codage « one hot ». Pour étendre cette représentation et représenter un digit en base N, N fils sont alors utilisés : chaque fil représente une valeur et l'état invalide est déterminé par la remise à zéro de tous les fils. Les combinaisons où au moins deux fils sont à « 1 » sont interdites. Ce codage, représenté dans la Figure 6, est appelé dans ce manuscrit codage « 1-parmi-N », ou codage multi-rails. Il est également possible d'étendre ce codage au codage « M-parmi-N ».

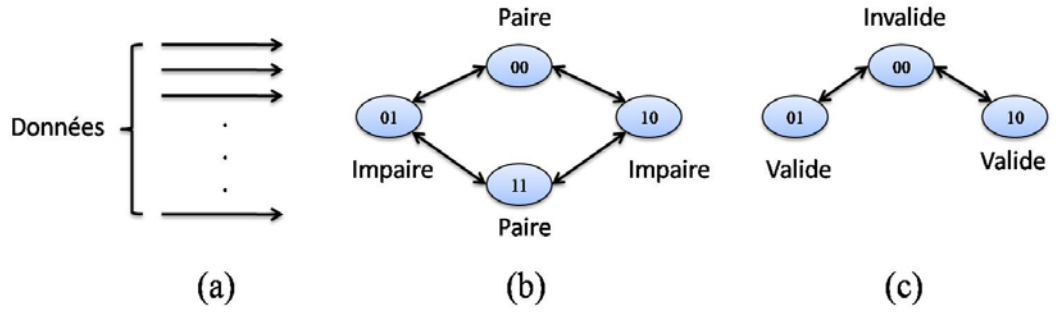


Figure 6 : Codage insensible au délai 4 états (b) 3 états (c)

D'un point de vue structurel, les circuits asynchrones peuvent être classés en deux catégories principales : Pipelines Linaires "LP" et Pipelines Non Linéaires "PNL". Les structures de base des circuits asynchrones sont présentées dans les Figure 7 et 8. La fonction "F" est l'équivalent asynchrone de circuits combinatoires. Il est transparent pour les signaux du protocole de communication. Les registres asynchrones représentent les tampons de stockage des jetons de données, elles gèrent l'exécution du protocole de connexion qui contrôle le débit de jeton. Les registres peuvent être des registres linéaires "R" comme celui qui est représenté dans la Figure 7(b). Ce registre a un canal d'entrée unique ainsi qu'un canal de sortie unique.

Il stocke le jeton reçu par l'entrée et ensuite il répond par un acquittement sur ce canal d'entrée. De même, si ce jeton est transmis à un autre registre, ce dernier confirme la réception par

un acquittement du canal de sortie. Lorsque ce registre est le producteur de jetons, il a seulement un canal de sortie et est appelé Transmetteur "TX" (Figure 7 (c)). Toutefois, dans le cas où ce registre est uniquement un consommateur de jetons, il a seulement un canal d'entrée et est appelé récepteur «Rx» (Figure 7 (d)).

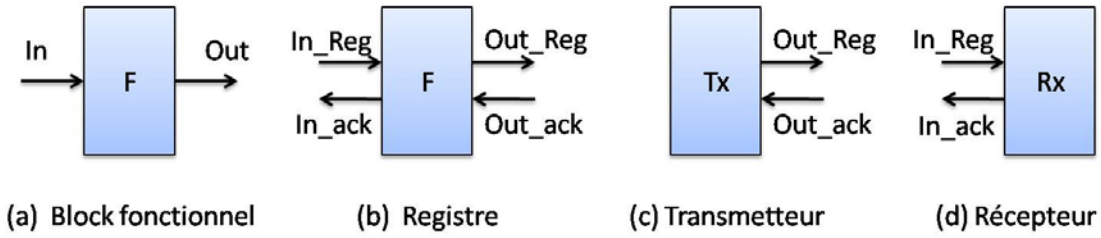


Figure 7: Structures asynchrones linéaires de base

Si l'entrée du jeton est dupliquée et distribuée sur des canaux de sortie multiples, ce registre est appelé Fourche "FK", (Figure 8 (a)).

Au contraire, si des jetons de multiples entrées sont traités et injectés à un canal de sortie unique, le registre est dit convergent "J"(*join*), (Figure 8 (b)). Si le registre injecte le jeton d'entrée à un canal de sortie sélectionné (qui est déterminé par une entrée de commande), ce registre est appelé Démultiplexeur "S" (Split), (Figure 8 (c)). En revanche, si le registre sélectionne une entrée, qui est déterminée par une entrée de commande, et l'injecte ensuite à son canal de sortie, ce registre est appelé Multiplexeur "M" (Merge) (Figure 8 (d)).

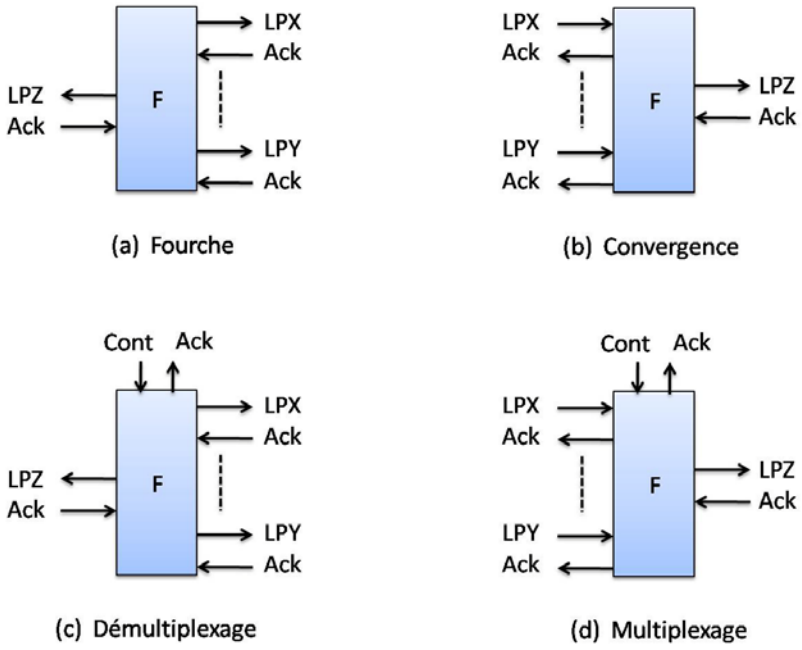


Figure 8: Structures asynchrones non linéaires de base

En utilisant ces structures de base, les concepteurs sont capables de mettre en œuvre leurs circuits asynchrones. Dans certains classements, les circuits qui sont composés de composants (F, R, Tx, Rx) sont appelés Pipelines linéaires. Les circuits qui contiennent des composants (F, J) sont appelés Pipelines non contrôlées non linéaires. Lorsque les circuits contiennent des composants (S, M), ils sont de type Pipelines non linéaires contrôlés. D'autres littératures [6] classent les circuits qui ne contiennent pas (S, M) comme des pipelines déterministes. Alors qu'ils sont appelés Pipelines non-déterministes lorsque (S, M) sont utilisés dans leur conception.

2. Implantations de la porte de Muller :

Pour implanter les protocoles asynchrones, les portes logiques élémentaires ne suffisent pas. Ce paragraphe présente rapidement le fonctionnement des portes de Muller (autrement appelées «C élément », proposées par Muller dans [7]. Elles sont essentielles pour l'implantation de circuits asynchrones : elles réalisent le rendez-vous entre plusieurs signaux. La sortie copie la valeur des entrées lorsque celles-ci sont identiques, sinon elle mémorise la dernière valeur calculée. La Figure 9 (a) représente le symbole d'une porte de Muller à deux entrées et sa spécification sous forme de table de vérité (b) :

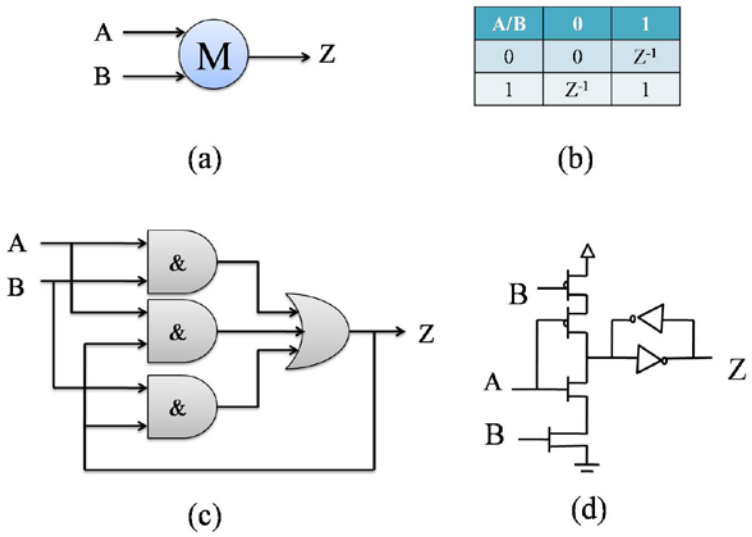
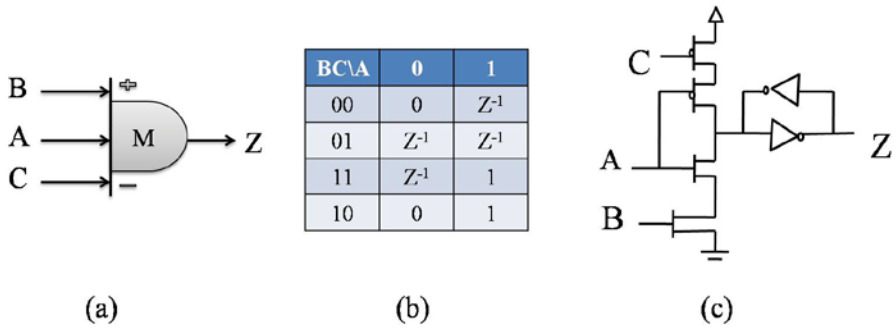


Figure 9 : Porte de Muller

A partir de cette spécification, la Figure 9 donne plusieurs implantations possibles, sous forme de portes logiques élémentaires (c) et sous forme de transistors (d).

Il existe des variantes de la porte de Muller [8]. La porte de Muller généralisée est une porte de Muller dans laquelle les signaux qui font monter la sortie à « 1 » ne sont pas toujours les mêmes que ceux qui la font descendre à « 0 ». Dans ce cas, la porte de Muller est également dite

dissymétrique. A titre d'exemple, la Figure 10 illustre une porte de Muller généralisée (son symbole, sa spécification et sa réalisation au niveau transistor). Dans cet exemple, il suffit que les entrées « B » et « C » aient la valeur « 1 » pour que la sortie passe à « 1 » et que les entrées « A » et « B » aient la valeur « 0 » pour que la sortie passe à « 0 ». Sinon la sortie est mémorisée.



BC\A	0	1
00	0	Z^{-1}
01	Z^{-1}	Z^{-1}
11	Z^{-1}	1
10	0	1

Figure 10 : Porte Muller dissymétrique

3. Performance et analyse :

L'objectif de cette thèse est de réaliser des contrôleurs séquentiels asynchrones robustes à faible consommation. Afin de pouvoir évaluer les méthodes ou les architectures proposées, il est important d'étudier les paramètres de conception d'un circuit intégré : consommation, robustesse, etc. Évidemment, pour des questions de coûts et délais, il est impossible de concevoir, fabriquer et tester l'ensemble des solutions envisagées. En conséquence, les simulations électriques des circuits au travers des modèles, au niveau transistor, sont nécessaires.

3.1. Robustesse :

La robustesse d'un circuit contre les variations (PVT) dépend de plusieurs facteurs, et le choix du modèle de délai et du mode de fonctionnement forme un compromis entre la complexité et la robustesse des circuits. La Figure 11 présente la terminologie habituellement utilisée pour classifier les circuits asynchrones [9]. Plus le nombre d'hypothèses temporelles est élevé (contraintes sur les délais relâchés), plus simple sera le circuit. Cependant, dans ce dernier cas, le circuit sera moins robuste.

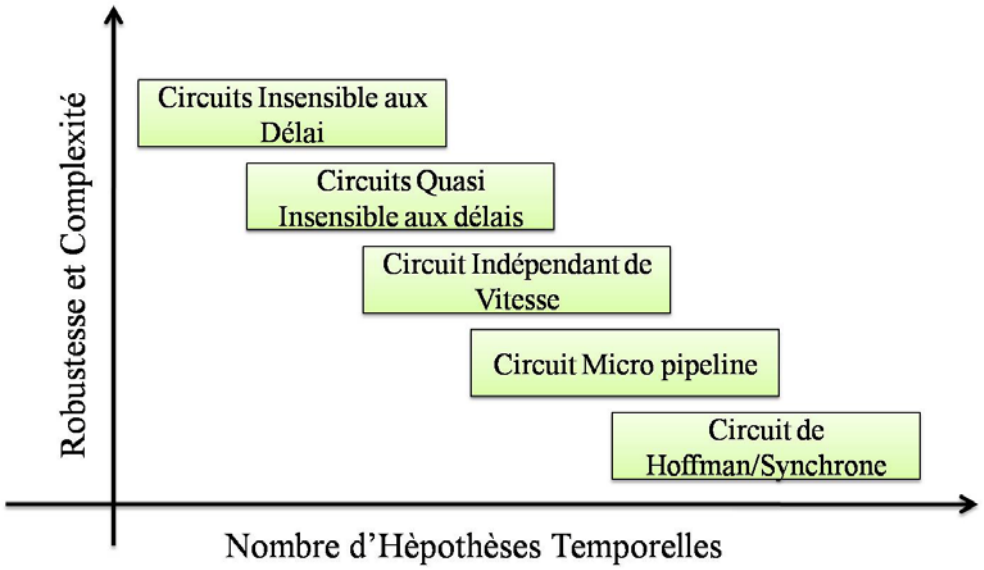


Figure 11: robustesse vis-à-vis du nombre d'hypothèses temporelles

Le circuit de Huffman [10], comme les circuits synchrones, se base sur un modèle de délais bornés et connus dans les portes et les fils. Les circuits synchrones fonctionnent notamment avec une hypothèse globale : la période de l'horloge doit être supérieure au chemin critique.

Les circuits micropipeline se basent sur le même principe (délai connu et borné), en revanche, ils adoptent des hypothèses temporelles localisées à chaque bloc combinatoire (se situant entre des registres). En effet, les délais des signaux de requêtes et d'acquittements sont calculés en fonction de ce chemin critique local à chacun de ces blocs.

Les circuits indépendants de la vitesse présument que les délais dans les fils sont nuls, autrement dit, que les délais dans les fourches sont identiques (toutes les fourches sont isochrones) et que les délais dans les portes sont bornés mais non connus.

Les circuits Quasi insensible aux délais se basent également sur les principe des fourches isochrones, par contre toutes les fourches ne sont pas considérées comme isochrones, d'où le fait que ces circuits sont plus robustes que les circuits insensibles à la vitesse.

Les circuits insensibles aux délais sont des circuits qui n'utilisent pas d'hypothèses temporelles, les délais ne sont ni bornés ni connus, ni dans les fils ni dans les portes. Une telle hypothèse limite les circuits réalisables.

En résumé, les circuits QDI présentent l'avantage d'être très robustes mais aussi réalisables dans le sens où toutes les fonctions peuvent être implémentées d'une façon QDI. Dans cette thèse, on s'intéresse à l'implémentation de contrôleurs séquentiels robustes de type QDI. Nous

mesurerons la robustesse par le nombre de fourches isochrones considérés dans un circuit : moins il y a de fourches isochrones, plus les circuits sont robustes. Un circuit où toutes les fourches sont considérées non isochrones est un circuit insensible aux délais.

3.2. Consommation d'énergie :

Il est nécessaire de comprendre les sources de consommation dans un circuit électronique afin de diminuer sa consommation. Il existe trois sources de consommation présentes dans les circuits électroniques, composés de transistors CMOS. Les trois sources de consommation sont les suivantes

- Courant de fuite

- Courant de court circuit

- Courant de charge des capacités

a. Courant de fuite :

Cette source de consommation dépend de la technologie employée afin de fabriquer le circuit. Il se compose de deux éléments, d'abord le courant de fuite des jonctions dans les transistors. Celui-ci apparaît entre la source ou le drain à travers les diodes de polarisation inverses lorsque le transistor est bloqué. Le deuxième est le courant sous le seuil qui est le courant drain-source du transistor bloqué. L'ordre de grandeur du courant sous le seuil est fonction de la température, de la tension d'alimentation, de la taille et des paramètres de fabrication pour lesquels la tension d'alimentation joue un rôle très important. Dans les technologies inférieures à 65nm, le courant de seuil est prédominant par rapport aux autres courants de fuites dans les systèmes CMOS. Ce courant est donné par l'Équation 1 [11], il faudra bien sûr tenir compte que ce courant commence à avoir le même ordre de grandeur que le courant dynamique. Le courant de fuite (puissance statique) peut représenter jusqu'à 50% de la consommation totale du circuit.

$$I_{DS} = K\left[1 - e^{-\frac{V_{DS}}{V_T}}\right] e^{\frac{(V_{GS} - V_T + \eta V_{DS}}{nV_T}}$$

Équation 1 : Courant de seuil dans un transistor CMOS

K et n sont fonctions de la technologie et η est le coefficient de réduction dû à la barrière induite par le drain. La Figure 12 [12] montre le schématique d'un inverseur avec les tensions de polarisation. La diminution de la taille des transistors et de la tension de seuil augmente le courant

de fuite pour les nouvelles technologies. Enfin, les courants de fuite par effet tunnel au travers de la grille d'un transistor représentent également une valeur qui peut s'avérer importante (jusqu'à 1/3 de la consommation statique).

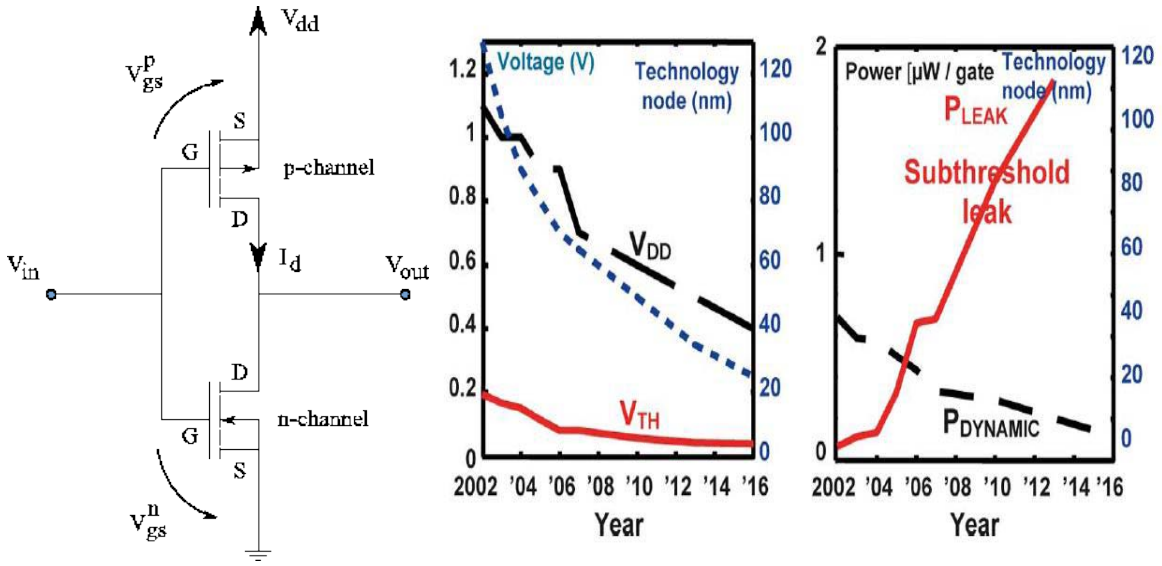


Figure 12: Schématique d'un inverseur et tendance des consommations dans les nouvelles technologies

b. Courant de court circuit

Dans un circuit CMOS, les réseaux de transistors nMOS et pMOS ne conduisent pas en même temps. Leur fonctionnement est complémentaire, et la sortie d'une porte est donnée soit par le réseau nMOS soit par le réseau pMOS en fonction du niveau logique souhaité. Lorsque la sortie doit basculer, les transistors nMOS et pMOS doivent passer d'un état à un autre. Comme les transistors sont complémentaires, ils devront inverser leurs états de « passant » à « bloqué » et vice versa en fonction de la nouvelle valeur d'entrée. Ce type de fonctionnement implique l'existence pendant un (très court) moment d'un court circuit entre l'alimentation et la masse lorsque les deux transistors sont dans leur région saturée. Dans un inverseur, la consommation de court circuit est proportionnelle au temps de montée/descente de la rampe d'entrée, de la charge et de la taille de la grille dans les transistors [11, 13]. Le courant de court circuit est maximum lorsque la charge à la sortie est nulle; ce courant diminue si la charge en sortie augmente. L'estimation du courant de court circuit peut être obtenue à l'aide de quelques équations qui donnent une estimation du courant en fonction des approximations du modèle du courant :

$$P_{sc} = \eta C_{sc} {V_{DD}}^2 f = \eta f \frac{V_{DD}^4 (1-b)^2}{6} \left[\frac{\alpha_1' \beta_p t_{HL(i-1)}}{t_{HL(i-1)} + t_{HL(i)}} + \frac{\alpha_1'' \beta_N t_{HL(i-1)}}{t_{HL(i-1)} + t_{LH(i)}} \right]$$

Équation 2: Puissance de court circuit

L'Équation 2 présentée dans [14] montre la dépendance de la dissipation de puissance de court circuit dans le circuit avec les paramètres de performance comme la taille des transistors, les temps des rampes d'entrée et de sortie et de la charge à la sortie. Cette approche permet de définir la dissipation d'énergie avec une capacité équivalente de court circuit C_{SC}.

c. Courant de charge des capacités

La consommation dynamique continue à être une source de consommation importante dans un circuit CMOS classique. Cette consommation provient de la charge et de la décharge des capacités dans chaque nœud du circuit [11, 15-16]. La puissance dynamique « P » consommée dans un circuit CMOS est donnée par l'équation 3. Avec « P » représentant la puissance consommée, « A » le facteur d'activité, « C » la capacité totale de transition et « F » la fréquence d'horloge du circuit. Cette équation nous montre que si la capacité « C » est chargée et déchargée à une fréquence d'horloge « F » avec une tension d'alimentation « V », alors la charge par cycle est de « CV » et la charge par seconde est de CVF.

La charge étant alimentée avec une tension de « V », l'énergie dissipée par cycle est de CV^2F. L'activité du système ne pouvant commuter qu'une fois par cycle, la puissance dans le circuit est ½(CV^2F). Dans un circuit synchrone, le facteur d'activité « A » est employé lorsqu'on utilise du 'clock gating' ou lorsque les bascules ne sont pas actives au cours de tous les cycles. Ce facteur d'activité peut prendre les valeurs $0 \leq A \leq 1$, avec ce facteur il est possible d'obtenir une moyenne de la consommation de ce type de circuit.

$$P = ACV^2F$$

Équation 3: Puissance dynamique dans un circuit CMOS

Dans un circuit asynchrone, la consommation n'est pas cadencée par le signal d'horloge. Il est possible d'utiliser la même équation en remplaçant la fréquence F par l'inverse du temps que l'on veut calculer. Autrement dit, pour calculer la puissance consommée par le circuit pendant un temps t, il suffit de diviser par t l'énergie E dissipée par le circuit. Le facteur d'activité dans ce

type de circuit représente le nombre de transitions réalisées à la sortie de chaque porte logique. La puissance de chaque porte est ainsi donnée par l'Équation 4 [11, 16]:

$$P = \frac{ACV^2}{t}$$

Équation 4: Puissance consommée par une porte dans un circuit asynchrone

3.3. Délai/débit :

Le modèle dit de l'effort logique [17] décrit le retard d'une porte logique comme dépendant de la capacité que la porte logique pilote, mais aussi de sa complexité. Plus la charge de sortie est importante, plus la porte nécessite un temps élevé pour commuter, mais le retard est également dépendant de la fonction logique implémentée par la porte.

Le modèle de l'effort logique exprime le retard d'une porte comme une variable. Le retard absolu d'un porte (d_{abs}) est donnée par :

$$d_{abs}=d\tau$$

Où τ est le retard d'un inverseur idéal (sans capacité parasite) qui pilote un autre inverseur identique. Il est suffisant de calculer « d » pour comparer plusieurs architectures. A partir d'ici, le mot retard fait référence à la variable « d ».

En conséquence, le retard total est donné par :

$$d = f + p$$

Où f est l'effort de l'étage, il dépend des charges et de la capacité de la porte à piloter ces charges

$$f = gh$$

Où g est l'effort logique, il peut être défini comme étant le rapport entre la capacité d'entrée d'une porte et la capacité d'entrée d'un inverseur qui peuvent délivrer le même courant de sortie. h est l'effort électrique. Il décrit comment l'environnement électrique de la porte affecte sa performance et comment les dimensions des transistors déterminent la capacité de la porte à piloter ses charges. L'effort électrique est défini par :

$$h= (C_{out} / C_{in}).$$

Où C_{out} est la capacité pilotée par la porte logique et C_{in} est la capacité de grille de l'entrée sur laquelle le retard est calculé. Il est important de signaler que C_{in} pour l'effort électrique n'est pas la capacité totale du nœud sur laquelle l'entrée est connectée, mais simplement la capacité de grille.

La principale contribution aux capacités parasites (P) est la capacité des régions de diffusion des transistors connectés à la sortie de la porte logique.

La formulation du retard proposé par [17] est définie par :

$$d_{abs} = (gh + P)\,\tau.$$

Dans ce paragraphe, nous avons montré le modèle de délai pour comprendre les origines des délais dans un circuit CMOS afin de l'optimiser par la ensuite, par contre au cours de cette thèse, le retard d'un circuit est calculé directement par simulation électrique. Le délai « T » d'une fonction logique « F » est calculé comme le temps moyen entre le changement des valeurs d'entrée de « F » à l'acquittement de son buffer de sortie.

Le débit de F est ainsi D=1/T.

3.4. La Métrique Et^n :

Il est important de connaître l'efficacité des architectures en termes de consommation d'énergie et de vitesse d'opération. Pour ce faire, les critères de comparaison qui combinent l'énergie et la vitesse en un unique nombre sont particulièrement intéressantes. En effet, ces métriques sont exprimées par la relation Et^n [18], où « n » transcrit l'importance du paramètre vitesse par rapport à la consommation.

La métrique Et^n permet de comparer :

- Deux solutions à l'égard d'une certaine transformation ;
- L'efficacité d'une optimisation sur deux solutions distinctes ;
- L'efficacité d'une optimisation pour une même solution.

La théorie des circuits montre que la métrique Et^2 est, en première approximation, constante, quelle que soit la tension d'alimentation. Cependant, il est vrai que les concepteurs de circuits synchrones préfèrent utiliser la métrique Et compte tenu de leur prudence à propos du choix de la fréquence d'horloge. Cette métrique est aussi intéressante pour comparer les performances de plusieurs combinaisons logiques.

Dans cette thèse, nous utilisons la métrique « ET » car elle est montrée constante, dans les simulations électriques effectuées, pour des tensions d'alimentation variées, ce qui la rend intéressante pour la classe de circuits asynchrones de type « QDI », car ceux-ci restent fonctionnels sur une large gamme de valeurs de tension d'alimentation.

3.5. Surface :

La première façon d'estimer la surface d'un circuit est de compter le nombre de portes contenues dans la description du circuit. Cette méthode a le défaut qu'elle ne prend pas en compte la complexité des portes utilisées.

Il est possible de capturer la complexité des portes en calculant le nombre de transistors minimum nécessaires à son implémentation. En conséquence, il est facile de définir le nombre total de transistors nécessaires à l'implémentation d'un réseau de portes logiques données.

Le nombre de transistors possède une plus forte corrélation avec la surface que le nombre de portes du réseau. Aussi, le nombre de transistors peut être utilisé comme première approximation de la surface des circuits [16].

Par contre, le nombre de transistors d'une porte ne donne pas d'indice suffisant pour estimer son implémentation car plusieurs portes différentes en surface peuvent avoir le même nombre de transistors. Par exemple une porte « OR » a le même nombre de transistors qu'une porte « AND », alors que leurs surfaces relatives sont différentes. Il en va de même pour les différentes sortances (« drive ») d'une même porte.

Tableau 1: La relation entre la surface d'une porte est la somme des largeurs de ses transistors

	W(relative à AN2LL)	L	√L/W (relative à AN2LL)
AN2LL	1,0	3,035	1,0
OR2LL	1,0	2,97	1,0
M2X0	1,2	3,19	0,9
AN4LL	1,3	4,475	0,9
OR4LL	1,3	5,715	1,0
AON	1,3	5,36	1,0
AO1ALL	1,3	5,87	1,1

En supposant que la surface d'une porte soit la somme des surfaces de ses transistors, elle est, par conséquent, proportionnelle à la somme des largeurs des transistors (L) dans la schématique de porte. La Tableau 1 montre plusieurs exemples de portes de la bibliothèque CMOS 130nm de ST et de la bibliothèque TAL. Ce tableau montre une corrélation entre la surface W et L.

3.6. Le rayonnement électromagnétique et les appels de courant :

Avec l'augmentation de la complexité et de la vitesse de fonctionnement des circuits intégrés, le problème des émissions électromagnétiques devient de plus en plus important. En effet, la consommation de courant lors des commutations du circuit peut atteindre plusieurs ampères pendant quelques nanosecondes, provoquant des émissions parasites en mode conduit et rayonné. Les pics de courant principaux dans un circuit synchrone correspondent au front de l'horloge, alors que dans le cas asynchrone, les pics de courant ne sont pas synchronisés et peuvent arriver à n'importe quel moment. L'émission parasite générée par les circuits intégrés a un rapport direct avec le courant consommé lors de la commutation des portes logiques, c'est la principale source de l'émission parasite au travers des rails d'alimentation VDD et VSS [19].

L'énergie normalisée W(t) d'un signal x(t) s'exprime de la façon suivante :

$$W_x(t_1, t_2) = \int_{t1}^{t2} x^2(t)dt$$

Équation 5 : L'énergie normalisée W(t) d'un signal x(t)

La distribution spectrale du signal x(t) peut-être obtenue grâce à l'application de la transformée de Fourier qui introduit le principe de dualité entre l'espace temps et l'espace fréquence :

$$X(f) = \int_{-\infty}^{+\infty} x(t) \exp(-j2\pi ft)\, dt$$

Équation 6 : La distribution spectrale du signal x(t)

Le bruit généré dépend de l'amplitude du signal x(t), c'est pour cela que le pic du courant est une valeur intéressante à voir afin d'estimer le bruit électromagnétique d'un circuit.

4. Montage électrique :

Il reste important de préciser que tous les circuits réalisés dans cette thèse sont implémentés en technologie CMOS130 de STMicroelectronics. Les simulations électriques sont réalisées soit via le simulateur spectre de Cadence et soit avec l'outil Nanosim de Synopsys (Annexe A).

5. Synthèse asynchrone :

La synthèse des circuits asynchrones est un domaine actif depuis des années, nous présenterons les approches les plus reconnues. Ces approches peuvent être classifiées en deux grandes catégories : celles qui ciblent principalement la partie chemin de donnée, et celles qui ciblent la partie contrôle ou séquentielle.

5.1. Synthèse Asynchrone de parties combinatoires

5.1.1. Méthode Haste et Balsa :

Philips développe des circuits asynchrones de type micro-pipeline à partir du langage Tangram [20], dont le nom s'est transformé en Haste. Haste est en fait un ensemble comprenant un langage de description de haut niveau, également appelé Haste, et un compilateur associé qui permet de traduire les structures du langage en des structures de composants de type « poignée de mains ». Haste utilise une compilation dirigée par la syntaxe (Figure 13).

Balsa [21-22] est une autre approche pour la conception de circuits asynchrones avec un langage de description pour ce type de circuits, qui adopte lui aussi une approche de type compilation dirigée par la syntaxe : les structures de langage sont ciblées directement sur des composants communicants de type « poignée de mains ».

L'avantage de ces deux approches est leur capacité à décrire également à haut niveau des systèmes concurrents, complexes et hiérarchiques et donc d'éviter le problème d'explosion d'états en traduisant des structures de langages directement dans des blocs de circuits fixes(prédéfini). Par contre, Il est difficile d'optimiser un circuit obtenu à partir d'une telle approche de compilation. L'optimisation dans la méthode se fait au niveau primaire : il s'agit de remplacer les structures de composants « poignée de mains » par des équivalents plus efficaces. Les circuits générés par ces méthodes peuvent être inefficaces et redondants puisque les optimisations sont difficiles à appliquer avec une synthèse dirigée par la syntaxe. En effet, comme cette approche repose sur des transformations locales, elle manque de flexibilité en vue d'une optimisation globale. De plus, il n'existe pas de place pour la créativité : une conception pauvre peut facilement donner le même résultat qu'une conception élégante. Il faut que le concepteur ait une bonne connaissance de la conception des circuits asynchrones et ceci même s'il est bon concepteur dans le domaine synchrone. Finalement, il est important de noter que la synthèse dirigée par la syntaxe préserve la correction par construction, mais n'assure pas que le circuit global soit correct. Il est possible que le circuit présente un blocage (« deadlock ») par exemple.

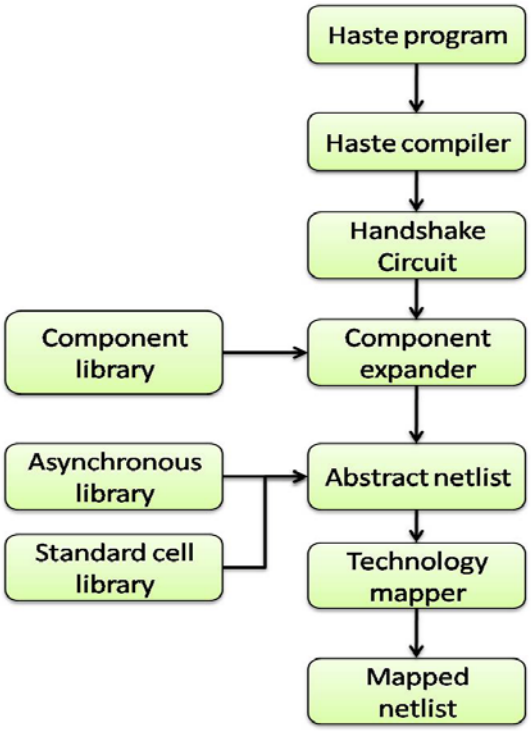


Figure 13: Flot de conception Haste

5.1.2. Méthode Caltech :

Le principe de la méthode « Caltech » [23-24] repose sur la modélisation de processus utilisant des canaux de communication : au plus haut niveau, le protocole est implicite, seules les actions de communication (lecture, écriture) apparaissent. Le développement du code permet de faire intervenir différents types de protocole, de codage et de conventions (choix du processus actif sur le canal, séquencement de la communication) pour aboutir, par étapes successives, à une description explicite des signaux. Sophistiquées et complexes, les étapes de développement préservent la sémantique mais sont pour la plupart effectuées manuellement. Appliquée à plusieurs conceptions dont le microprocesseur asynchrone très connu MIPS R3000 [25], la méthode de synthèse du professeur Alain Martin génère des circuits QDI avec le protocole de communication quatre phases. Elle génère des circuits plus optimisés que ceux générés par les méthodes dirigées par la syntaxe car des optimisations peuvent être appliquées à la fois sur les abstractions les plus élevées comme sur les plus basses. Cette méthode n’est, à notre connaissance, pas totalement automatisée.

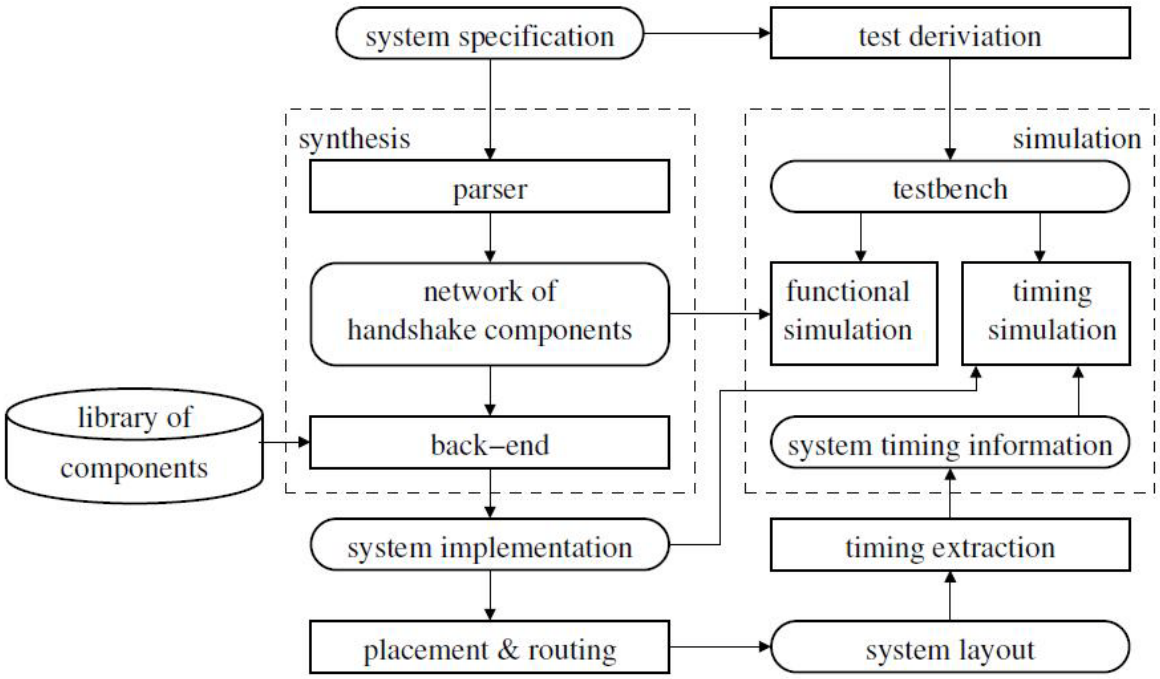


Figure 14 : flot de synthèse dirigé par la syntaxe

5.1.3. Méthode TAST :

TAST (Figure 15) est un outil de conception et de synthèse de circuits asynchrones quasi insensibles aux délais [26]. C'est dans le cadre du développement de cet outil que s'est effectué ce travail de thèse. Le langage de description est le CHP [24], comme dans CAST (basé sur CSP [27] et les processus communicants). Le flot de synthèse s'articule autour d'un format intermédiaire, qui est un réseau de Pétri dont les places sont étiquetées par des graphes de type flot de données (Data Flow Graph, DFG). La Figure 15 montre le flot de conception de TAST.

De la même façon qu'il existe en synchrone des règles RTL (Register Transfer Level), spécifiant si un programme VHDL ou Verilog est synthétisable, TAST définit des règles DTL (Data Transfer Level). Ces règles identifient les mémoires fonctionnelles, et restreignent leur usage. Si un processus ne respecte pas ces règles, il est décomposé en un ensemble de processus qui les respectent.

Le flot est automatisé depuis le langage de description comportementale jusqu'à la netlist. Il ne nécessite l'intervention de l'utilisateur que pour lancer les différentes étapes de la synthèse, et vérifier que tout se passe bien. En particulier, le chemin d'acquittement est calculé automatiquement et de manière totalement transparente. De plus, on peut utiliser les outils de

placement-routage classiques du monde synchrone sur la netlist obtenue après synthèse. En effet, puisque la netlist est quasi insensible aux délais, quelle que soit la manière dont le placement et le routage sont faits, elle fonctionnera. La seule contrainte à respecter est celle des fourches isochrones, qui n'est pas un problème en pratique pour les circuits générés. En effet, il faudrait que la différence de temps de propagation entre deux branches du circuit soit très importante pour que le circuit ne fonctionne pas, et il est facile de contraindre l'outil de placement-routage pour que ceci n'arrive pas.

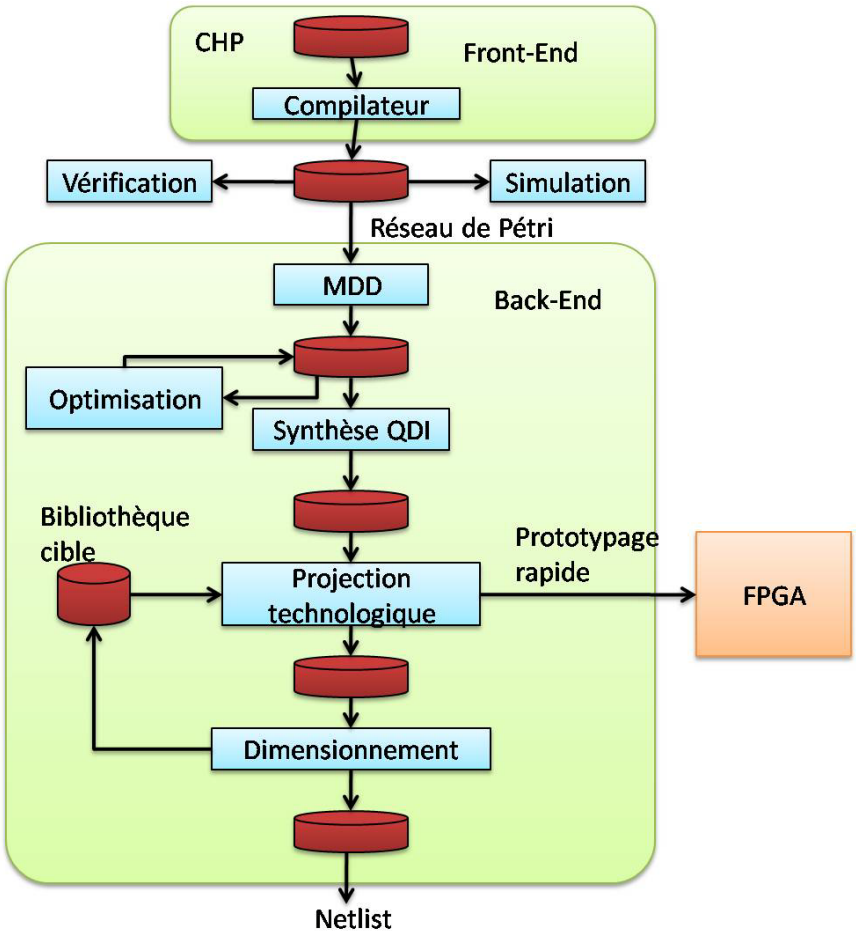
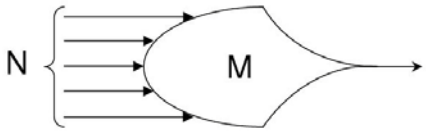


Figure 15: Flot de conception TAST

5.1.4. Méthode NCL :

La philosophie des circuits générés par la méthode NCL [28-29] repose sur le motif de base représenté Figure 16. Il s'agit de séparer fortement la partie combinatoire (proche du circuit synchrone équivalent) et la partie acquittement/gestion de la validité des sorties (spécifique à l'asynchrone). Ainsi, l'apposition d'une « barrière de registres » en sortie de la partie

combinatoire (Figure 17) permet d'éviter de propager des données non valides. La génération de l'acquittement se fait dans des blocs logiques distincts (blocs de détection de complétion ou CD) dont la synthèse n'est pas automatisée.



Entrées	Sortie
M(ou plus) entrée valides	1
Toutes les entrées sont nulles	0

Figure 16: Fonctionnement d'une porte à seuil (M < N).

Cette méthode comporte un certain nombre de limitations. Le style de conception proposé est très typé. Il apparaît que non seulement cette description générique n'est pas universelle, mais de plus est limitée à plus d'un point de vue. En effet, si la distinction opérée entre logique directe et logique d'acquittement semble permettre une relative simplification de la conception, il s'agit tout de même d'une très forte restriction : il n'est dès lors plus possible de générer des acquittements partiels en vue de réaliser des optimisations (n'acquitter que certaines branches du circuit), ni de procéder à toutes les combinaisons complexes où l'acquittement dépend du chemin suivi par les données dans la partie calculatoire.

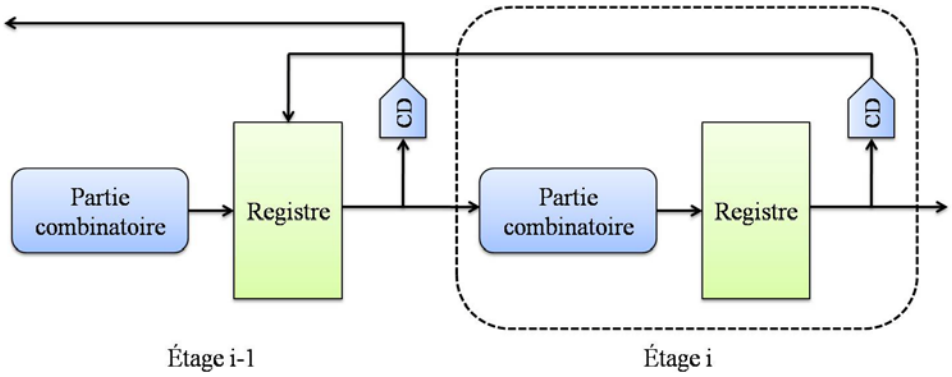


Figure 17 : Circuits obtenus par la méthode NCL

Les implantations du type « lecture conditionnelle » (comme la fonction de multiplexage) ne sont pas synthétisables par cette méthode. Contrairement à Haste, Balsa ou Caltech, où les méthodes d'acquittement sont bien définies, ce principe de communication simplifiée, dans laquelle le concepteur implante manuellement les acquittements, se révèle surtout un manque de flexibilité et de robustesse vis-à-vis de la spécification « haut niveau ». En effet, il n'est nullement possible de vérifier si la génération des signaux d'acquittement est correcte, ni si elle réalise le

schéma de communication souhaité. La Figure 18 montre le flot de conception de la méthode NCL.

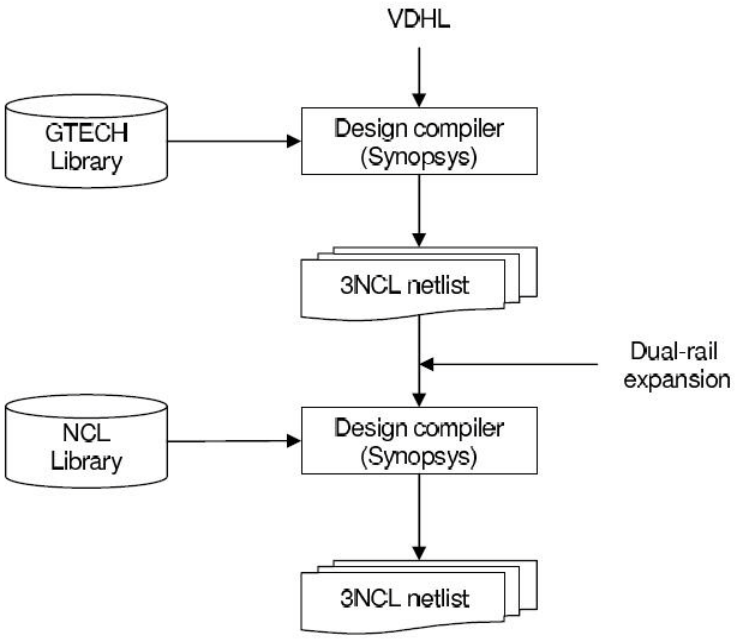


Figure 18: Flot de synthèse de la méthode NCL

5.2. Synthèse partie séquentielle :

5.2.1. Machine à états en mode rafale (Burst-mode state machines) :

Une spécification en mode rafale [30] (Figure 19) est une variation d'une machine de Mealy qui permet un changement multiple des valeurs d'entrée.

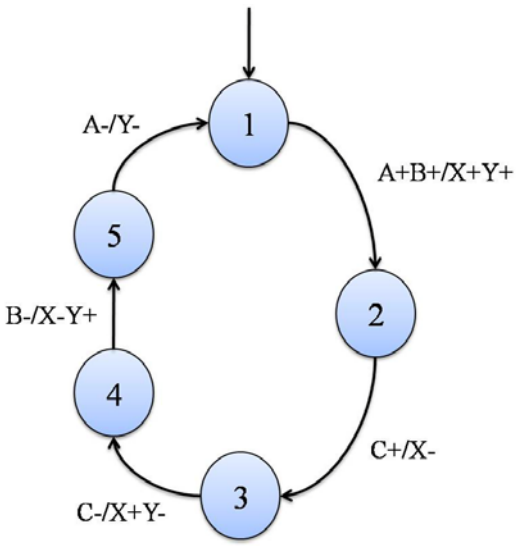


Figure 19: Exemple de spécification en mode rafale

a. Minimalist :

Cette méthode, proposée par Steven Nowick [31-32], traite la conception de contrôleurs asynchrones fonctionnant en mode rafale. Elle génère des circuits de Huffman. Elle se base sur une spécification de machine à états. Du point de vue du calcul, cette spécification se base sur les états : à chaque état, la machine peut recevoir des entrées, génère des sorties, et avance jusqu'à l'état suivant. Cette spécification est plus concise que le STG, notamment lorsque la concurrence du système est importante. Pour que le contrôleur fonctionne correctement, des contraintes sur la spécification sont définies. Tout d'abord, une seule entrée est autorisée à changer dans un état (Single Input Change). De plus, chaque état a un point d'entrée unique, ce qui simplifie l'optimisation et garantit une logique sans aléa. Pour finir, tout changement d'état nécessite un changement sur une entrée, ce qui signifie que le système reste stable tant qu'aucune entrée ne change. S. Nowick a également proposé une méthode de synthèse basée sur une machine à états synchronisée localement (Figure 20). Cette machine est une machine de Huffman à laquelle on a ajouté une unité d'auto-synchronisation, qui fonctionne comme une horloge locale. La première étape de la méthode de synthèse consiste à générer une table de transitions pour les sorties et les prochains états. L'étape suivante optimise la table pour réduire le nombre d'états, et choisit un codage des états. La dernière étape génère les fonctions booléennes pour l'horloge, chaque sortie et chaque variable d'état.

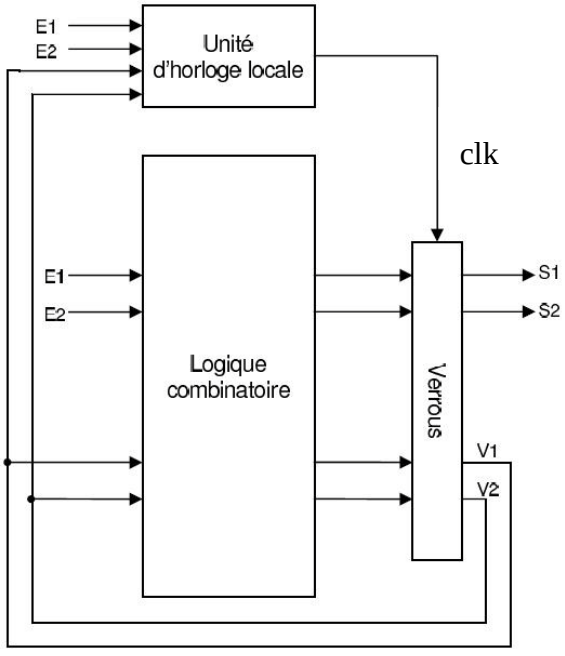


Figure 20: Schéma de la machine auto-synchronisée.

b. Machine asynchrone 3D :

Dans [33] une autre méthode de synthèse pour les machines à états asynchrones, toujours en mode rafale est proposée. La méthode se base sur une représentation 3D des états.

Une machine à états asynchrone à trois dimensions (3D) [33] est un quadr-uplet (X, Y, Z; δ) où X est un ensemble de symboles primaires pour les entrées, Y un ensemble de symboles primaires pour les sorties, Z (éventuellement vide) un jeu de symboles des variables d'état interne et $\delta : XY \times Z \rightarrow Y \times Z$ est la fonction de l'état suivant. L'implémentation matérielle de cette machine 3D est un réseau de portes logiques à deux niveaux ET-OU où les sorties (et des variables d'état supplémentaires) sont renvoyées en tant qu'entrées pour ce réseau. Ce réseau ne contient pas d'éléments de stockage comme les tampons (latches), les bascules ou les portes de Muller. La mémoire est maintenue uniquement par une rétroaction statique.

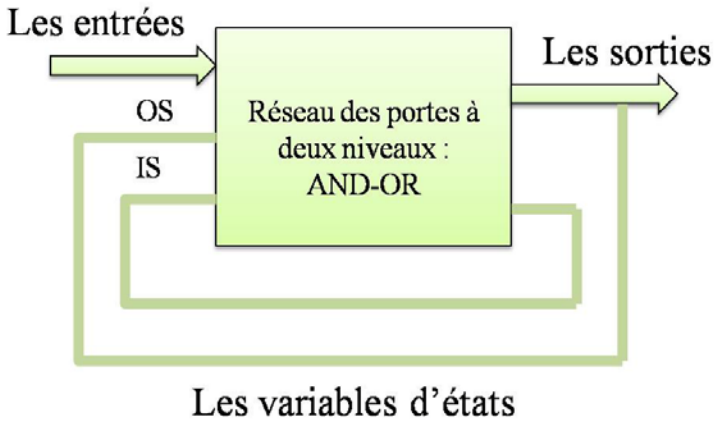


Figure 21 : Machine à états 3D

La mise en œuvre 3D d'une machine en mode rafale est obtenue par la réalisation d'un tableau de fonctions à trois dimensions (3D) appelé le tableau de l'état suivant : une représentation tabulaire de la fonction de l'état suivant δ. Chaque état atteignable doit être précisé; les états non atteignables qui restent dans ce tableau sont considérés comme indifférents. Le fonctionnement de la machine à état 3D est similaire à celle d'une machine synchrone de type Mealy.

Un cycle complet de machine est composé de 3 phases: une rafale d'entrée suivie par la production de sorties suivie par la production de nouvelles variables d'état. Pendant l'état de

repos, la machine attend un changement sur les entrées. Le prochain ensemble de transitions ne doit se produire qu'à partir du moment où la machine est stabilisée.

L'inconvénient principal d'une spécification d'une machine à état en mode rafale est l'hypothèse de mode fondamental de l'environnement (l'environnement ne produit aucun changement sur les entrées avant que la machine se stabilisent, il faut considérer aussi que les délais sont bornés et connus dans les portes et les fils du circuit : circuit de Huffman) qui rend leurs implémentations vulnérables face aux variations PVT. L'algorithme de synthèse de cette approche est compliqué et doit garantir l'absence des aléas qui mettent en péril le fonctionnement global des circuits. Ces méthodes de synthèse sont toutefois convenables mais uniquement pour des machines asynchrones de petites tailles.

5.2.2. Le graphe de transition des signaux « STG » :

STG ou graphe de transition des signaux est un Petri-Net [34] (Figure 22) où les événements sont interprétés via les fronts montants (+) ou les fronts descendants (-) des transitions des signaux.

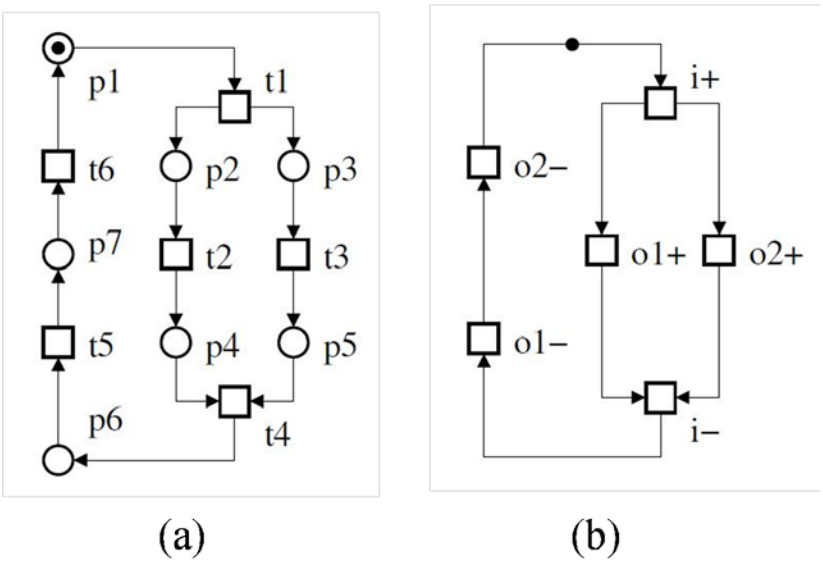


Figure 22 : Patri-Net (a), STG (b)

a. « Petrify » :

Petrify [35] est un outil de synthèse logique explicite, il accepte en entrée une spécification bas niveau du système (niveaux de transition des signaux), et cet outil produit des équations Booléennes des sorties du circuit en utilisant la fonction de l'état suivant obtenu à partir du STG.

Petrify (Figure 23) synthétise des circuits Indépendants de la vitesse (SI), à partir d'une spécification sous forme de graphes de transitions de signaux (STG). Ce modèle, basé sur le réseau de Pétri, est une re-formalisation du diagramme temporel, qui spécifie les relations de causalité entre les transitions. Il permet de modéliser la concurrence et une version limitée de choix entre les entrées.

Pour calculer la fonction logique de chaque sortie du circuit, qui est nécessaire pour la synthèse, l'outil doit d'abord calculer l'espace des états atteignables. Mais le nombre d'états de cet espace augmente exponentiellement avec le nombre de signaux du STG. C'est le principal point faible de cette méthode, qui se limite aux petits circuits. De plus, un problème d'ambiguïté peut se poser lors du calcul des fonctions de signaux : des états différents peuvent avoir le même codage. Dans ce cas, le circuit ne respecte pas la propriété d'État de Codage Complet (CSC, Complete State Coding), et il faut ajouter des variables d'état supplémentaires pour la respecter. Les équations logiques des sorties sont ensuite calculées, pour les fonctions de charge (set) et décharge (reset). À partir de ces équations, le circuit est synthétisé en utilisant une bibliothèque de cellules complexes.

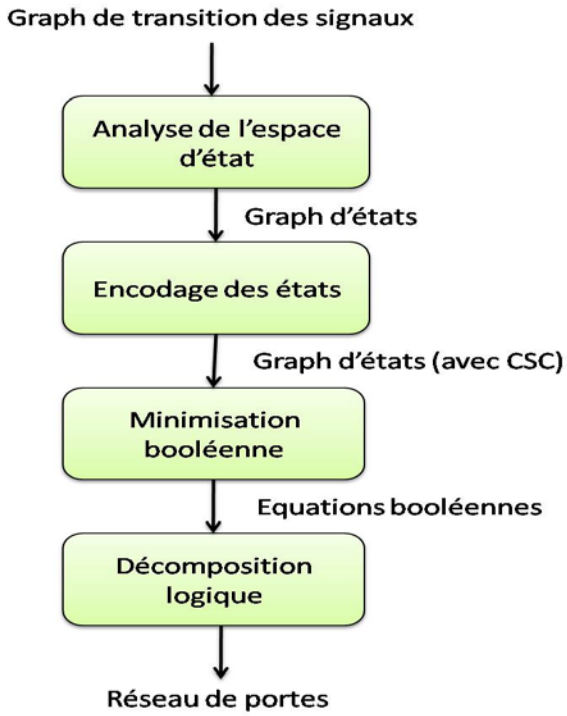


Figure 23 : Flot de conception logique de la méthode STG

Enfin, le circuit est modifié pour pouvoir être initialisé, via l'utilisation d'un signal de Reset. L'outil de synthèse Petrify automatise cette méthode. Il prend en entrée un STG décrivant le

circuit, qui peut être saisi sous forme de texte ou sous forme graphique. La description de circuit sous forme de STG est ardue et sujette aux erreurs ; ceci constitue aussi une limitation de la méthode. De plus, l'étape d'exploration de l'espace d'états limite la complexité des circuits pouvant être réalisés (environ 20 signaux au maximum).

b. « Direct mapping » :

L'idée principale de l'implémentation directe est qu'une spécification graphique d'un circuit peut être traduite directement en Netlist de composants de telle façon que les nœuds du graphe correspondent aux éléments de circuits et les arcs graphiques correspondent aux interconnexions. Une méthode de synthèse d'une spécification en STG basée sur cet algorithme est proposée en [36].

Dans cette méthode, la spécification d'un système est d'abord divisée en deux STG, un STG-périphérique qui reflète le comportement de l'environnement, et un STG-dispositif qui décrit la réaction du composant (Figure 24, Figure 25). Le STG-dispositif est examiné séparément : il se compose d'un traqueur « tracker » et d'un videur « bouncer ». Le traqueur suit l'état de l'environnement et est utilisé comme un point de référence par les sorties de dispositif. Le videur fait l'interface avec l'environnement et génère-les sorties en fonction des entrées en se basant sur l'état du traqueur. Cette architecture à deux niveaux de dispositifs fournit une interface efficace à l'environnement. Elle est pratique pour utiliser l'algorithme de l'implémentation directe afin de générer une Netlist de portes.

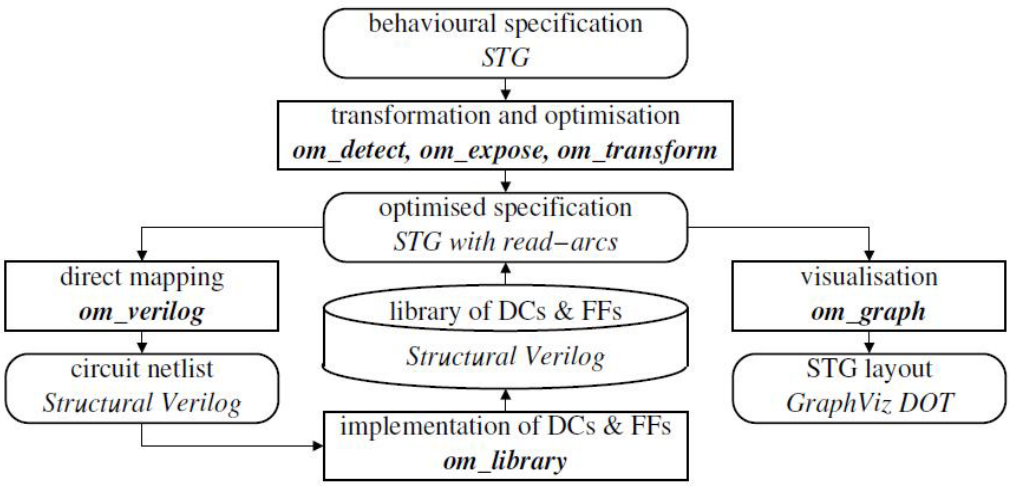


Figure 24 : Flot de conception optimiste

La méthode est étendue par un ensemble d'algorithmes et heuristiques d'optimisation. Ces optimisations sont mises en œuvre dans un outil logiciel appelé « OptiMist» (Figure 25). Les circuits indépendants de la vitesse obtenus par cette méthode ont une architecture à deux niveaux, ce qui contribue à une interface à faible latence pour l'environnement. Le temps de calcul pour générer le circuit avec OptiMist croît d'une façon linéaire avec la taille de la spécification, ce qui permet d'appliquer la méthode sur des STG de grande taille. Le transfert d'un état à l'autre est implémenté par un module asynchrone D-élément. La Figure 24 montre le flot de conception d'OptiMist.

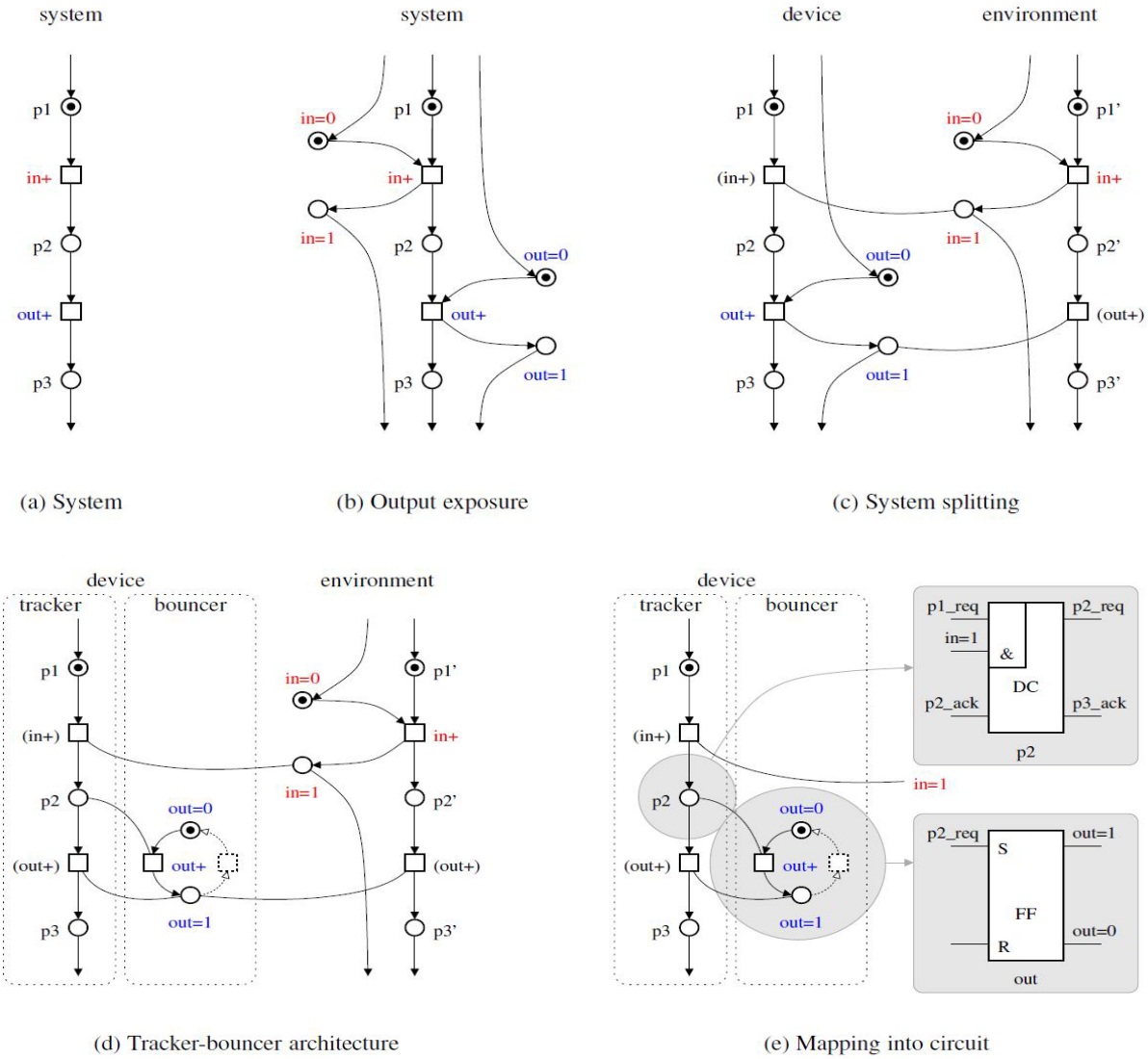


Figure 25: Méthode de l'implémentation directe des STG (OptiMist)

L'inconvénient de cette méthode est que chaque place dans les réseaux de STG est remplacée par des D-éléments, et lorsque le circuit communique avec l'environnement avec des canaux d'entrées et sorties qui respectent un protocole poignée de main, la taille du circuit augmente

rapidement. En plus, les circuits résultants de la méthode sont indépendants de la vitesse ce qui ne convient pas dans plusieurs cas à une conception QDI visée par le travail de cette thèse.

6. Problématique :

Dans ce travail de thèse, nous ciblons une famille particulière de contrôleurs asynchrones séquentiels, ceux que l'on peut trouver dans un circuit QDI où l'interface des contrôleurs est un groupe de canaux d'entrée et de sorties qui adoptent un codage des données QDI et une synchronisation par un protocole de type poignée de main (Figure 26). La spécification et la synthèse de ce type de contrôleurs pour les systèmes à grande complexité, comme les cœurs de processeurs ou les routeurs, etc…, reste un problème difficile en raison du manque d'une méthode de synthèse efficace.

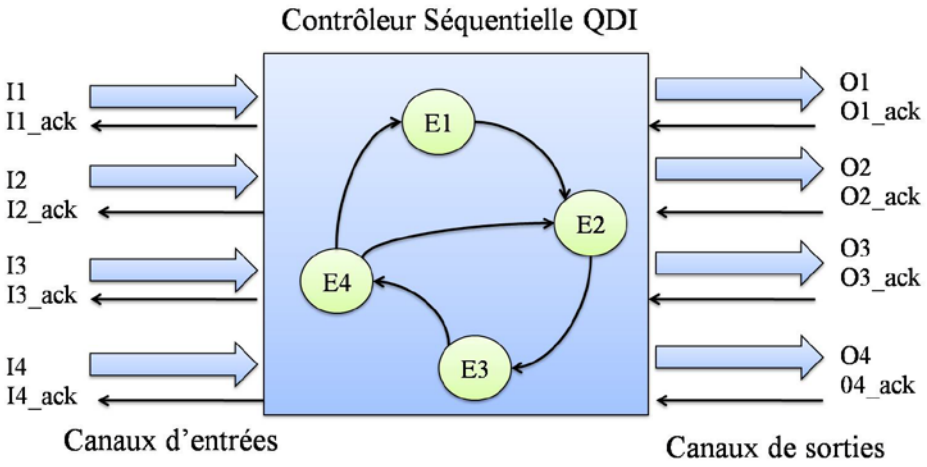


Figure 26: Contrôleurs séquentielle QDI

Dans la conception asynchrone QDI, il y a un besoin de modèle générique qui peut, à la fois, capturer la concurrence et les choix. Nous avons présenté la plupart des méthodes de conception de la logique de contrôle asynchrone. Certaines méthodes telles que Tangram (ou Haste), ou Balsa, utilisent CSP-HDL comme des langages pour la spécification des systèmes, et elles adoptent une synthèse dirigée par la syntaxe. Elles ne sont pas particulièrement bien adaptées à la conception de la partie contrôle des circuits car elles réalisent la conception entière comme un ensemble de processus et de canaux. Le contrôle est donc implicite dans ces canaux.

La version actuelle de TAST est encore loin d'être comparable à un outil de synthèse de circuits synchrones, et il reste beaucoup d'améliorations à effectuer. C'est le but de cette thèse que d'y contribuer. Tout d'abord, l'outil est efficace pour synthétiser des composants de chemins de données mais aucune des méthodes étudiées n'étaient parties sur la conception d'une partie de

contrôle séquentielle ou les machines à états. Les circuits de contrôles séquentiels synthétisés par TAST sont sous-optimaux : seules des optimisations naïves sont effectuées. De plus, la structure du circuit séquentiel généré est encore proche de celle du programme CHP d'où l'on vient : la structure des choix et des gardes est préservée à travers les transformations. Or, le CHP est un langage qui ne permet pas de manipuler directement des opérations sur des types complexes. Ainsi, la structure de choix et de gardes est très souvent utilisée dans un programme CHP, de manière imbriquée. Il serait essentiel de s'affranchir de cette structure imbriquée pour générer des circuits vraiment optimisés.

Les autres méthodes telles que Machine à états en mode rafale, ainsi que des réseaux de Petri (PN) et STG, sont plus appropriées pour la conception de la logique de contrôle parce qu'elles permettent de capter les comportements concurrents à un niveau très bas de granularité. Les circuits produits par ces méthodes sont plus compacts et plus rapides (en termes de latence) si on compare avec ceux issus des méthodes de la traduction de syntaxe HDL.

Toutefois, les méthodes de synthèse de machines à états en mode rafale ou des STG ciblent généralement des contrôleurs de petites tailles avec un petit nombre de choix où chaque option est plutôt unique. Dans de nombreuses applications, comme le contrôleur de processeur, le concepteur est souvent confronté à un problème de modélisation de comportements différents et complexes définis sur le même domaine des unités opérationnelles.

Appliquer la modélisation de machines en mode rafale, réseaux de Petri (ou STG) à des systèmes aussi complexes conduirait à une procédure longue et susceptible d’erreurs à cause d’une modélisation bas niveau des comportements. Ces méthodes résultent en des circuits non performants en termes de surface, consommation, etc. en raison de leur notion de contrôle explicite par la transition d’état. Ces modèles fonctionnent sur la base du calcul de l’état suivant ce qui nécessite une logique de transition assez complexe et une mémoire interne considérable.

Dans ce travail, nous avons essayé de trouver un modèle qui permettrait de conserver les avantages des modèles de Petri (ou STG) et de machines à états finis (Finite State Machine). Les premiers sont avantageux pour la modélisation d’un grand degré de concurrence tandis que le second l’est pour le choix. Nous avons essayé aussi de trouver une méthode de synthèse qui est simple pour couvrir une large gamme des contrôleurs séquentiels et efficaces en termes de consommation de surface. En fin nous avons cherché à prouver formellement la correction des circuits générés.

7. Conclusion :

Ce chapitre présente un ensemble de méthodes pour la conception de circuits asynchrones. Nous avons également présenté un état de l'art sur les méthodes de synthèse existantes dans le domaine asynchrone. Nous avons également évoqué la problématique de cette thèse. Dans la suite de ce manuscrit, nous allons proposer une solution pour modéliser le comportement d'un contrôleur séquentiel QDI, et nous proposerons deux solutions pour synthétiser des contrôleurs asynchrones QDI. La première est basée sur le flot de conception TAST, et la deuxième est basée sur des techniques d'implémentation directe.

Bibliographie

1. Marc Renaudin, J.L., *Asynchronous Circuits Design: An Architectural Approach. (2003).*
2. Beerel, P.A., *Asynchronous Circuits: An Increasingly Practical Design Solution.* Proceedings of ISQED'02, IEEE Computer Society (2002).
3. Jens Spars, S.F., *Principles of Asynchronous Circuit Design. A System Perspective.* Kluwer Academic Publishers (2001).
4. C.H. (KEES) Van Berkel, M.B.J., Steven M. Nowick, *Scanning the Technology. Applications of Asynchronous Circuits.* Proceedings of the IEEE, Vol. 87, No 2 (1999).
5. Al Davis, S.M.N., *An Introduction to Asynchronous Circuit Design.* UUCS-97-013, (1997).
6. Beerel, S.K.a.P., *Pipeline Optimization for Asynchronous Circuits: Complexity Analysis and an Efficient Optimal Algorithm.* IEEE Transactions of Integrated Circuit and Systems , VOL. 25, NO. 3, 2006.
7. MILLER, R.E., *Sequential Circuits and Machines, volume 2 of Switching Theory.* John Wiley & Sons, 1965.
8. RIGAUD, J.-B., *Spécification de bibléothèques pour la synthèse de circuits asynchrones* THèse à Grenoble-INP, 2002.
9. Marc Renaudin, J.B.R., *Etat de l'art sur la conception des circuits asynchrones : persepectives pour l'intégration des systèmes complexes* publication labortoire TIMA, CNRS, université de grenoble.
10. **Huffman, D.A.**, *The Synthesis of Sequential Switching Circuits.* J. Symbolic Logic Volume 20, 1955: p. 69-70.
11. Rios, D., *Systeme a microprocesseur asynchrone basse consommation* These Docteur-Ingénieur, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2008.
12. Kawa, J., *Low Power and Power Management for CMOS—An EDA Perspective.* IEEE TRANSACTIONS ON ELECTRON DEVICES, VOL. 55, NO. 1, JANUARY 2008, 2008.
13. S. R. Vemuru, N.S.S.-C.P.D.E.f.C.L.G., *Short circuit Power dissiption estimation for CMOS Logic gates.* IEEE Transactions of Integrated Circuit and Systems-I: Fundamental Theory and Applications VOL. 41, NO. 11, Novomber 1994.

14. S. Turgis, N.A.a.D.A., *Explicit evaluation of short circuit power dissipation for CMOS logic structures.* In Proceedings of the 1995 International Symposium on Low Power Design, pages 129-134, April 1995.

15. C. Y. Tsui, M.P., A. M. Despain. , *Efficient estimation of dynamic power consumption under a real delay model.* IEEE Int. Conf. Computer-Aided Design. Santa Clara, CA, Novomber 7-11, 1993. pp 224-228.

16. J. Fragoso, G.S.a.M.R., *Automatic generation of 1-of-M QDI asynchronous adders.* SBCCI 2003 Proceedings, 16th Symposium on Integrated Circuits and Systems Design., 2003: p. 149-154.

17. I. Sutherland, B.S., D.Harris, *Logical effort : designing fast CMOS circuits.* Morkan Kaufmann Publishers, Inc. San Francisco, CA , 1999. ISBN 1-55860-557-6, PP. 240.

18. Pénzes, P., *Energy-Delay Complexity of Asynchronous Circuits.* Thesis, CalTech, Pasadena, California, USA. 202.

19. PANYASAK, D., *Reduction de l'émmision electromagnetique des circuits integres : l'alternative asynchrone.* THèse à Grenoble-INP, 2004.

20. BERKEL, K.V., *Handshake Circuits : an asynchronous architecture for VLSI Programming.* volume 5 of International Series on Parallel Computation. Cambridge University Press, 1993.

21. EDWARDS., A.B.a.D., *Compiling the Language Balsa to Delay-Insensitive Hardware.* Hardware Description Languages and their Applications. Kloos C.D. and Cerny E., 1997.

22. EDWARDS, A.B.a.D., *The balsa asynchronous circuit synthesis system.* In Forum on design Languages, 2000.

23. A. J. Martin, M.N., *CAST : Caltech Asynchronous Synthesis Tools.* Proc. of the Fourth Asynchronous Circuits Design Working Group Workshop., 2004.

24. Martin, A.J., *Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits.* California Institute of Technology, Pasadena, CA., 1989.

25. Lee, A.J.M.a.A.L.a.R.M.a.M.N.a.P.P.a.R.S.a.U.C.a.T.K., *The Design of an Asynchronous MIPS R3000 Microprocessor.* in Advanced research in VLSI, 1997: p. 164--181.

26. A. Dinh Duc, J.F., M. Renaudin, J. B. Rigaud and A. Sirianni. , *TAST : CAD Tools Tutorial.* Summer School on "Asynchronous Circuit Design", Proceedings of ASYNC'02, 2002.

27. HOARE., C.A.R., *Communicating sequential processes.* Communications of the ACM 21, 8 :666–677, August 1978.

28. FANT., K.M., *Logically Determined Design.* John Wiley & Sons, 2005.

29. BEEREL., M.F.a.P.A., *High performance asynchronous design using single-track fullbuffer standard cells.* IEEE Journal of Solid-State Circuits, 41(6) :1444–1454, 2006.

30. Nowick, S., *Automatic synthesis of BURST-MODE asynchronous states controllers* Stanford University Ph.D, Departements of electrical engineering, 1995.

31. NOWICK., S.M., *Automatic Synthesis of Burst-Mode Asynchronous Controllers.* PhD thesis, Standford University, 1993.

32. R.M. Fuhrer, S.M.N., M. Theobald, N.K. Jha, B. Lin and L. Plana, *Minimalist : An environment for the synthesis, verification and testability of burst-mode asynchronous machines.* TR CUCS-020-99, 1999.

33. Yun K., D.D., Nowick S.M., *Synthesis of 3D asynchronous states machines.* ICCD Proceedings, 1992.

34. Peterson, J.L., *Petri Net Theory and the Modeling of Systems.* Prentice Hall, ISBN 0-13-661983-5, 1981.

35. Cortadella J., K.M., Kondratyev A, Lavagno L., Yakovlev A., *Petrify: A Tool for Manpulating Concurrent Specifications and Synthesis.* IEICE Transactions on Information and Systems, 1997. **E80-D**(3): p. 315-325.

36. Yakovlev, A.B.a.A., *Asynchronous Circuit Synthesis by Direct Mapping: Interfacing to Environment.* Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems (ASYNC'02), 2002.

CHAPITRE 3 : SYNTHESE DES CONTROLEURS SEQUENTIELS QDI PAR TAST

La valeur morale ne peut pas être remplacée par la valeur intelligence et j'ajouterai : Dieu merci !

Le plus beau sentiment du monde, c'est le sens du mystère. Celui qui n'a jamais connu cette émotion, ses yeux sont fermés.

Un problème sans solution est un problème mal posé!

Albert Einstein

Figure 1: plan de thèse

Sommaire

Table des figures :

Table des tableaux :

1. Introduction

Nous avons montré les particularités de contrôle QDI par rapport aux autres types de contrôles asynchrones existants. Le chemin de données dans un circuit asynchrone QDI peut être construit d'une façon pipeline ou séquentielle. Dans tous les cas, la partie contrôle est répartie dans l'architecture sous forme de petits contrôleurs séquentiels fonctionnant en parallèles. Plusieurs travaux ont été effectués au sein de l'équipe CIS de laboratoire TIMA pour optimiser la conception de la partie opérative d'un circuit asynchrone comme les multiplieurs, l'additionneur et les fonctions concurrentes [1-4]. Dans ces travaux, il y avait plusieurs exemples de machines à états asynchrones QDI ou de contrôleurs séquentiels, en revanche aucun n'a montré leur conception de façon méthodique.

TAST est l'outil développé par notre équipe afin de réaliser la partie opérative. Il est possible d'utiliser son flot de conception afin de concevoir également des contrôleurs séquentiels. Ce chapitre est ainsi consacré à la présentation de la conception des contrôleurs asynchrone QDI séquentiels selon ce flot de conception. Nous classifions ces contrôleurs QDI en plusieurs types et présentons celui étudié au cours de cette thèse. Nous introduirons dans un deuxième temps le besoin des canaux de contrôle dans le chemin de données. Puis nous présenterons les différentes étapes pour implémenter le circuit à partir de sa description en langage haut niveau. Le flot de conception TAST sera évalué à travers un exemple. Enfin, nous proposerons des alternatives afin d'améliorer la performance.

2. Contrôleurs Séquentiels QDI :

Un contrôleur séquentiel QDI est une fonction où les valeurs envoyées sur ses sorties dépendent des valeurs d'entrée et de l'état interne de ce contrôleur. Ici, A l'inverse de la machine à états mode rafale [5-6], les entrées et les sorties sont des canaux et non pas des signaux simples. Un protocole de poignée de main contrôle donc la communication entre ce contrôleur et l'environnement sur chaque canal de son interface. Les valeurs sur ces canaux sont codées par un codage insensible aux délais (DI). Selon le nombre d'entrées et sorties, plusieurs types peuvent exister pour ces contrôleurs :

MIMO (Multi Input Multi Output contrôleurs) : Ce contrôleur possède plusieurs canaux d'entrée et de sortie (Figure 4 : a).

SIMO (Single Input Multi Output contrôleur) : Ce contrôleur possède un seul canal d'entrée et plusieurs canaux de sorties (Figure 4 : b).

MISO (Multi Input et Single Output contrôleur) : Ce contrôleur possède plusieurs canaux d'entrée et un seul canal de sortie (Figure 4 : c).

SISO (Single Input Single Output contrôleur) : Ce contrôleur possède un seul canal d'entrée et un seul canal de sortie (Figure 4 : d).

Séquenceurs (SO ou MO) : Ce contrôleur ne possède pas d'entrée mais une ou plusieurs sorties (Figure 4 : e ; f).

Cette classification prend en compte le nombre d'entrées et sorties parce que chaque canal d'entrée représente des signaux d'entrées mais également un signal de sortie, c'est le signal d'acquittement pour le canal. C'est la même chose pour les sorties : chaque canal de sortie est composé des signaux de sorties et un signal d'acquittement entrant. La gestion de ces signaux d'acquittement implique un traitement particulier selon chaque type. Nous ne rentrerons pas dans l'explication précise de ce sujet parce qu'il est très large et il demande beaucoup d'investigations supplémentaires, hors des objectifs fixés dans ce travail de thèse.

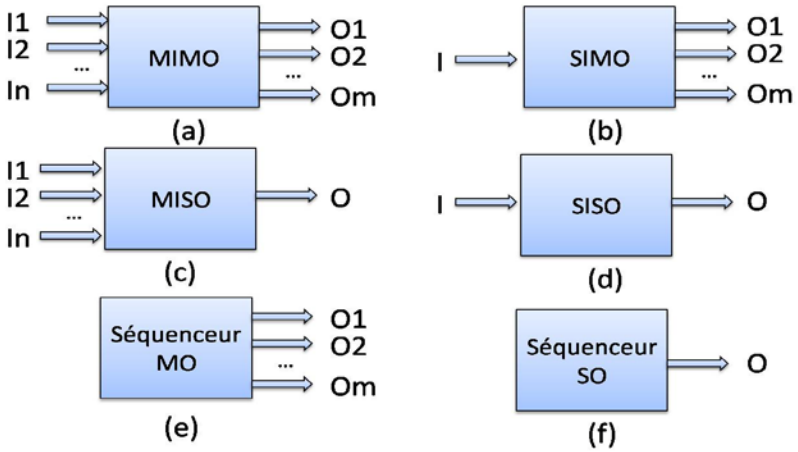


Figure 2 : Contrôleurs Séquentiels QDI

La fonction de ces contrôleurs est de générer les canaux de contrôles nécessaires pour commander les structures de multiplexage, démultiplexage, UAL, buffers, etc. qui se trouvent sur le chemin de données.

Cette thèse traite uniquement les contrôleurs type SIMO, SISO et les fonctions Séquenceurs à cause d'un manque de temps et des moyennes.

3. Le rôle du contrôle dans un chemin de données QDI

Un contrôleur intervient dans un architecture QDI parce que certains de ses composants disposent de plusieurs choix afin d'effectuer leurs fonctions. Les fonctions de multiplexage ou démultiplexage ont par exemple besoin d'un canal de contrôle qui précise la direction du flot de données. Une unité UAL peut réaliser des fonctions logiques (ET, OU, etc.) ou arithmétiques (addition, décalage, etc.). Cette unité a ainsi besoin d'un canal de contrôle pour décider de la fonction à effectuer. Certains tampons, équivalents des registres dans les circuits synchrones, ont également besoin d'un canal qui précise si une mémorisation peut se réaliser ou non, etc. Nous allons illustrer nos propos par deux exemples de circuits de chemin de données QDI conçu selon l'approche TAST.

3.1. Exemple 1 : multiplexeur QDI

La Figure 2 montre le circuit d'un multiplexeur dont « A » et « B » sont deux canaux d'entrée et « S » est la sortie. Ces canaux sont codés en double rail. Le canal de contrôle « C » est également codé en double rail. Lorsque « C [0] =1 », le canal « A » est connecté à « S » alors que lorsque « C [1]=1 », c'est le canal « B » qui est connecté à « S ».

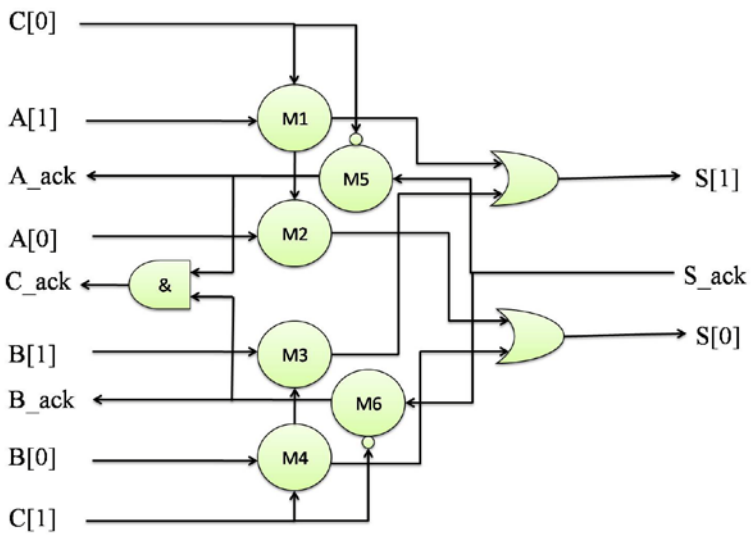


Figure 3 : Multiplexeur QDI

3.2. Exemple 2 : unité de décalage QDI

Le circuit d'une unité de décalage est montré dans la Figure 3. Elle effectue un décalage à droite ou à gauche d'une suite de bits. Les données de « A » sont codées en 1-parmi-4. « C » est le canal de contrôle en double rail qui décide si le décalage est à droite « C[0]=1 » ou à gauche « C[1]=1 ».

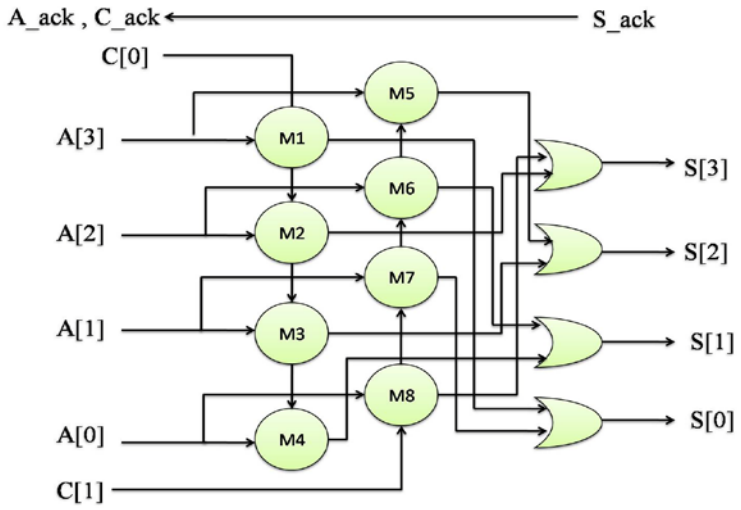


Figure 4 : Unité de décalage QDI

Dans ces deux exemples, un protocole 4 phases se produit sur le canal de contrôle « C » à chaque passage de données.

Dans cette thèse, on ne considèrera que le protocole 4 phases comme protocole de communication et un codage 1-permis-n pour le codage des données sur l'entrée et les sorties de contrôleurs QDI.

4. Contrôleurs Séquentiels QDI SIMO:

La Figure 6 montre le block diagramme de ce contrôleur à une seule entrée et à plusieurs sorties. Le contrôleur fonctionne de la manière suivante : lorsqu'une nouvelle valeur arrive sur son entrée, une séquence de valeurs est envoyée sur ses sorties. Chaque valeur d'entrée implique une séquence différente. Ces séquences sont différentes par l'ordre des sorties activées et par leurs valeurs. L'envoi de deux valeurs concurrentes sur deux canaux différents est aussi possible.

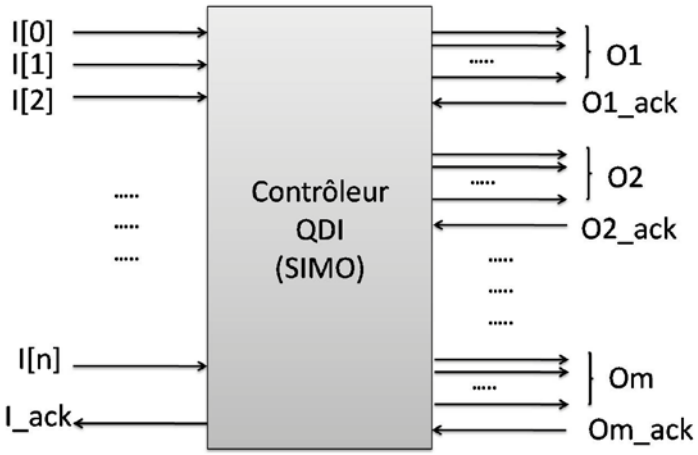


Figure 5 : Contrôleurs Séquentiel QDI (SIMO)

Un Décodeur Séquentiel d'Instructions « DSI » est un exemple de ce type de contrôleurs (Figure 6) où son entrée représente le mot à décoder alors que ses sorties représentent les instructions. Imaginons que ce DSI contrôle trois unités « U1, U2, U3 ». Imaginons que « U1 » exécute trois opérations (a : addition, b : soustraction, e : décalage), « U2 » exécute une seule opération (d : démultiplexage) et « U3 » exécute deux opérations (c : multiplication et f : division). Le « DSI » contrôle « U1 » par la sortie « O1 » codé 1-parmis-3 « O1 [a], O1 [b], O1[e] ». Il contrôle « U2 » par une sortie simple rail « O2 [d] » et « U3 » par « O3 » en double rail « O3 [c] », « O3 [f] ». Son entrée « I » se compose de 3 fils « I [0], I [1], I[2]».

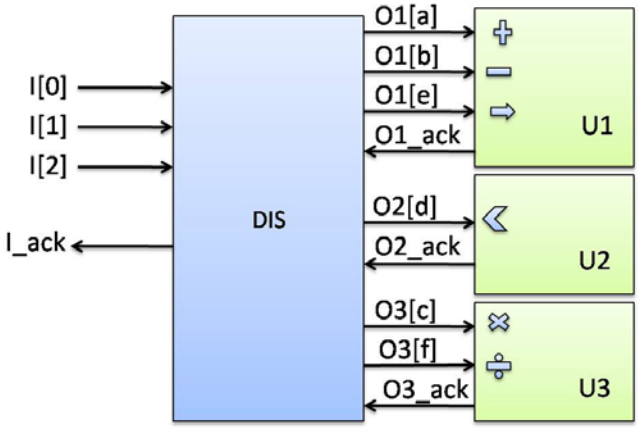


Figure 6 : Décodeur Séquentiel d'Instruction (DSI)

Ce contrôleur effectue 3 séquences de commandes:

1- Lorsque I = 1 : exécute une addition, puis une soustraction, un démultiplexage et enfin un décalage.

2- Lorsque I = 2 : exécute une addition, puis une soustraction, une multiplication, un démultiplexage et enfin un décalage.

3- Lorsque I = 3 : exécute une addition, puis une soustraction et une multiplication et enfin un décalage et une division.

A travers cette thèse, nous allons étudier plusieurs architectures et méthodes pour réaliser ce type de contrôleurs. Notre objectif est de trouver une méthodologie efficace, en termes de consommation, débit et surface, pour réaliser ces circuits séquentiels QDI. Dans la suite de cette thèse, nous citons ces contrôleurs SIMO comme contrôleurs QDI sauf si le contexte nécessite plus de précisions.

5. Méthodologie et flot de conception TAST :

Depuis quelques années, le groupe de recherche CIS du laboratoire TIMA développe un environnement de conception pour les circuits asynchrones QDI [1, 7-8]. Cet outil TAST est une contribution au manque d'outils de conception de circuits asynchrones dans le but d'essayer de faire évoluer la technologie des circuits asynchrones. L'objectif de cet outil est de générer des circuits asynchrones QDI décrits avec le langage de haut niveau CHP[9].

TAST (Figure 7) est constitué de plusieurs outils qui réalisent différentes étapes de la conception des circuits asynchrones QDI. La Figure 5 montre le flot de conception de TAST. Cet outil utilise un format intermédiaire basé sur les réseaux de Pétri. La partie front-end de l'outil transforme le circuit décrit en CHP au format intermédiaire en réseau de Pétri. Ce format permet la simulation et la vérification du circuit décrit en CHP.

La deuxième étape consiste à transformer ce réseau de Pétri en une description du circuit à l'aide d'un Diagramme de Décisions Multi-value (MDD)[1]. La synthèse s'effectue en utilisant des cellules standard à deux entrées avec des portes "et" et "ou", ainsi qu'avec la cellule de base asynchrone, la porte de Muller.

TAST délivre un fichier de description en portes logiques (netlist) en format Verilog ou VHDL avec des instances de portes standard. Cette *netlist* permet de simuler le circuit avec des logiciels commerciaux pour vérifier le fonctionnement du circuit. Le circuit peut être optimisé en utilisant des cellules complexes pour en réduire la taille et la consommation.

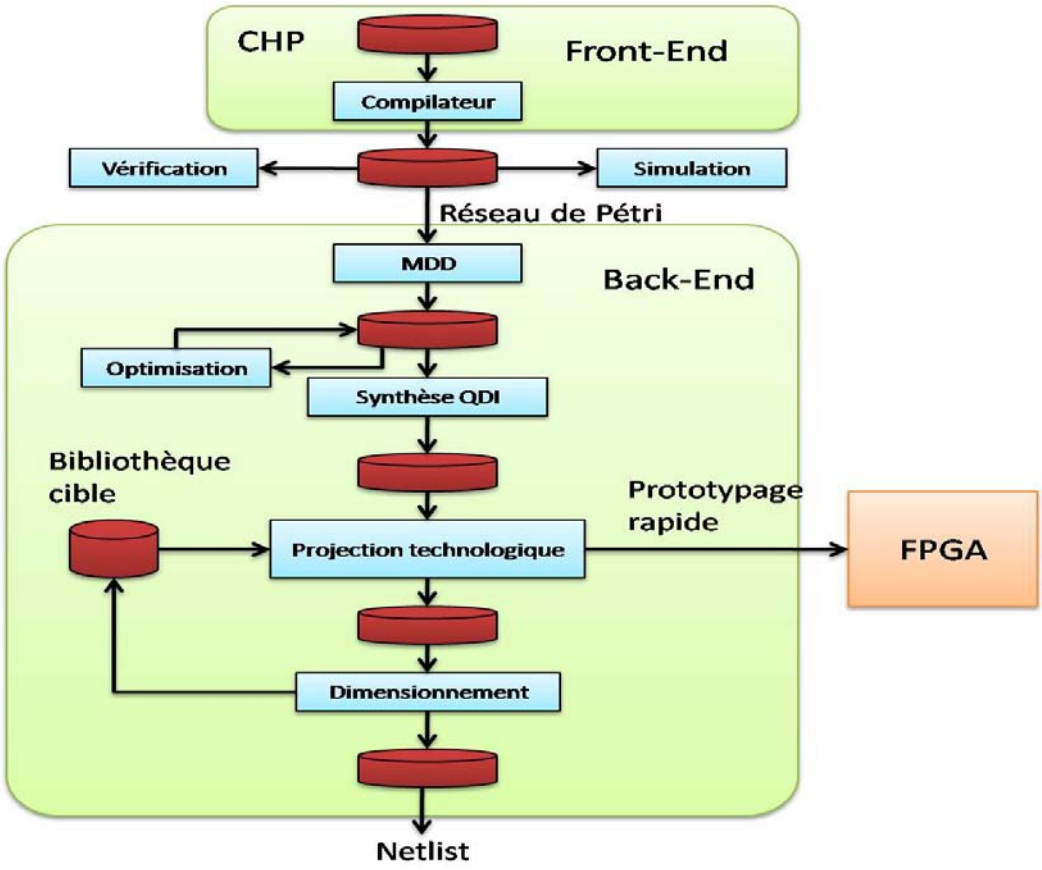


Figure 7 : flot de conception TAST

Le circuit est en effet optimisé en utilisant la librairie TAL (pour « TIMA Asynchrones Library »). TAL est une librairie asynchrone composée de portes complexes asynchrones. Cette étape est connue comme la projection technologique (« library binding » en anglais). Elle consiste à transformer un réseau logique indépendant en un réseau logique dépendant d'une technologie, ce qui revient à transformer un circuit ciblant des portes génériques en un circuit composé de portes standard et complexes issues d'une bibliothèque spécifique. Une dernière étape de dimensionnement est nécessaire afin de satisfaire les besoins de Fan-in et Fan-Out des portes pour une implémentation ASIC.

Une projection sur une plate-forme de prototypage rapide est également possible grâce à une bibliothèque des portes équivalentes à celles de TAL en FPGA.

6. Synthèse des contrôleurs QDI par flot de conception TAST :

La synthèse d'un contrôleur séquentiel selon TAST se compose des étapes suivantes : spécification en code CHP, transformation de ce code sous une forme

synthétisable, traduction en réseau de Petri et ensuite en MDD, implémentation du MDD, gestion des acquittements, décomposition et projection technologique.

Nous montrerons uniquement la partie nécessaire pour comprendre l'architecture du circuit de contrôleur.

6.1. Description en CHP:

Alain Martin dans [9] propose le langage CHP spécifique à l'asynchrone inspiré du langage CSP [10] avec l'utilisation des commandes gardées montrées dans [11]. Ce langage est bien un langage de haut niveau qui décrit les circuits asynchrones mais reste encore très basique par rapport au langage haut niveau existant comme VHDL, Verilog et C. Elle permet cependant de décrire un circuit asynchrone d'une manière assez simple.

Un circuit est décrit par des blocs qui communiquent entre eux, et qui réalisent des opérations arithmétiques, logiques, de comparaison, communication et conversion. Les données sont représentées par des canaux multi-rails de type booléen. Les canaux peuvent être définis avec une taille allant du mono rail à un canal à n rails. Le nombre de rails correspond au nombre de valeurs que peut prendre la donnée. Si la donnée est une donnée binaire en base deux, le canal pour chaque bit du signal sera un canal 1 parmi 2. Avec ce même principe, il est possible de créer des canaux 1-parmi-n.

Nous allons expliquer la syntaxe de ce langage en présentant la partie nécessaire à cette thèse :

Un processus « P » est un bloc fonctionnel, il est composé d'une interface de communication, une partie déclarative et un corps ; « ? » représente une lecture d'un canal ; « ! » Représente une écriture de canal ; « # » Sonder une valeur dans un canal d'entrée ; « ; » est un opérateur séquentiel ; « , » est un opérateur concurrent : L'opérateur de séquence I1 ; I2 indique que l'instruction I2 doit commencer après l'exécution de l'instruction I1. Au contraire, l'opérateur concurrence I1, I2 indique que les instructions I1 et I2 doivent être exécutées en parallèle.

Les structures de contrôle sont basées sur des blocs de commande gardée, il en existe deux types : un déterministe et un non déterministe [12]. Nos contrôleurs QDI seront décrits uniquement avec des structures déterministes. La structure déterministe attend jusqu'à ce qu'une seule garde soit vraie, et quand la garde est vraie, le bloc correspondant est exécuté. La syntaxe pour une sélection déterministe est la suivante :

Code 1 : @[

guarde_1 => block_1

guarde_2 => block_2

…

guarde_2 => block_2]

Le contrôleur QDI sera modélisé comme un processus qui lit un mot de contrôle sur son canal d'entrée et qui écrit d'une manière séquentielle sur plusieurs canaux de sortie.

Si on définit « I » comme le canal d'entrée et « O_m » comme le canal de sortie « m ». Le modèle CHP de l'exemple de DSI présenté à la figure 6 est ainsi le suivant :

Code 2 : IDS = [I? i; [@i=0 => O1! a; O1! b; O2! d; O1! e
@i=1 => O1! a; O1! b, O3! c; O2! d; O1! e
@i=2 => O1! a; O1! b, O3! c; O1! e, O3! f]

Dans ce code, l'opérateur séquentiel (;) marque la séquence dans la génération des valeurs de sortie comme « O1! a ; O1 ! b ». Alors que l'opérateur concurrent (,) marque la génération de deux valeurs concurrentes sur deux sorties comme « O2 ! b, O3 ! c ».

Une groupe concurrent de sorties est le groupe d'instructions qui concerne la génération des sorties, et qui se trouve entre deux opérateurs séquentiels « ; ». Le Code 2 a ainsi 11 groupes concurrents de sortie, dont 6 identiques.

D'une manière générale, un contrôleur QDI, avec un canal d'entrée « I » à « n » fils et « m » canal de sortie, peut être modélisé en CHP de la manière suivante :

Code 3 : P= [I ?i ; [@ i=0 => O1 ! v0 ; O1 ! v1 ; O2 ! v0 ;…;Om ! vk]
@ i=1 => O1! v_2, O2! v_1; O2! v_2; …
…
@ i=n => O1! v_m, O2! v_j; O2! v_{j+1}; …]

Ce code n'est pas synthétisable selon les règles DTL(Data Transfer Level) [12] car les accès séquentiels à un canal ne sont pas autorisés puisqu'ils imposent de la mémoire.

6.2. Transformation en code synthétisable :

Les Code 1 et Code 2 ne sont pas synthétisables, la solution consiste à traduire l'opérateur séquentiel sous la forme d'une machine à états (Figure 8). Le bloc P2, avec le

canal CS (Current State, état courant) en sortie et le canal NS (Next state, Etat suivant) en entrée, est un ensemble d'éléments de mémoire qui assurent le passage de l'état suivant au bloc de calcul des sorties (P1). Les états associés aux opérateurs séquentiels sont codés et extraits comme des variables tournantes. Le nombre d'états nécessaire à la décomposition est le nombre d'actions de l'opérateur séquentiel plus « Un». Ce « Un » traduit la nécessité d'avoir un état supplémentaire qui correspond à l'initialisation.

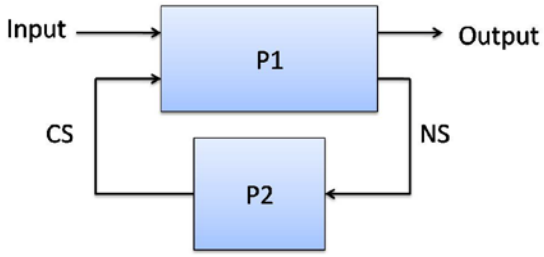


Figure 8 : Architecture de contrôleur en machine à états

Dans l'exemple « DSI » le nombre d'opérateurs séquentiels est de « 8 », le nombre d'états pour coder le machine est ainsi de « 9 », la transformation de ce code en machine à états donne les deux processus suivants :

Code 4 : P1= [CS? cs; [@cs=0 => I? i; [@i=0 => O1! a, NS! 1
@i=1 => O1! a, NS! 4
@i=2 => O1! a, NS! 7]
@cs =1 => O1! b, NS! 2
@cs =2 => O2! d, NS! 3
@cs =3 => O1! e, NS! 0
@cs=4 => O1 ! b, O3 ! c, NS ! 5
@cs=5 => O2 ! d, NS ! 6
@cs=6 => O1 ! e, NS ! 0
@cs=7 => O1 ! b, O3 ! c, NS ! 8
@cs=8 => O1 ! e, O3 ! f, NS ! 0]]

P2= [CS!0 ; [NS?ns; CS!ns]]

Ce code n'est pas optimal, si l'on remarque que chaque ligne dans le Code 2 est indépendante des autres, la séquentialité dans chaque ligne est donc marquée par les opérateurs séquentiels dans la ligne. Le nombre d'états pourrait être réduit au nombre maximum d'opérateurs dans une ligne plus un. Cette solution donne le code suivant :

Code 5 : P1= [Cs ? cs => [@cs=0 => [#I=> @I=0 => O1 ! a, NS!1
@I=1=> O1 ! a, NS!1
@I=2=> O1 ! a, NS!1]
@ cs =1 => [#I=> @I=0 => O1 ! b, NS ! 2
@I=1=> O1 ! b, O3 ! c, NS ! 2
@I=2=> O1 ! b, O3 ! c, NS!2]
@ cs =2=> [#I=> @I=0 => O2 ! d, NS ! 3
@I=1=> O1 ! b, O3!c, NS ! 3
@I=2=> O2 ! e, O3!f, NS ! 0, I?]
@cs=3 => [I# => @i=0 => O1 ! e, NS ! 0, I?
@i=1=> O1 ! e, NS ! 0, I?]]]

P2= [CS!0 ; [NS?ns; CS!ns]]

Le fait que « I » est acquitté au dernier état (I ?) permet donc d'éviter de mémoriser tous les états du processus. Il faut également noter que le canal « I » est consommé à la fin de l'exécution des actions de P1, tandis que le programme initial exécute cette lecture au début. Ceci peut être corrigé en ajoutant une mémoire tampon sur le canal « I ».

6.3. Synthèse à partir d'un diagramme de décisions multi-value :

Avec TAST, la description de circuit en CHP est transformée en un réseau de Petri, puis transformé en un Diagramme de Décisions Multi-valué (MDD)[1]. Cette partie contient une brève explication de la synthèse des contrôleurs asynchrones QDI à partir de cette description en MDD.

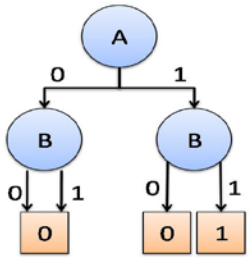


Figure 9 : exemple de MDD

Un exemple de MDD est présenté dans la Figure 9. Ce diagramme implémente la fonction "et" logique entre deux opérandes « A » et « B ». L'idée de ce diagramme est de spécifier la valeur de la sortie en fonction de toutes les combinaisons possibles des entrées. Avec ce diagramme de décisions multi-valué (MDD), le circuit peut être généré à l'aide de portes de Muller et de portes "ou" logiques. Les circuits générés avec cette

méthode présentent une série d'étages de portes de Muller pour recevoir toutes les entrées. La sortie est alors construite avec des étages de portes "ou" logiques. Cette méthode de synthèse garantit que le circuit obtenu est QDI.

Dans le cas de processus P1 du Code 3, le circuit est décrit avec une série de gardes déterministes qui calculent les résultats en fonction des entrées. P1 possède deux entrées (I, CS) et 4 sorties (O1, O2, O3, NS). La partie de Code 6 est liée à la génération de la valeur de sortie O1[a], elle était extraite du Code 4.

Code 6 : [@cs=0 => I? i => [@i=0 => O1 !a
@i=1 => O1! a
@i=2 => O1! a]

Avec cette description en langage CHP, l'outil de synthèse effectue dans un premier temps un MDD qui est montré dans la Figure 10 : a. Dans cet exemple, l'état courant « CS » est traité en premier, c'est aussi la racine de l'arbre, l'entrée « I » étant un fils de la racine. Dans ce circuit, la génération de la sortie commence par des portes de Muller qui reçoivent un signal de l'état courant « CS » et un signal de l'entrée « I ».

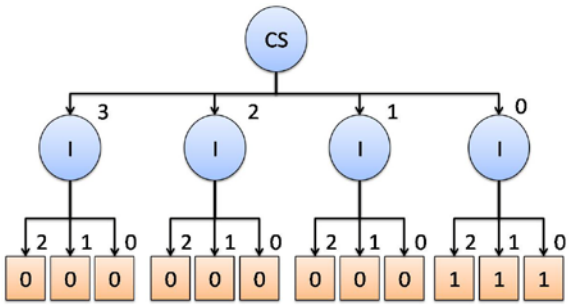


Figure 10 : MDD du signal O1 [a]

A partir de ce MDD, le circuit est réalisé en portes standard à deux entrées. Les branches de l'arbre qui terminent en une valeur « 1 » sont uniquement implémentées par des séries des portes Muller-OR (Figure 11).

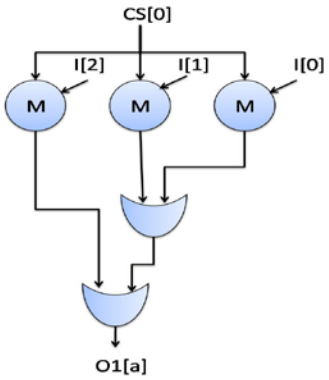


Figure 11 : Circuit du signal O1 [a] (Code 4)

Lorsque l'opérateur « # » est utilisé pour un canal d'entrée comme dans le Code 7, une porte Muller dissymétrique est utilisée à la place d'une porte standard pour sonder la valeur d'entrée. Code 7 est la partie du code 5 qui traite la génération de la valeur « a » sur la sortie O1 :

Code 7 : [cs=0 => [#I=> @I=0 => O1 ! a
@I=1=> O1 ! a
@I=2=> O1 ! a]

Le circuit correspondant après synthèse est montré sur la Figure 12.

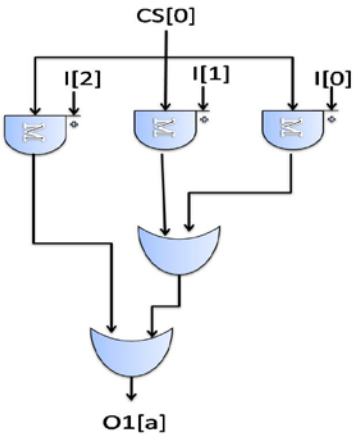


Figure 12 : Circuit du signal O1 [a] (Code 5)

6.4. Implémentation du protocole quatre phases :

Les circuits asynchrones utilisent l'équivalent d'un verrou (ou latch) pour les circuits synchrones pour assurer le protocole quatre phases. Ce registre est connu sous le nom de Half-Buffer. La Figure 13 montre le fonctionnement d'un circuit asynchrone QDI. Entre

blocs asynchrones, un Half-Buffer permet de mémoriser localement les données. Cheque étage doit acquitter l'étage précédent.

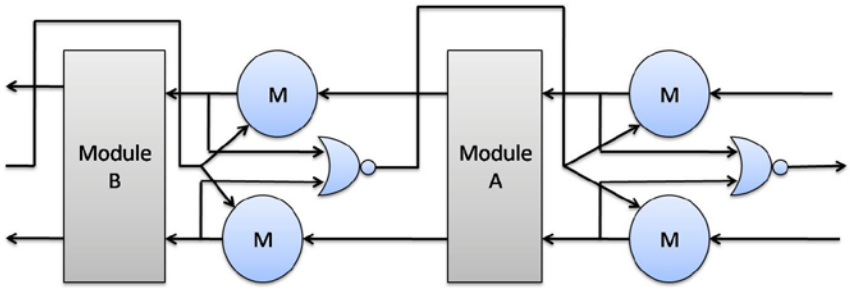


Figure 13 : Halh-Buffer double rail entre les blocks asynchrones QDI

Une lecture de canal dans le code CHP est implémentée par un Half-Buffer. Une écriture dans un canal est également implémentée par un Half-Buffer, P2 (Code 6 ou Code 7) est donc synthétisé par deux Half-buffers. La Figure 14 montre le circuit de P2, exemple DSI, dans le cas de machine à 4 états (Code 5).

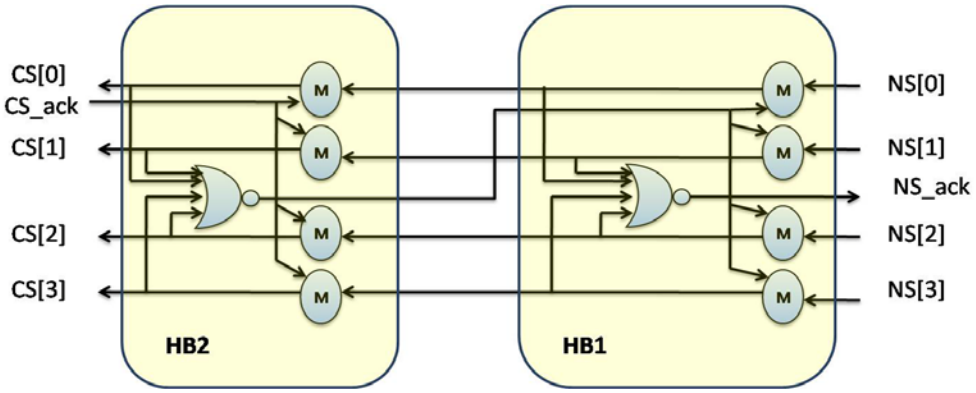


Figure 14 : Architecture P2 (deux Half-buffers)

Afin d'assurer la vivacité du protocole 4 phases dans un anneau asynchrone, il est indispensable d'avoir au moins trois étages de mémoire tampon [13-14]. L'architecture en machine à états du contrôleur QDI exige donc un troisième étage : les portes de Muller de P1 représentent ce troisième étage.

6.5. Acquittement :

P1 est une structure non conventionnelle contrairement à celles qui existent dans le chemin de données. D'un côté, P1 éprouve une structure de « marge » [12] entre l'entrée « I » et l'état courant « CS ». D'un autre côté, elle éprouve plusieurs structures de fourche, une fourche entre chaque sortie et l'état suivant (NS). Ainsi, trois conditions doivent être satisfaites afin de garantir une implémentation QDI de cette structure :

1- Aucun jeton de l'état suivant (NS) n'est produit avant que le précédent jeton ne soit consommé par P2.

2- Aucun nouveau jeton n'est transmis sur la même sortie avant que le précédent jeton ne soit consommé par son canal.

3- La valeur d'entrée est consommée une fois pour chaque séquence de sorties.

En résumé, d'une part quant à la structure de « Marge » [12] entre CS et I : CS est acquitté à chaque itération d'état ; l'entrée « I » est acquitté une fois, soit en début de séquence, soit en fin de séquence.

D'autre part, quant au structure de fourche : chaque porte de Muller, qui appartient à une fourche entre les sorties et NS, doit être acquittée par ces sorties et NS_ack à la fois.

L'arbre d'acquittement est en effet la projection inverse du code « CHP ». Une porte de Muller du processus « P1 » fait partie de la fonction qui génère l'état suivant « NS » et les sorties. Cette porte doit donc être acquittée par l'acquittement de l'état suivant « NS » du processus « P2 » et des sorties de ce groupe.

Prenons le Code 8 : [cs=1 => # I=1 => O1 ! b, O3 ! c, NS!2], la Figure 15 montre la porte Muller équivalente, celle qui génère O1 [b], O3 [c] et NS [2]. Cette porte reçoit ainsi trois acquittements O1_ack, O3_ack et NS_ack.

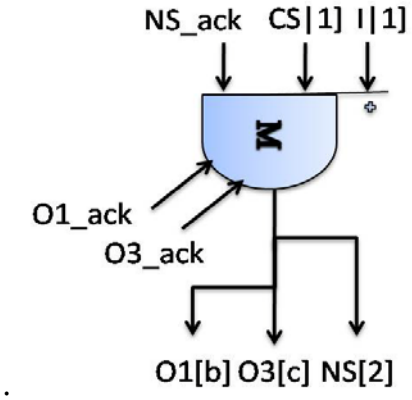


Figure 15 : Une porte de Muller de P1

Le front descendant du signal d'acquittement indique l'arrivée de jetons valides de données dans cette architecture de contrôleur. C'est pourquoi une porte NOR est utilisée dans chaque étage-tampon afin de produire l'acquittement pour l'étage précédent (Figure 14). Une Porte NOR est utilisée dans « P2 » par exemple pour produire l'acquittement de

l'état courant « CS ». Les signaux de l'état suivant « NS » forment les entrées de cette porte.

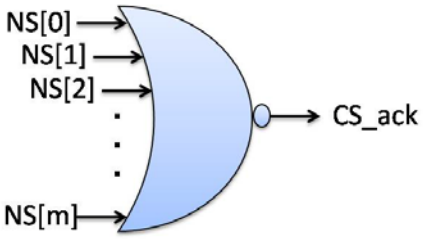


Figure 16 : CS_ack génération dans P1

Le signal I_ack est produit par la synthèse de code CHP correspondant à l'opérateur « I ? » car celui-là représente la fin de lecture de l'entrée. Il est considéré comme un signal de sortie de block P1 et donc synthétisé à partir de son MDD. Dans le cas du Code 4, ce MDD de l'I_ack est en effet identique avec celui d'O1 [a] alors qu'il est identique avec celui de O1[e] dans le Code 5. Après factorisation logique I_ack est dérivé donc de O1 [a] sur Figure 17 alors qu'il est dérivé de O1[e] sur la Figure 18.

6.6. Implémentation

Le DSI a été synthétisé selon la méthode TAST. Le circuit 1 (Figure 17) montre l'implémentation de P1 à partir du Code 4, alors que le circuit 2 (Figure 18) montre son implémentation à partir du Code 5. Quant à P2, c'est la même architecture que dans la Figure 14, la (seule) différence est le nombre d'états qui est de 9 par rapport au Code 4.

Les différences entre les deux implémentations se résument par les points suivants :

1- Le nombre des portes Muller dépend du nombre de groupe concurrent identiques des sorties dans le code, c'est pourquoi cela ne change pas d'une implémentation à une autre. Par contre, en Figure 18, l'entrée « I » est connectée à toutes les portes alors qu'elle est uniquement connectée à « n » portes en Figure 17, où « n » est le nombre de séquences produites par le contrôleur. Les portes Muller de circuit 2 sont ainsi plus larges (Fan-in, Fan-out) que celles du circuit 1.

2- L'optimisation du nombre d'états (Figure 18) a en effet conduit à une augmentation du nombre d'étage des portes OR en P1. Cela dit, un étage est ajouté pour produire l'état suivant.

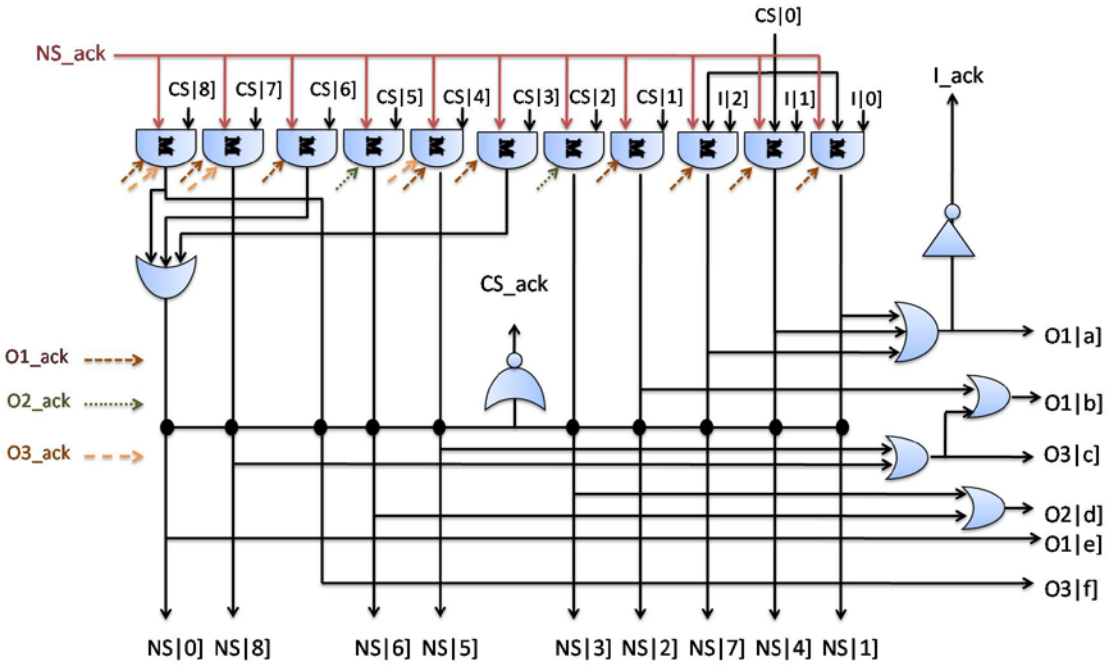


Figure 17 : Implémentation 1 du P1 (Code 4)

3- L'arbre d'acquittement (porte NOR) de circuit 1 est plus large (Fan-in=9, Fan-out=9-11) que celui du circuit 2 (Fan-in=4, Fan-out = 4-11).

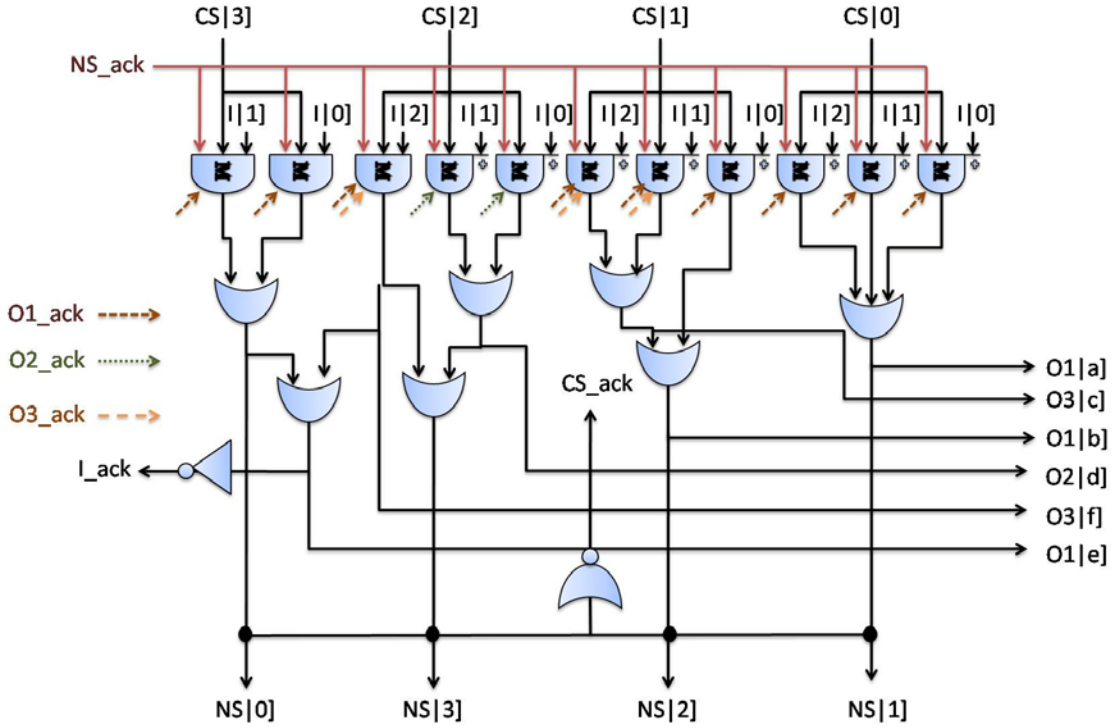


Figure 18: Implémentation 2 du P1 (Code 5)

4- La partie mémoire de la machine (P2) est proportionnelle au nombre d'états, c'est pourquoi l'implémentation 1 du P2 est 9/4 fois plus grande que celle de l'implémentation 2.

Ces remarques sont générales pour cette architecture de contrôleur QDI SIMO en machine à états.

6.7. Initialisation :

A l'état initial, les signaux d'acquittement des sorties, de l'entrée, de l'état actuel, et de l'état suivant (Figure 17, Figure 18, Figure 14) ont la valeur « 1 » car l'acquittement de jeton valide de données se fait par le front descendant des acquittements. En plus, les sorties et les entrées portent un jeton invalide (zéro). C'est pourquoi les portes Muller doivent avoir une valeur précise à leurs sorties. Un signal de réinitialisation « RST » est ainsi nécessaire.

D'un côté, P1 (Figure 17, Figure 18) et HB 1 de P2 (Figure 14) ont une valeur zéro, ou un jeton invalide, à leurs sorties. Leurs portes de Muller sont donc réinitialisées à zéro. D'une autre côté, HB 2 (Figure 14) de P2 a un jeton valide « CS [0]=1», une porte de Muller doit donc être initialisée à une valeur « 1 » alors que le reste des portes à un zéro.

6.8. Bilan 1

Le tableau 1 montre le résultat d'une simulation électrique pour ces deux implémentations de la fonction DSI. L'implémentation 2 est plus efficace en termes de surface (-15%), pic de courant (-53%) et énergie et puissance dissipées (-37 %) que celle du Code 4. Ceci est justifié par la très forte diminution du nombre d'états (de 9 à 4). Par contre le circuit 1 montre un débit égal à celui du circuit 2 à cause des points 1 et 2 illustrées avant.

Tableau 1: Résultat d'une simulation électrique pour les deux circuits (Code 4 et Code5)

Architecture en anneau(MAE)	*Nombre de transistors*	*débit M. séq.s-1*	*pic du courant*	*puissance*	*Energie*
Circuit 1 (Code 4)	526	36	683,3	183	5,09
Circuit 2 (Code 5)	449	35,8	319,1	115	3,21

Les résultats de ce travail ont été utilisés dans la thèse de David Rios [15] afin d'implémenter une machine à états qui contrôle un registre à décalage configurable.

Malgré l'optimisation du nombre d'états qui a conduit à un meilleur résultat, cette architecture en anneau a plusieurs inconvénients :

D'un côté, chaque étape dans l'anneau dépend de l'étage précèdent pour effectuer ses calculs. Le circuit devient de plus en plus lent en augmentant le nombre de sorties dans une séquence (nombre d'états), cela diminue le débit du contrôleur. D'un autre côté, l'arbre d'acquittement (Porte NORs) de chaque étage-tampon est proportionnel au nombre d'états, la consommation augmente et la vitesse diminue donc proportionnellement au nombre d'états.

Il est possible de simplifier l'arbre d'acquittement du circuit 2 en utilisant un codage double rail pour l'état actuel et l'état suivant [CS1, CS2, NS1, NS2]. Par contre, cela complique la logique du processus P1. Chaque valeur de sortie de P1 dépendra en effet des trois entrées CS1, CS2 et I. Cela implique un deuxième niveau dans l'arbre MDD pour générer une sortie, en conséquence, un deuxième étage de portes Muller. En conséquence, augmentation de consommation et diminution de débit du circuit résultant. Cela empire proportionnellement au nombre de canaux pour coder l'état actuel ou l'état suivant. Nous avons donc choisi d'étudier d'autres alternatives pour améliorer la performance obtenue par le circuit 2.

7. Eléments séquentiels (bilan 2) :

Le séquenceur [16] est un composant qui permet de produire une séquence de protocole 4 phases. Ce composant possède deux canaux, un canal gauche qui se compose d'un signal requête passif « LR » et un signal d'acquittement « LA » ; le canal droit se compose d'un signal de requête actif « RR » et d'un signal d'acquittement RA. Lorsque le séquenceur reçoit un requête (jeton 1) sur « LR », il génère une séquence de 4 phases sur son canal droite (RR+ ; RA- ; RR- ; RA-). Il acquitte ensuite « LR+ » par « LA+ ». Lorsque le signal « LR » se remet à zéro (Jeton 0), l'acquittement « LA » envoie un zéro.

La Figure 19 montre le circuit (a) et le STG (b) de ce composant.

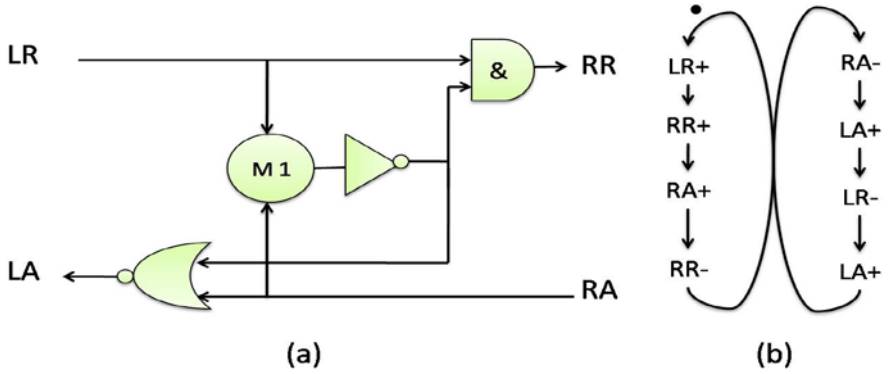


Figure 19 : composant séquenceur

Lorsque le signal « LA » d'un séquenceur 1 est connectée au « LR » d'un séquenceur 2, une séquence de deux protocoles 4-phase est obtenue (Figure 20). Lorsqu'un jeton 1 (requête) arrive sur LR du séquenceur « A », une séquence de protocole 4-phases se produit sur A et A_ack. Le jeton est ensuite transmis au séquenceur B par LA, un protocole 4-phases est produit sur B et B_ack.

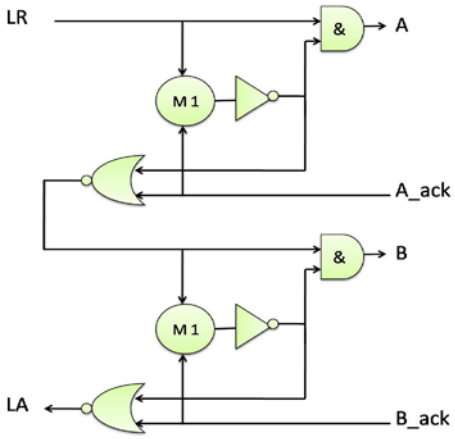


Figure 20 : Contrôleur Séquenceur double sorties coder en simple rail

Si nous voulons obtenir une séquence de deux protocoles 4 phases sur le même canal double rail {A[0], A[1], A_ack}, les deux séquenceurs doivent partager le même signal d'acquittement A_ack. Il est nécessaire d'ajouter une porte de Muller à chaque séquenceur (Figure 21) pour empêcher le signal d'acquittement A_ack de modifier l'état interne du séquenceur, celui qui n'est pas actif pendant la séquence de protocole poignée de main.

La Figure 22 montre le circuit QDI obtenu pour générer une double séquence de 4 phases sur un canal « A » double rails.

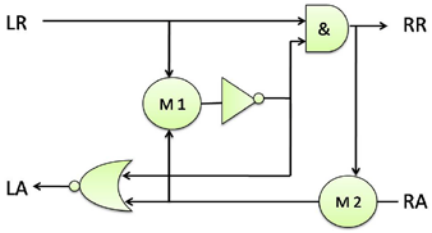


Figure 21 : Composant séquenceur optimisé

Nous avons implémenté deux circuits « contrôleur séquenceurs SO (paragraphe 3) » à 4 rail de deux manières : Le circuit 1 implémente l'architecture en anneau précédemment présentée ; Le circuit 2 utilise des séquenceurs de la même manière expliquée juste avant.

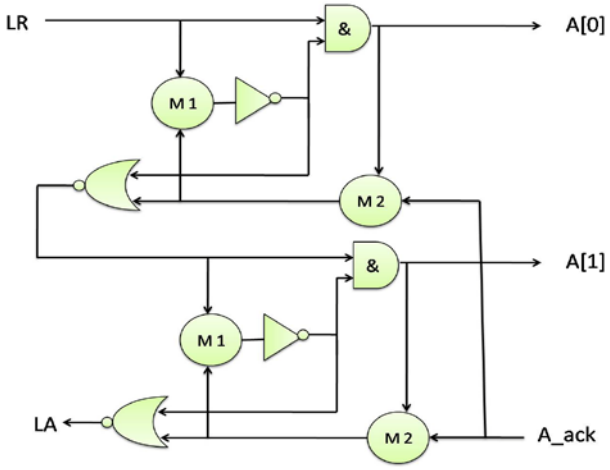


Figure 22 : Contrôleur Séquenceur à une sortie codé en double rail

Le tableau 2 montre les résultats de la simulation électrique pour ces deux circuits. Le circuit 2 est meilleur en terme du surface (-38% moins), débit (2,2 fois plus rapide), puissance (-37%) et énergie (3,5 fois moins consommant) que le circuit 1.

Tableau 2 : Simulation électrique des deux contrôleurs Séquenceurs

Contrôleur Séquenceur SO (4 rails)	*Nombre de transistors*	*Débit M.seq. s-1*	*Avg. pic du courant (µA)*	*Puissance (µW)*	*Energie (pJ)*
Circuit 1 (Machine à états)	206	109	234	108	0,99
Circuit 2 (séquenceurs)	128	239	204	68	0,29

En effet, chaque état dans l'architecture séquenceur est mémorisé par une seule porte de Muller. De plus, chaque séquenceur acquitte le séquenceur précédent d'une manière rétroactive, il n'est donc pas besoin d'un arbre d'acquittement (porte NORs). De plus, l'acquittement des sorties est isolé de l'acquittement de l'état suivant, à l'inverse de

l'architecture en anneaux (section 6.5). Tout cela conduit à une implémentation plus optimale que celle du circuit 1.

8. Conclusion

Ce chapitre a illustré les différents types de contrôleurs QDI, puis a montré l'utilisation de ce contrôleur dans un circuit QDI. Il a ensuite présenté le type SIMO qui est traité au cours de cette thèse. La méthode TAST a été appliquée afin d'implémenter des fonctions séquentielles. La description CHP de ce contrôleur s'est transformée en anneaux asynchrones ou machine à états. Il a été montré qu'il est possible d'optimiser le code CHP afin de minimiser le nombre d'états de machine. Ces mesures ont conduit à une meilleure performance de point de vue de la consommation. Nous avons finalement illustré que l'utilisation d'un composant séquentiel appelé séquenceur peut conduire à des architectures de contrôleurs meilleures en surface, consommation, débit et pic de courant.

Notre contribution dans ce chapitre est la méthodologie de synthèse des contrôleurs QDI à partir de leur description en CHP selon l'approche TAST. C'est une première étude de l'arbre d'acquittement pour des contrôleurs QDI multi-sorties. Nous avons également proposé une architecture en séquenceurs afin de réaliser des fonctions « Séquenceurs SO ». Les bons résultats obtenus grâce à cette dernière nous encouragent à explorer des architectures plus complexes de contrôles en utilisant le séquenceur. Dans ce chapitre nous n'avons pas montré les techniques utilisées pour générer les *netlists* des circuits présentés, les décomposer [17] et effectuer la projection technologique[2].

9. Bibliographie

1. Bregier, V., *Automatic synthesis of Asynchronous Proven Quasi Delay Insensitive Circuits.* PHD thesis, Institut National Polytechnique Grenoble, 2007.
2. Folco, B., *Contribution à la synthèse des circuits asynchrones Quasi Insensibles aux Délais, application aux systèmes sécurisés.* These, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2007.
3. Fragoso., J., *Data Paths Automatic Generation in QDI Asynchronous Logic.* Ph.D thesis, Institut National Polytechnique de Grenoble, Laboratoire TIMA, 2005.
4. Rios, D., *Systeme a microprocesseur asynchrone basse consommation* These Docteur-Ingénieur, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2008.
5. Nowick, S., *Automatic synthesis of BURST-MODE asynchronous states controllers* Stanford University Ph.D, Departements of electrical engineering, 1995.

6. R.M. Fuhrer, S.M.N., M. Theobald, N.K. Jha, B. Lin and L. Plana, *Minimalist : An environment for the synthesis, verification and testability of burst-mode asynchronous machines.* TR CUCS-020-99, 1999.

7. A.-V. Dinh-Duc, L.F., and M. Renaudin, *Synthesis of QDI Asynchronous Circuits from DTL-style Petri Nets.* 11th IEEE/ACM International Workshop on Logic and Synthesis, 2002.

8. Folco, B., *Contribution to Synthesis of Asynchronous Quasi Delay Insensitive Circuits, Application to Secured Systems.* PHD thesis, Institut National Polytechnique Grenoble, 2007.

9. Martin, A.J., *Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits.* California Institute of Technology, Pasadena, CA., 1989.

10. Hoare, C.A.R., *Communicating sequential processes.* Communications of the ACM, Aug. 1978. **21**(8): p. 666-677.

11. Dijkstra, E.W., *A Discipline of Programming.* Prentice Hall PTR, Upper Saddle River, NJ., 1997.

12. DINH-DUC, A.-V., *Synthese Automatique de Circuits Asynchrones QDI.* Thèse en microélectronique, INPG, Ecole doctroral EEATS, Grenoble, 2003.

13. Williams, T., *Latency and Throughput Tradeoffs in Self-Timed Speed-Independent Pipelines and Rings.* Technical Report No. CSL-TR-90-431, 1990.

14. Renaudin, E.Y.a.M., *QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis* Springer Berlin / Heidelberg : Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, 2006. **Volume 4148/2006**.

15. Arámbula, D.R., *SYSTEMES A MICROPROCESSEURS ASYNCHRONES BASSE CONSOMMATION*. Thèse à Grenoble-INP, 2008.

16. Nowick, L.P.a.S., *Concurrency-Oriented Optimization for low-power asynchronous Systems.* International Symposium on low Power Electronics and Design, 1996: p. 151-156.

17. RIGAUD, J.-B., *Spécification de bibléothèques pour la synthèse de circuits asynchrones* THèse à Grenoble-INP, 2002.

1. Bregier, V., *Automatic synthesis of Asynchronous Proven Quasi Delay Insensitive Circuits.* PHD thesis, Institut National Polytechnique Grenoble, 2007.

2. Folco, B., *Contribution à la synthèse des circuits asynchrones Quasi Insensibles aux Délais, application aux systèmes sécurisés.* These, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2007.

3. Fragoso., J., *Data Paths Automatic Generation in QDI Asynchronous Logic.* Ph.D thesis, Institut National Polytechnique de Grenoble, Laboratoire TIMA, 2005.

4. Rios, D., *Systeme a microprocesseur asynchrone basse consommation* These Docteur-Ingénieur, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2008.

5. Nowick, S., *Automatic synthesis of BURST-MODE asynchronous states controllers* Stanford University Ph.D, Departements of electrical engineering, 1995.

6. R.M. Fuhrer, S.M.N., M. Theobald, N.K. Jha, B. Lin and L. Plana, *Minimalist : An environment for the synthesis, verification and testability of burst-mode asynchronous machines.* TR CUCS-020-99, 1999.

7. A.-V. Dinh-Duc, L.F., and M. Renaudin, *Synthesis of QDI Asynchronous Circuits from DTL-style Petri Nets.* 11th IEEE/ACM International Workshop on Logic and Synthesis, 2002.

8. Folco, B., *Contribution to Synthesis of Asynchronous Quasi Delay Insensitive Circuits, Application to Secured Systems.* PHD thesis, Institut National Polytechnique Grenoble, 2007.

9. Martin, A.J., *Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits.* California Institute of Technology, Pasadena, CA., 1989.

10. Hoare, C.A.R., *Communicating sequential processes.* Communications of the ACM, Aug. 1978. **21**(8): p. 666-677.

11. Dijkstra, E.W., *A Discipline of Programming.* Prentice Hall PTR, Upper Saddle River, NJ., 1997.

12. DINH-DUC, A.-V., *Synthese Automatique de Circuits Asynchrones QDI.* Thèse en microélectronique, INPG, Ecole doctroral EEATS, Grenoble, 2003.

13. Williams, T., *Latency and Throughput Tradeoffs in Self-Timed Speed-Independent Pipelines and Rings.* Technical Report No. CSL-TR-90-431, 1990.

14. Renaudin, E.Y.a.M., *QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis* Springer Berlin / Heidelberg : Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, 2006. Volume 4148/2006.

15. Arámbula, D.R., SYSTEMES A *MICROPROCESSEURS ASYNCHRONES BASSE CONSOMMATION.* Thèse à Grenoble-INP, 2008.

16. Nowick, L.P.a.S., *Concurrency-Oriented Optimization for low-power asynchronous Systems.* International Symposium on low Power Electronics and Design, 1996: p. 151-156.

17. RIGAUD, J.-B., *Spécification de bibléothèques pour la synthèse de circuits asynchrones* THèse à Grenoble-INP, 2002.

CHAPITRE 4 : SYNTHESE DES CONTROLEURS SEQUENTIELS QDI PAR L'IMPLEMENTATION DIRECTE

Ne t'inquiète pas si tu as des difficultés en maths, je peux t'assurer que les miennes sont bien plus importantes !

Ce qui compte ne peut pas toujours être compté, et ce qui peut être compté ne compte pas forcément.

Si vous ne pouvez expliquer un concept à un enfant de six ans, c'est que vous ne le comprenez pas complètement.

Albert Einstein

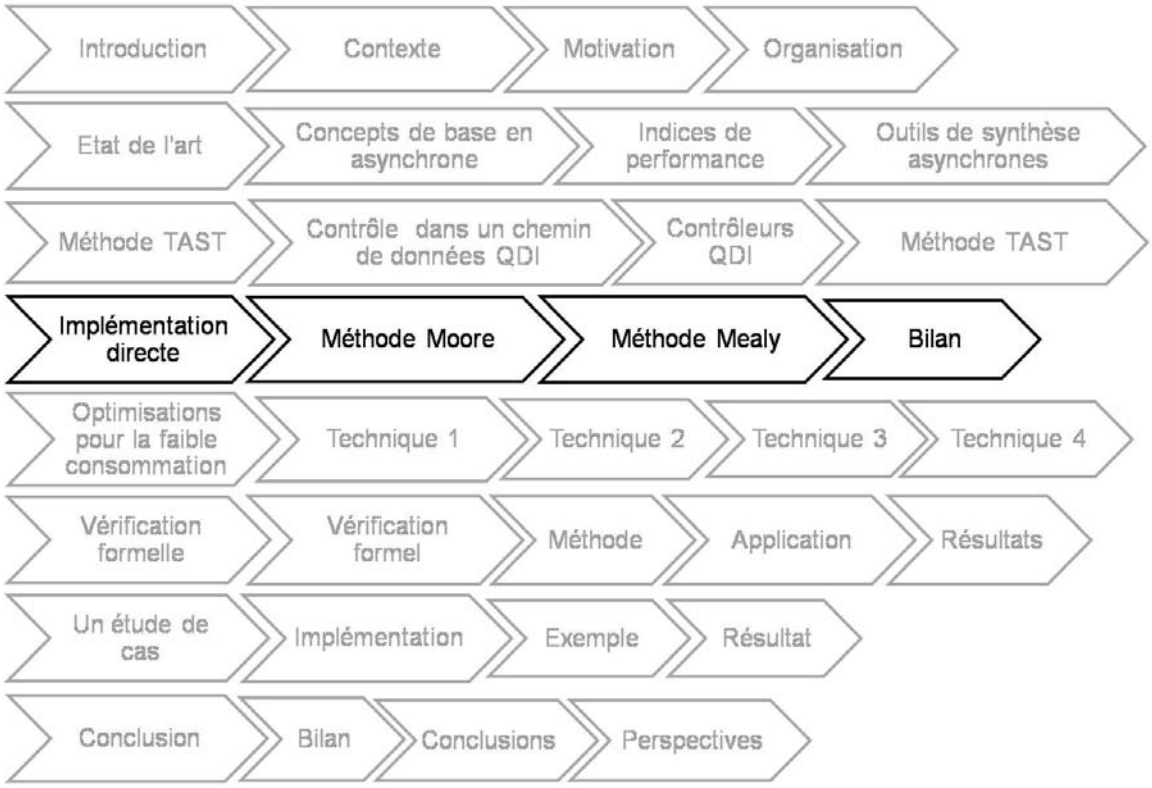


Figure 1 : plan de thèse

Sommaire

Table des figures :

Table des tableaux :

1. Introduction

Le chapitre 3 a présenté une architecture de machine à états asynchrone QDI. Par ailleurs, nous avons également montré la méthode pour synthétiser de tels contrôleurs en suivant un flot de conception « asynchrone » comme TAST. Cependant, nous avons montré que l'architecture des machines à états à base d'anneaux asynchrones n'est pas optimale en termes de vitesse, de consommation et de surface. Nous avons donc introduit un composant séquentiel dit « Séquenceur » permettant de réaliser des contrôleurs séquentiels simples.

Dans ce chapitre, nous allons élaborer une méthode de synthèse de contrôleurs plus complexes en utilisant ce composant. L'objectif est d'implémenter des architectures de contrôle meilleures à la fois en termes de débit et de consommation.

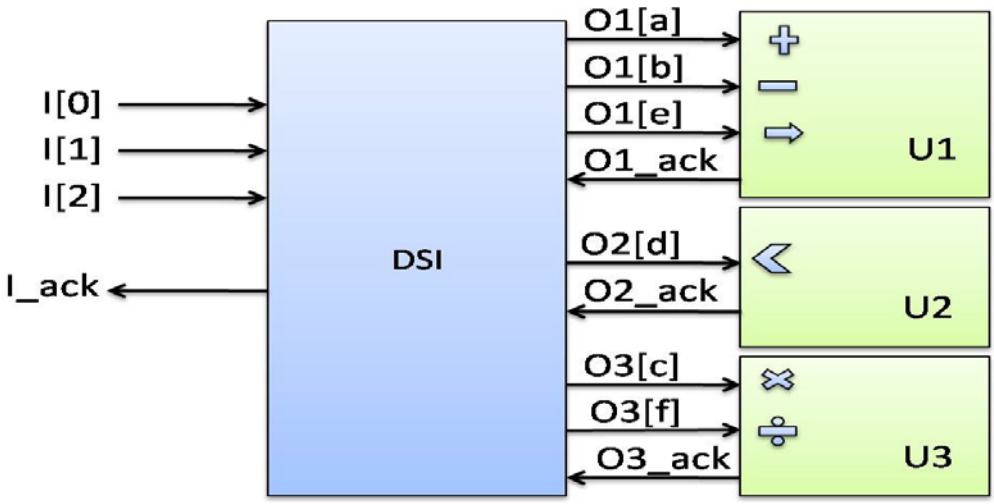


Figure 2 : DSI (Contrôleur d'Instructions Séquentiel)

Nous introduisons tout d'abord un graphe d'ordre partiel conditionnel CPOG (voir définition §2.2) [2-3]. Ce graphe est choisi pour modéliser les comportements des contrôleurs QDI de deux façons. La première est une description inspirée des machines à états type « Moore » alors que la deuxième est une description inspirée des machines à états type « Mealy ». Cette dernière présente l'avantage de réduire le nombre d'états (actions) séquentiels d'un contrôleur par rapport à la représentation Moore. L'optimisation de ces deux descriptions est ensuite présentée à travers plusieurs algorithmes.

A partir de ces graphes, le circuit peut ensuite être implémenté en appliquant des règles simples. L'algorithme de synthèse est en effet une implémentation directe de la

description graphique de contrôleur utilisant les séquenceurs. La méthode est illustrée sur l'exemple du DSI (Décodeur Séquentiel d'Instruction) présenté à la Figure 2.

2. Modèle graphique de contrôleurs séquentiel QDI

Un modèle graphique de contrôleur QDI doit décrire des comportements séquentiels et concurrents à la fois. Il doit aussi être assez abstrait pour pouvoir décrire des contrôleurs complexes. Un graphe d'ordre partiel a ces trois propriétés [4-5], il permet une modélisation du même type que les Petri nets [6] (modélisation des comportements concurrents et séquentiels), les machine à états [7] (modélisation des comportements séquentiels) ou les STG [8] (description bas niveau d'événements). Nous nous basons dans ce chapitre sur un travail antérieur de l'équipe « Async » de l'Université de Newcastle[2-3].

2.1. Le Graphe d'Ordre Partiel

Les graphes d'ordres partiels sont naturellement bien adaptés pour la spécification de l'ordre des événements dans un système où certains sont contraints de se produire avant les autres.

Soit E = (1, ..., n) un ensemble de « n » événements. Leur ordre peut être spécifié avec un graphe dirigé POG = (V, R) où V est un ensemble de nœuds et « R » est l'ensemble des couples de nœuds ou des arcs. Chaque nœud représente ainsi une action ou un événement. Chaque arc représente ainsi une relation d'ordre entre deux paires d'événements. Si « a » et « b » sont deux actions de « V » : « a→b » est défini comme la relation d'ordre entre « a » et « b » si « a » doit strictement se produire avant « b ». Si ni « a→b » ni « b→a » ne sont spécifiés, les actions « a » et « b » peuvent donc se produire dans n'importe quel ordre, éventuellement simultanément.

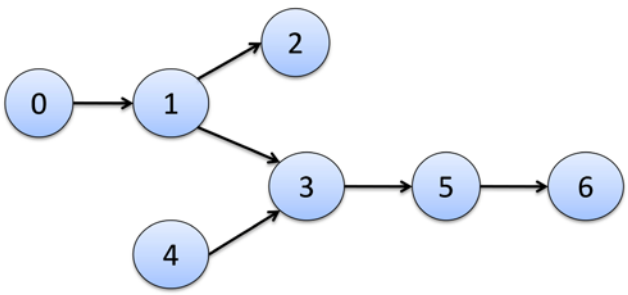


Figure 3 : Exemple de graphe d'ordre partiel

La figure 3 montre un exemple de graphe d'ordre partiel : L'action « 1 » peut se produire seulement après que l'action « 0 » ait eu lieu, ce qui est symbolisé par un arc « 0 →1 ». L'action « 3 » peut se produire uniquement lorsque ses deux actions précédentes, « 1 » et « 4 », ont eu lieu et ainsi de suite. Si deux actions ne sont pas reliées par un arc, alors elles peuvent se produisent indépendamment l'une de l'autre (« 2 » et « 5 » par exemple). Néanmoins, il peut exister une dépendance indirecte entre deux actions, comme « 1 » et « 6 ». Ces deux actions ne sont pas directement dépendantes, mais clairement l'action « 6 » ne peut se produire qu'après l'action « 1 ».

Les ordres partiels ont les propriétés suivantes :

Relation irréflexive : $\forall\, a \in A, Non(a \to a)$;

Relation asymétrie : $\forall\, a, b \in A, (a \to b) \Longrightarrow Non(b \to a)$;

Relation transitivité : $\forall\, a, b, c \in A, (a \to b) \wedge (b \to c) \Longrightarrow (a \to c)$.

2.2. Le Graphe d'Ordre Partiel Conditionnel

Mochove et Yockvlove [2] ont introduit le concept de condition dans un graphe d'ordre partiel. Lorsque les arcs du graphe sont marqués par des signaux de contrôle, il devient Graphe d'Ordre Partiel Conditionnel CPOG (Conditional Partial Order Graph). Ce graphe marqué a été utilisé ensuite par Mochove pour modéliser un type de contrôleur séquentiel asynchrone indépendant de la vitesse appelé « Encodeur des Séquences des Transitions » ou TSE (Transition Sequence Encoder).

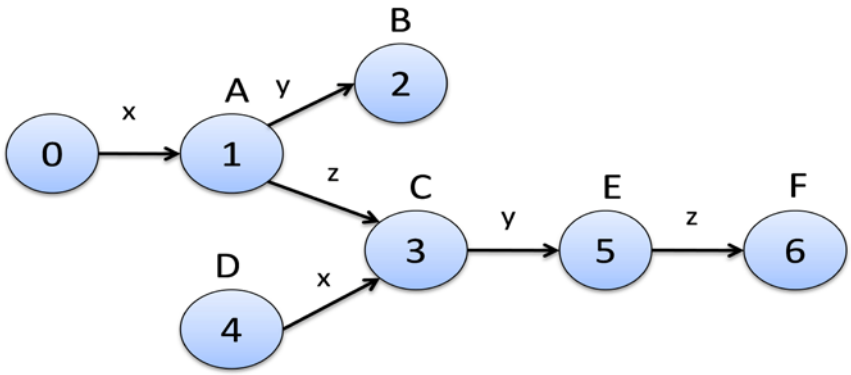


Figure 4 : Exemple d'ordre partial conditionnel

Le graphe d'ordre partiel conditionnel (Figure 4) est un n-uplet CPOG (V, A, T, X, λ, σ) où V est l'ensemble des nœuds ; T : V × V est l'ensemble des arcs ; A : est

l'ensemble d'actions (ou événements) ; λ : V→A est une fonction d'étiquetage qui établit la correspondance entre les nœuds du graphe et les actions qui se déroulent dans le système modélisé ; X est l'ensemble des variables booléennes et la fonctionσ : T→F(X) attribue une condition à chaque arc dans le graphe. Une condition sur un arc $t \in T$ $t \in T$ est une fonction booléenne où $\sigma(T) \in F(X)$ est l'ensemble de toutes les fonctions booléennes sur les variables dans X. Les TSE sont des contrôleurs séquentiels asynchrones, ils se distinguent de nos contrôleurs QDI en trois points essentiels : d'abord, ces contrôleurs adoptent des hypothèses temporelles plus fortes que la fourche isochrones : ils ne sont donc pas QDI. En fait, ils supposent que les signaux de contrôle sont émis avant un signal « Go » qui déclenche la séquence de sorties et que ces signaux restent constants pendant toute l'opération de protocole de poignée de main. Ensuite, la génération de sorties se produit d'une façon monotone [2]. Une fois qu'un signal de sortie passe au niveau haut, il y restera jusqu'à ce que tous les autres signaux de sorties deviennent hauts. En conséquent, la performance totale du système diminue car chaque unité de calcul est obligée d'attendre que les autres unités activées auparavant se réinitialisent avant d'effectuer sa propre réinitialisation. De plus, de tels contrôleurs ne supportent pas un pipeline avec un *feed-back* à cause de la dépendance créée entre les signaux de contrôle pendant l'étape de réinitialisation. En plus, la génération de la même valeur de sortie dans la même séquence n'est pas possible. Enfin, ce contrôleur ne supporte pas des sorties à multi rail.

Notations

Un groupe de notations est nécessaire pour comprendre la formalisation des algorithmes dans la suite.

●a est un nœud précédant nœud « a » : $b = \bullet a \Leftrightarrow a, b \in V \wedge \exists t \in T: t = b \longrightarrow a$;

●A est le groupe de nœuds précédant nœud « a » : $\bullet A = \{b: b \in V \wedge b = \bullet a\}$;

a● est un nœud suivant « a » : $b = a\bullet \Leftrightarrow a, b \in V \wedge \exists t \in T: t = a \longrightarrow b$;

A● est le groupe des nœuds suivant « a » : $A\bullet = \{b: b \in V \wedge b = a\bullet\}$;

$t\blacksquare(a)$ est un arc « t » qui débute en « a » et $\blacksquare t(a)$ est un arc « t » qui termine en a ;

T■(a) est le groupe des arcs qui débutent en « a » : $T\blacksquare(a) = \{t: t \in T \wedge t = t\blacksquare(a)\}$;

■T(a) est le groupe des arcs qui terminent en « a » : $\blacksquare T(a) = \{t: t \in T \wedge t = \blacksquare t(a)\}$;

t● est un nœud où « t » termine : $a = t\bullet$: $a \in V \wedge \exists b \in V \Longrightarrow t = a \longrightarrow b$;

●t est un nœud d'où « t » débute : a= $\bullet t$: $a \in V \wedge \exists b \in V \Longrightarrow t = b \longrightarrow a$;

λ (a) est l'ensemble d'actions pour le nœud « a » ;

σ (t) est l'ensemble de signaux de contrôles sur l'arc t ;

Si T_x : $\{t : \forall t \in T_x => t \in T\}$ est un ensemble d'arcs, σ(T_x) est donc l'ensemble des signaux de contrôle pour toutes les transitions en T_x : $\sigma(T_x) = \{i: i \in I\ , t \in T\ \wedge x \in \sigma(t)\}$.

3. Modélisation des contrôleurs QDI par CPOG :

En premier lieu, nous présentons une modélisation CPOG type Moore des contrôleurs séquentiels QDI. Les nœuds (actions) du graphe modélisent ici le protocole de poignée de main. Nous introduisons le concept de vecteur de sortie pour modéliser l'écriture dans un canal de sortie à multi-rails. Les nœuds du graphe sont marqués par les vecteurs de sortie et les arcs sont marqués par les signaux de contrôle. Cette modélisation est nommée Moore_MR_CPOG.

En deuxième lieu, nous présentons une modélisation CPOG de type Mealy. Les nœuds du graphe sont ici marqués par un numéro correspondant à l'ordre de sortie dans la séquence générée par le contrôleur. Les arcs sont marqués à la fois par les signaux de contrôle et les vecteurs de sorties. Cette modélisation est nommée Mealy_MR_CPOG.

3.1. Modélisation Moore_MR_CPOG de contrôleurs QDI

3.1.1. Définition

Moor_MR_CPOG est un n-uplet (V, A, I, 0, e, O[X], λ, σ, δ) (Figure 5) où :

A : est le groupe d'actions où une action a ∈ A modélise ici une séquence de protocole 4 phases [a+ ; a_ack+ ; a- ; a_ack-] ;

I : est le groupe de signaux du canal d'entrée ou de contrôle : I[k] est le signal « k » du canal d'entrée « I ». Ces signaux sont mutuellement exclusifs pour la valeur 1 ou jamais « 1 » ensemble ;

O[X] : est un groupe des couples ou vecteurs de sortie noté Om[k], où « m » est l'indice de canal de sortie et « k » correspond au signal k de ce canal ;

δ : A → O[X] est la fonction qui attribue un groupe de vecteurs de sortie à une action « a » ;

0 : est une action qui représente l'état initial du système

e : est une action qui modélise l'acquittement « I_ack » pour l'entrée;

Les autres objets gardent leurs significations.

Le figure 5 montre Moore_MR_CPOG qui modélise une séquence de sorties. Cette séquence se produit lorsque le canal d'entrée prend la valeur I[010] ou lorsque I[1]=1.

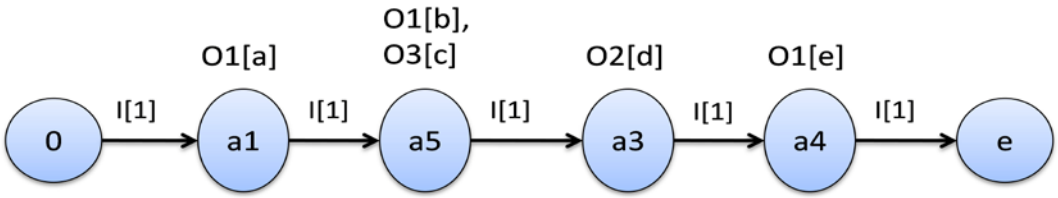


Figure 5 : Modélisation type Moore_MR_CPOG de la séquence S[1] de DSI

Les actions sont uniques dans chaque séquence de sortie. Elles remplacent l'opérateur séquentiel « ; » dans le code CHP (chapitre 3). Plusieurs vecteurs de sorties peuvent être attribués à une action, ce qui modélise une écriture concurrente (parallèle) sur les canaux de sortie (Figure 5**)**. L'action « a5 » a deux vecteurs de sortie : δ (a5)= {O1 [b], O3[C]}). Ce parallélisme est symbolisé par le signe «,» entre les vecteurs sur le graphe. Par contre, une action ne peut pas avoir deux vecteurs de sorties appartenant au même canal de sortie parce que cela viole le principe d'un codage QDI (1-parmis-n) sur les sorties de contrôleur.

Un vecteur de sorties $O_m[k]$ modélise les deux fils ($O_m[k]$, O_m_ack). Il peut être attribué à plusieurs actions en même temps, même si elles appartiennent à la même séquence de sortie. Ainsi générer deux fois la même valeur de sortie dans la même séquence est possible.

Nous définissons la projection inverse δ^{-1} d'un vecteur de sortie $O_m[k]$ comme le groupe d'actions marquées par ce vecteur ou formellement :

$$\delta^{-1} : O[X] \to A : \ \delta^{-1}(Om[k]) = \{a : a \in A \wedge \delta(a) = Om[k]\}$$

δ^{-1}(O1 [b]) est ainsi {a2, a5} dans le Figure 5.

3.1.2. Composition

La première étape pour modéliser un contrôleur QDI est d'écrire un Moore_MR_CPOG partiel (Figure 5) pour modéliser chaque séquence. Cela est valable pour chaque valeur d'entrée. L'étape suivante est de composer un seul Moore_MR_CPOG complet à partir de ces graphes partiels. La figure 6 montre un modèle complet de l'exemple du DSI.

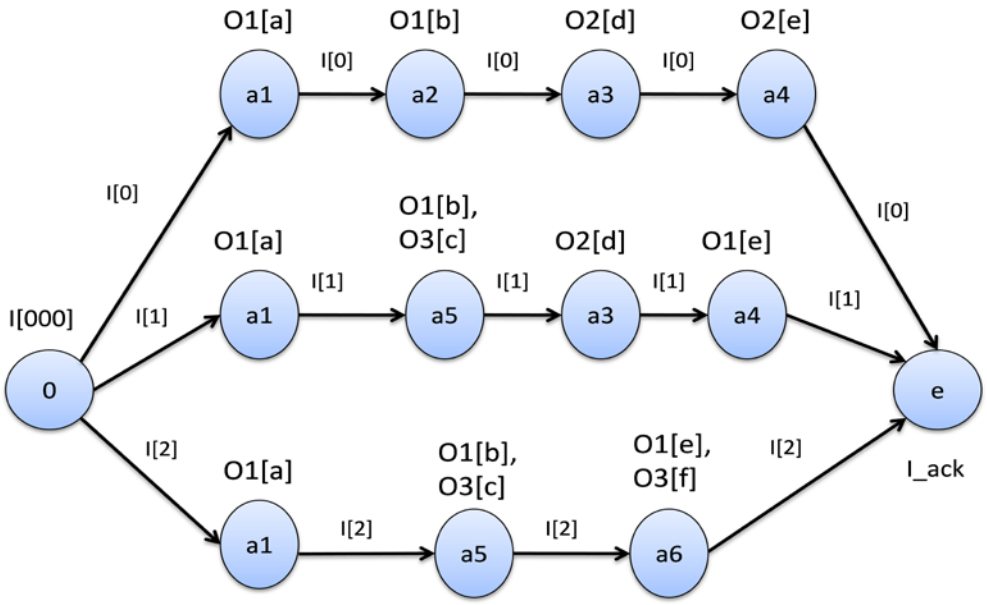


Figure 6: Composition d'ordre partiel conditionnel type Moore à multi-rails

3.1.4. Optimisation du graphe

La synthèse directe de Moore_MR_CPOG (Figure 6) n'est pas optimale parce que certaines actions sont répétées. L'action a1 par exemple appartient aux trois séquences S[1], S[1] et S[3]. Il est possible d'optimiser ce graphe de la manière suivante [4, 9-10]:

1. Réduire le nombre de nœuds de graphe.
2. Réduire le nombre d'arcs.
3. Réduire le nombre des signaux de contrôle sur les arcs.

L'objectif de ces trois étapes est de minimiser la taille de l'implémentation physique et de simplifier la logique dans le circuit et donc la consommation.

1. **Réduire le nombre de nœuds de graphe par élimination d'actions redondantes :**

   Deux actions « a » et « b » modélisent deux événements équivalents. Il est donc

possible de les remplacer par une seule action « c » équivalente. « a » et « b » sont considérés redondants ($a \equiv b$) sous deux conditions :

1- Ils ont le même groupe de vecteurs de sorties.

2- Ils n'appartiennent pas à la même séquence.

$$\forall a, b \in A : a \equiv b \iff \delta(a) = \delta(b) \wedge (\nexists x \in I : x \in \sigma(\blacksquare T(a)) \wedge x \in \sigma(\blacksquare T(b))$$

Le remplacement de « a » et « b » est fait par l'algorithme suivant :

$$\forall a, b \in V \wedge \lambda(a) \equiv \lambda(b) \implies c : \lambda(c) \equiv \lambda(a) \equiv \lambda(b)$$

$$\forall t_x \in \blacksquare T(a) + \blacksquare T(b) \Rightarrow t_y \in \blacksquare T(c) : \bullet t_y = \bullet t_x \wedge \sigma(t_y) = \sigma(t_x)$$

$$\forall t_x \in T\blacksquare(a) + T\blacksquare(b) \Rightarrow t_y \in T\blacksquare(c) : t_y \bullet = t_x \bullet \wedge \sigma(t_y) = \sigma(t_x)$$

A=A+c, A=A-a, A=A-b, T=$T - \blacksquare T(a) - T\blacksquare(a), T = T - \blacksquare T(b) - T\blacksquare(b)$

2. **Réduire le nombre d'arcs par enlèvement des arcs redondants entres actions équivalentes :**

t_x est redondant avec t_y si ces deux arcs connectent entre actions identiques :

$$\forall t_x, t_y \in T : t_x \equiv t_y \Leftrightarrow \bullet t_x = \bullet t_y \wedge t_x \bullet = t_y \bullet$$

L'algorithme pour enlever les arcs redondants est :

$$\forall a \in A, \forall t_x, t_y \in \sigma(\blacksquare T(a)),\ t_x \equiv t_y \implies t_z : t_z \equiv t_x \equiv t_y$$

$$\sigma(t_z) = \sigma(t_x) + \sigma(t_y), T = T + t_z - t_x - t_y$$

Appliquer les étapes «1 » et « 2 », au graphe représenté par la Figure 6, donne le graphe optimisé de la figure 7 : le nombre de nœuds est réduit de 13 à 8, le nombre d'arcs est réduit de 14 à 9.

Dans la figure 7, chaque transition est marquée par un ensemble de signaux de contrôle, le symbole « + » symbolise la relation d'exclusivité entre ces signaux.

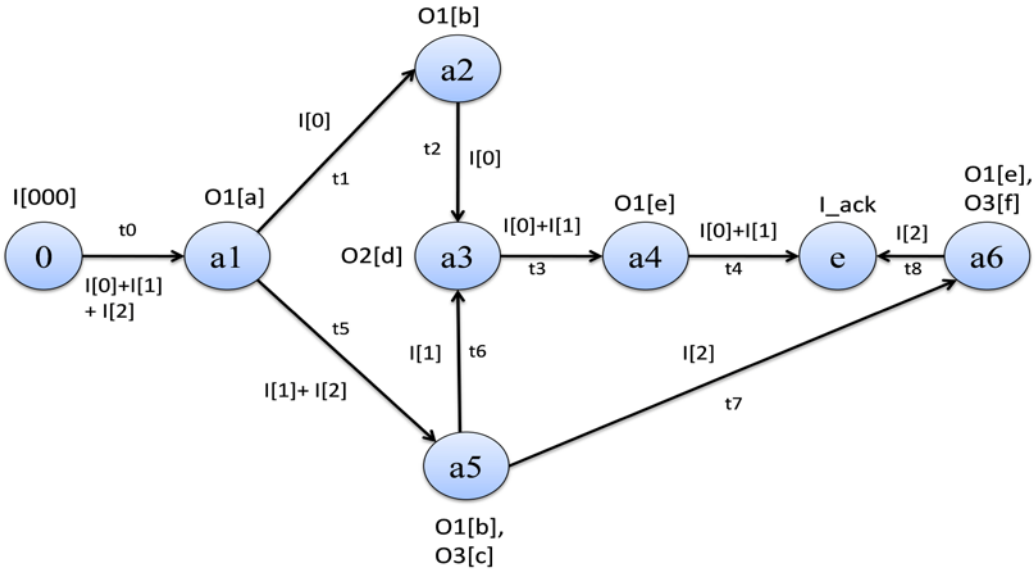


Figure 7 : Optimisation de Moore_MR_CPOG étape 2

3. Réduire les signaux de contrôle par élimination des signaux redondants :

Un signal x est redondant pour une action « a » si « x » appartient à tous les arcs issus de a : $x \in I\ est\ redondant : \ a \ \in A \Leftrightarrow \ \forall t \in T\blacksquare(a) \Longrightarrow x \in \sigma(t)$

Ce signal est éliminé des arcs qui débutent à l'action « a » car ce signal ne décide pas de l'action suivante dans la séquence active. Le signal I[1] sur l'arc (a2→a3) est un signal redondant pour l'action a2 (Figure 7). En effet, une fois « a2 » activé, « a3 » est sûrement activé. L'élimination de I[1] de cet arc simplifie la logique de l'implémentation physique de cet arc.

L'élimination de signaux de contrôle redondants se fait par l'algorithme suivant :

$$\forall a \in A\backslash\{0, e\} \Longrightarrow \forall t \in \ T\blacksquare(a) \ \Longrightarrow \forall x \in \sigma(t) : x\ redondant \ \Longrightarrow$$

$$\sigma(t) = \sigma(t) - x$$

La figure 8 montre le modèle final de DSI où le nombre des signaux de contrôle est réduit de 14 à 7.

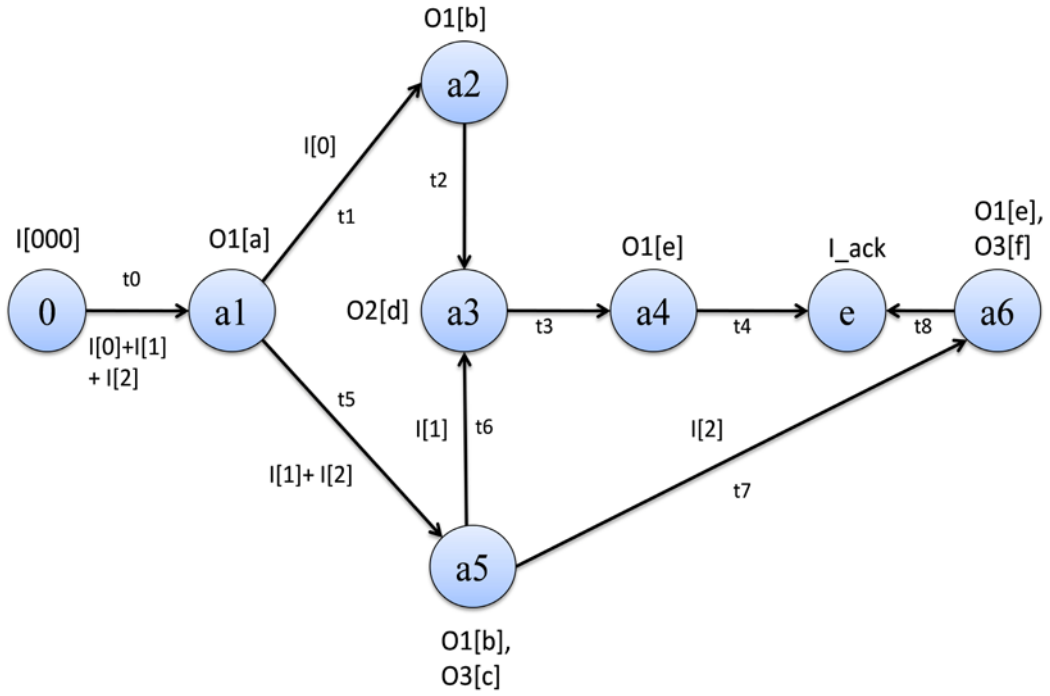


Figure 8 : Optimisation de Moore_MR_CPOG (étape 3)

3.2. Synthèse : Implémentation directe de Moore_MR_CPOG

La technique de l'implémentation directe permet de traduire un graphe directement en circuit [11-13] [23]. Cette technique a plusieurs avantages : l'algorithme est simple et la synthèse de contrôleurs complexes ne génère pas une explosion du nombre d'états [12]. Le circuit synthétisé correspond à la spécification, le débogage du circuit est donc simple [14]. En plus, les règles de l'algorithme sont simples à vérifier formellement.

Dans ce paragraphe, nous présenterons l'algorithme de l'implémentation directe d'un contrôleur QDI à partir d'une description Moore_MR_CPOG. Cet algorithme doit garantir d'abord que le protocole poignée de main se termine sur le canal de sortie avant toute écriture dans un autre canal. Lors d'une écriture parallèle sur deux canaux de sortie, leurs acquittements doivent être synchronisés. Il faut donc considérer que les fourches d'entrée ne sont pas isochrones.

Les séquenceurs sont référencés par Seq_x [LR_x, LA_x, RR_x, RA_x] ; où LR_x, RA_x sont les entrées et LA_x, RR_x les sorties. Les portes logiques sont référencées par NP [I(i), Z()] où NP est le nom de portes. I() représente les entrées de porte alors que Z() référence la sortie de porte. L'entrée I() peut être aussi référencée comme A() et B() ,etc.

M[A(),B(),Z()] par exemple est une porte Muller à deux entrées A et B et une sortie Z. OR[I(),Z()] référence une porte OU, etc.

Le circuit est implémenté à partir d'un Moore_MR_CPOG par les étapes suivantes :

1. **Chaque action « a » sauf {0, e} est implémentée par un séquenceur (Figure 9) :**

$$\forall\, a \in A\backslash\{0,e\} \iff Seq_a[LR_a\,, LA_a, RR_a, RA_a]$$

Un séquenceur reçoit une requête (jeton 1) sur LR, il génère un protocole 4 phases sur un vecteur de sortie (ou plusieurs vecteurs de sortie) par (RR, RA). Ensuite, il passe le jeton à un autre séquenceur par LA. Lorsqu'un séquenceur reçoit un jeton 0, il se réinitialise et il passe ce jeton 0 au séquenceur suivant.

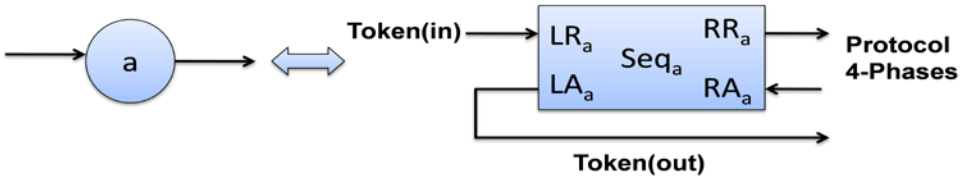


Figure 9 : Implémentation des actions par des séquenceurs

2. **Initialisation :**

Chaque arc (0→a) est implémenté par une porte OR (Figure 10) dont les entrées sont les signaux de contrôle qui marquent l'arc et la sortie connectée à LR_a du séquenceur « a ».

$$\forall\, t \in T\blacksquare(0) \iff OR[I\big(\sigma(t)\big), Z(RA_a)]$$

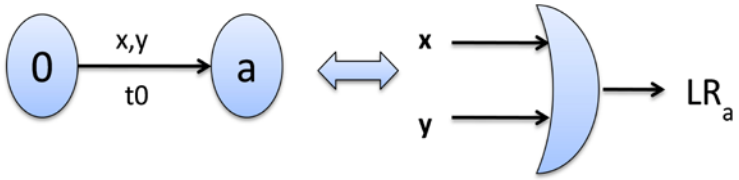


Figure 10 : Initialisation

3. **Génération de la logique des arcs :**

3-1- Chaque arc t_k qui ne débute pas à 0 est implémenté par la structure Muller-OR (Figure 11). D'une part, une porte de Muller représente un point de rendez-vous entre le signal LA_a (le jeton) et un des signaux de contrôle qui marquent l'arc t_k. D'autre part, une porte OR regroupe les sorties de ces portes Muller pour générer un signal t_k.

$$\forall t_k \in T: \bullet t_k \neq 0 \Longrightarrow$$

$$\forall x \in \sigma(t): M\left[A(x), B(LA_a), Z\left(t_{k_i}\right); i++\right]$$

$$OR[I(t_{k_i}); i++, Z(t_k)]$$

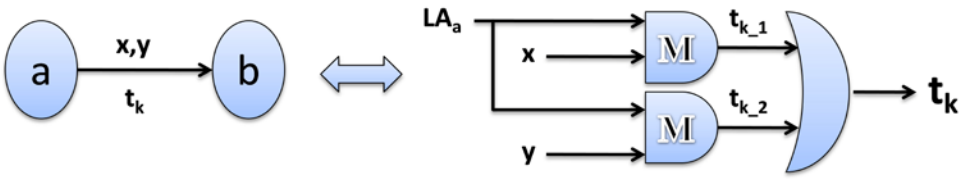


Figure 11 : Implémentation des arcs par des structures M-OR

3-2- L'ensemble des arcs terminant une action « a » est implémenté par une porte « OR » (Figure 12) dont les entrées sont les fils « t_k » produits par l'étape (3-1-).

Lorsque « a≠e » la sortie de cette porte est le signal LR_a :

$$\forall\, a \in A\backslash\{e\}, \forall t \in \sigma\left(\blacksquare T(a)\right) \iff OR[I(t), Z(LR_a)]$$

Lorsque « a=e » la sortie de porte OR est le signal I_ack :

$$a = e \Leftrightarrow OR[I(t), Z(I\_ack)]$$

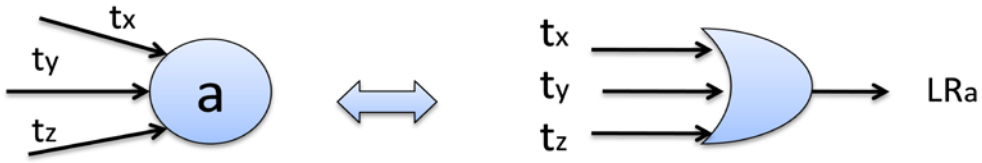


Figure 12 : Génération des signaux de requêtes (LR) pour les séquenceurs

4- Génération des sorties :

4-1- Chaque signal k d'un canal de sortie « O_m » est implémenté par une porte OR (Figure 13) dont les entrées sont les signaux « RR_a », où « a » est une action de la projection des vecteurs de sortie δ^{-1} (O_m[k]). La sortie de cette porte est le signal « k » du canal de sortie « O_m ».

$$\forall\, O_m[k] \in O[X] \Longrightarrow \forall a \in \delta^{-1}(O_m[k]) \Leftrightarrow OR[I(RR_a), Z(O_m[k])]$$

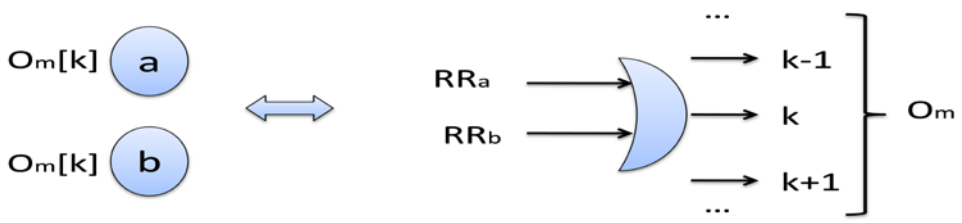


Figure 13 : Génération des sorties

4-2- Chaque signal d'acquittement RA_a, où « a ≠0 et a≠e », est implémenté par une porte de Muller (Figure 14) dont les entrées sont les signaux d'acquittement des vecteurs de sorties, qui marquent l'action « a », et dont la sortie est le signal RA_a.

$$\forall a \in A, \forall\, O_m[k] \in O[X]: O_m[k] \in \delta(a) \iff M[I = O_m\_ack, Z(RR_a)]$$

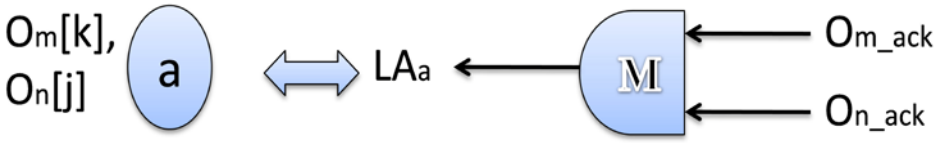


Figure 14 : Génération des signaux d'acquittement (RA)

La synthèse de décodeur d'instructions séquentiel (DSI), à partir du graphe Moore_MR_CPOG (Figure 8), donne le circuit de la figure 15.

Une étape de factorisation logique est nécessaire pour éliminer les duplications de certaines portes comme la porte Muller entre O3_ack et O1_ack.

Le circuit fonctionne en trois modes :

Mode initialisation : Ce circuit est automatiquement initialisé lorsque la valeur des signaux d'entrée et d'acquittement sont à zéro. Ainsi, Il n'y a pas besoin d'un signal reset.

Mode Token 1 : Lorsqu'un des signaux d'entrée est activé par exemple I[1]=1 , le séquenceur « a » est activé et un protocole 4 phases est généré sur le canal O1[a]. Le séquenceur passe le jeton « 1 » à Seq(a5) à travers t5, etc. A la fin de la séquence, le jeton « 1 » passe à I_ack par le signal t4.

Mode Token 0 : Le signal I[1] est remis à zéro, touts les séquenceurs activés par l'étape précédente se sont réinitialisés et le jeton « 0 » est passé au signal I_ack à travers t4.

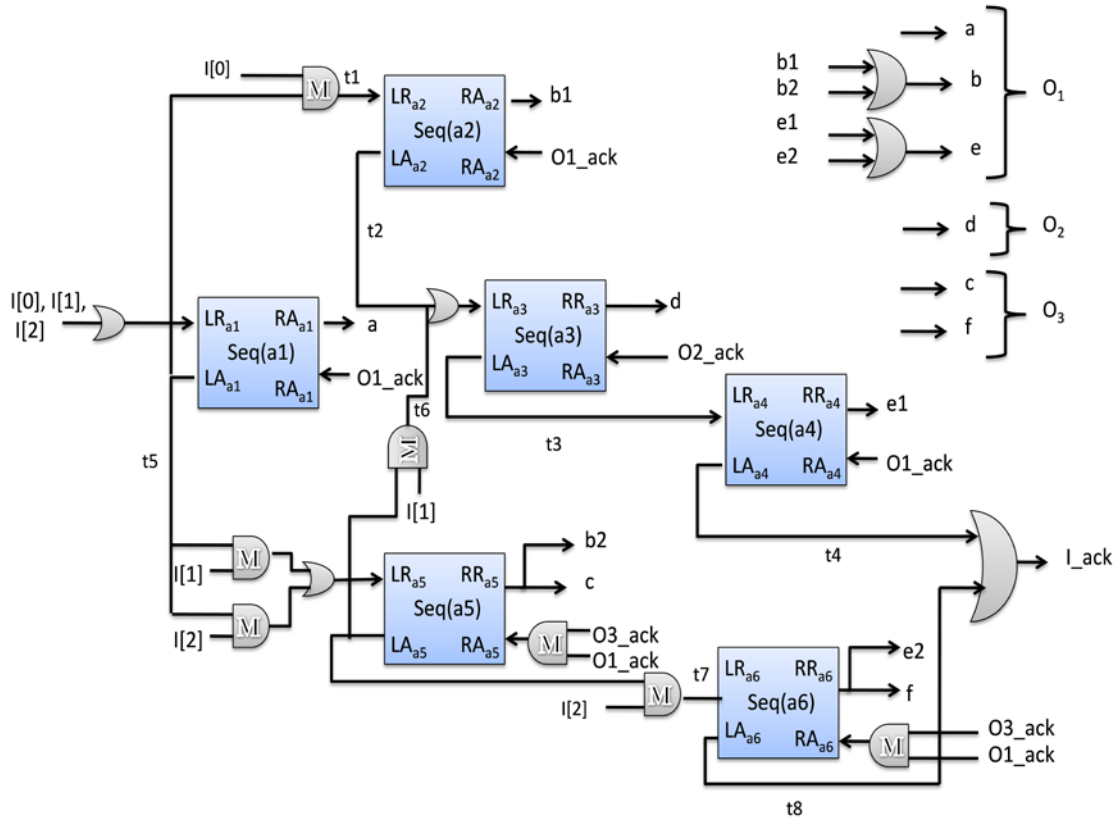


Figure 15 : Circuit du DSI après une implémentation directe (Moore_MR_CPOG)

Le circuit fonctionne d'une manière complètement quasi-insensible aux délais, car la seule hypothèse temporelle à respecter est que les signaux d'acquittement sont des fourches isochrones. Il est plus robuste que le circuit présenté par l'architecture machine à état où les signaux d'entrée sont considérés également isochrones (chapitre 3- paragraphe 6.8).

3.3. Modélisation Mealy_MR_CPOG des contrôleurs QDI

Nous optons maintenant pour une représentation Mealy des contrôleurs (ou Mealy-MR-CPOG) lorsqu'une action modélise l'ordre de sortie dans une séquence du contrôleur. Dans ce modèle, les nœuds du graphe sont marqués par un numéro qui reflète cet ordre. Les arcs sont marqués par des signaux de contrôle et des groupes de vecteurs de sorties.

3.3.1. Définition :

Mealy_MR_CPOG est un n-uplet (V, ℕ, T, I, O[X], 0, e, σ, λ, δ) :

$\mathbb{N}$: est le groupe des numéros « n ». max ($\mathbb{N}$) est le nombre des sorties dans la séquence la plus longue.

O[X] : est le groupe des vecteurs de sorties. Un vecteur de sortie $O_m[k]$ modélise aussi le protocole 4 phases sur le canal de sortie O_m : $\{O_m[k]^+ ; O_m\_ack^+ ; O_m[k]^- : O_m\_ack^-\}$.

λ : V→N : est la fonction qui donne à chaque nœud son ordre dans la séquence de sortie.

δ : T→O[X] : est la fonction qui donne à chaque arc le groupe de vecteurs de sorties.

Les autres éléments du n-uplets gardent leurs significations.

Figure 16 montre le modèle Mealy_MR_CPOG de séquence S[1] de l'exemple de DSI.

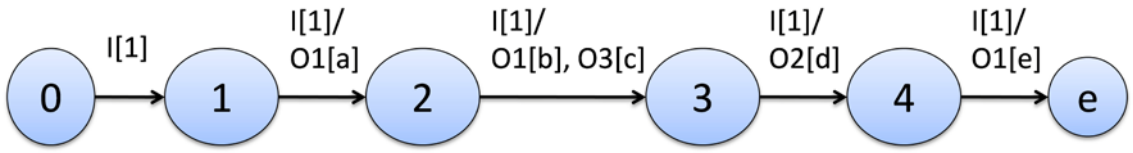


Figure 16 : Modélisation Mealy_MR_CPOG de la séquence S[1] de l'exemple de DSI

3.3.2. Composition

Une description Mealy_MR_CPOG complète du contrôleur est composée à partir des graphes partiels qui modélisent ses séquences de sortie. La figure 17 montre une modélisation Mealy de l'exemple de DSI.

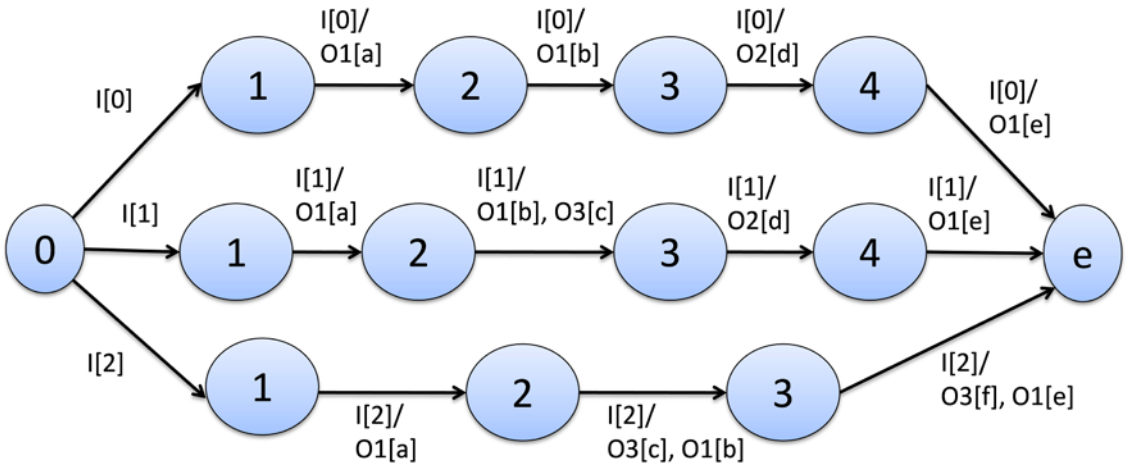


Figure 17 : Composition de Mealy_MR_CPOG

3.3.3. Optimisation de graphe :

La réalisation d'un modèle compact de système se déroule en trois étapes. Ces étapes sont légèrement différentes de celles définies pour optimiser Moore_MR_CPOG :

1. La réduction des nœuds du graphe est obtenue en remplaçant des actions redondantes « n » par une seule action équivalente (voir le point 1 de le paragraphe 3.1.4).

La Figure 18 montre le graphe du DSI après avoir appliqué l'étape 1.

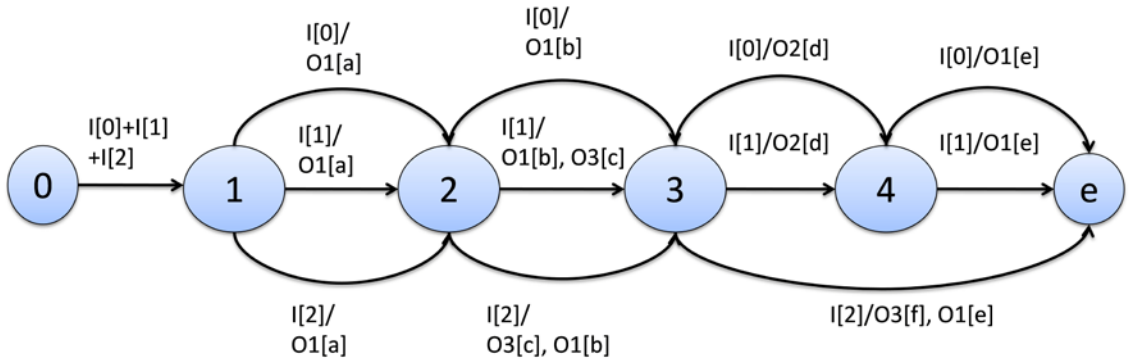


Figure 18 : Optimisation de Mealy_MR_CPOG étape 1.

2. La réduction des arcs est faite par remplacement des arcs redondants par des arcs équivalents :

Dans le modèle de Mealy, deux arcs « t_x , t_y » sont redondants « $t_x \equiv t_y$ » sous deux conditions :

1- Les deux arcs relient les mêmes nœuds « a » et « b » ;

2- « t_x » et « t_y » ont deux groupes de vecteurs de sortie identiques.

$$\forall t_x, t_y \in T: \bullet t_x = \bullet t_y \wedge t_x \bullet = t_y \bullet \wedge \delta(t_x) = \delta(t_y) \Longrightarrow t_x \equiv t_y$$

Un arc équivalent « t_z » est un arc redondant avec « t_x » et « t_y » et qui possède l'ensemble des signaux de contrôle qui marquent « t_x » et « t_y ». L'algorithme pour remplacer les arcs redondants par un arc équivalent est le suivant :

$$\forall n \in \mathbb{N}, \forall t_x, t_y \in \sigma(T\blacksquare(n)): t_x \equiv t_y \Longrightarrow t_z : t_z \equiv t_x \equiv t_y$$

$$\sigma(t_z) = \sigma(t_x) + \sigma(t_y), T = T - t_x - t_y + t_z$$

La Figure 18 montre le graphe du DSI après avoir appliqué l'étape 2.

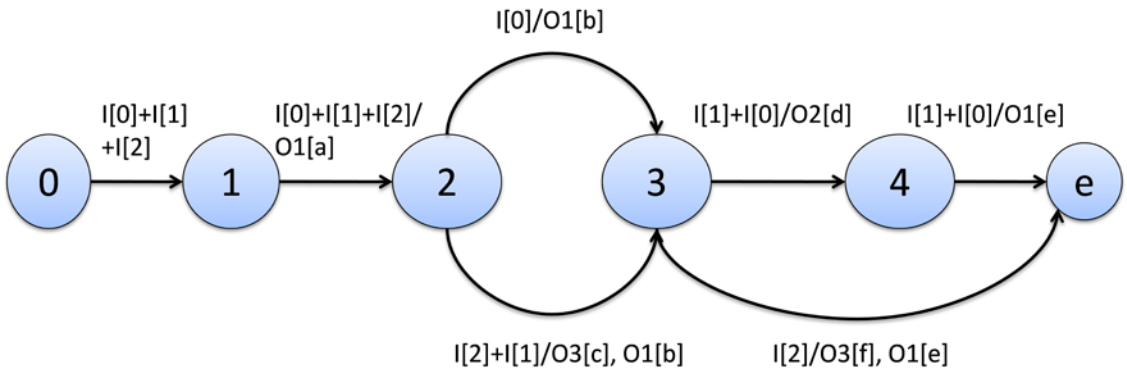


Figure 19 : Optimisation de Mealy_MR_CPOG étape 2

3. Réduire les signaux de contrôle par élimination des signaux redondants (voir le point 3 de la sous-section 3.1.4) :

Appliquer l'étape 3 donne le graphe optimisé de la figure 20.

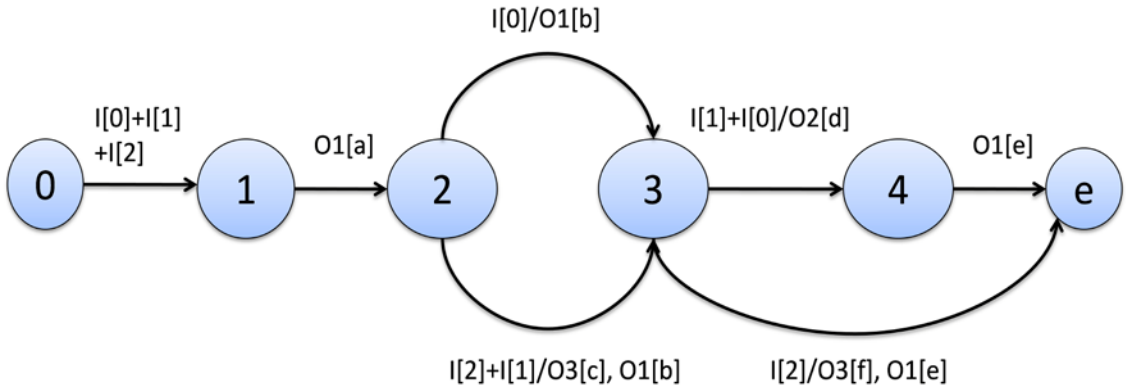


Figure 20 : Optimisation de Mealy_MR_CPOG étape 3

Le nombre d'actions obtenues par l'utilisation de ce deuxième modèle est le nombre d'actions dans la séquence de sorties la plus longue, ce qui est 4 dans le cas du DSI.

3.4. Synthèse : Implémentation directe de Mealy_MR_CPOG :

L'algorithme de l'implémentation directe de Mealy-MR-CPOG (V, ℕ, T, I, O[X], 0, e, σ, λ, δ) se compose des étapes suivantes :

1. Chaque action « a » sauf {0, e} du graphe est implémentée par un séquenceur dans le circuit. Il y a donc max(ℕ) séquenceurs.

$$\forall\, n \in \mathbb{N}\backslash\{0, e\} \iff Seq_n[LR_n\, , LA_n, RR_n, RA_n]$$

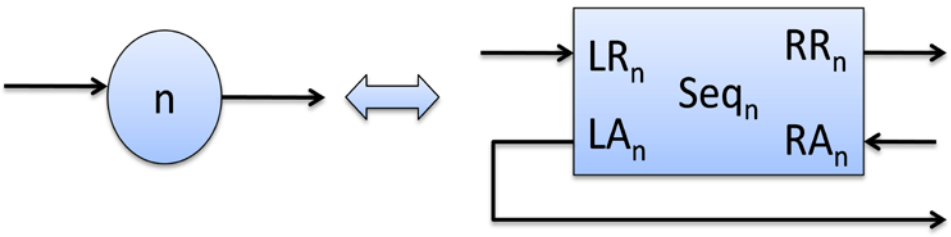


Figure 21 : Implémentation des ordres par des séquenceurs

2. Initialisation :

Chaque signal de contrôle « x » démarre une séquence qui possède au moins une action. Les signaux de contrôle doivent être donc connectés au séquenceur « Seq1 ». Ces signaux sont mutuellement exclusifs pour la valeur 1. Une porte « OR » est alors utilisée pour regrouper ces signaux et générer le signal « LR1 » (Figure 22).

$$\forall x \in I \Leftrightarrow OR[I(x), Z(RA_1)]$$

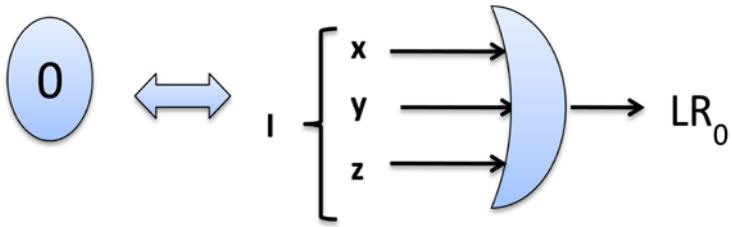


Figure 22 : Porte OR (Initialisation)

3. Logique de contrôle pour les arcs :

3-1- Pour chaque action $n \in \mathbb{N}\backslash\{0, e\}$:

3-1-1- Lorsque « e » n'est pas une action suivant « n » : $e \notin N\bullet$ alors N●= {n+1}. C'est le cas lorsque il y a au moins « n » valeurs de sorties dans les séquences générées par le contrôleur. Dans ce cas, LA_n de séquenceur Seq_n est connecté au signal LR_{n+1} des séquenceurs Seq_{n+1} (Figure 23).

$$\forall n \in \mathbb{N}: e \notin N\bullet \Leftrightarrow N\bullet = n + 1 \Leftrightarrow LR_{n+1} = LA_n$$

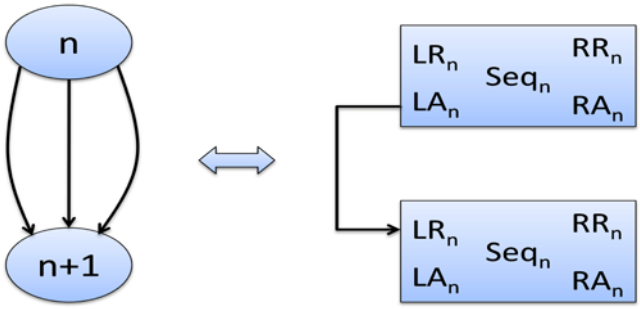


Figure 23 : Implémentation des arcs par connexion directe (LA-LR)

3-1-2- Dans le cas où « e » est une action suivant « n » mais où « n » ne représente pas l'ordre maximum :

- Il faut utiliser un structure M-OR pour implémenter les arcs qui terminent en « n+1 » (Figure 24). Les portes de Muller réalisent ainsi le rendez-vous entre le signal « LA_n » et un des signaux de contrôle. La porte « OR » génère le signal « LR_{n+1} ».

$$\forall n \in \mathbb{N}: e \in N\bullet \wedge n \neq max(\mathbb{N}) \Longrightarrow \forall t \in T: t\bullet = (n+1)$$

$$\Longrightarrow \forall x \in \sigma(t) \Leftrightarrow M\left[A(LA_n), B(x), Z\left(t_{n+1}^{i}\right)\right], i++$$

$$\forall n \in \mathbb{N}: e \in N\bullet \Leftrightarrow OR\left[I = t_{n+1}^{i}, Z(LR_{n+1})\right], i++$$

- Il faut aussi implémenter les arcs terminant en « e » par des portes de Muller (Figure 25) qui génèrent des signaux d'acquittement partiel pour le canal d'entrée. Pour tout $t \in T\blacksquare(n)$ l'algorithme est :

$$\forall n \in \mathbb{N}: e \in N\bullet \Rightarrow \forall t \in T: t\bullet = e \Rightarrow \forall x \in \sigma(t)$$

$$M\left[A(LA_n), B(x), Z\left(I_{\_ack\ j}\right)\right], j++$$

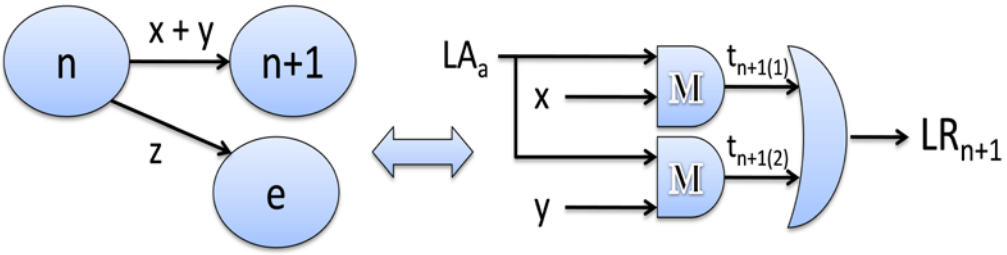


Figure 24 : Implémentation des arcs par les structures M-OR

3-2- Pour l'action « e », il faut implémenter une porte OR dont les entrées sont les signaux I_ack(i) et le signal « LR_N », issu du séquenceur Seq(N) (Figure 25). La sortie de cette porte est le signal d'acquittement pour le canal d'entrée I_ack.

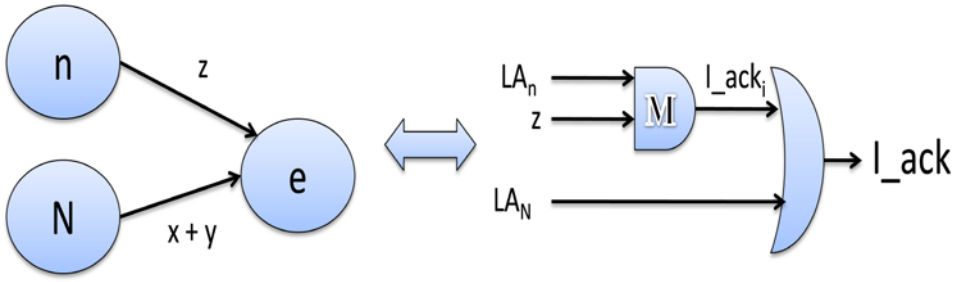


Figure 25 : génération de signal d'acquittement pour le canal d'entrée I_ack

4. Logique de sorties :

Chaque vecteur de sortie $O_m[k]$ dépend de deux facteurs : l'ordre « n » de l'exécution et le signal de contrôle « x » qui identifie cette sortie. Le signal « x » est utilisé pour définir sur quel vecteur de sortie le protocole 4 phases du séquenceur Seq_n est transmis. Ce signal doit donc contrôler le front montant de « RR_n » et « O_m_ack ». C'est pourquoi « x » est connecté à l'entrée dissymétrique de la porte de Muller M2D1P (notation pour les Mullers dissymétriques dans la bibliothèque TAL) (Figure 26) :

$$\forall\, O_m[k] \in O[X] \Rightarrow \forall t \in T : t \in \delta^{-1}(O_m[k]) : \bullet t = n \Rightarrow$$

$$\forall x \in \sigma(t) \Leftrightarrow M2D1P\left[A(RR_n), B(x), Z\left(O_m[k](i)\right)\right], i++$$

$$OR\left[I\left(O_m[k](i)\right),\ Z = O_m[k]\right], i++$$

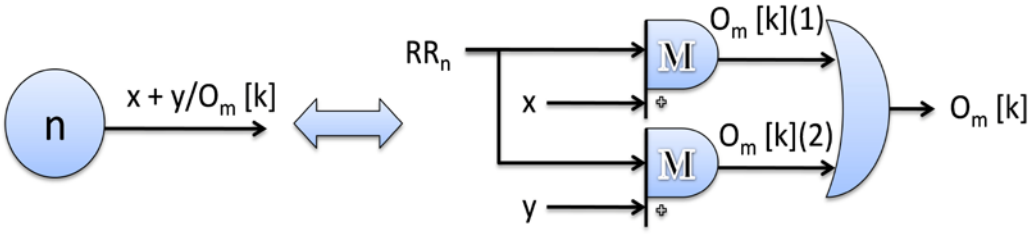


Figure 26 : Génération des sorties

L'acquittement « RA_n » pour un séquenceur Seq(n) est fonction de l'ensemble des acquittements des vecteurs de sorties qui marquent les arcs T■(n), c'est-à-dire le groupe d'arcs qui débute en « n ». Lorsque deux vecteurs de sorties sont activés en parallèle, un point de rendez-vous (Muller) doit être ajouté pour synchroniser leurs acquittements. Les signaux d'entrée « x » contrôlent le passage du bon acquittement au séquenceur.

- Création des signaux partiels d'acquittement :

$$\forall n \in \mathbb{N}\backslash\{0,e\} \Longrightarrow \forall t \in T\blacksquare(n) \Longrightarrow \forall O_m[k] \in \delta(t), \forall x \in \sigma(t),$$

$$\Leftrightarrow MD1P\left[A\left(O_{m\,ack}\right), B(x), Z(ack_i)\right], i++$$

- Création des signaux d'acquittement LA :

$$\forall t \in T\blacksquare(n) \Leftrightarrow OR\left[I(\,I(ack_i), Z(t_j\_ack\,)\right]$$

$$\forall n \in \mathbb{N}\backslash\{0,e\} \Leftrightarrow OR\left[I(\,I(t_j\_ack), Z(LR_n\,)\right]$$

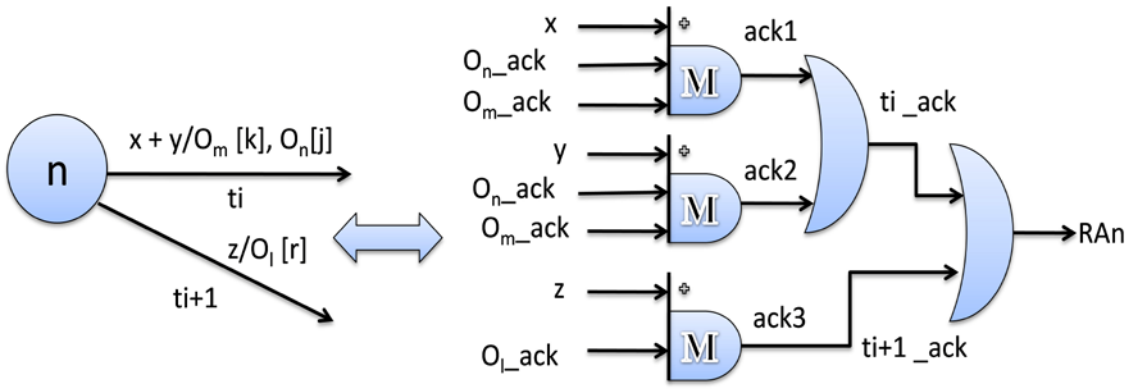


Figure 27 : Génération des signaux d'acquittement RA

La synthèse de Mealy-MR-CPOG du décodeur d'instructions séquentiel (DSI) présenté par le graphe optimisé en Figure 20 donne le circuit de la Figure 28.

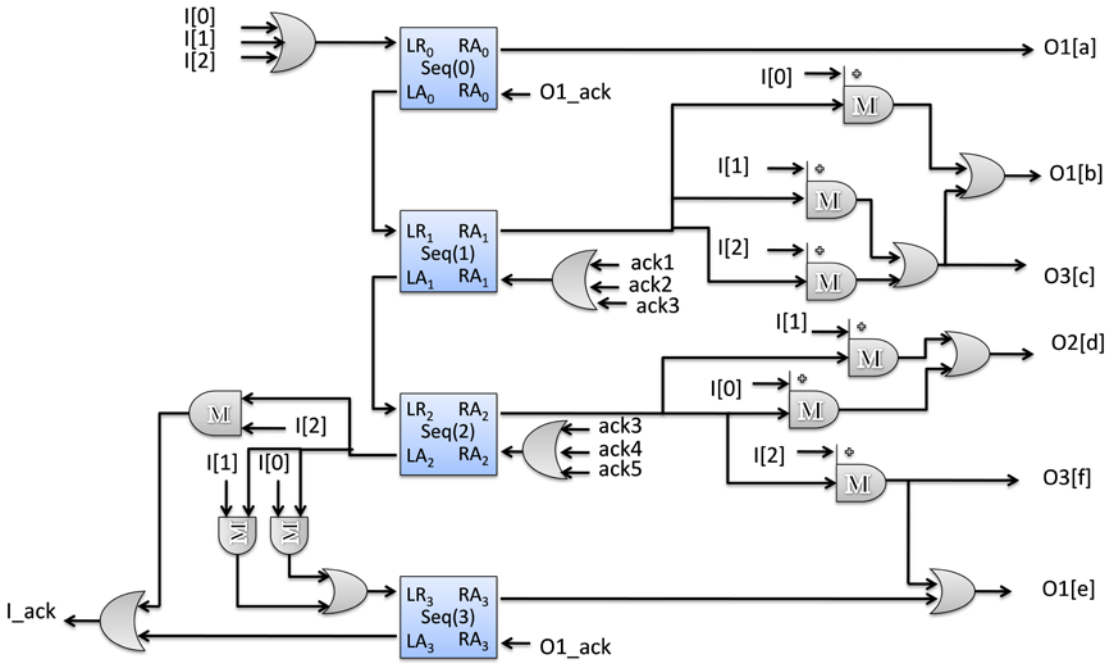


Figure 28 : circuit de DSI après une implémentation directe (Mealy_MR_CPOG)

La génération des signaux d'acquittement ack est illustrée par la figure 29.

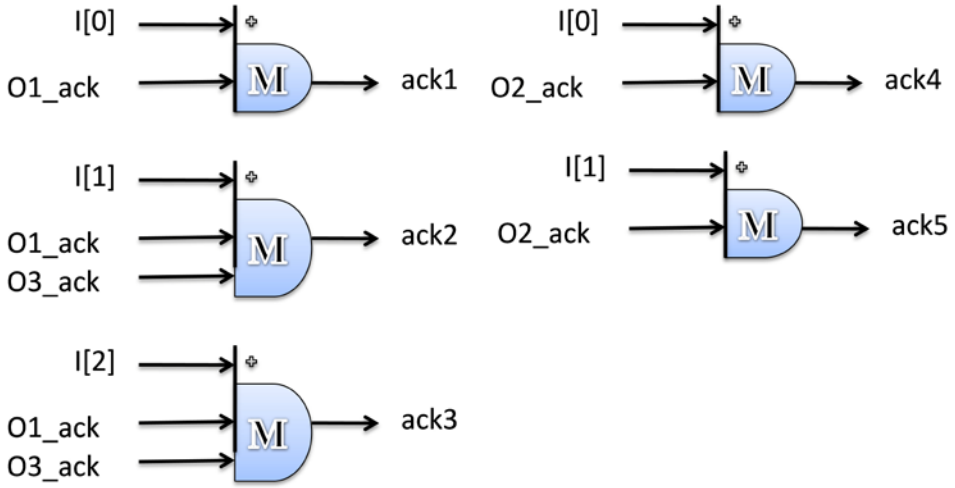


Figure 29 : Génération des signaux d'acquittement ack

Ce circuit fonctionne en trois modes : mode initialisation ; mode jeton 1 ; mode jeton 0.

Dans ce circuit, il y a trois hypothèses temporelles : Les signaux d'acquittement de sorties sont des fourches isochrones. Les fourches des signaux d'entrée sont isochrones également. Le délai des portes OR dans l'arbre d'acquittement de LA_n est inférieur aux

délais des portes OR dans l'arbre d'acquittement de $LA_{n+1.}$ C'est pourquoi cette architecture n'est pas QDI. Par contre, elle est QDI dans le cas de contrôleurs ayant un canal de sortie unique.

Tableau 1: Comparaison entre les trois architectures de l'exemple de DSI

Spectre électrique simulation	Nombre de transistors	débit M. Séq.s-1	Pd (µW)	Ed (pJ)
Arch. 1 (Anneau asynchrone)	449	30,7	92,70	3,02
Arch. 2 (Moore_MR_CPOG)	**283**	**68,0**	**82,65**	**1,21**
Arch. 3 (Mealy_MR_CPOG)	305	57,8	84,65	1,46

4. Bilan

Nous avons simulé électriquement les trois circuits, ceux de la figure 28 et figure 15 et celui réalisé avec une architecture anneau (selon la méthode TAST choisie dans le chapitre 3), afin d'évaluer et de comparer leurs performance. Le tableau « 1 » montre les résultats de cette simulation électrique avec Nanosim.

Les deux nouvelles architectures sont meilleures par rapport à l'architecture en anneau de TAST. La seconde architecture (Moore_MR_CPOG) montre en effet une réduction de 37% du nombre de transistors, un débit 2,22 fois plus rapide et une consommation en énergie réduie de 60%. La troisième architecture (Mealy_MR_CPOG) montre une réduction de 32% en nombre de transistors, un débit 1,88 fois plus rapide et une consommation réduite de 52%.

En effet, la première architecture (anneau asynchrone) nécessitait trois portes de Muller pour mémoriser l'état et générer le protocole 4 phases, alors qu'il suffit d'une seule porte de Muller dans le séquenceur pour réaliser tout cela.

La logique de transition dans les seconde et troisième architectures est isolée de la logique des sorties. Cela implique une réduction du fan-in et fan-out des portes, en conséquence, une diminution de leurs efforts logiques et donc un gain en consommation et en vitesse. Le signal d'acquittement est aussi connecté à moins de portes que dans le cas de l'anneau.

5. Conclusion :

Dans ce chapitre nous avons présenté deux nouvelles architectures pour les contrôleurs asynchrones QDI. Une méthode de modélisation graphique des algorithmes d'optimisation et une méthode de synthèse ont été présentées pour chaque architecture.

L'architecture de Moore est intéressante pour des contrôleurs QDI à sorties multiples. Il est montré avec l'exemple qu'elle est plus robuste, plus rapide, plus petite et moins consommatrice que l'architecture à anneau. L'architecture de Mealy est quant à elle attractive pour des contrôleurs QDI à canal de sortie unique. Elle présente aussi des avantages concernant la surface, la consommation et le débit par rapport à l'architecture en anneau.

Nos contributions dans ce chapitre sont les suivantes :

1- La modélisation des contrôleurs QDI par des graphes d'ordres partiels conditionnels.

2- La présentation du concept de vecteur de sortie afin de pouvoir modéliser les contrôleurs à sortie multi-rails et l'introduction d'une modélisation de type Mealy.

3- Les méthodes de synthèse basées sur le principe de l'implémentation directe et les deux architectures de contrôleurs résultants.

Les perspectives sont :

- L'optimisation de ces architectures pour réduire la consommation et améliorer la performance ;

- La vérification formelle de ces circuits ;

- La validation de la méthode avec une implémentation plus conséquente ;

- L'optimisation de l'architecture de Mealy afin de la rendre QDI pour les contrôleurs à sorties multiples.

6. Bibliographie

1. 1850-2005, I.S., IEEE Standard for Property Specification Language (PSL). 2005. p. 01-143.
2. Mokhov, A.a.Y., Alex, Conditional Partial Order Graphs and Dynamically Reconfigurable Control Synthesis. Design, Automation and Test in Europe, 2008. DATE '08, 2008: p. 1142-1147.

3. Mokhov, A.Y., A. , Verification of conditional partial order graphs. ACSD 2008. 8th International Conference on Application of Concurrency to System Design., June 2008: p. 128-137.
4. Thulasiraman, K.S., M. N. S. , Acyclic Directed Graphs, Graphs: Theory and Algorithms, john Wiley and Son. 1992.
5. Lew, A., Computer Science: A Mathematical Introduction, Prentice-Hall, 1985.
6. Peterson, J.L., Petri Net Theory and the Modeling of Systems. Prentice Hall, ISBN 0-13-661983-5, 1981.
7. Hopcroft, J.E.R.M., Jeffrey D. Ullman Introduction to Automata Theory, Languages, and Computation (2nd ed.). Reading Mass: Addison-Wesley. ISBN 0201441241, (2001).
8. T. A. Chu, C.K.C.L.e.T.S.W., A design methodology for Concurrent VLSI Systems. ICCD Proceedings, 1985: p. 407-410
9. T. H. Cormen, C.E.L., R. L. Rivest, and C. Stein., Introduction to Algorithms. MIT Press. 2001.
10. Yakovlev, A.M.a.A., Conditional Partial Order Graphs and Dynamically Reconfigurable Control Synthesis. Technical Report, NCL-EECE-MSD-TR-2007-119, Newcastle University, 2007.
11. David, R., Modular Design of Asynchronous Circuits Defined by Graphs. IEEE Trans. Comput., 1977. **26**(8): p. 727-737.
12. Sokolov D., B.A., Yakovlev A., Dirct mapping of low-latency asynchronous controllers from STGs. IEEE Transaction on Computers-Aided Design of integrated circuits and systems, 2007. **26**(6): p. 993-1009.
13. Bartky., D.E.M.a.W.S., A theory of asynchronous circuits. In Proceedings of an International Symposium on the Theory of Switching. Harvard University Press., April 1959: p. 204-243.
14. Hollaar, L.A., Direct Implementation of asynchronous control Units. IEEE Transaction on Computers, 1982. **C-31**(12): p. 1133-1141.

CHAPITRE 5 : OPTIMISATIONS POUR LA FAIBLE CONSOMMATION

La valeur d'un homme tient dans sa capacité à donner et non dans sa capacité à recevoir

Il est plus facile de désintégrer un atome qu'un préjugé

Rien n'est plus proche du vrai que le faux

Le mental intuitif est un don sacré et le mental rationnel est un serviteur fidèle. Nous avons créé une société qui honore le serviteur et a oublié le don

Albert Einstein

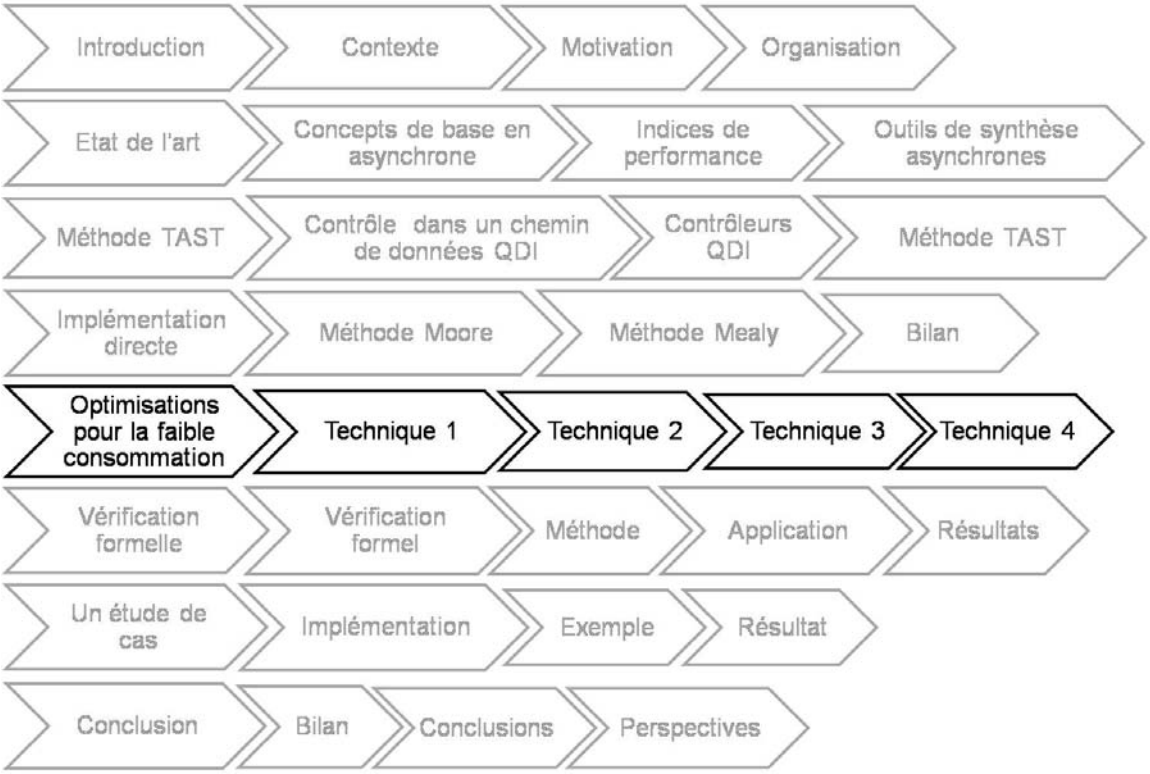


Figure 1 : Plan de thèse

Sommaire

Table des figures :

Table des Tableaux :

1. Introduction :

La réalisation de contrôleurs asynchrones QDI a été illustrée avec trois méthodes : TAST, Moore_MR_CPOG et Mealy_MR_CPOG. Parmi ces trois, l'architecture Moore a montré des propriétés intéressantes en termes de robustesse, de consommation et de débit. Nous avons souhaité investiguer quelques techniques au niveau logique dans l'objectif d'optimiser ces architectures en terme de consommation. Ce chapitre est donc consacré à la présentation de ces techniques. Pour chacune d'elle, nous présenterons les avantages, les inconvénients et le lieu dans le circuit du contrôleur QDI où elles seront appliquées. Puis, ces techniques seront évaluées et validées à travers l'exemple du DSI (Décodeur Séquentiel des Instructions). Nous tirerons enfin quelques conclusions sur cette partie du travail.

2. Optimisations en consommation :

Quatre stratégies sont valables afin d'optimiser l'énergie au niveau logique [1-4]:

1- Réduire le nombre de commutations afin de diminuer la consommation dynamique.

2- Réduire l'effort logique ou la charge sur les entrées et les sorties des portes afin de diminuer le courant court-circuit.

3- Réduire la surface des portes avec comme objectif de diminuer la consommation des portes.

4- Réduire la tension d'alimentation afin de diminuer la consommation du circuit.

L'importance de chaque stratégie par rapport aux autres dépend, d'une part, de la nature et du débit relatif de l'application ; d'autre part, de la technologie employée. Nous exploitons ces trois techniques indépendamment de la technologie.

Le montage présenté en figure 2 a pour objectif de comparer les différentes implémentations qui seront présentées dans ce chapitre. Vdd est la tension d'alimentation pour la partie extérieure au circuit (comme les Buffers). Vdd_sous_test est la tension d'alimentation pour le circuit sous test. V(a), V(b),…, V(n) sont des sources des pulses qui alimentent le circuit par des niveaux logiques (000…0, 100..0, 110..0,…, 111…1, 011…1, 001…1, 000…1,…, 000…0). L'ensemble des vecteurs permet de créer des stimuli comprenant des fronts montants et descendants. Les effets parasites liés aux

successions de 1 et 0 ne sont pas pris en compte. Les performances (consommation dynamique, consommation statique, délai, …) sont mesurées selon la méthode présentée dans l'état d'art (chapitre 2).

La simulation a été effectuée avec Nanosim et spectre de cadence sur une technologie CMOS 130nm de STMicroelectronics.

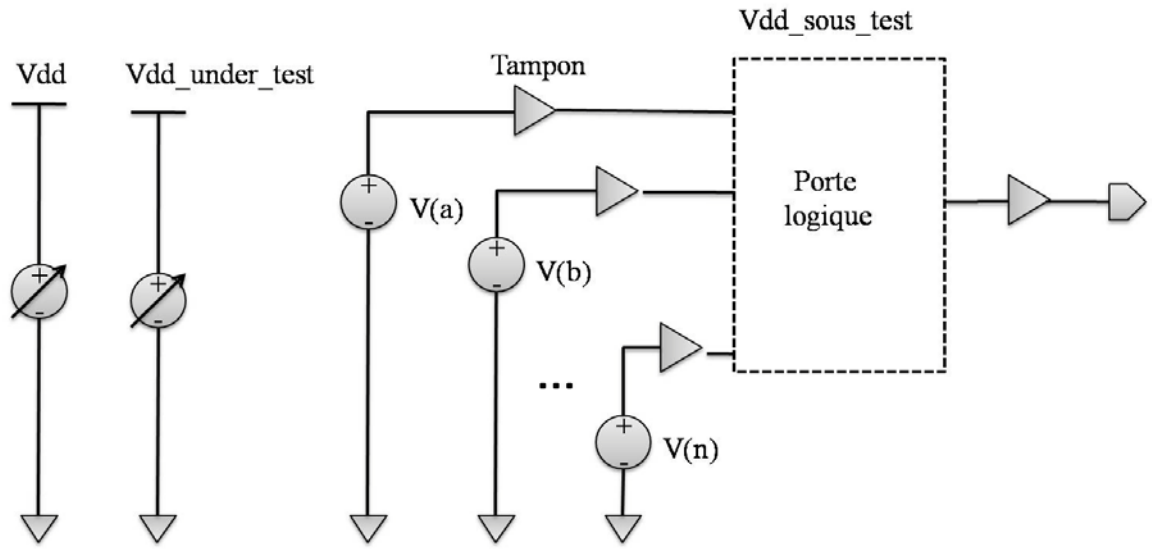


Figure 2 : Montage pour la simulation électrique des portes

2.1. Optimisation logique 1 (Structure OM vis-à-vis MO)

La figure 3 montre la fonction logique basique «2M2-O2 » dans un circuit QDI. Lorsqu'il y a un signal « A » connecté aux portes Muller et lorsque « B » et « C » sont mutuellement exclusives pour la valeur « Un » (jamais B=1 & C=1). Il est possible de remplacer cette structure logique par la structure « O2-M2 » où le signal « A » est connecté à la porte Muller et les deux signaux « B » et « C » sont connectés à la porte « OR ».

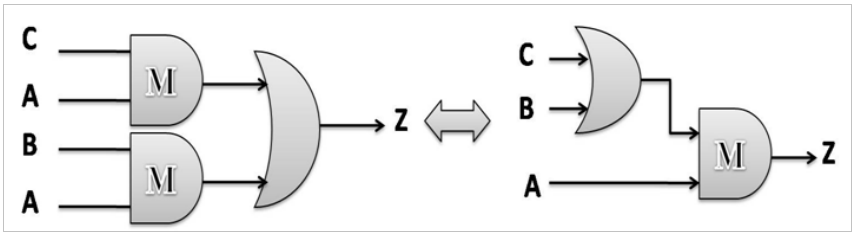


Figure 3 : Fonctions logiques (a) M2-O2 (b) O2-M2

Le tableau 1 montre les résultats de la simulation électrique effectuée sur les deux portes sous une tension opérative 1,2 V.

Tableau 1 : Résultats des simulations électriques pour les fonctions 2M2-O2 et O2-M2

Paramètres de performance	2M2-OR2	OR2-M2
Ed (Energie dynamique) (pJ)	0,115	0,086
Pd (Puissance dynamique) (µW)	1513,8	1093,5
Ps (Puissance statique) (pW)	2069	1346,0
LdT (Largeur de Transistors) (µm)	9,34	6,15
D (Délais) (ps)	380	420
ET (Energie fois Délais) ($p^2w.s^2$)	43,6	36,2

Les avantages :

OR2-M2 est meilleur en termes de consommation en énergie dynamique (-25%), puissance statique (-35%) (Figure 4) car la charge sur chaque fil d'entrée est réduite grâce à un effort logique moins important. Le signal « A » par exemple a un effort logique de 2 résultant d'une seule porte Muller dans la structure O2-M2 au lieu de 4 résultant de deux portes Muller dans la structure 2M2-O2. « B » et « C » sont connectés à une porte « OR » dont l'effort logique est (5/3) au lieu de (2).

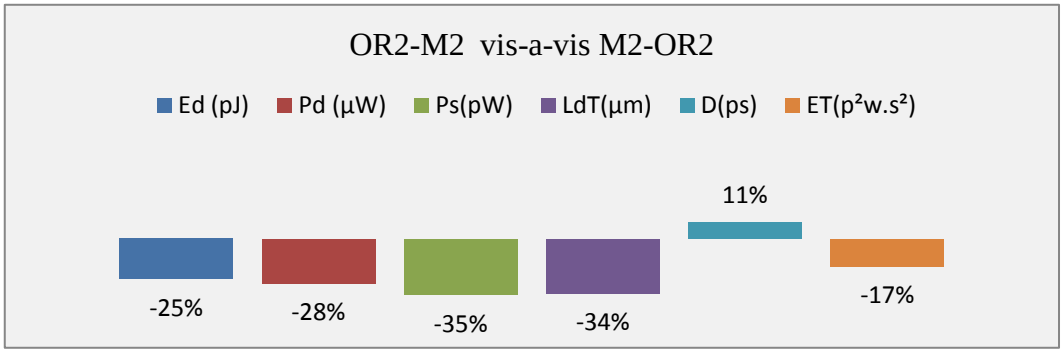


Figure 4 : Gain de OR2-M2 vis-à-vis 2M2-OR2

Malgré le fait que OR2-M2 a un délai plus long (+11%) que M2-O2, elle a un facteur « ET » meilleur (-17%). Cette structure prouve, enfin, un gain en performances proportionnelle au nombre de ses entrées exclusives (B, C, D,…).

Les inconvénients :

Pourquoi la structure OR2-M2 n'existe-t-elle pas dans la bibliothèque TAL ? Tout simplement parce que lorsqu' il y a une fourche sur « C » ou « B », cette fourche n'est pas isochrone selon le modèle QDI. Le délai de la porte OR est indéterminé et le délai de remise à zéro du signal « B » ou « C » ne peut ainsi pas être considéré identique sur les

branches de leurs fourches. Cela peut se traduire par un dysfonctionnement dans un circuit QDI.

Cette fonction logique reste cependant intéressante à utiliser lorsque les deux signaux « B » et « C » sont mutuellement exclusifs pour le « Un » et lorsque ni « B » ni « C » possède une fourche. Cette structure OR-M fonctionne sous une hypothèse temporelle fondamentale [5](délai borné comme dans les circuits synchrones).

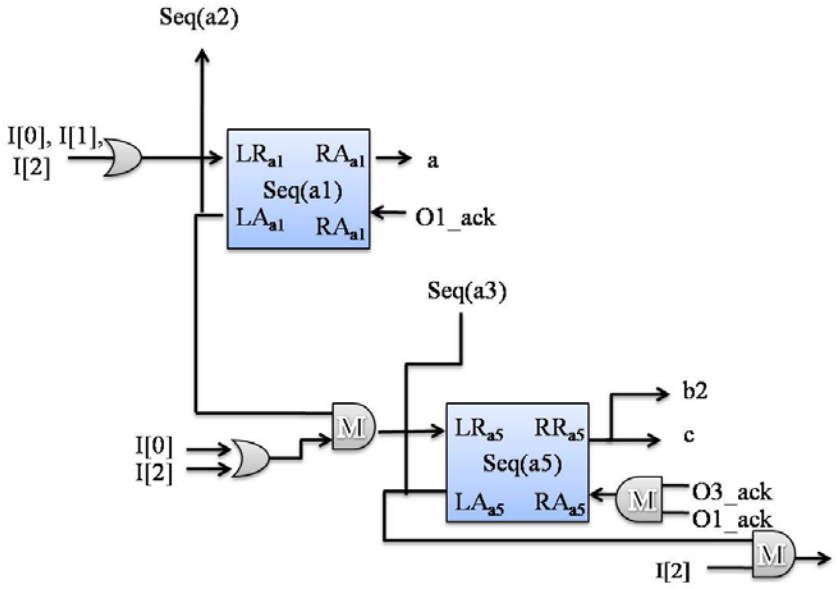


Figure 5 : Fonction logique OR2-M2 dans la transition a1 → a5

Les composants du chemin de données ont des comportements concurrents, autrement dit, les signaux arrivent concurremment. Cette fonction logique ne peut donc pas être appliquée. En revanche, dans un composant séquentiel certains signaux peuvent arriver avant d'autres, il y a ainsi toujours une possibilité d'utiliser cette fonction.

Dans l'architecture de Moore du circuit des contrôleurs QDI par exemple, cette structure est utilisable dans la logique de transition pour remplacer la structure M-OR (Chapitre 4-paragraphe 3.2). La figure 5 montre une partie du circuit de l'exemple du DSI dans laquelle la fonction O2-M2 remplace la fonction 2M2-O2. Dans ce circuit, il y a une corrélation entre le signal « LA_{a1} » et les deux signaux I[0] ou I[1]. Cela dit la remise à « un » ou à « zéro » du « LA_{a1} » est conditionnée par la remise à « un » ou à « zéro » de I[0] ou I[1]. Cette corrélation garantit ainsi la nature QDI du circuit.

2.2. Optimisation logique 2 : portes de Muller dissymétriques :

Une porte Muller dissymétrique (MD) est une porte qui réagit différemment en fonction de l'entrée activée (Figure 6). Un Muller positivement dissymétrique (M(n)D1P) réagit au front montant (+) de son entrée positive et elle est indifférente vers le front descendant de cette entrée. Alors qu'une Muller négativement dissymétrique (M(n)D1N) réagit au front descendant (-) de son entrée négative et elle est indifférente vers le front montant de cette entrée, où (n) est le nombre d'entrées de la porte. Par exemple une porte Muller M2D1P est positivement dissymétrique, elle génère « zéro » sur la sortie lorsque son entrée « A » est en état bas, alors qu'elle génère « un » lorsque ses deux entrées « A » et « B » sont en état haut. La porte est donc sensible au front montant (+) de l'entrée « B » lorsque « A » est « un ». Une porte dissymétrique occupe moins de surface qu'une porte de Muller standard possédant le même nombre d'entrées. L'effort logique de ses entrées est également réduit parce que le nombre de transistors à l'entrée de la porte est réduit.

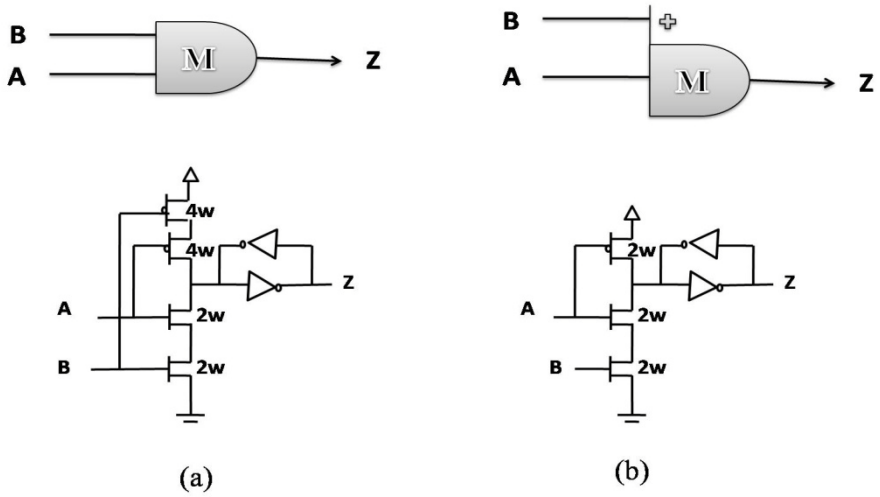


Figure 6 : Porte Muller standard vis-à-vis d'une porte Muller dissymétrique

En fait, l'effort logique de Muller M2 est égal à « 2 » sur chaque entrée, alors que l'effort logique de l'entrée symétrique « A » de la porte Muller M2D1P est de « 4/3 ». En même temps, l'entrée dissymétrique « B » a « 2/3 ». Il est donc intéressant de remplacer une porte Muller par un équivalant dissymétrique où cela est possible afin de diminuer la consommation dynamique due à courant court-circuit.

Le tableau 2 montre une simulation électrique pour comparer ces deux portes.

Les avantages :

La réduction en énergie dynamique atteint 28% et les délais décroissent de 9%. Cela est justifié par la réduction de l'effort logique.

Tableau 2 : Résultats des simulations électriques pour les fonctions M2 et M2D1P

Porte	Ed (pJ)	Pd (µW)	Ps (pW)	D (ns)	ET ($p^2W.s^2$)
M2	0,113	1474	723	250	28,4
M2D1P	0,083	1060	801	230	19,1
Gain	-27%	-28%	10%	-9%	-49%

Les inconvénients :

La consommation statique peut augmenter jusqu'à +10% car le nombre de transistors en série, sur l'étage d'entrée de la porte, a diminué. Cette porte est employée lorsque l'opérateur « # » est utilisé dans le code CHP, autrement dit, lorsqu'il y a un sondage pour la valeur de données (voire chapitre 3, paragraphe 6.1).

La question à se poser à ce stade est la suivante : où est-il possible de remplacer une porte Muller par son équivalent dissymétrique ? En d'autres mots, est-ce qu'il est possible de changer l'entrée « B » d'une porte Muller standard par une entrée « B » positivement dissymétrique ?

Cela en effet implique l'hypothèse suivante dans un circuit QDI : lorsque la sortie « Z » de la porte M2 est à « un », « B » doit toujours passer à « zéro » avant que « A » ne passe à « zéro ». Dans ce cas-là, le seul point de rendez-vous à réaliser est le front montant de « Z », parce que le circuit garantit le rendez-vous pour réaliser le front descendant où $\{Z^+ \rightarrow B^- \rightarrow A- \rightarrow Z-\}$. Le STG sur la Figure 7(a) montre la séquence des signaux pour avoir une entrée « B » positivement dissymétrique.

La deuxième séquence Figure 7 (b) montre la condition pour que « B » soit négativement dissymétrique. Cela dit, lorsque « Z » est « zéro », le circuit garantit que « B » passe à « un » avant que « A » ne passe à « un » $\{Z- \rightarrow B+ \rightarrow A+ \rightarrow Z+\}$.

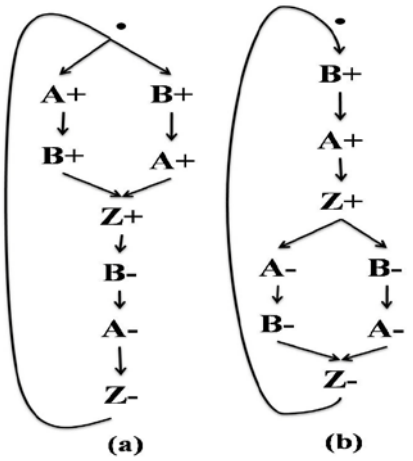


Figure 7 : STG des comportements du circuit permettant l'utilisation d'une porte M2D1P (a) et M2D1N (b)

Le circuit du séquenceur (chapitre 3, paragraphe 7) possède une première région où la condition (a) est valide : il est possible de remplacer les deux portes Müller standard par deux portes dissymétriques. La figure 8 montre le circuit de séquenceur après modifications.

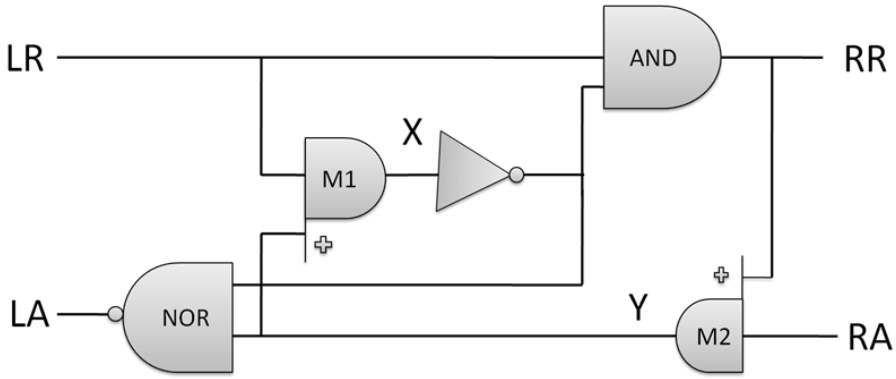


Figure 8 : Circuit optimisé de séquenceur

M2 est ainsi remplacé par une porte positivement dissymétrique car lorsque « Y » est « Un », l'environnement QDI et le protocole 4 phase garantit que « RR » passe à « Zéro » avant « RA ». « RR » est donc connecté à l'entrée positive de M2D1P.

De la même manière, lorsque « X » est égal à « un », il est également garanti que « Y » passe à « zéro » avant que « LR » ne passe à « zéro ». En fait, le passage de « Y » à zéro implique le passage de « LA » à « 1 », cela conduit « LR » à passer à zéro grâce au protocole 4-phase sur « LR » et « LA ».

Pour garantir cette séquence d'événements (Figure 7 (a)), les deux fourches « LR » et « RR » doivent être considérées isochrones. Il suffit donc d'implémenter cette cellule et de l'ajouter à la bibliothèque TAL.

La logique de transition des contrôleurs QDI offre donc une deuxième région où la condition (a) est valide (Figure 9). Lorsqu'une porte Muller reçoit un signal I[x] et un signal « LA_{n-1} » pour générer un signal « LR_n », alors I[x] peut être connecté à l'entrée positive d'une porte M2D1P. La remise à zéro du signal « LA_{n-1} » est conditionnée par la remise à zéro du I[x]. Si la fourche sur I[x] est considérée comme isochrone, la conception de circuit garantit alors que l'entrée « B » revient à zéro avant l'entrée « A » lorsque la sortie « Z » est à l'état haut.

Quant à la structure « OR(n)-M2 » employée pour générer « LR_n », il est possible de la remplacer par la structure OR(n)-M2D1P. « LA_{n-1} » serait connecté à l'entrée « A » de M2D1P alors que la sortie de « OR(n) » serait connectée à l'entrée « B ». Cette optimisation implique deux hypothèses temporelles :

D'un côté, les fourches sur les signaux d'entrée sont isochrones.

D'un autre côté, la remise à zéro de la porte « OR(n) » doit être plus rapide que la remise à « un » d'un signal « LA_{n-1} ». Cela implique que le délai de cette porte « OR(n) » est plus court que celui de la réinitialisation sur le chemin de la séquence la plus courte du contrôleur. Ce circuit n'est donc pas « QDI » !

Pour garantir un fonctionnement correct du circuit, il est possible d'ajouter des buffers sur le signal I_ack qui augmentent le délai sur le chemin de réinitialisation de LA_{n-1}. Le circuit fonctionnera en modèle fondamental. Cependant, il y a une meilleure solution qui sera présentée dans la suite.

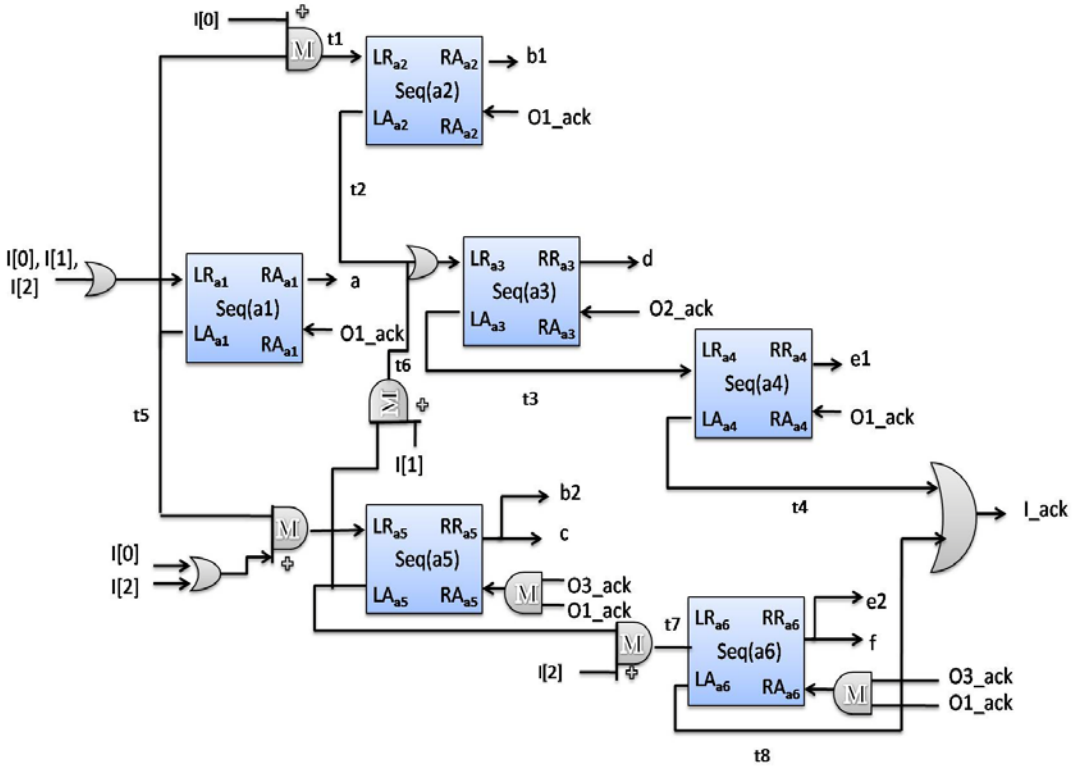


Figure 9 : Circuit DSI après avoir remplacé M2 par M2D1P

2.3. Optimisation logique 3 (Projection technologique) :

Plusieurs portes logiques complexes existent déjà dans la bibliothèque TAL comme les portes « MO » ou Muller-OR. Par contre, Il y manque plusieurs structures logiques intéressantes en termes de performance et qui pourraient exister dans un circuit asynchrone séquentiel QDI, en particulier les portes complexes « porte_logique-Muller » comme « ORM » ou OR-Muller. En plus, l'utilisation des portes dissymétriques offre plusieurs implémentations en transistor avantageuses en termes de rapidité, de surface et de consommation. Par la suite, nous traiterons le cas avec une porte de Muller positivement dissymétrique mais ces études sont aussi valables pour une porte de Muller négativement dissymétrique (l'Annexe « B » montre toutes les portes logiques complexes que nous nous proposons d'ajouter à la bibliothèque TAL).

La figure 10 montre les fonctions logiques (OR2M2, OR2M2D1P) et leurs schémas transistors équivalents :

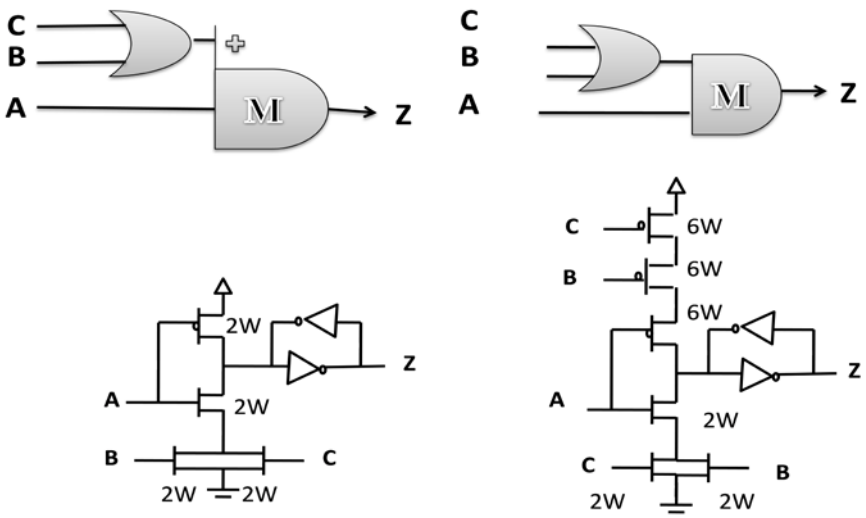


Figure 10 : Porte complexe OR2M2D1P (a) et OR2M2 (b)

Nous allons montrer par une simulation électrique les avantages de cette implémentation. Le tableau 3 montre les résultats de cette simulation effectués sur ces deux fonctions logiques : La première simulation considère deux portes séparées (OR2-M2, OR2-M2D1P) et la seconde l'utilisation des implémentations réduites en nombre de transistors (OR2M2, OR2M2D1P).

Tableau 3 : Simulation électrique des portes complexes

Porte	Tension	Pd (µW)	Ps (pW)	Largeur de transistors (µm)	Délai (ps)	ET ($p^2w.s^2$)
OR2-M1	1,2	1093	1346	6,2	420	36,2
OR2M2	1,2	1494	715	5,7	240	28,9
OR2-M2D1P1	1,2	915	1424	4,9	290	21,3
OR2M2D1P1	1,2	666	801	2,1	240	12,5

La porte complexe OR2M2 montre une consommation dynamique (27%) supérieure à celle des deux portes OR2-M2 (Figure 11). Cette porte a en effet un effort logique plus important (8/3) qui justifie sa consommation dynamique plus grande du fait d'un courant de court-circuit plus important. Par contre, la consommation statique a diminué (-47%), ceci est dû à l'étage d'entrée qui contient plus de transistors en série. La performance totale (ET) est également améliorée de 20% parce que le délai a diminué de 43%. Cette amélioration en performance est également proportionnelle au nombre des entrées. Ces résultats montrent que cette porte est intéressante pour des applications avec une activité relativement faible ou pour améliorer le débit du circuit.

Le seul problème avec cette structure est qu'elle est limitée dans les technologies (130, 90, 65, 45 et 32) où le nombre maximum de transistors dans l'étage d'entrée ne peut pas dépasser quatre (pour 130, 90, 65) ou trois pour (45, 32).

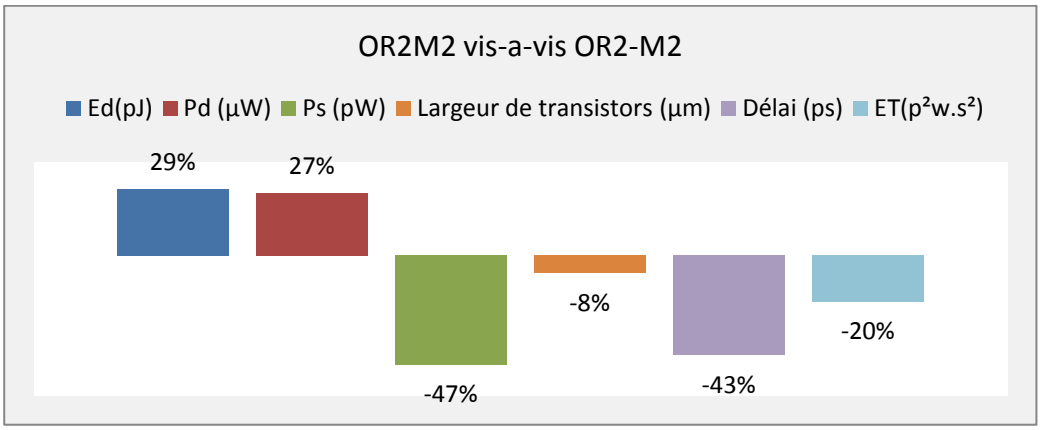


Figure 11 : Gain en performance (OM21 vis-à-vis OR2-M1)

La structure OR2M2D1P ne subit en revanche pas cette limitation grâce à une schématique en transistors NMOS parallèles. La nouvelle implémentation transistor de cette porte est meilleure dans tous les plans (Figure 12). En plus, elle permet d'avoir autant d'entrées possibles où, pour chaque entrée, l'effort logique sera toujours (4/3) indépendant du nombre d'entrées de portes, ou de la technologie. Ces résultats sont également applicables au transistor PMOS, pour la structure NAND2M2D1N qui remplace la fonction NAND2-M2D1N (Annexe B).

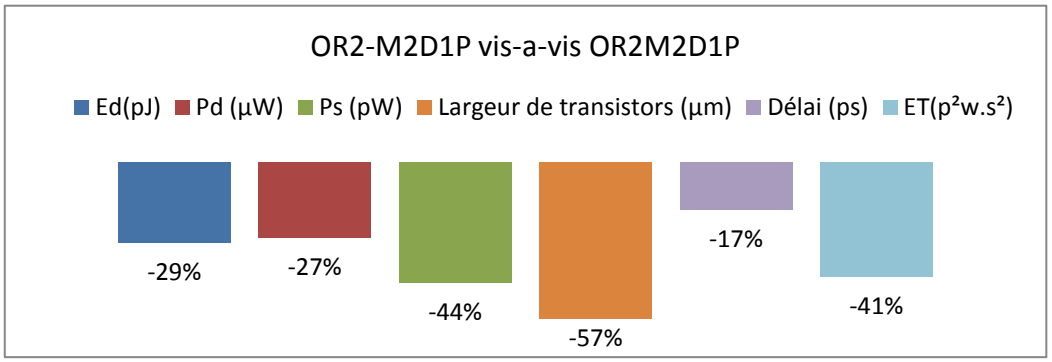


Figure 12 : Gain en performance (OR2M2D1P vis-à-vis OR2-M2D1P)

2.4. Bilan 1:

L'utilisation de la porte OR(n)-M2D1P dans le circuit optimisé de contrôleurs QDI ne modifie pas son caractère QDI car la seule hypothèse temporelle à respecter est que les signaux d'entrées soient des fourches isochrones.

Tableau 4 : Architecture Moore vis-à-vis architecture Moore optimisée

Simulation électrique (spectre / nanosim)	Nombre de transistors	Débit (M. Seq.s-1)	Pd (µW)	Ed (pJ)	Ps (nW)
Arch. 2 (Moore_MR_CPOG)	281	68,5	74,59	1,09	25,88
Arch. 2 Moore (optimisée)	251	73,5	69,40	0,94	24,55

La simulation électrique du circuit de l'exemple du DSI (Tableau 4) montre une réduction de 11% du nombre de transistors, une augmentation de 7% du débit, une diminution de 21% des pics de courant et de 13% de l'énergie consommée (Figure 13). En effet, il y a une réduction du nombre des commutations dans le circuit, des charges sur les signaux, internes et externes, et de la taille du circuit.

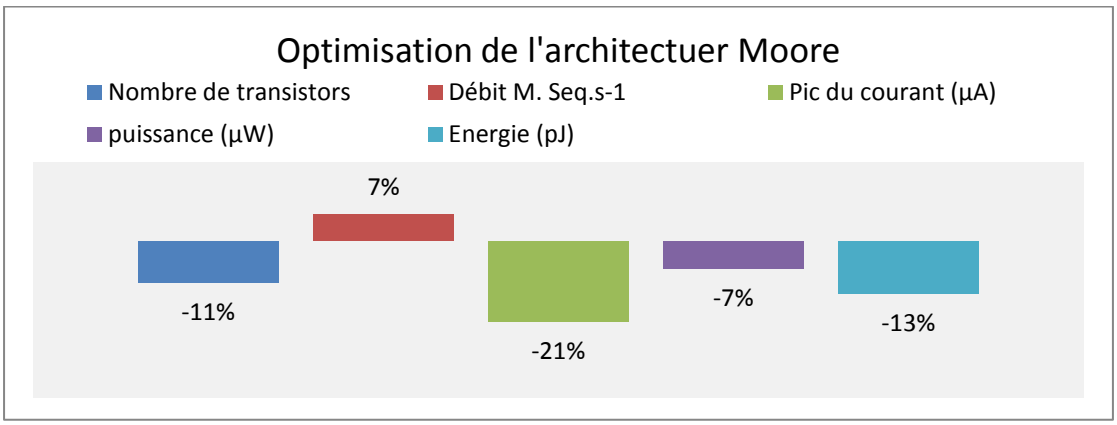


Figure 13 : Gain après optimisation de l'architecture Moore

2.5. DVS (Dynamic Voltage Scaling, Variation Dynamique de Tension)

La technique de DVS est une technique qui a déjà fait ses preuves et qui consiste à réduire la consommation d'énergie dans un circuit CMOS en réduisant la tension d'alimentation. Plusieurs travaux peuvent être trouvés sur l'étude de cette technique [6-9]. Les idées présentées par Weiser et al. [10] et celles de Govil et al. [11] forment les bases de ces algorithmes : le temps d'inactivité du processeur doit être minimisé en réduisant la vitesse de ce dernier.

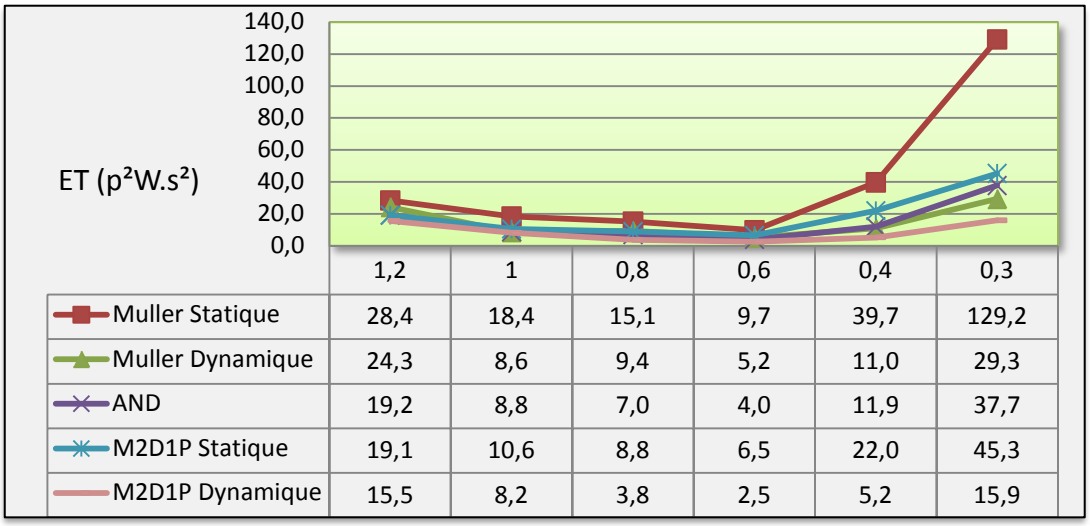
L'énergie consommée par le circuit CMOS peut être donnée par l'équation suivante :

$$E \propto C * V^2$$

« C » étant la capacité totale commutée et « V » la tension d'alimentation [1].

Nous voulons mesurer l'efficacité de cette technique sur notre circuit contrôleur QDI. Plusieurs portes logiques ont été simulées dans la technologie 130nm en utilisant le montage du Figure 2 et cela avec plusieurs tensions d'alimentation. Nous avons donc

défini une tension optimale. En fait, cette tension donne le meilleur profil consommation-délai. Il n'y a ainsi pas besoin de réduire la tension plus que ce seuil de 0,6 V.



	1,2	1	0,8	0,6	0,4	0,3
Muller Statique	28,4	18,4	15,1	9,7	39,7	129,2
Muller Dynamique	24,3	8,6	9,4	5,2	11,0	29,3
AND	19,2	8,8	7,0	4,0	11,9	37,7
M2D1P Statique	19,1	10,6	8,8	6,5	22,0	45,3
M2D1P Dynamique	15,5	8,2	3,8	2,5	5,2	15,9

Figure 14 : Tension nominale pour l'application de technique DVS

La figure 14 montre que cette tension nominale est de 0,6 V pour la technologie 130. A partir de cela, nous avons réalisé deux simulations pour le circuit du DSI avec l'architecture de Moore. L'une pour la tension de 1,2 V et la deuxième pour la tension de 0,6 V. La tableau 5 montre les résultats de la simulation électrique.

Tableau 5 : Circuit de DSI architecture Moore en mode DVS

Nanosim électrique simulation	Tension	Débit M. Seq.s-1	puissance (µW)	Energie (pJ)	Ps (nW)	ET
Moore_DSI	1.2	73,53	69,4	0,94	24,55	12,8
	0.6	16,18	3,5	0,22	6,99	13,4

La puissance dynamique est réduite d'un facteur 20 entre la tension 1,2 et 0,6V, et la puissance statique d'un facteur 3,5. Par contre, le débit a diminué d'un facteur 4,5.

Lorsque la tension est réduite de 1,2 V à 0,6V, le délai entre la remise à zéro d'un signal d'entrée I[x] et I_ack, a augmenté d'un facteur 4,5. L'hypothèse temporelle (fourches isochrones sur les signaux d'entrées) s'affaiblit ainsi et le circuit devient encore plus robuste avec la diminution de la tension, c'est pourquoi l'utilisation de cette architecture s'accommode parfaitement de la technique DVS. Cela est également valable pour des technologies avancées (90, 65, 45, 32,…).

3. Conclusion:

Dans ce chapitre nous avons proposé plusieurs techniques qui permettent de réduire la consommation dans un circuit asynchrone QDI. Il a été démontré en particulier leurs utilisations dans le cadre de l'architecture des contrôleurs de type Moore. Ces optimisations ont réduit la consommation dynamique et statique tout en garantissant ses propriétés QDI. En plus, l'architecture optimisée devient plus robuste en diminuant la tension de l'alimentation, ce qui rend ces techniques adaptables à un système fonctionnant en mode DVS.

Nos contributions dans ce chapitre se résument par les points suivants :

- Une nouvelle technique pour remplacer les structures MO par des structures OM

- Une proposition de remplacement des portes de Muller par des portes dissymétriques dans certaines fonctions logiques tel que le séquenceur.

- Une proposition des nouvelles portes complexes (séquenceurs optimisés, OM, OMD1P…)

- Une évaluation de ces techniques sur l'architecture de Moore avec des contrôleurs QDI.

- Une évaluation des techniques DVS pour l'architecture de Moore optimisée de contrôleur.

Les perspectives de ce chapitre sont les suivantes :

- L'implémentation layout des nouvelles portes complexes proposées.

- L'intégration de ces techniques dans le flot de conception des circuits asynchrones QDI (TAST). Cela demandera des études des méthodes de placement-routage avec des analyses temporelles [12].

4. Bibliographie

1. 1. Brodersen, T.B.a.R., *Energy Efficient CMOS Microprocessor Design.* Proceedings of the Hawaii International Conference on System Sciences, 1995: p. pp.288-297.
2. BRODERSEN, A.P.C.A.R.W., *Minimizing Power Consumption in Digital CMOS Circuits.* PROCEEDINGS OF THE IEEE, VOL. 83, NO. 4, APRIL 1995, 1995.

3. Hao Dongyan, Z.M., Zheng Wei, *Design Methodology of CMOS Low Power.* Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific, 2003: p. 394- 399.

4. Mi-Chang Chang, C.-S.C., Chih-Ping Chao, Ken-Ichi Goto, Meikei Ieong, Lee-Chung Lu, and Carlos H. Diaz, Fellow, *Transistor- and Circuit-Design Optimization for Low-Power CMOS.* IEEE TRANSACTIONS ON ELECTRON DEVICES, VOL. 55, NO. 1, JANUARY 2008, 2008.

5. **Huffman, D.A.**, *The Synthesis of Sequential Switching Circuits.* J. Symbolic Logic Volume 20, 1955: p. 69-70.

6. Kawa, J., *Low Power and Power Management for CMOS—An EDA Perspective.* IEEE TRANSACTIONS ON ELECTRON DEVICES, VOL. 55, NO. 1, JANUARY 2008, 2008.

7. Monteiro, J.e.C., *Power Optimization using Dynamic Power Management.* 1998.

8. K. Flautner, S.K.R., and T. N. Mudge, *Automatic performance setting for dynamic voltage scaling.* In Mobile Computing and Networking, 2001: p. pages 260–271.

9. K. Govil, E.C., and H.Wasserman., *Comparing algorithm for dynamic speed-setting of a lowpower CPU.* In Mobile Computing and Networking, 1995: p. pages 13–25.

10. M.Weiser, B.W., A. J. Demers, and S. Shenker, *Scheduling for reduced CPU energy.* In Operating Systems Design and Implementation, 1994: p. 13–23.

11. K. Govil, E.C., and H.Wasserman., *Comparing algorithm for dynamic speed-setting of a lowpower CPU.* In Mobile Computing and Networking, 1995: p. 13–25.

12. Yahya, E., *«Modélisation, Analyse et Optimisation des Performances des Circuits Asynchrones Multi-Protocoles ».* Thèse à Grenoble-INP, 2009.

CHAPITRE 6 : VERIFICATION FORMELLE DES CIRCUITS ASYNCHRONES QDI

L'imagination est plus importante que le savoir

La vie, c'est comme une bicyclette, il faut avancer pour ne pas perdre l'équilibre

Ne fais jamais rien contre ta conscience, même si l'Etat te le demande

Albert Einstein

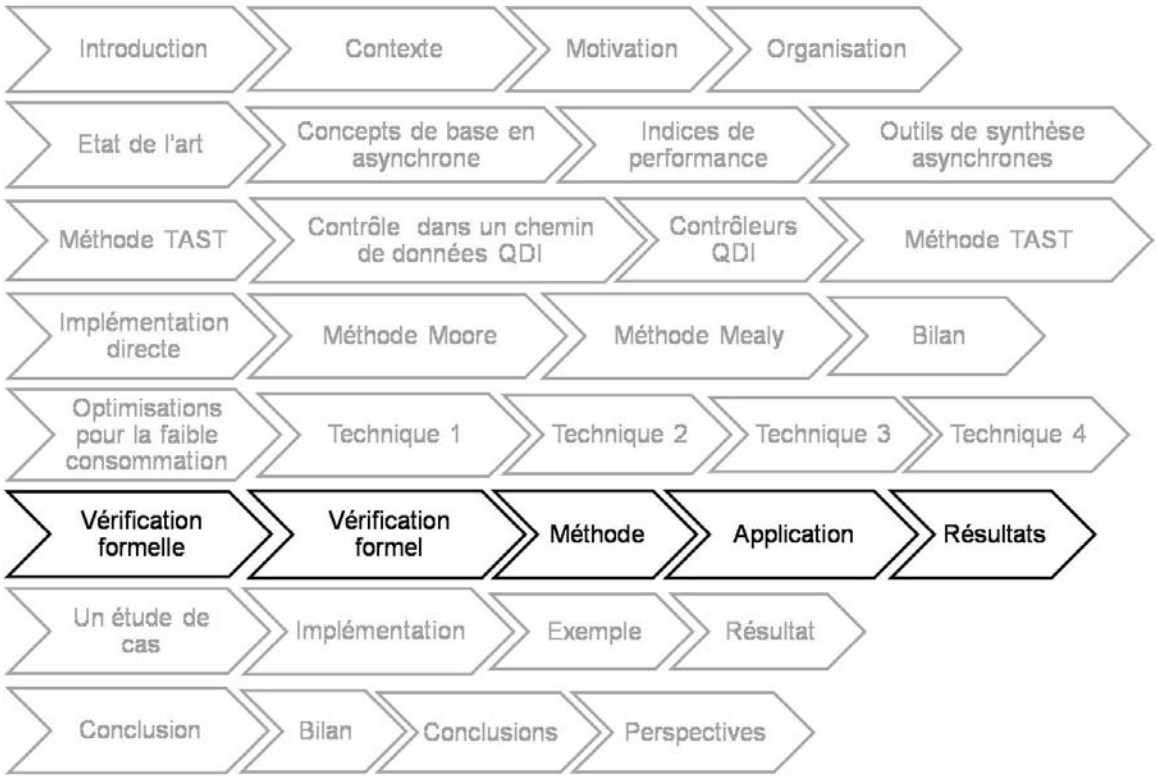


Figure 1: Plan de thèse

Sommaire

Table des figures :

Table des tableaux :

1. Introduction

Les chapitres 3, 4 et 5 ont expliqué les principes de la synthèse des contrôleurs asynchrones séquentiels QDI. A ce stade du travail, nous voulons nous assurer de la validité des circuits résultants après synthèse. Les approches traditionnelles, comme le test ou la simulation, demandent un temps considérable et ne permettent pas de révéler tous les défauts du circuit. Dans le cas des circuits asynchrones, toutes les compositions de délais de tous les composants doivent être prises en compte, ce qui rend la simulation et le test difficile.

Les défauts non-découverts représentent un coût élevé s'ils ne sont pas localisés pendant les étapes primaires de la conception. De plus, dans les applications à sécurité-critique « *safety-critic applications* » comme les centres nucléaires ou l'aviation, les erreurs ne sont pas acceptables. La vérification formelle est une approche alternative pour s'assurer de la qualité et de la justesse de la conception matérielle.

La figure 2 illustre le flot de vérification formelle. A partir de l'idée que l'on se fait du circuit, une première spécification est faite dans un modèle formel. La vérification de cette spécification est faite indirectement : on ne peut vérifier que des propriétés dessus. Cette étape s'appelle validation formelle. Puis par raffinement, nous obtenons différentes implémentations. A chaque étape de raffinement, il faut vérifier que l'implémentation est conforme à la spécification quels que soient les vecteurs d'entrées. C'est l'étape de vérification formelle. Elle est soit directe (équivalence) soit indirecte.

Il existe différentes méthodes de vérification selon le niveau d'abstraction ou l'on travaille et selon le type de vérification que l'on veut faire. Le principe est de modéliser la spécification et l'implémentation à l'aide de modèle mathématique, puis de prouver mathématiquement des relations entre les deux modèles.

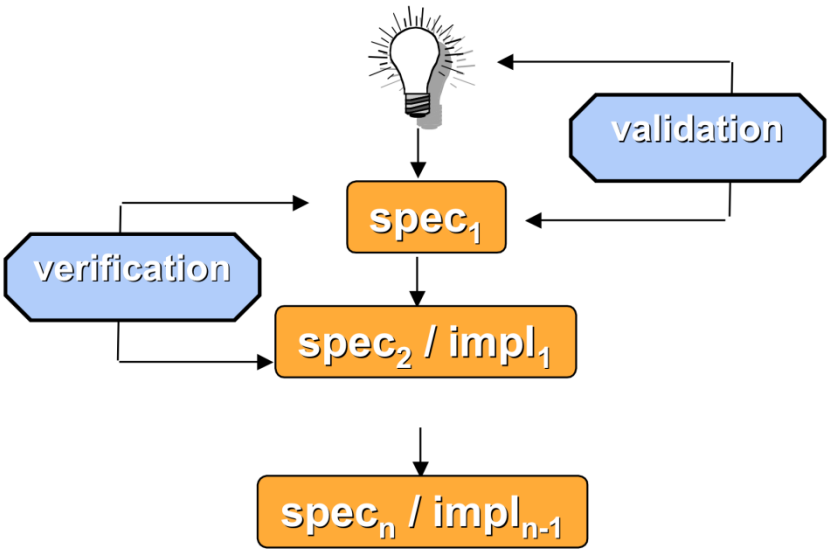


Figure 2: Flot de vérification formelle

Techniques utilisées :

- Vérification de tautologie : Le modèle mathématique utilisé est la logique propositionnelle (des variables et des opérateurs logiques). Cette technique s 'applique au niveau bit et permet de prouver l'équivalence de circuits combinatoires.
- Equivalence entre FSM : Le modèle mathématique utilisé est celui des machines à états finis et la logique propositionnelle. Elle s'applique au niveau bit et permet de prouver l'équivalence de deux machines à états.
- Vérification de propriétés : La modélisation est faite à partir d'automate au niveau bit. Le but est« Prouveurs » de théorèmes et assistants de preuve : Les modèles mathématiques utilisés sont la logique du premier ordre ou d'ordre supérieur. Ils peuvent être utilisés à tous les niveaux d'abstractions. Ils permettent de vérifier la correction d'un algorithme, d'une étape de raffinement ou de faire des preuves sur des implémentations paramétrées.

Les techniques présentées ici sont classées de la plus simple (plus automatique et moins puissante) à la plus complexe (moins automatique, plus puissante) à utiliser. Elles sont complémentaires entre elles. Dans la suite on s'intéressera à la vérification de modèle et à la vérification de propriété.

Ce chapitre s'organise de la manière suivante : après une courte introduction, nous présentons l'état d'art sur la vérification de circuits asynchrones, suivi des bases méthodologiques concernant la logique temporelle, le langage PSL et les techniques de vérification de modèle. La méthodologie de vérification est expliquée en détails dans le quatrième paragraphe à travers l'exemple du circuit de séquenceur. La méthode utilisée est évaluée sur plusieurs exemples de circuits à base de séquenceur.

2. État de l'art

Plusieurs travaux concernant la vérification de circuits asynchrones existent à différents niveaux d'abstraction :

Dans [13], le langage CHP (*Communicating Hardware Processes*) est utilisé comme langage de haut niveau de spécification où il est transformé en LOTOS : un autre langage similaire inspiré de CSP. Le cadre de l'outil de vérification est CADP [16], ce dernier utilise des techniques de vérification de modèle « *model checking* » pour vérifier des spécifications écrites en LOTOS. Cette méthode est capable de vérifier des propriétés comme la justesse de protocole et l'absence de blocage (*deadlock-freedom*) sur des sous-blocs. Par contre cette approche ne garantit pas l'équivalence entre l'implémentation et la spécification. LOTOS a été également proposé comme langage HDL pour spécifier et modéliser les circuits indépendants de la vitesse [17]. Ce travail visait à vérifier la conformité (*conformance*) entre une spécification et son implémentation. La vérification de l'implémentation ou du circuit se base sur la génération automatique d'un vecteur de test pour une couverture raisonnable des fautes de l'implémentation. Ce travail ne garantit pas une vérification exacte et n'aborde pas le problème d'espace d'état, la méthode n'est donc applicable que sur des circuits de petites tailles.

Dans l'équipe CIS, la méthode de synthèse TAST a été vérifiée formellement dans les travaux de V. Bregier et B. Folco [16, 18]. Ces travaux abordent l'adéquation de la méthode de synthèse avec la projection technologique, dédiée aux composants du chemin de données asynchrones QDI, à partir d'une spécification en MDD (*Multi Décision Diagram*). Par contre ces travaux ne vérifient pas le circuit final.

Sirianni, dans [20], a tenté de vérifier indépendamment la spécification et le circuit. La spécification, modélisée avec un réseau de Petri, est traduite en machine à états pour être ensuite vérifiée en utilisant des techniques traditionnelles de vérification de modèle.

Le circuit, par contre, est vérifié à l'aide de Rulebase [21] : un outil de vérification de modèle commercial qui est, à la base, destiné à vérifier les circuits synchrones. Cet outil vérifie des propriétés temporelles écrites en PSL (Properties Spacification Language) et utilise l'horloge comme signal de référence. De ce fait, certains états restent cachés et ne sont pas pris en compte pendant la vérification d'un circuit asynchrone. Cette méthode ne garantit pas ainsi une vérification complète.

Des outils de validation du protocole ont été proposés par [22] pour vérifier les implémentions asynchrones. Les circuits asynchrones sont modélisés par un langage de validation de Protocol PROMELA. L'outil SPIN est utilisé ensuite pour vérifier des propriétés temporelles sur le circuit, comme l'indépendance de vitesse. SPIN utilise des techniques d'analyses d'accessibilité (*reachability*) ou de vérification de modèle. Cette approche ne traite pas non plus le problème de l'explosion de l'espace d'états, elle n'est donc applicable que sur des circuits de moyenne ou petite taille.

Une approche hiérarchique a été proposée par [10] pour traiter le problème de l'explosion de l'espace d'états. Cette méthode consiste à traduire automatiquement une *netlist* de portes simples par une *netlist* de portes complexes équivalente. Les signaux internes sont éliminés pendant la procédure de vérification alors que sont uniquement considérés les sorties des portes complexes ou les signaux d'éléments de mémorisation (Porte Muller ou bascules). Le temps de vérification dépend alors du nombre des éléments de mémorisation plutôt que du nombre de signaux. En conséquence, une réduction drastique de l'espace d'états est obtenue. La méthode emploie des techniques de vérification de modèle symbolique (*symbolic model checking)* pour représenter l'espace d'états du circuit.

Dans cette dernière approche, toutes les portes de Mullers sont considérées comme des éléments de mémorisation. Il faut noter que, parmi ces portes, certaines sont utilisées uniquement pour réaliser un rendez-vous entre les signaux et pour garantir les propriétés d'indépendance de vitesse. De fait, ces portes ne représentent pas la mémoire active, pour réaliser le protocole du composant, et peuvent être également éliminées de la procédure de vérification.

3. Vérification par conformité de protocoles

Dans notre approche, nous nous basons sur une théorie dénommée « vérification par conformité de protocole» (protocole conformance verification) proposée dans [23]. Elle concerne la validation des systèmes asynchrones constitués d'une *Netlist* de composants matériels. Cette théorie montre qu'il suffit de remplacer les composants par leurs protocoles pour vérifier le système. Comme un protocole est normalement beaucoup moins complexe qu'un composant, une réduction drastique de l'espace d'état est ainsi atteinte. Les contrôleurs QDI s'identifient par leurs protocoles et leurs circuits se composent des briques basiques : les Séquenceurs ; le choix de cette approche hiérarchique apparait ainsi logique pour vérifier ces circuits.

3.1. Le langage PSL :

PSL est un langage donnant un moyen standard pour spécifier des propriétés sur un design avec une syntaxe concise et une sémantique clairement définie. Dans le but de comprendre le travail réalisé, une introduction au langage PSL est nécessaire. Cette section porte sur la sémantique et la syntaxe du langage mais n'est pas exhaustive. Pour plus de détails se reporter à [IEE05].

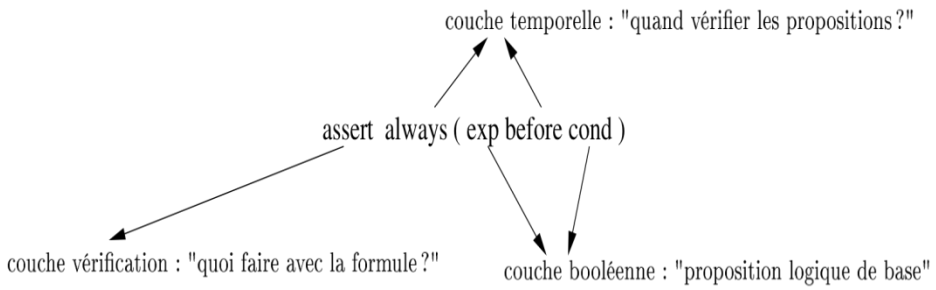


Figure 3: PSL : les couches booléenne, temporelle et vérification

Les assertions PSL sont construites en s'appuyant sur quatre niveaux complémentaires qui garantissent la bonne compréhension du langage.

- Le niveau booléen : utilisé pour des expressions de bases qui seront utilisées par les autres niveaux. Une expression booléenne est évaluée en un cycle.
- Le niveau temporel : c'est le cœur du langage. Ce niveau permet de décrire des relations temporelles complexes entre signaux et ceci sur plusieurs cycles.

- Le niveau vérification : fournit les informations nécessaires aux outils de vérifications. Il permet de savoir comment utiliser les propriétés.
- Le niveau modélisation : utile pour décrire le comportement des entrées/sorties du design. La figure 3 illustre les couches booléenne, temporelle et vérification.

Les différentes catégories d'opérateurs

Les opérateurs du langage PSL sont classés dans quatre catégories différentes :

- Les expressions booléennes sont des relations entre booléens.
- Les SEREs, ou « *Sequential Extended Regular Expression* » sont construits grâce aux expressions booléennes.
- Les FLs, ou Fondation Langage sont des opérateurs temporels construits grâce aux opérateurs des SEREs.
- Les OBEs, ou Optionnel Branching Extension construits grâce aux FLs.

Dans la suite on ne s'intéressera pas au OBEs. Les quatre catégories précédentes sont définies sur un ensemble P non vide de propositions atomiques. Les propositions atomiques sont des expressions les plus simples possibles. Par exemple a =`1' est une proposition atomique. De plus on se restreint au sous-ensemble simple de PSL. Travailler avec ce sous ensemble permet d'avoir des propriétés où l'on ne peut pas voir de retour en arrière dans le temps. Cela nous permet d'être proches de la réalité de fonctionnement et de simulation d'un circuit intégré.

Exemple :

```
P= Property P is

    assert {always ( A->( B until C) )}
```

La propriété P signifie qu'à chaque occurrence de A, le signal B doit être maintenu à '1' jusqu'à ce que C prenne la valeur '1'. La figure 4 montre un chronogramme pour les signaux A, B et C, à l'instant 2, les signaux A et B sont vrais. B est maintenu à 1 jusqu'à l'instant 3 et à l'instant 4 C est vrai. La propriété est donc satisfaite pour la trace commençant à l' instant 0 et finissant à l'instant 6. En revanche, elle est fausse pour la

trace finissant à 8, car pour la deuxième occurrence de A à l'instant 7, B n'est pas maintenu à 1 jusqu'à une occurrence de C.

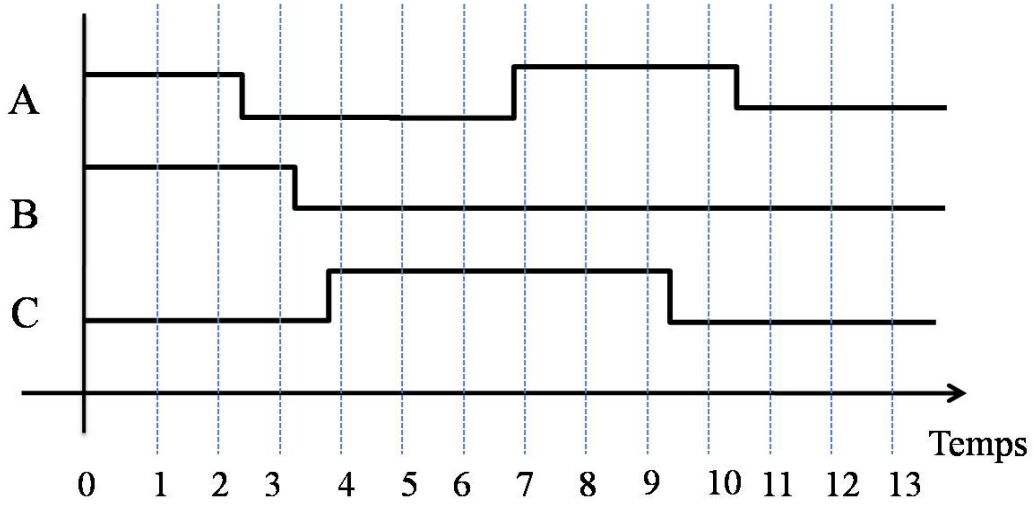


Figure 4 : Une trace qui satisfait la propriété P1

Il y a deux types d'attributs sur les opérateurs de base :

1. Opérateurs *forts* ou *faibles* : cet attribut définit la manière dont les traces doivent vérifier la propriété. Dans le cas d'un opérateur fort, si la propriété est vérifiée sur une trace, elle sera vérifiée sur toute extension de cette trace, en revanche dans le cas d'un opérateur faible, si la propriété est vérifiée sur une trace, elle peut devenir fausse sur une extension. Reprenons le cas de la propriété P, elle est satisfaite sur la trace finissant à l'instant 6 car B est à 1 et qu'il n'est pas obligatoire de rencontrer C, en revanche elle n'est pas satisfaite sur l'extension de la trace jusqu'à l'instant 7. Si l'on avait utilisé l'opérateur until ! (fort), la propriété serait fausse à l'instant 6, car dans ce cas, il faut obligatoirement rencontrer C pour satisfaire la propriété.

2. Opérateurs *recouvrants ou non recouvrants* : cet attribut est utilisé pour les opérateurs « until » et « before ». Il est défini comme non recouvrant par défaut. Dans le cas recouvrant, la marque « _ » est ajouté à la suite du nom de l'opérateur : « until_ » et « before_ ». Le chronogramme de la figure 4 illustre une trace dont le « until » n'est pas recouvrant, il n'est pas obligatoire que B soit vrai en même temps que C.

3.2. Model checking (Vérification de modèle):

*L*a vérification de modèle (Model checking) [26, 29] est une technique qui repose sur la construction d'un modèle fini d'un système et vérifie si ce modèle *satisfait* la propriété désirée. Il désigne une famille de techniques de vérification automatique des systèmes dynamiques (souvent d'origine informatique ou électronique). Les spécifications sont souvent formulées en termes de logique temporelle.

Le principe de la vérification de modèle est le suivant. Le circuit à vérifier est modélisé à l'aide d'un automate, les propriétés sont représentées soit à l'aide d'un automate soit dans une logique temporelle. Le problème se réduit ensuite à montrer si certains états sont atteignables, et s'appuie sur des algorithmes de parcours de graphe. Si le parcours des états est explicite, il y a un risque d'explosion, on préfère alors utiliser des techniques symboliques qui réduisent le nombre d'état à parcourir. Ces techniques symboliques s'appuient sur les diagrammes de décision binaire.

4. RAT et vérification de modèle

RAT (Requirement Analysis Tool) [33], est un outil formel permettant de localiser une erreur dans la spécification d'un système donné sous la forme d'un ensemble de conditions écrites en PSL.

Le comportement du système est décrit à l'aide d'un ensemble de propriétés appelées restrictions, et les propriétés à satisfaire sont appelées assertions. L'outil s'appuie sur des techniques symboliques de vérification de modèles. Il se base sur deux outils NuSMV et VIS [32]. Il permet trois sortes de vérification formelle : assurance, simulation et réalisabilité de propriété. Le choix de RAT est principalement lié à l'utilisation de PSL pour spécifier les circuits.

5. Méthodologie de vérification :

Dans ce travail nous proposons l'utilisation du RAT pour vérifier les circuits asynchrones. La méthode est constituée des trois étapes suivantes :

- Modélisation du circuit : le comportement de toutes les portes et les connexions du circuit sont représentées à l'aide de propriétés temporelles.
- Modélisation de l'environnement : les protocoles de communication avec l'interface du circuit sont modélisés à l'aide de propriétés temporelles.

- Modélisation des caractéristiques à satisfaire : Les propriétés d'insensibilité au délai, ou d'autres caractéristiques sont modélisées.

Ces propriétés peuvent se séparer en deux groupes : le groupe des conditions (modélisation du circuit et du protocole des signaux d'entrées), et le groupe des assertions (protocole sur les signaux de sorties et les caractéristiques à satisfaire).

La preuve est hiérarchique : on commence par vérifier la correction de petits blocs élémentaires, puis ces blocs sont remplacés par leur protocole pour vérifier des blocs un peu plus gros.

Cette méthode est illustrée à travers l'exemple de séquenceur (Figure 5).

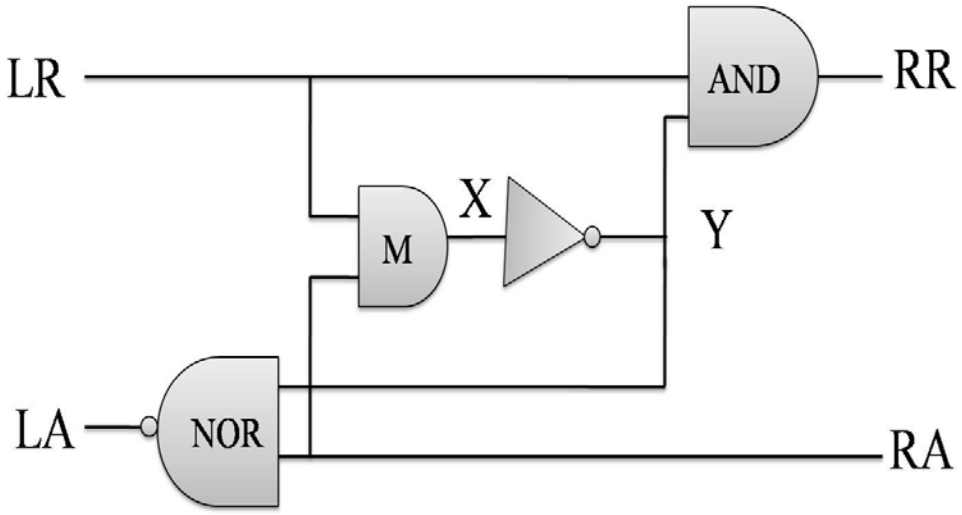


Figure 5 : Circuit du séquenceur (Simple-Rail)

5.1. Modélisation d'un bloc:

i) Modélisation d'une porte

Un modèle d'une porte logique se compose de quatre propriétés :

- OHLP (Output Low Holding Property) définit l'état de la porte pour retenir une valeur zéro sur la sortie;
- OHHP (Output High Holding Property) définit l'état de la porte pour retenir « 1 » sur la sortie ;
- OSP (Output Setting Property) définit l'état de la porte lorsqu'elle change sa valeur de sortie de « 0 » à « 1 »;

- ORP (Output Resetting Property) définit l'état de la porte lorsqu'elle change sa valeur de « 1 » à « 0 ».

Notons « SE » l'expression logique qui donne « 1 » sur la sortie « Z » de la porte (Setting Expression) et « RE » l'expression logique qui donne « 0 » sur la sortie « Z » (Resetting Expression). Dans le cas d'une porte sans mémoire l'expression logique « SE » est la négation de l'expression logique « RE » : RE= not (SE). Pour une porte OU, par exemple, SE= A | B et RE = ! A & ! B => SE = not (RE). Dans le cas d'une porte avec mémoire, RE ne correspond pas nécessairement la négation de SE. Par exemple dans le cas d'une porte de Muller symétrique : SE = A & B alors que RE= ! A & ! B => RE ≠ not (SE).

Le modèle de porte en PSL consiste en quatre propriétés :

- OLHP : always (! Z => ! Z until_ SE) *(P 1)*

Pendant la vérification, un signal sans restriction peut prendre n'importe quelle valeur à n'importe quel moment. Cette propriété restreint ainsi la sortie de la porte à garder la valeur zéro jusqu'à ce que la condition « SE » soit valide. Elle couvre donc les états « ! Z & ! SE » qui sont les états stables de la porte. L'opérateur « until » est donc utilisé dans ce cas.

- OHHP: always (Z => Z until_ RE) *(P 2)*

De la même manière, cette propriété couvre les états (Z & ! RE) où la porte maintient « 1 » sur sa sortie.

- ORP: always (RE & Z => eventually! ! Z) *(P 3)*

Cette propriété couvre les états « RE & Z » de la porte où sa sortie change de « 1 » à « 0 ». L'opérateur « eventually ! » est ainsi utilisé pour modéliser un changement de valeur après un délai borné mais indéterminé.

- OSP: always (SE & ! Z => eventually! Z) *(P 4)*

De même, cette propriété couvre les états « SE & !Z » où la sortie de la porte change de « 0 » à « 1 ».

La figure 6 montre le graphe d'états et le modèle en PSL de la porte OU.

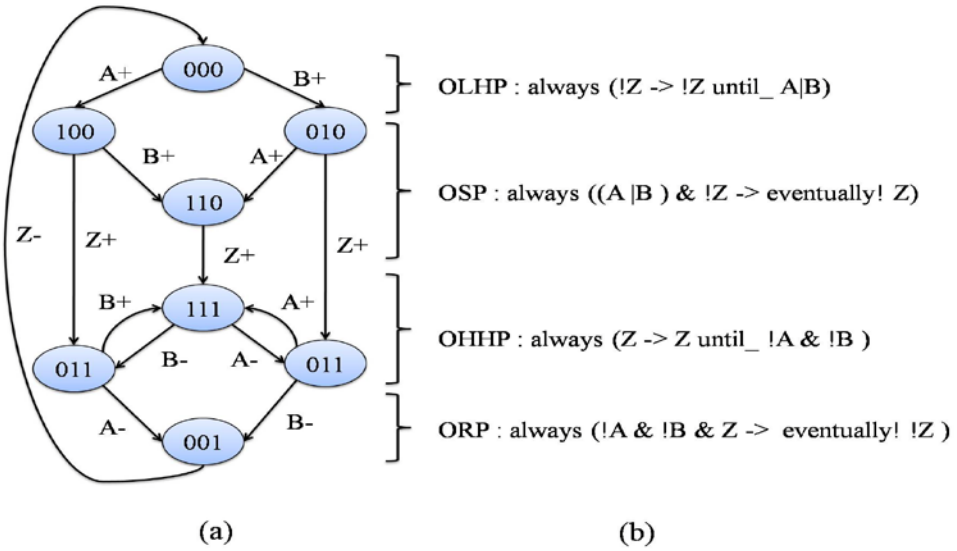


Figure 6 : Porte OU : (a) graphe d'états (A B Z), (b) son modèle en PSL

La figure 7 montre le graphe d'états d'une porte de Muller avec l'ensemble des propriétés PSL qui modélisent cette porte.

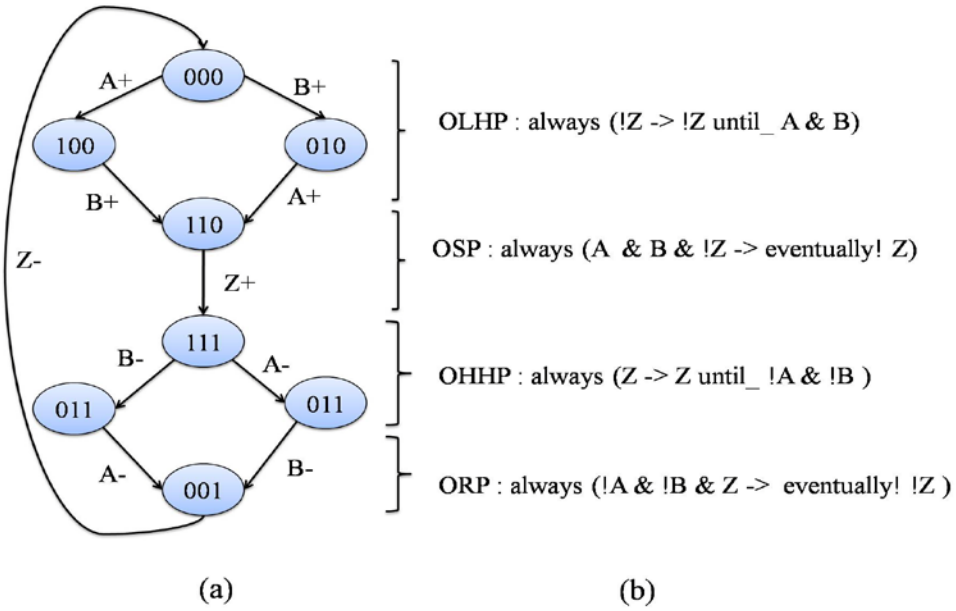


Figure 7 : Porte Muller : (a) graphe d'états (A B Z), (b) modèle en PSL

Une bibliothèque des modèles en PSL a été construite pour la plupart des portes logiques existantes dans la bibliothèque TAL et CMOS 130 de STMicroelectronics (Annexe C).

ii) Modélisation des fils (connexions) :

Plusieurs cas de figures existent :

- une sortie peut être connectée à une seule entrée, on parle dans ce cas d'un fil,

- une sortie peut être connectée à plusieurs entrées et on parle alors de fourches. Dans ce deuxième cas, le modèle utilisé dépendra des hypothèses temporelles effectuées sur la fourche : isochrone ou pas.

Le composant modélisant les connexions entre les portes se crée par trois méthodes :

● Dans le cas d'un fil ou d'une fourche isochrone, il suffit d'utiliser le même nom du signal pour connecter les portes. Dans ce cas, on considère que les fils ou les fourches n'introduisent aucun délai.

● En utilisant des fourches non isochrones : P5 donne le modèle de ce type de fourche avec une entrée A et deux sorties Z1 et Z2. Ces dernières ne peuvent pas changer de valeur en même temps. Par contre, « Z1 » devient « 1 » implique toujours que « Z2 » devient « 1 » et vice-versa.

always (!A → eventually! ! Z1 & ! Z2)

&&

always (A → eventually! Z1 & Z2)

&& (P 5)

always (! Z1 & ! Z2 → ! Z1 & ! Z2 until_ A)

&&

always (Z1 & Z2 → Z1 & Z2 until_ !A)

● En utilisant des fourches isochrones : P6 montre un exemple où il n'y a pas besoin de modéliser le délai, c'est pourquoi « Z1 » et « Z2 » copient fidèlement « A » en utilisant la relation directe implique « → ».

always (! A → ! Z1 & ! Z2)

&& (P 6)

always (A → Z1 & Z2)

Modélisation d'un bloc Chaque porte du design est remplacée par sa description PSL, en affectant à chaque propriétés les noms des signaux de chaque porte, les connexions sont elles aussi remplacées par des propriétés (par contre dans l'exemple de séquenceur, on considère toutes les fourches sont isochrones, c'est pourquoi les noms des signaux sont réutilisés dans les modèles des portes pour effectuer les connexions).

Le séquenceur présenté figure 5 est ainsi représenté par l'ensemble de propriétés suivantes :

always (LR & Y & ! RR → eventually! RR)

&&

always ((! LR | ! Y) & RR → eventually! ! RR) *(P 7: Porte AND)*

&&

always (RR → RR until_ ! LR | ! Y)

&&

always (! RR → ! RR until_ LR & Y)

always((RA | Y) & LA → eventually! ! LA)

&&

always(! RA & ! Y & ! LA → eventually! LA) *(P 8: Porte NOR)*

&&

always(! LA → ! LA until_ ! RA & ! Y)

&&

always(LA → LA until_ RA | Y)

always (LR & RA & ! X → eventually! X)

&&

always (! LR &! RA & X → eventually! ! X) *(P 9 : Porte Muller)*

&&

always (X → X until_ ! LR &! RA)

&&

always (! X → ! X until_ LR & RA)

always(X & Y → eventually! !Y)

&&

always (! X & ! Y → eventually! Y) *(P 10 : Porte NOT)*

&&

always (! Y → ! Y until_ ! X)

&&

always (Y → Y until_ X)

5.2. Modèle de l'environnement:

L'environnement du circuit doit refléter le comportement QDI. Cela consiste en deux points : d'abord, définir le protocole et le codage de données sur l'interface du composant, et ensuite définir l'état initial du système.

Les canaux d'entrée et de sortie respectent les règles du protocole 4-phases. Il y a deux types de canaux :

● Un canal actif, celui qui initialise la communication avec le circuit. Il envoie une requête et attend l'acquittement du circuit. Le canal (LR, LA) du séquenceur est un exemple de ce type de canal.

● Un canal passif, celui qui répond à une requête du circuit par un acquittement. Le canal (RR, RA) du séquenceur est un exemple de ce type de canal.

Chaque signal du système est un élément de mémoire, il peut prendre n'importe quelle valeur au début de la vérification, c'est pourquoi les valeurs initiales des signaux des portes et des composants doivent être définies.

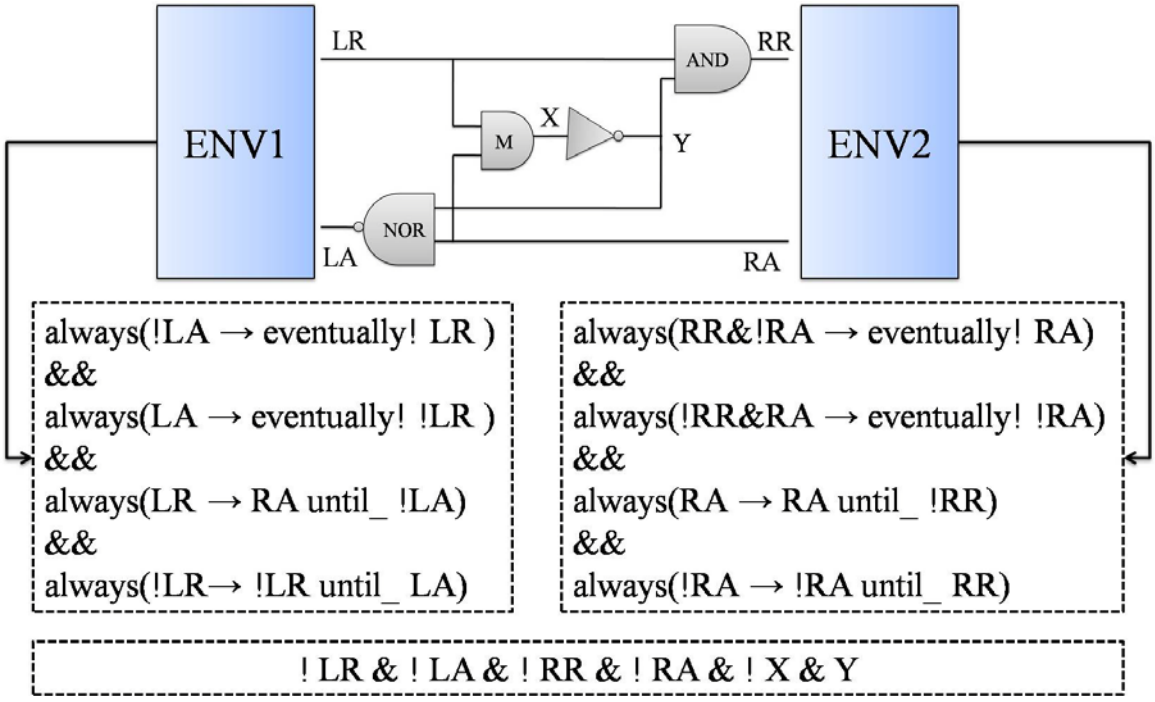


Figure 8 : Modèle de l'environnement du circuit de séquenceur sous vérification

L'initialisation d'un signal « S » est faite par une propriété avec le nom de signal « S » s'il prend « 1 » ou procédé par « ! » s'il prend zéro. La propriété P 11 par exemple initialise le circuit du séquenceur :

! LR & ! LA & ! RR & ! RA & ! X & Y *(P 11)*

5.3. Vérification de la quasi insensibilité aux délais du circuit

La quasi-insensibilité aux délais est une propriété intrinsèque de certains circuits asynchrones. Elle caractérise les comportements du circuit dans son environnement (QDI) [10]. Un circuit est QDI lorsqu'il a les trois propriétés suivantes:

- le circuit est sécurisé et semi-modulaire :

Définition [10]: un signal « ai » est semi-modulaire par rapport à un signal d'entrée a_k « $a_k \neq a_i \ if \ a_k \in fan\_in(a_i)$ » et « a_i », une fois excité, ne redevient pas stable en changeant la valeur de « a_k ». Un signal « a_i » est semi-modulaire s'il est semi-modulaire par rapport à ses signaux d'entrée (fanin). Un circuit est semi-modulaire si tous ses signaux sont semi-modulaires.

Pour prouver qu'un composant est semi-modulaire, il faut prouver que la projection de l'environnement sur son circuit ne génère pas d'aléas sur ses sorties ou des signaux non désirés : un acquittement sans requête par exemple « *P 12* » est une propriété qui garantit la sécurité sur le canal (RR, RA) du séquenceur :

always (!LA → !LA until_ LR)

&& (P 12)

always (LA → LA until_ !LR)

Elle affirme que « LA » ne passe jamais à « 0 » sauf s'il était déjà précédé par « LR+ », idem « LA » ne revient pas à « 1 » sauf si « LR » était déjà revenu à « 0 ». Elle garantit aussi qu'aucun aléa est généré sur « LA » car un aléa implique que « LA » change de valeurs au moins deux fois (1→0→1 ou 0→1→0).

● le circuit est vivace : si pour n'importe quel signal « a_i », il existe un état « S_i » dans lequel « a_i » est acquitté. Le circuit doit donc réagir en permanence aux requêtes de l'environnement et il ne doit pas se mettre dans un état de blocage interne. Un circuit vivace répond à une requête par un acquittement ou une nouvelle requête.

« P 13 » est un exemple de propriété de vivacité:

Always (LR & ! LA → eventually! LA)

&& (P 13)

Always (!LR & LA → eventually! !LA)

P 13 garantit une réponse sur « LA » pour chaque requête de « LR ».

La Figure 9 illustre le chronogramme pour les deux propriétés *(P 12* et P 13) où *P 12* vérifie les états stables alors que P 13 vérifie les transitions.

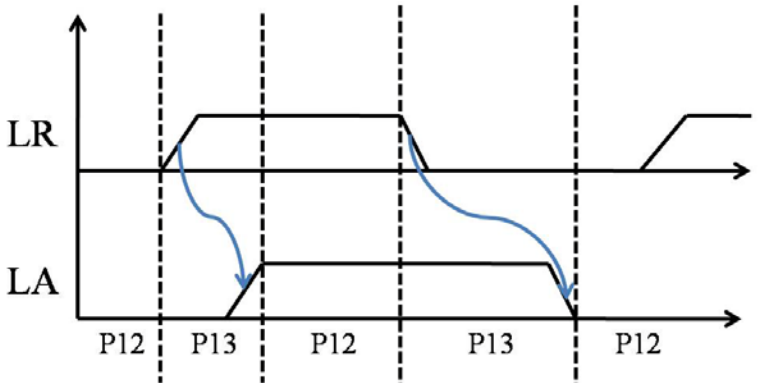


Figure 9 : Une trace graph pour les propriétés P12 et P13

● Le codage de données :

Il faut également garantir que les sorties de circuit ont un codage QDI comme (1-parmi-N), cela sera illustré sur l'exemple DSI plus tard (chapitre 7).

Le concepteur peut également ajouter d'autres assertions pour vérifier la spécification de circuit, cela est relatif à la nature de l'application.

5.4. Extraction d'un modèle équivalent de protocole :

Une fois que le composant est vérifié, en particulier que l'on a prouvé qu'il respectait le protocole de communication, il est substitué par les propriétés modélisant le protocole. C'est une abstraction. Remplacer un composant par son protocole a pour objectif de cacher ses signaux internes afin de réduire l'espace d'état pris par le modèle de son circuit. Un protocole occupe en effet un espace d'état plus réduit que son circuit [23].

L'extraction d'un modèle équivalent se décompose en quatre étapes, décrites comme suit:

1- Le protocole du composant est défini et représenté par un STG dans lequel seuls les signaux d'interface et de mémoire active interne sont considérés.

2- Un graphe d'états *actifs* est extrait à partir de son STG.

3- Pour chaque signal de sortie ou de mémoire, quatre propriétés PSL sont définies, similaires à celles d'une porte simple (voir paragraphe 5.1). Ce sont des assertions. De la même manière, pour chaque signal d'entrée, quatre propriétés PSL sont définies, ce sont des contraintes de l'environnement.

4- Prouver la correction du composant, revient à prouver que si les contraintes de l'environnement sont satisfaites, alors le modèle PSL du composant satisfait les assertions.

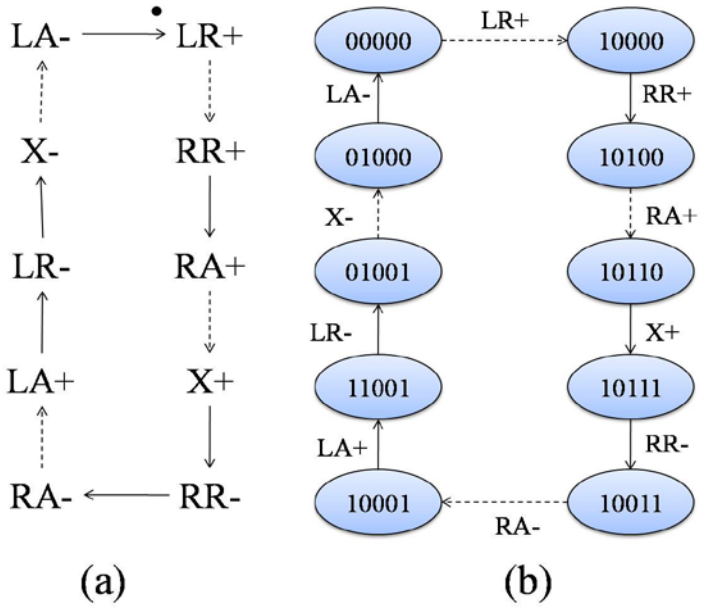


Figure 10 : Séquenceur simple : (a) protocole (STG) (b) graph d'états (LR LA RR RA X)

La figure 10 montre le protocole STG (a) et le graphe d'états (b) pour un séquenceur simple (Figure 5). Son graphe d'états reflète uniquement les états activés au cours de la séquence d'événements de son protocole.

Le séquenceur a deux signaux de sorties « LA » et « RR » et un signal de mémoire « X ». Un ensemble d'assertions {A} est mis pour vérifier que le séquenceur respecte son protocole. Afin d'obtenir le modèle le plus optimisé, « RE » et « SE » sont deux fonctions des signaux d'entrée et de la mémoire active uniquement (les signaux de sortie ne sont pas considérés). De cette manière, quatre propriétés décrivent le comportement du signal RR :

OSP_{RR} *: always (((LR & ! RA & ! X) & ! RR →* $\diamond$ *! RR)* *(P 14)*

Cette propriété décrit la transition ($RR^- \rightarrow RR^+$) dans l'état (10000) où l'expression logique « SE » est égale à « LR & ! RA & ! X »

ORP_{RR} *: Always ((LR & RA & !X) & RR→* $\diamond$ *! ! RR)* *(P 15)*

Cette propriété décrit la transition ($RR^+ \rightarrow RR^-$) dans l'état (10111) où l'expression logique « RE » est égale à « LR & RA & ! X ».

$OHHP_{RR}$: Always (RR → RR until_ (LR & RA & !X)) *(P 16)*

Cette propriété restreint la valeur de RR à « 1 » de l'état (10111) jusqu'à l'état (10000).

$OLHP_{RR}$: Always (! RR → ! RR until_ (LR & ! RA & !X)) *(P 17)*

Cette propriété restreint la valeur de RR à « zéro » de l'état (10000) jusqu'à l'état (10111).

Huit propriétés similaires (assertions) décrivent le comportement de la sortie « LA » et la mémoire active « X ». Si le composant satisfait ces propriétés, il est alors abstrait par ces propriétés.

5.5. Abstraction et vérification hiérarchique

La vérification hiérarchique signifie que la spécification au niveau bas du système pourrait être utilisée comme description comportementale dans un niveau supérieur. Elle permet de diviser la tâche de vérification en groupe de tâches plus petites.

Pour tout composant Z, nous noterons {Z} l'ensemble des contraintes modélisant Z, {A(Z)} l'ensemble des assertions vérifiées par Z, {E(Z)} l'ensemble des contraintes modélisant l'environnement de Z, et {P(Z)} l'ensemble des assertions modélisant le protocole respecté par Z ($\{P(Z)\} \subset \{A(Z)\}$).

{E(Z)} & {Z} → {A(Z)} → *{E(Z)} & {Z} → {P(Z)}* *(P 18)*

Soit C_n un composant à un niveau « n » d'une architecture, et soit C_{n-1} un sous-composant de C_n (voir figure 11). Notons X le sous-composant de Cn tel que l'interconnexion de Cn-1 à X donne C_n (X sur la figure 11 est représenté par les composants X0 et X1). La modélisation de C_n est l'intersection de l'ensemble des assertions $\{C_{n-1}\}$ et {X}. Pour prouver l'ensemble d'assertions $\{A(C_n)\}$, il faut prouver

$\{E(C_n)\}$ & $\{C_n\} \rightarrow \{A(C_n)\}$

Ou encore

$\{E(C_n)\}$ & $\{C_{n-1}\}$ & {X}→ $\{A(C_n)\}$ *(P 19 : vérification de Cn)*

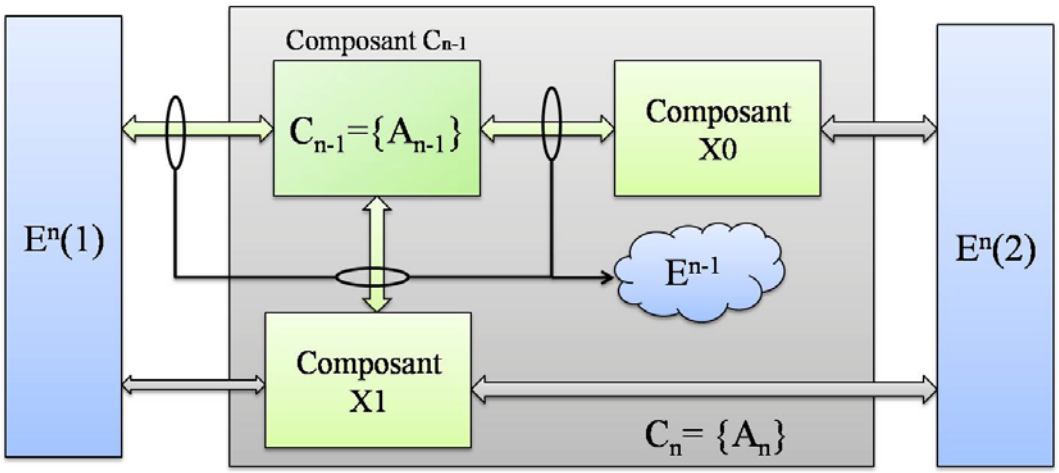


Figure 11 : La vérification de composant Cn

La propriété P 19 peut être trop complexe à prouver directement si l'ensemble d'assertions de C_n est trop gros. Il peut alors être intéressant de substituer $\{C_{n-1}\}$ par tout ou partie $\{A(C_{n-1})\}$ (il faut au moins garder la modélisation du protocole). Pour effectuer cette abstraction, il faut s'assurer que $\{E(C_{n-1})\}$ est satisfait. Il faut donc prouver la propriété suivante :

$$\{E(C_n)\} \,\&\, \{X\} \rightarrow \{E(C_{n-1})\} \qquad (P\ 20\ )$$

Notons P21 la propriété :

$$\{E(C_n)\} \,\&\, \{A(C_{n-1})\} \,\&\, \{X\} \rightarrow \{A(C_n)\} \qquad (P\ 21)$$

Si P 20 et P 21 sont satisfaites alors P 19 sera satisfait. De manière similaire {X} peut lui même être substitué par tout ou partie de {A(X)}.

5.6. Bilan et résultats

Nous avons exécuté plusieurs expériences sur le vérificateur RAT. L'exemple étudié est un circuit de « Contrôleur Séquenceur » (chapitre 3 – paragraphe 7). C'est un circuit séquentiel QDI évolutif, c'est-à-dire qu'il peut être agrandi en augmentant simplement le nombre des cellules de base (séquenceurs).

a. Vérification à plat contre vérification hiérarchique

La figure 12 montre le circuit d'un double-séquenceur, qui se compose de deux séquenceurs connectés en série. Il génère deux séquences de protocoles quatre phases sur deux sorties, chacune d'entre elles étant simple rail. Un quadri et octo-séquenceur en plus

de ce circuit ont également été vérifiés deux fois : en tant que circuits à plats qui se composent de portes simples (AND, OR, Muller, NOT) et en tant que circuits hiérarchiques en considérant le protocole de séquenceur à la place de son circuit.

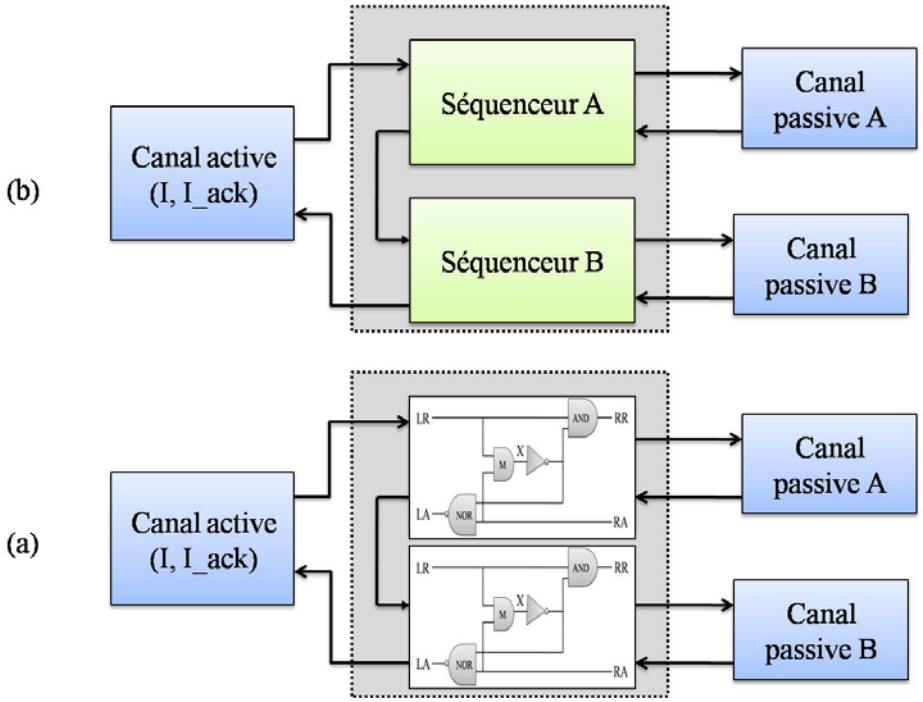


Figure 12: Netlist plat (a) vis-à-vis Netlist hiérarchique (b) pour un circuit de bi-séquenceurs

Le Tableau 1 montre le résultat en temps de CPU pour vérifier les trois circuits. Ce temps a été calculé après avoir utilisé l'option (miniSAT) de RAT qui effectue une vérification symbolique bornée du modèle (SBMC : Symbolique Bounded Model Checking) basée sur la vérification de la « *satisfiability* » [38]. Le nombre d'itérations était fixé pour les deux exercices à plat et hiérarchique.

L'approche hiérarchique montre une amélioration significative en temps de calcul du CPU.

Tableau 1: vérifications à plat vis-à-vis des vérifications hiérarchiques

Description du circuit			**Vérification à plat**			**Vérification hiérarchique**		
Circuit	portes	SMCB	signaux	état	Temps(s)	signaux	Etat	Temps(s)
bi-séquenceur	8	30	12	4096	173	11	2048	88
quadri-séquenceur	16	50	21	2097152	4048	17	131072	931
octo-séquenceur	21	50	41	2,2E+12	Non validé	33	8,6E+9	2340

L'espace d'état est réduit car de nombreuses variables sont éliminées lors de la réduction en modèle de séquenceur. En fait, la taille des BDD résultants est directement

liée au nombre de nœuds dans le graphique du BDD, qui est contrôlé par le nombre de variables (signaux) d'entrée et de leur ordre. L'amélioration dans le temps de CPU est donc due à la réduction de taille du BDD.

b. Réduction de nombre d'itérations :

La réduction du nombre des signaux, ayant pour conséquence la réduction de la taille du BDD, a un effet bénéfique sur le nombre d'itérations nécessaires pour avoir une vérification correcte grâce à une réduction de la profondeur logique du circuit. Autrement dit, cela influe directement sur le nombre d'itérations nécessaires pour atteindre un point fixe au cours de la remise à jour de nœuds du BDD [10, 39].

La figure 13 montre une vérification des quadri et octo–séquenceurs où les deux nombres d'itération suivants ont été considérés : 50 et 30. Le graphe montre ainsi un gain significatif en temps de calcul du CPU.

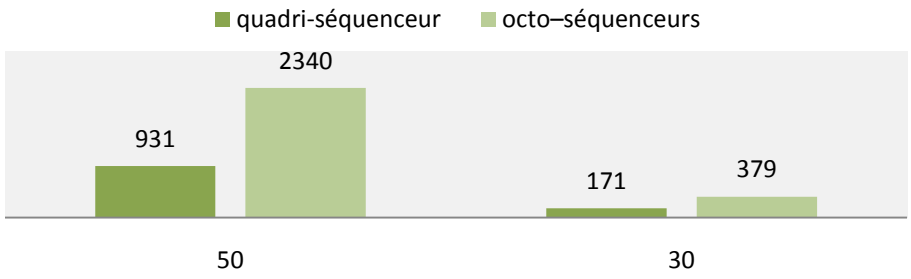


Figure 13: Réduction en temps CPU due à une réduction du nombre d'itérations ou la borne de la vérification symbolique du modèle en RAT

c. Niveau hiérarchique « n » contre niveau « n-1 » :

L'approche hiérarchique peut être appliquée sur plusieurs niveaux. La figure 14 montre le circuit quadri-séquenceur, il a été vérifié en tant que circuit composé de deux bi-séquenceurs connectés en série (Figure 14), où le modèle de bi-séquenceur a remplacé les modèles des séquenceurs.

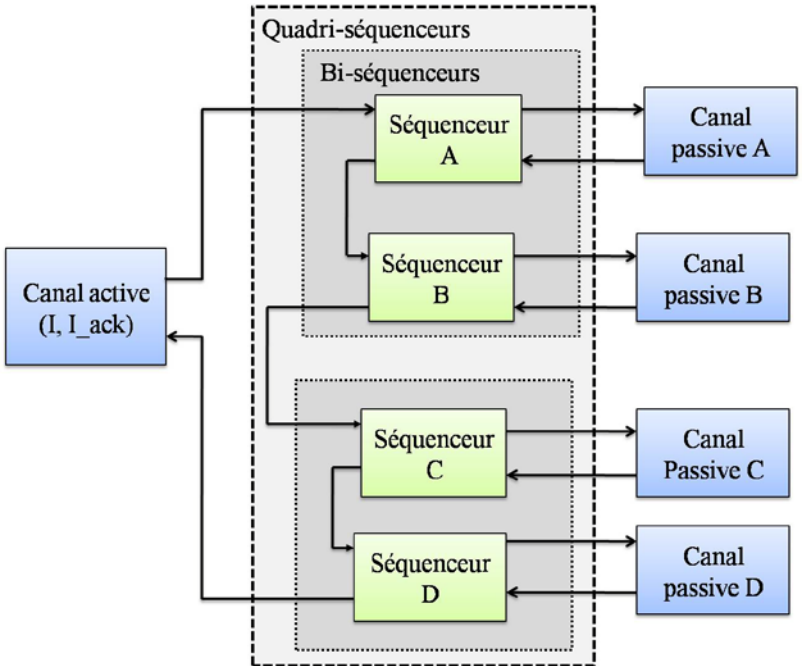


Figure 14: Représentation hiérarchique de quadri-séquenceur sur deux niveaux

La figure 15 montre les résultats de vérification en considérant 30 comme le nombre d'itérations. Un gain de 1,2 en temps de vérification du CPU a été calculé grâce à une réduction en nombre de signaux de 17 à 15.

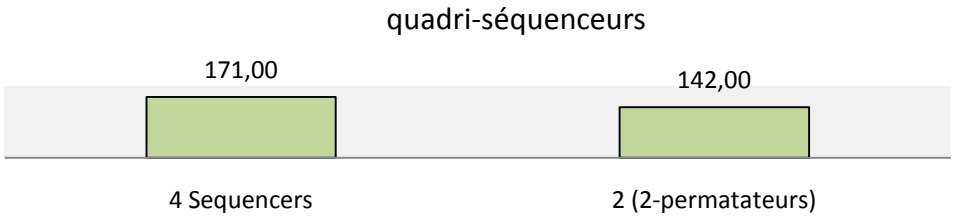


Figure 15 : Réduction en temps CPU de vérification due à un deuxième niveau hiérarchique

d. Modélisation optimisée des protocoles des composants

Nous volons évaluer l'effet du nombre des signaux considérés pour l'expression SE et RE dans le modèle d'un séquenceur. Nous avons ainsi proposé deux modèles : Dans le premier (celui présenté dans le paragraphe 5.5), les expressions SE et RE pour chaque signal de sortie sont extraites des signaux d'entrée et de la mémoire (Annexe C- Séquenceur Modèle SR 1). Dans le deuxième (Modèle SR 2 : Annexe C), seuls les signaux qui ont un effet direct sur la valeur du signal de sortie ont été considérés (Annexe C- Séquenceur modèle SR 2) : Les 4 propriétés, qui modélisent les comportements du signal RR, s'écrient ainsi de la manière suivante :

always (! RR & LR & ! X → eventually! RR)

&&

always (RR & X → eventually! ! RR)

&&

always (! RR → ! RR until_ LR & ! X)

&&

always (RR → RR until_ X)

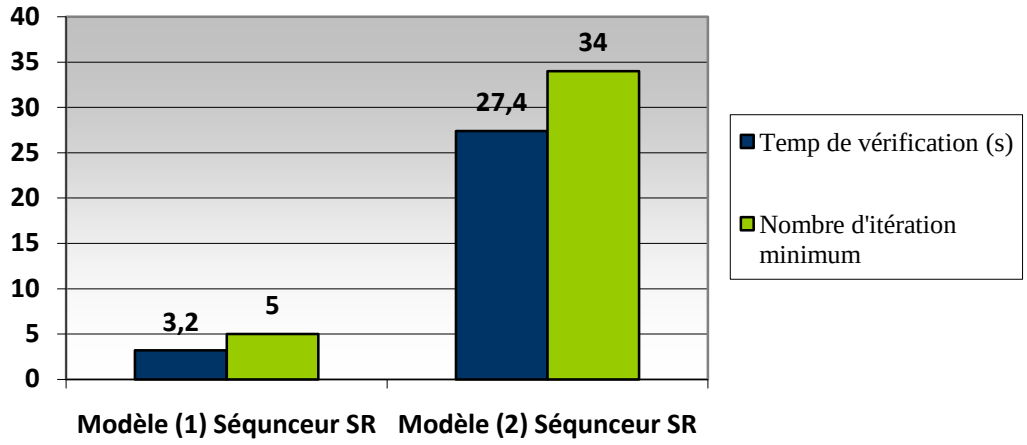


Figure 16: Modèle (1) vis-è-vis Modèle (2) pour Séquenceur SR

La vérification du quadri-séquenceur a pris 3,2 secondes pour un nombre d'itérations de 5 dans le cas du modèle (1). En revanche, elle a pris 200 secondes pour 34 itérations dans le cas du modèle (2) (Figure 16). D'où l'intérêt de considérer tous les signaux d'entrée et de mémoire dans l'expression SE et RE pendant la modélisation d'un composant, car cela permet de réduire la taille du BDD résultant grâce à une restriction plus sévère pour l'espace d'état équivalent du modèle.

e. Elimination de la mémoire inactive du modèle d'un composant :

Le circuit du séquenceur multi-rail (Figure 17) proposé dans le chapitre 3 – paragraphe 7, contient une porte de Muller en plus (M2). Cette porte ne représente pas de mémoire active et elle sert uniquement à faire le rendez-vous entre le front montant de RR et RA. Dans ce cas, l'environnement du séquenceur peut envoyer un acquittement (RA+) même si RR n'est pas dans l'état haut (Figure 17).

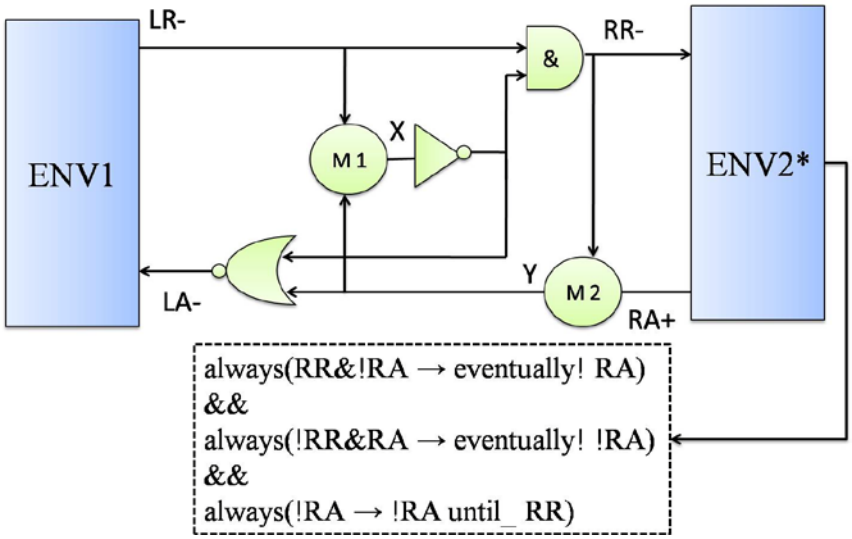


Figure 17: Séquenceur MR et ENV2 modifié

Nous avons vérifié les circuits quadri-séquenceurs multi-rails dans la figure 19 (chapitre 3, paragraphe 7) en utilisant deux modèles pour le séquenceur :

Dans le premier modèle (Modèle (3) : Annexe C), M2 n'était pas considéré, ce modèle étant similaire au modèle présenté au paragraphe 5.5, sauf que l'environnement équivalent du protocole permet « RA+ » pour un « RR- » comme dans la figure 17. Dans le deuxième modèle (Modèle (4) : Annexe C), M2 était considéré comme une porte mémoire. Il était donc représenté dans le protocole du composant comme un signal Y, où les propriétés suivantes modélisent le comportement de RR (comme un exemple) en fonction de LR, RA, X et Y.

always (! RR & LR & ! RA & ! X & ! Y → eventually! RR)
&&
always (RR & LR & RA & X & Y → eventually! ! RR)
&&
always (! RR → ! RR until_ LR & ! RA & ! X & ! Y)
&&
always (RR → RR until_ LR & RA & X & Y)

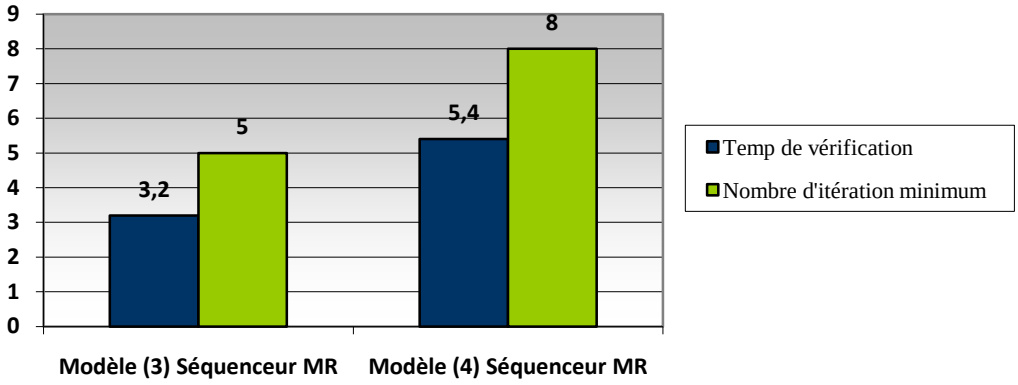


Figure 18: Modèle 3 vis-à-vis Modèle 4 pour séquenceur MR

La vérification de quadri-séquenceur était possible pour 5 itérations dans le cas du modèle (3) et pour un temps (CPU) 3,2 secondes, il a nécessité 8 itérations et un temps de calcul de 5,4s dans le cas du modèle (4) (Figure 18).

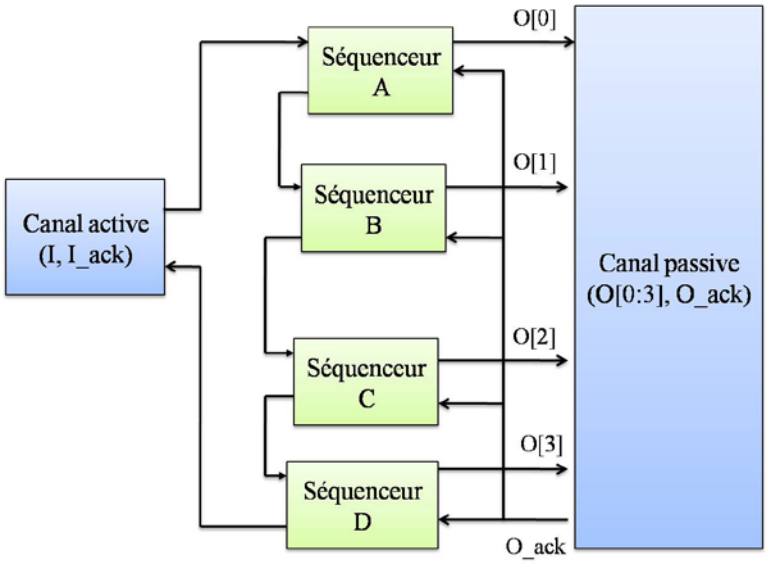


Figure 19: Vérification d'un quadri-séquenceur à 4 rails

Les deux dernières expérimentations montrent l'entrée d'une description adéquate du protocole des composants, cela représente un gain considérable au niveau du temps de vérification.

Dans un premier temps, il faut limiter l'espace d'états du modèle lui-même en considérant tous les signaux d'entrée et de mémoire dans ses expressions SE et RE.

Dans un deuxième temps, il faut jouer sur la modélisation de l'environnement, s'il est possible, afin de réduire l'espace d'état de son modèle équivalent.

5.7. Outil de transformation des *netlists*

Un outil a été développé en se basant sur cette méthode présentée. Il transforme une « Netlist » de portes en un ensemble de propriétés (Conditions). Les modèles sont pris d'une bibliothèque développée à partir de la bibliothèque TAL et la bibliothèque CMOS130 de ST Microelectronic. L'outil génère un fichier « RAT » qui contient, d'un côté, l'ensemble des « Conditions » qui décrivent le circuit, son environnement et l'état initial des signaux ; d'un autre côté, l'ensemble d'assertions nécessaires pour la vérification des propriétés QDI du circuit comme le protocole 4-phases, le codage de données, etc. Le concepteur doit rajouter à la main des propriétés pour vérifier la spécification du circuit cible. Cette dernière partie ne peut pas être automatisée parce que cela dépend de la nature du circuit et de son utilisation. Le concepteur a également besoin de vérifier un ensemble de propriétés qui reflète le protocole du circuit cible afin de pouvoir générer son modèle correspondant en PSL. Cette dernière partie est automatisable et en cours de développement pour être ajouté à l'outil (Figure 20). L'outil a été réalisé sous Windows (832 lignes du code C++).

Cette méthode est générale et a été utilisée au sein de notre équipe pour vérifier des composants asynchrones QDI comme le « Synchro@ », nouveau composant proposé par Alexandre Porcher dans sa thèse pour l'interface synchrone-asynchrone de ses moniteurs [40].

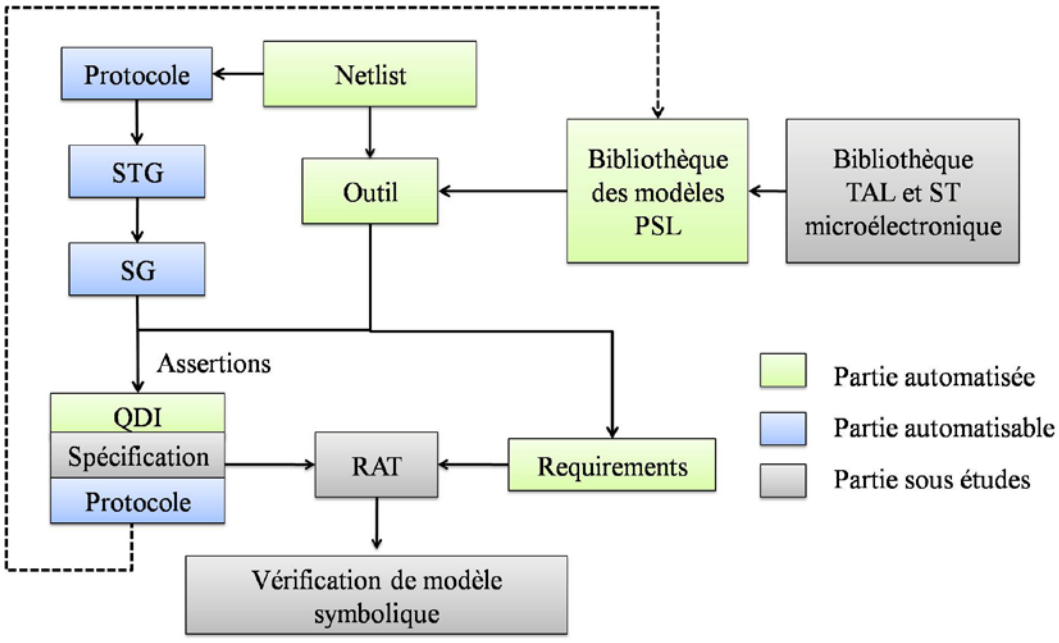


Figure 20 : Outil de vérification semi-automatisé

5.8. Limite de la méthode en utilisant RAT

A cause de l'abstraction, certaines informations du circuit sont limitées, ce qui fait que certaines propriétés ne seront plus vérifiables, en particulier celles concernant les comportements des signaux internes d'un circuit.

Bien que les résultats obtenus par cette méthode soient très encourageants pour poursuivre notre recherche en se basant sur RAT, l'outil a montré une instabilité et certaines limites concernant la vérification des circuits de taille relativement moyenne (d'ordre 40 portes). De plus, l'opérateur « Never » en PSL n'est pas validé dans l'outil alors que cet opérateur est essentiel dans plusieurs cas, comme la vérification de codage de données ou la simplification de certains modèles (composant multi-rail en annexe C).

De ce fait, un travail plus approfondi est nécessaire afin d'améliorer les résultats et valider la méthode. Nous proposons en particulier de travailler le cœur de « *solveur* » miniSAT en prenant en compte l'approche hiérarchique. L'idée est de limiter l'espace d'états ou l'OBDD résultant pendant l'étape de vérification d'un composant aux états actifs de son protocole. En plus, les CBDD (Connective Binary Decision Diagram) ou les Digrammes connectés à décision binaire sont intéressants dans ce contexte. Ces digrammes montrent en effet un avantage principal : ils ne souffrent pas d'une croissance exponentielle lorsque le nombre d'entrées (signaux) et les interconnexions logiques croissent [39].

6. Conclusion

La complexité de la vérification formelle des circuits asynchrones dépend fondamentalement de la taille du circuit. Dans ce chapitre, nous avons présenté notre approche pour vérifier les circuits de contrôle asynchrone QDI. Un vérificateur basé sur la vérification symbolique de modèle a ainsi été mis en œuvre par l'outil RAT. La méthode est exploitée d'une manière hiérarchique et les composants basiques, comme le séquenceur, sont remplacés par les modèles de leurs protocoles. Il a donc été montré qu'une vérification exacte demande une vérification des comportements de l'environnement pour chaque composant tout au long de cette procédure. Réduire le nombre de variables pertinentes a conduit à une réduction de la taille du BDD et du temps de calcul. Plusieurs expériences ont démontré les avantages de cette approche.

Notre contribution dans ce chapitre est la suivante:

- Une nouvelle méthode pour vérifier des circuits asynchrones se basant sur l'outil RAT.

- Une mise en application de la théorie présentée par [23],

- Une approche hiérarchique de vérification,

- Une modélisation efficace du protocole des composants,

- Une bibliothèque de modèles en PSL.

Les perspectives de ce chapitre sont les suivantes : l'automatisation complète du flot de vérification, l'application de la méthode sur des contrôleurs asynchrones séquentiels QDI, la mise en place d'une méthodologie efficace pour modéliser un composant à partir de son STG et SG et l'étude de l'approche hiérarchique avec l'outil miniSAT en utilisant des CBDD.

7. Bibliographie

1. Emerson, E.A., *Temporal and modal logic*. Handbook of Theoretical Computer Science. 1990: Elsevier. 995--1072.
2. Hodges, W., *Classical Logic I: First Order Logic,*. The Blackwell Guide to Philosophical Logic. 2001: Blackwell.
3. Melham, T., *Abstraction Mechanisms for Hardware Verification*. VLSI Specification, Verification and Synthesis. 1987: Kluwer Academic Publishers. 129--157.
4. Lamport, M.A.a.L., *The Existence of Refinement Mappings.* Theoretical Computer Science, 1991. **82**(2): p. 253--284.
5. Thomas, W., *Automata on infinite objects*. Handbook of theoretical computer science (vol. B): formal models and semantics. 1991 MIT Press Cambridge, MA, USA. 133 - 191
6. Rajeev Alur Lalita, R.A., Rajeev Alur, Lalita Jategaonkar Jagadeesan, Lalita Jategaonkar Jagadeesan, Joseph J. Kott, Joseph J. Kott, James E. Von Olnhausen, James E. Von Olnhausen *Model Checking of Real-Time Systems: A Telecommunications Application.* In Proceedings of the 19th International Conference on Software Engineering, 1997: p. 514--524.
7. Henzinger, R.A.a.T.A., *A Really Temporal Logic.* Journal of the ACM, 1989: p. 164--169.
8. Vaandrager, N.L.a.F., *Forward and backward simulations for timing-based systems.* In de Bakker et al, 1991: p. 397--446.
9. Shankar, N., *Verification of Real-Time Systems Using PVS*. Lecture Notes in Computer Science, ed. C.A. Verification. Vol. 697. 1993: Springer-Verlag. 280--291.

10. Roig, O.a.C., J. and Pastor, E., *Hierarchical gate-level verification of speed-independent circuits* Asynchronous Design Methodologies, 1995. Proceedings., Second Working Conference on, 1995: p. 128-137.

11. 1850-2005, I.S., *IEEE Standard for Property Specification Language (PSL)*. 2005. p. 01-143.

12. Tchaltsev, R.B.a.R.C.a.I.P.a.M.R.a.A., *RAT: A Tool for the Formal Analysis of Requirements.* Computer Aided Verification,Springer Berlin / Heidelberg, 2007: p. 263-267.

13. G. Salaun, W.S., Y. Thonnart, P. Vivet, *Formal Verification of CHP Specification with CADP Illustartion on an Asynchronous Network-on-Chip.* ASYNC '07: Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems, 2007: p. 73--82.

14. ISO/IEC, *LOTOS — a formal description technique based on the temporal ordering of observational behaviour. International*

Standard 8807, International Organization for Standardization Information Processing Systems Open Systems Interconnection. Sept. 1989.

15. Martin, A.J., *Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits.* California Institute of Technology, Pasadena, CA., 1989.

16. H. Garavel, F.L., and R. Mateescu., *An overview of CADP 2001*. EASST Newsletter, Aug. 2002: p. 4:13–24.

17. Turner, J.H.a.K.J., *Verifying and Testing Asynchronous Circuits using LOTOS.* Proc. Formal Methods for Distributed System Development 2000(Kluwer Academic Publishers): p. 267--283.

18. Bregier, V., *Automatic synthesis of Asynchronous Proven Quasi Delay Insensitive Circuits.* PHD thesis, Institut National Polytechnique Grenoble, 2007.

19. Folco, B., *Contribution à la synthèse des circuits asynchrones Quasi Insensibles aux Délais, application aux systèmes sécurisés.* These, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2007.

20. Sirianni., M.B.a.D.B.a.E.D.a.M.R.a.J.-B.R.a.A., *An Approach to the Introduction of Formal Validation in an Asynchronous Circuit Design Flow*. HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9, 2003: p. 279.2.

21. I. Beer, S.B.-D., C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthai, *RuleBase : Model checking at IBM.* Proc. 9th Intl. Conference in computre aided verification CAV'97, 1997: p. 480-483.

22. Rahardjo, B.a.M., R.D., *Verification of speed-independent asynchronous circuits with protocol validation tools.* Communications, Computers, and Signal Processing, 1995. Proceedings. IEEE Pacific Rim Conference on, 1995: p. 257-260.

23. Wang, X.a.K., M., *On process-algebraic verification of asynchronous circuits.* Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on, 2006: p. 37-46.

24. Manna, Z., Pnueli, Amir *The Temporal Logic of Reactive and Concurrent Systems Specification*. Springer. 1992.

25. Brinksma, *Tools and Algorithms for the Construction and Analysis of Systems.* Third International Workshop, TACAS'97, Enschede, The Netherlands, April 2-4, 1997, Proceedings, 1997.

26. Clarke, E.M., *Model Checking.* Lecture Notes in Computer Science, 1997. **1346**.

27. IEEE1850-2005, *IEEE Standard for Property Specification Language (PSL).* 2005.

28. Accellera, *Property Specification Language : Reference Manual.* 2004.

29. E. M. Clarke, E.A.E., A. P. Sistla, *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.* ACM Transactions on Programming Languages and Systems (TOPLAS), 1986. **Volume 8**: p. 244 - 263

30. @, *http://nusmv.irst.itc.it/NuSMV/.*

31. McMillan, K.L., *Fitting formal methods into the design cycle.* Proc. 31st Conference on Design Automation, June 1994: p. 314--319.

32. CIMATTI Alessandro, G.E., PISTORE Marco, ROVERI Marco, SEBASTIANI Roberto, TACCHELLA Armando., *Integrating BDD-based and SAT-based symbolic model checking.* FroCos 2002 : frontiers of combining systems 2002. **2309**: p. 49-56.

33. R. Bloem, R.C., I. Pill, M. Roveri, and A. Tchaltsev, *Rat: A tool for the formal analysis of requirements.* Computer Aided Verification,Springer Berlin / Heidelberg, 2007: p. 263-267.

34. Khaled Alsayeg, K.M.-A.a.L.F., *RAT-based formal verification of QDI asynchronous controllers.* FDL2009 : Forum on specification & Design Languages, 2009.

35. G. Fey, D.G., and R. Drechsler, *Avoiding false negatives in formal verification for protocol-driven blocks.* Proc. Design, Automation and Test in Europe DATE '06, March 2006. **1**: p. 1--2.

36. Mokhov, A.a.Y., Alex, *Conditional Partial Order Graphs and Dynamically Reconfigurable Control Synthesis.* Design, Automation and Test in Europe, 2008. DATE '08, 2008: p. 1142-1147.

37. Mokhov, A.Y., A. , *Verification of conditional partial order graphs.* ACSD 2008. 8th International Conference on Application of Concurrency to System Design., June 2008: p. 128-137.

38. K. Heljanko, T.J., and T. Latvala, *Incremental and complete bounded model checking for full pltl.* In: CAV. Volume 3576 of LNCS: Springer-Verlag, 2005: p. 98–111.

39. Wright, R.L.S., M.A. , *Reducing BDD size by exploiting structural connectivity.* Proceedings. Ninth Great Lakes Symposium on VLSI, 1999: p. 132-135

40. A.Porcher, K.Morin-Allory, L.Fesquet: "Synthesis of Asynchronous Monitors for Critical Electronic Systems". Proc. IEEE Symposium on Design and Diagnostics of Electronic Systems, Vienna (Austria), April 2010.

CHAPITRE 7 : UNE ETUDE DE CAS

La personnalité créatrice doit penser et juger par elle-même car le progrès moral de la société dépend exclusivement de son indépendance

Le monde est dangereux à vivre ! Non pas tant à cause de ceux qui font le mal, mais à cause de ceux qui regardent et laissent faire

N'essayez pas de devenir un homme qui a du succès. Essayez de devenir un homme qui a de la valeur

Ce qui fait la vraie valeur d'un être humain, c'est de s'être délivré de son petit moi

Albert Einstein

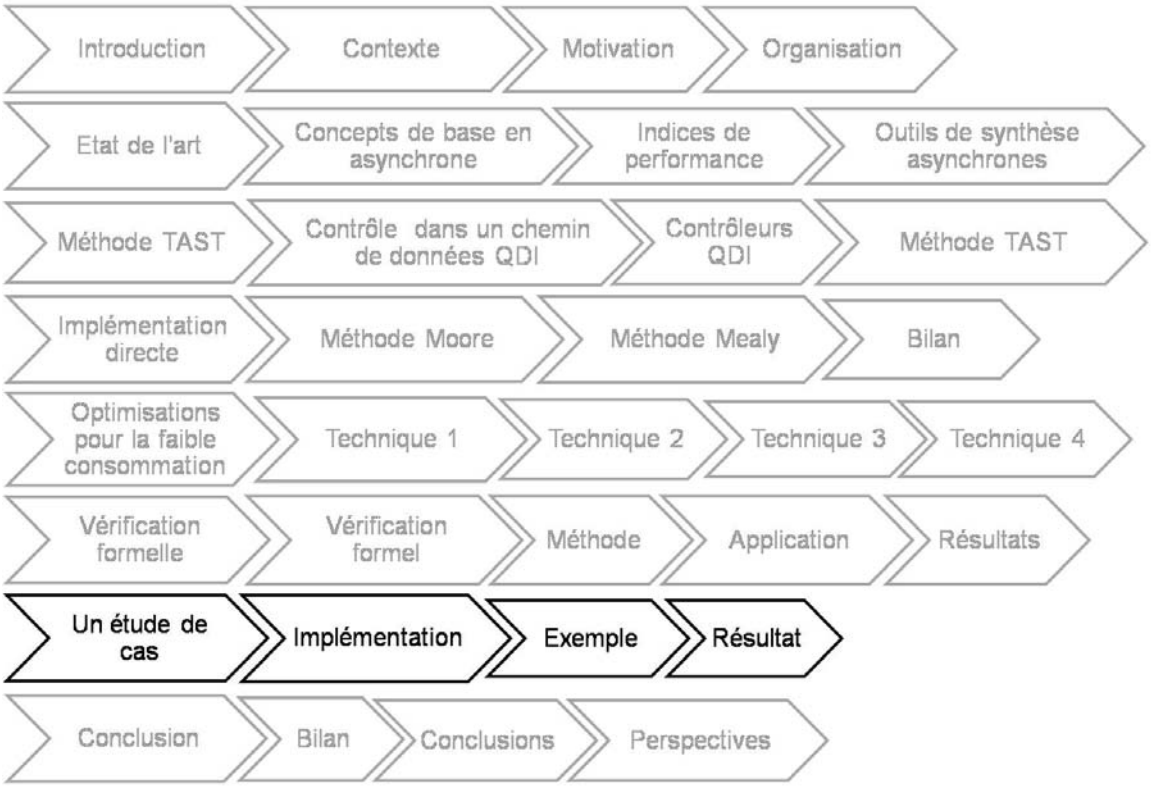


Figure 1: Plan de thèse

Sommaire

Table des figures :

Table des tableaux :

1. Introduction

A ce stade de la thèse, une méthodologie de synthèse a été proposée pour réaliser des contrôleurs asynchrones séquentiels QDI. Plusieurs techniques d'optimisation sont étudiées et présentées, aussi bien à haut niveau qu'au niveau portes logiques. Une méthodologie de vérification formelle permettant de valider les circuits après synthèse a également été présentée. L'objectif de ce chapitre est donc d'évaluer ces méthodes sur des circuits complexes.

Nous présenterons tout d'abord le circuit d'exemple étudié qui est un registre à décalage configurable. L'architecture de ce circuit contient un contrôleur séquentiel.

Ce chapitre est organisé de la manière suivante : ce contrôleur est d'abord décrit selon les modèles Moor_MR_CPOG et Mealy_MR_CPOG (chapitre 4). Une première version du circuit est synthétisée ensuite selon la méthode de l'implémentation directe à partir d'un modèle Moor_MR_CPOG (chapitre 4). Une deuxième version est synthétisée selon la méthode TAST (chapitre 3). Nous présenterons un ensemble de simulations sur les deux circuits afin d'évaluer la méthode de l'implémentation directe vis-à-vis de la méthode TAST. Les techniques d'optimisation de la consommation (chapitre 5) seront appliquées et évaluées par la suite sur le circuit à architecture de Moore. Enfin, Nous évaluerons la méthodologie de vérification formelle proposée (chapitre 6) à travers l'exemple du Décodeur Séquentiel d'Instructions (DSI) (chapitres 3,4 et 5).

2. Un registre à décalage configurable

L'architecture des registres à décalage configurable [1] se compose principalement de deux unités de décalages X1 et X4, où X1 réalise un décalage d'un bit (chapitre 3 – paragraphe 3.2), et X4 réalise un décalage de 4 bits. L'architecture proposée, dans la Figure 2, utilise ces deux étages. Un contrôleur (machine à état) contrôle le passage de la donnée à décaler lorsqu'elle traverse les étages de décalage nécessaires à son exécution. Il est important de noter que chaque bloc dans cette architecture possède un Half-Buffer en sortie. Ces tampons sont indispensables pour éviter un blocage des jetons dans les boucles fermées (Mux1, X1, MUX_DMUX), (MUX_DMUX, X4, DMUX2).

Les canaux de commande sont codés en 1-parmi-n. Le signal cdm1 [2 :0] contrôle le démultiplexeur (DMUX1), le signal CM1 [1 :0] contrôle le multiplexeur 1 MUX1, le signal cdmm [4 :0] contrôle le MUX_DMUX où cdmm[0] correspond à la position [0 :1],

cdmm[1] correspond à la position [1 :0], cdmm[2] correspond à la position [1 :1], cdmm[3] correspond à la position [1 :2] et cdmm[4] correspond à la position [2 :1], le signal cdm2 [1 :0] contrôle le démultiplexeur DMUX2 et enfin le signal cm2[2 :0] contrôle le MUX2. L'objectif de cet exemple est de montrer la différence en performance entre plusieurs architectures de contrôleur, le canal N étant codé en 1-parmi-32 afin de simplifier la tâche de la synthèse. (D'autres codages plus optimisés peuvent être utilisés mais ne sont pas présentés ici).

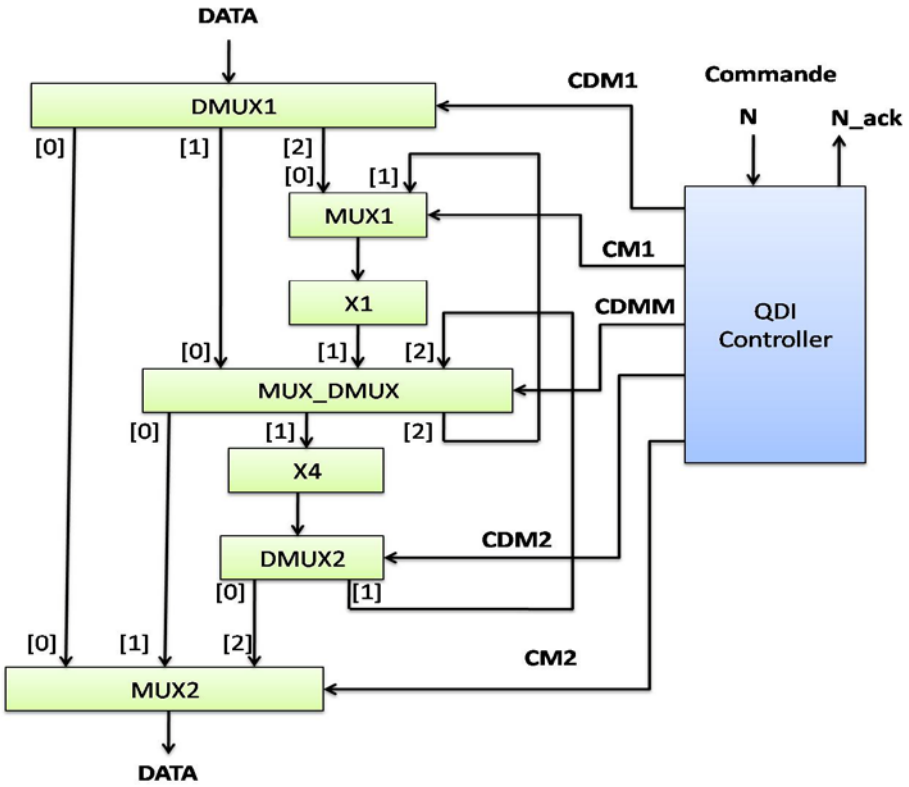


Figure 2 : Registre à décalage configurable

Pour chaque valeur de N, il y aura une génération d'une séquence de commandes afin de réaliser le décalage désiré sur les données. Par exemple pour N= 7, le contrôleur génère la séquence des commandes suivantes : {cdm[2] ; cm1[0] ; cdmm[3] ; cm1[1] ; cdmm[3] ; cm1[1] ; cdmm[2] ; cdm2[0] ; cm2[2]}. Cette séquence correspond à trois passages de données par X1 suivis d'un seul passage par X4, ce qui génère au total un décalage de 7 bits.

2.1. Unité de contrôle (machine à états)

La procédure de description du contrôleur, selon le modèle Moore, doit respecter la démarche suivante :

- D'abord, nous commencerons par décrire la séquence la plus longue (S_n), qui correspond à un décalage de 32 bits dans le cas de notre exemple.

- Ensuite, lorsque nous décrivons une deuxième séquence (S_{n-1}). Nous réutiliserons les actions redondantes avec S_n mais seulement celles qui arrivent en dernier dans la séquence.

Cette démarche permet de créer une convergence des actions redondantes à la fin des séquences. Cela simplifie l'arbre d'acquittement du canal d'entrée.

Par exemple, la séquence de commandes la plus longue est celle correspondant à un décalage de 31 bits. La Figure 3 montre le Moor_CPOG_MR de cette séquence.

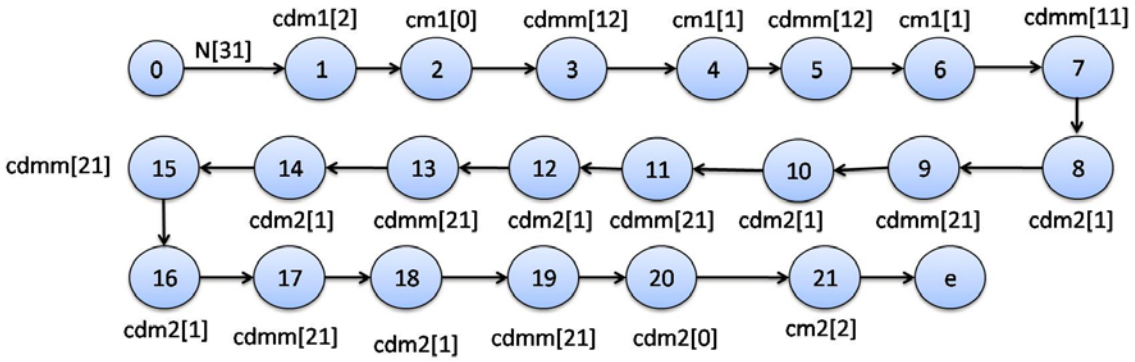


Figure 3 : La séquence de contrôle correspondant à un décalage de 31 bits

Pour réaliser un décalage de 30 bits, il faut enlever un seul passage par X1, ce qui peut être réalisé en enlevant les deux actions 3 et 4 ou les deux actions 5 et 6. Nous gardons les actions 5 et 6 parce qu'elles arrivent plus tard dans la séquence. La séquence correspondant à un décalage de 30 bits est ainsi celle de la Figure 4.

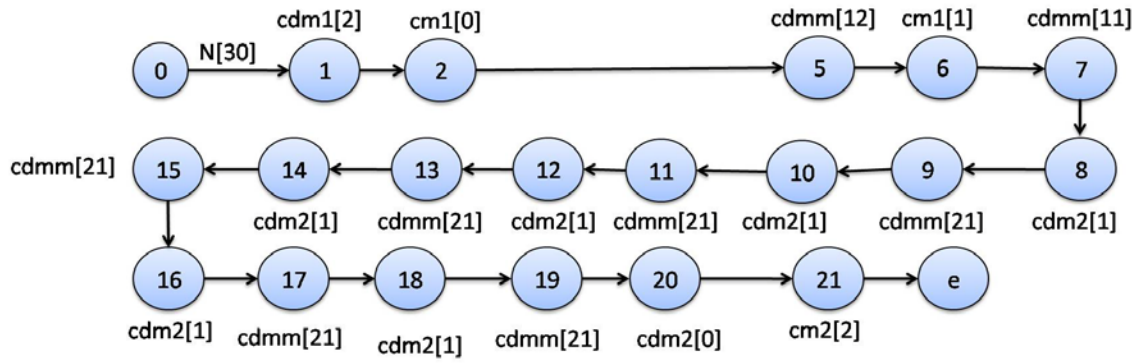


Figure 4 : La séquence de contrôle correspondant à un décalage de 30 bits

31 séquences ont été décrites, correspondant aux 32 valeurs de décalage pour les données. La composition et l'optimisation de ces séquences, selon les règles présentées au chapitre 4, donnent la description Moore_MR_CPOG complète du contrôleur (Figure 5).

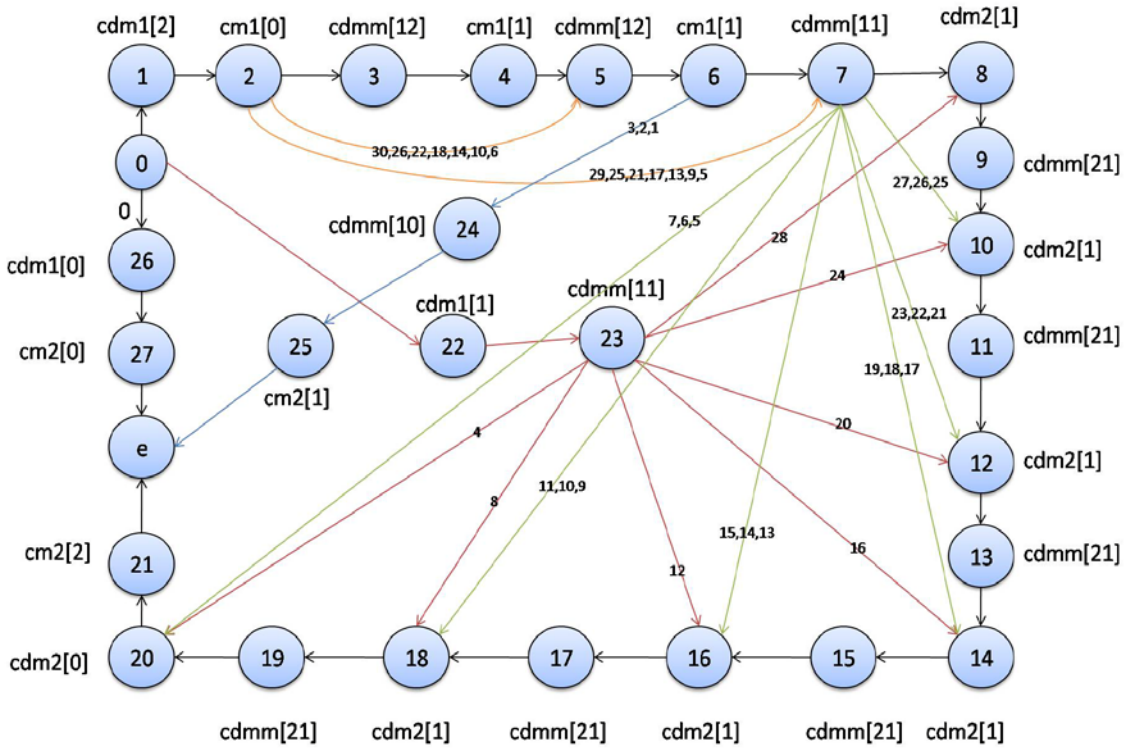


Figure 5: Modèle Moore_MR_CPOG complète pour le contrôleur du registre à décalage

Nous avons également décrit le contrôleur selon le modèle Mealy. Cela permet de modéliser ce contrôleur avec un nombre minimum d'actions séquentielles. Dans cet exemple, ce nombre est de 21 actions correspondant à un décalage de 31 bits. Une action modélise ici uniquement l'ordre d'un événement qui est le même que l'on commence par

la séquence la plus longue ou la plus courte, c'est pourquoi, Il n'y pas besoin de commencer la spécification par la séquence la plus longue.

La modélisation Mealy sert à rédiger une description plus compacte du contrôleur en CHP et permet par conséquent de réaliser une implémentation selon l'architecture FSM présentée au chapitre 3. Nous nous contentons uniquement de la réalisation FSM à cause d'un manque de temps et parce que l'architecture Mealy n'est pas QDI pour le moment comme nous l'avons vu au chapitre 4. La Figure 6 montre une partie de l'implémentation Mealy_MR_CPOG, nous ne montrons pas tout le modèle à cause de sa complexité. Ces séquences sont toutefois décrites dans l'annexe D.

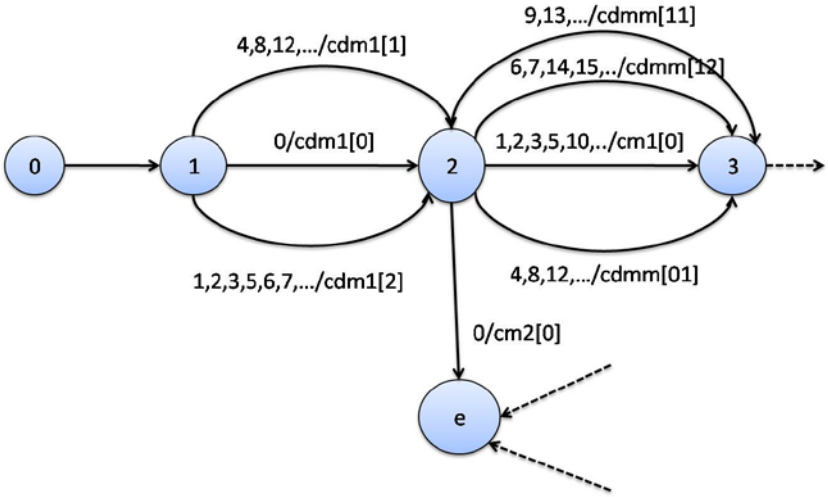


Figure 6 : Une partie de la desciption de l'unité de contrôle selon le modèle Mealy_MR_CPOG

Nous avons ainsi réalisé deux implémentations du circuit. La première est basée sur une architecture de Moore et la deuxième est une architecture de FSM obtenue avec TAST.

3. Synthèse de l'unité de contrôle (architecture de Moore vis-à-vis de l'architecture FSM)

Les deux circuits implémentés sont composés de portes simples à deux entrées (OR, NOR, …) de la bibliothèque CMOS130 de ST et des portes Muller complexes ou des séquenceurs de la bibliothèque TAL.

3.1. Simulations électriques :

Nous avons réalisé le montage suivant (figure 7) pour tester le circuit de contrôleur. La partie externe au circuit a été alimentée par la tension Vdd alors que le circuit sous test

est alimenté par la tension vdd_sous_test. Les portes ORs génèrent les signaux d'acquittement pour les sorties du contrôleur.

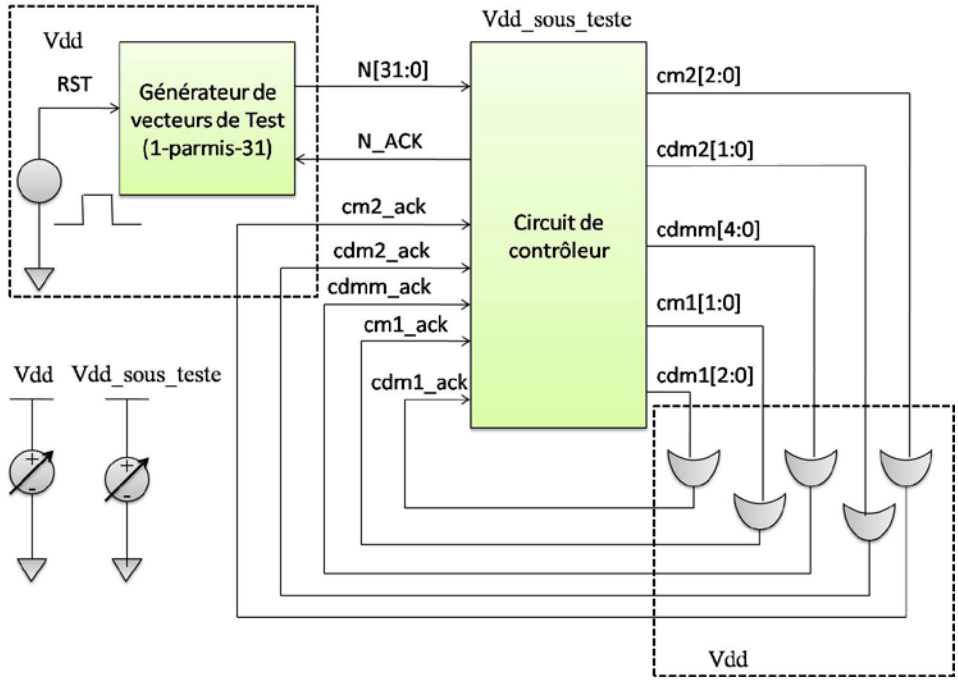


Figure 7 : Montage pour la simulation électrique de circuit de l'unité de contrôle

Le circuit générateur de vecteur de test, dans la Figure 8, a pour objectif de générer une nouvelle valeur, codée (1-parmi-31), sur le canal d'entrée « N » à chaque nouveau signal d'acquittement N_ack. Le générateur respecte également le protocole 4-phases sur ce canal.

Il est réalisé à l'aide de deux chaines de décalage de bascules D, commandées par l'horloge (N_ack), une par le front montant, l'autre par le front descendant, une série de portes AND génèrent les signaux N [i].

Le Tableau 1 montre le résultat de la simulation électrique de circuit par Nanosim. Comme prévu, l'architecture de Moore est plus performante que l'architecture FSM. L'architecture Moore est 14,6 fois moins consommatrice d'énergie dynamique et elle est 8,8 fois plus rapide que l'architecture FSM, ces résultats confirment ceux obtenus au chapitre 4 (pour l'exemple de DSI).

A ce stade nous pourrions conclure que l'architecture de Moore est la plus avantageuse parmi les trois architectures proposées dans cette thèse en termes de consommation, débit et robustesse. De plus, le gain en énergie et débit est proportionnel à

la complexité du contrôleur : plus le contrôleur est compliqué, plus nous avons un gain important en consommation et débit par rapport à l'architecture FSM.

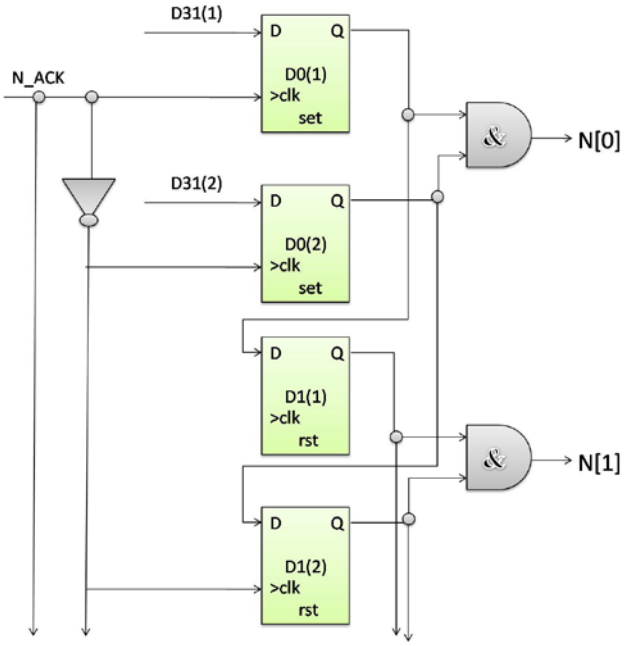


Figure 8 : Le circuit générateur de vecteur de test

Tableau 1 : résultat de la simulation électrique de deux circuits, l'un réalisé selon l'architecture FSM et deuxième selon l'architecture Moore

Simulation électrique /Nanosim	Nombre de transistors	Pic de crurent (uA)	Pd (uW)	Période (ns)	Ed (pJ)	ET	Ps (nW)
Arch_FSM_21_actions	7992	381,6	115	4918	566	2783892	724
Arch_Moor_27_actions	2170	249	69	560	39	21764	260
Gain	3,7	1,5	1,7	8,8	14,6	127,9	2,8

4. Optimisation de l'architecture de Moore

Quatre niveaux d'optimisation sont possibles pour améliorer la performance de l'architecture de contrôleur (chapitre 5):

- D'abord nous pouvons remplacer les fonctions M-OR par des fonctions OR-MD1P mais cela diminue la robustesse du circuit car le circuit n'est plus QDI (chapitre 5).

- Ensuite nous remplaçons les nouvelles structures OM-D1P par une implémentation en transistor OMD1P, ce qui rend à la fois le circuit QDI et réduit sa taille et sa consommation comme nous avons vu au chapitre 5.

- Ensuite, nous projetons la *Netlist* sur une bibliothèque de portes standard afin de remplacer les portes à deux entrées par des portes standards complexes (cette étape n'est pas réalisée pour cet exemple).

- Enfin, nous appliquons la technique DVS afin de réduire la consommation du circuit.

La figure 9 montre le gain en performance après avoir remplacé les structures M-OR par des structures OR-MD1P puis par des portes complexes OMD1P.

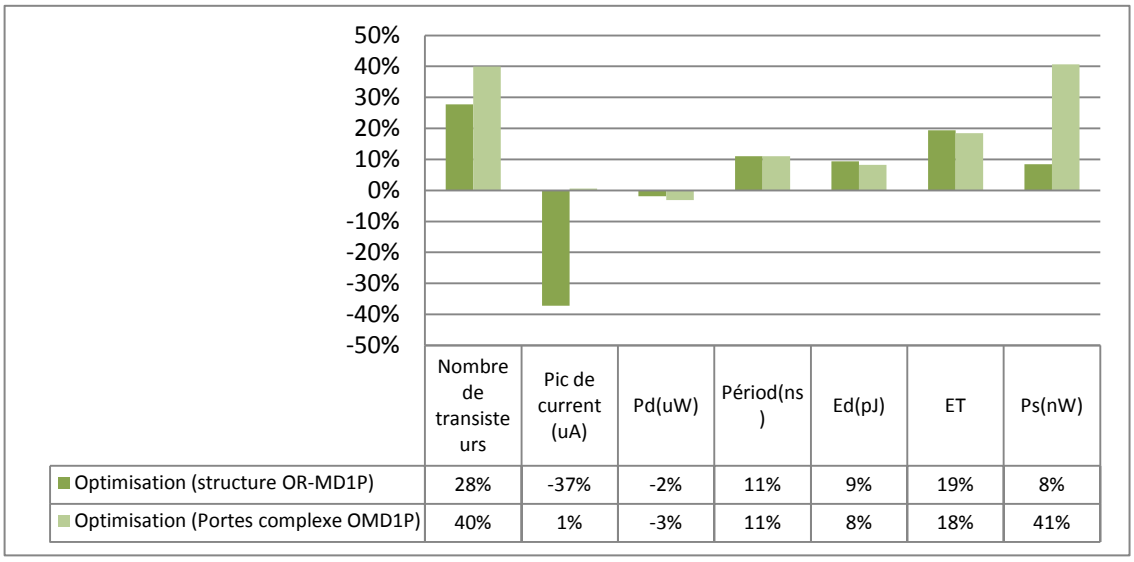


Figure 9 : Gain en performance après deux étapes d'optimisation

Le tableau 2 montre les résultats de la simulation électrique du circuit après optimisation pour une tension d'alimentation 0,6 V.

Ce qu'il est intéressant de regarder dans ce résultat est le facteur ET qui est indépendant de la tension. Ce facteur dépend de l'architecture employée. Ce résultat confirme nos premiers résultats obtenus au chapitre 5.

Tableau 2 : Application de technique DVS sur le circuit de c l'unité de contrôle

Structure OM vis-à-vis nouvelles portes OM	Tension	Pic de curant 1(uA)	Pd (uW)	Période (ns)	Ed (pJ)	ET	Ps (nW)
Arch. 2 Moor_optimisée (structure OMD1P)	1,2	247,6	72	498	36	17757	154
Arch. 2 Moor_optimisée (nouvelles portes OMD1P)	0,6	256,8	3	2382	7	17816	42
Gain	2,0	1,0	22,8	4,8	4,8	1,0	3,7

5. Vérification formelle

Comme nous l'avons vu à travers le chapitre 6, l'outil RAT fournit un cadre formel pour vérifier un circuit asynchrone d'une manière hiérarchique en se basant sur des modèles de composants en PSL.

La vérification du circuit de l'unité de contrôle dans l'outil RAT n'était pas possible car l'espace d'états du circuit dépasse la limite de calculs du cet outil. Nous avons donc choisi de vérifier formellement le résultat de la méthode de synthèse sur l'exemple DSI (Décodeur Séquentiel d'Instructions), car cet exemple (Figure 10) présente toutes les compositions logiques qu'on peut trouver dans un circuit de l'unité de contrôle après avoir été synthétisé selon l'architecture de Moore.

Ce circuit a été vérifié selon les étapes suivantes :

- Une modélisation en PSL de toutes les composantes de DSI où le séquenceur a été remplacé par son modèle (méthode hiérarchique).

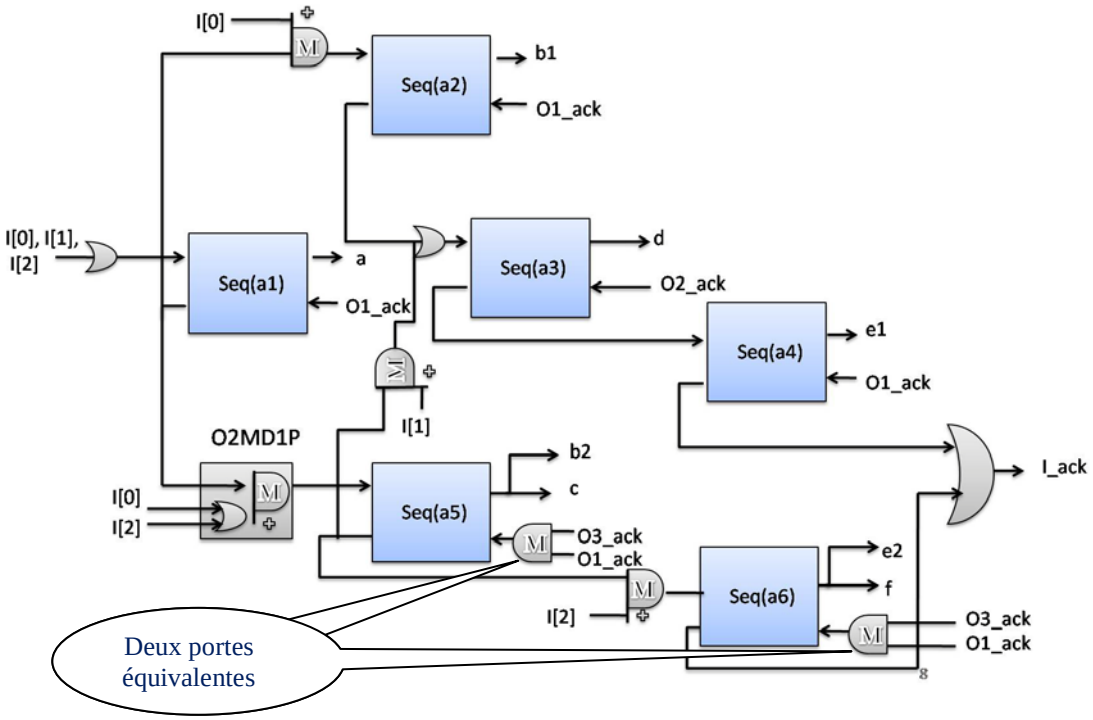


Figure 10 : Modélisation du circuit de DSI par des propriétés PSL dans RAT

- Une modélisation de l'environnement :

Dans le modèle du canal d'entrée, nous avons choisi de fixer la valeur de deux entrées et de vérifier le circuit formellement lorsqu'une seule entrée change. Cela

représente une approche hétérogène simulation-vérification formelle qui accélère et simplifie la procédure de vérification du circuit.

- Le canal d'entrée a été ainsi modélisé comme un canal actif à trois-rails (Code 1) :

```
Code 1: ENV_A_3R_1 ( I[0], I[1], I[2], I_ ACK)
always ( ! I_ACK &! I[0]→ eventually!  I[0]  )
&&
always ( ! I[0] → ! I[0] until_ ! I_ACK )
&&
always ( I[0] → I[0] until_  I_ACK )
&&
always ( ! I[1]  )
&&
always ( ! I[2] )}
```

Nous avons appliqué le modèle suivant (Code 2) pour les sorties (exemple : le canal O1 [A, B, E, O1_ACK]). Les sorties sont également des canaux multi-rails, elles sont modélisées comme des canaux passifs du côté de l'environnement (l'environnement reçoit une requête du circuit), elles provoquent un acquittement dès lors qu'une nouvelle valeur est produite sur le canal.

```
Code 2 : ENV_P_3R_1 (O1_ACK, A, B, E)
{Always ( ( A | B | E ) & ! O1_ack → eventually! O1_ack )
&&
Always ( ! A & ! B & ! E & O1_ack  →  eventually! ! O1_ack )
&&
Always ( ! O1_ack →  ! O1_ack until_  A | B | E )
&&
Always ( O1_ack →  O1_ack until_  ! A  & ! B & ! E )}
```

La troisième étape est la vérification des comportements du circuit. Cela se décompose en deux parties : tout d'abord, vérifier le protocole sur les entrées et les sorties des circuits. Ensuite, vérifier si le protocole 4 phases est respecté sur les entrées et les sorties de chaque séquenceur par son propre environnement.

L'assertion suivante (Code 3) donne un exemple sur la vérification du protocole 4 phases sur le canal O1.

```
Code 3 : A :{always ( A →  A until_  O1_ACK )
&&
always ( ! A   →  ! A until_ ! O1_ACK )
&&
always ( B →  B until_  O1_ACK )
&&
always ( ! B   →  ! B until_ ! O1_ACK )
&&
always ( E →  E until_  O1_ACK )
&&
always ( ! E   →  ! E until_ ! O1_ACK )}
```

L’assertion suivante (Code 4) donne un exemple sur la vérification de l’environnement du séquenceur Seq(a2).

```
Code 4 : always (! W1 & ! W2  → eventually! W2 )
&&
always ( W1 &  W2 → eventually! ! W2 )
&&
always ( ! W2 → ! W2 until_ ! W1 )
&&
always ( W2 → W2 until_  W1 )
&&
always ( AA & ! O1_ACK → eventually! O1_ACK )
&&
always ( ! AA & O1_ACK  → eventually! ! O1_ACK )
&&
always ( O1_ACK → O1_ACK until_  ! AA )}
```

Cette vérification formelle de circuit a montré par exemple que le circuit n’est pas QDI lorsqu’il y a deux portes de Muller différentes pour synchroniser le rendez-vous entre (O1_ACK et O3_ACK) (voire Figure 9). Cela a montré un défaut dans la méthode de synthèse. Nous avons, par la suite, modifié cette méthode pour éviter ce défaut.

La vérification formelle du circuit DSI garantit en même temps les propriétés QDI du circuit de l’unité de contrôle car les deux circuits sont synthétisés selon la même méthode de synthèse et ils ont les mêmes structures logiques. En revanche, cette démarche ne peut pas garantir que le circuit de l’unité de contrôle satisfasse sa spécification (Figure 5). Des simulations électriques et fonctionnelles sont ainsi nécessaires pour garantir la spécification.

6. Conclusion

Dans ce chapitre, nous avons pu montrer les avantages de la méthode de l’implémentation directe selon l’architecture de Moore par rapport à la méthode de TAST, au moyen de la réalisation d’une unité de contrôle de registre à décalage. Cela confirme notre précédent résultat. Ce résultat montre également l’intérêt de réaliser un outil de synthèse basé sur notre méthode. La méthode de vérification formelle proposée a été également validée et appliquée avec succès pour vérifier partiellement le circuit du contrôleur.

7. Bibliographie

1. Rios, D., *Systeme a microprocesseur asynchrone basse consommation* These Docteur-Ingénieur, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, 2008.

CHAPITRE 8 : CONCLUSION

The years of anxious searching in the dark, with their intense longing, their alternations of confidence and exhaustion and the final emergence into the light - only those who have experienced it can understand it.

La plus belle chose que nous puissions éprouver, c'est le côté mystérieux de la vie.

Quiconque prétend s'ériger en juge de la vérité et du savoir s'expose à périr sous les éclats de rire des dieux puisque nous ignorons comment sont réellement les choses et que nous n'en connaissons que la représentation que nous en faisons.

La seule chose absolue dans un monde comme le nôtre, c'est l'humour.

La science est une chose merveilleuse... tant qu'il ne faut pas en vivre !

Albert Einstein

Conclusion

De nos jours, le principal frein à l'adoption des circuits asynchrones dans l'industrie est le manque d'outils de conception d'un niveau « industriel » dédiés à ce type de circuit logique. Ce travail de thèse a donc pour objectif de participer au développement d'un environnement de conception de circuits asynchrones Quasi Insensibles aux Délais. Ce travail est intéressant pour des applications qui présentent des contraintes de sécurité, de consommation ou de robustesse aux variations PVT. A titre d'exemple, nous pouvons cités de très nombreuses applications, qui intègrent au moins l'une de ces contraintes, telles que les cartes à puces, les cœurs de microprocesseur, les IPs de contrôle, les réseaux de capteurs, etc.

Dans ce manuscrit de thèse, nous avons présenté une méthode novatrice pour synthétiser des circuits asynchrones séquentiels de type QDI. Dans la première partie, nous avons introduit les principaux concepts de la logique asynchrone ainsi que les différentes méthodes de conception académiques existantes. Les points forts et les points faibles de chacune de ces méthodes ont été présentés. Par la suite, nous avons présenté une classification des contrôleurs séquentiels QDI. Nous avons alors abordé les contrôleurs traités dans ce travail de thèse : les contrôleurs SIMO (Single Input Multi Output). Nous avons abordé trois méthodes de synthèse pour répondre aux exigences des circuits cibles.

La première méthode existante dans l'état de l'art est une application du flot de conception TAST. Nous avons présenté cette méthode et évoqué la problématique liée à l'architecture des contrôleurs réalisés suivant cette approche.

Nous avons également proposé de modéliser les comportements des contrôleurs par EXT-CPOG, un format graphique intermédiaire qui permet de décrire des comportements séquentiels et concurrents à la fois. Ce format apparait bien adapté pour représenter les contrôleurs QDI parce qu'il garantit un niveau d'abstraction élevé qui facilite la conception. Il a été ensuite optimisé par plusieurs algorithmes afin de réduire la taille et la consommation du circuit résultant.

Enfin, nous avons proposé une méthode de synthèse simple à mettre en œuvre, qui exploite un composant « le séquenceur » pour implémenter de façon directe la spécification EXT-CPOG en circuit. Il est possible de synthétiser les circuits à partir d'un

format Moore-EXT-CPOG et un Format Mealy-MR-CPOG. Nous avons également présenté tous les algorithmes nécessaires pour la synthèse. Cette technique simple permet de réaliser des contrôleurs de grande taille puisque le temps de calcul nécessaire par cet algorithme est linéairement proportionnel à la taille du contrôleur.

Nous avons montré ensuite que la technique de l'implémentation directe d'un contrôleur modélisé par Moore-MR-CPOG est la plus intéressante, parce que les circuits issus de cette méthode sont plus robustes, consomment moins d'énergie, sont plus petits en surface et plus rapides en débit. Le gain présenté par cette méthode est proportionnel à la taille et à la complexité des contrôleurs. Ces résultats ont été mis en évidence par un exemple concret et complexe. Nous avons cherché ensuite à améliorer les performances obtenues par cette méthode en appliquant plusieurs techniques de réduction d'énergie. Nous avons ainsi pu enrichir la bibliothèque des composants asynchrones TAL par de nouvelles portes complexes avantageuses en termes de surface, de consommation et de vitesse.

Enfin, nous avons proposé une méthode de vérification formelle simple à mettre en œuvre qui se base sur une modélisation matérielle en PSL, un langage de spécification de propriétés. Nous avons développé une bibliothèque de modèles PSL qui permet de vérifier les circuits QDI asynchrones de manière générale. Nous avons utilisé l'outil RAT comme outil de vérification de nos circuits décrits par des propriétés PSL. La méthode proposée a été exploitée de manière hiérarchique, ce qui permet de réduire le temps de vérification, et permet de vérifier des circuits bien plus grands que ce que permettrait une vérification à plat. Enfin, nous nous sommes basés sur les études théoriques présentées dans l'état de l'art pour garantir la justesse de l'approche hiérarchique.

Perspectives

Lorsque nous avons démarré ce travail, nous visions des contrôleurs asynchrones séquentiels simples de type « séquenceurs ». Nous avons pu réaliser des contrôleurs de plus en plus complexes grâce à une méthodologie adaptée. La méthode de l'implémentation directe se montre en effet intéressante pour couvrir une large famille de contrôleurs QDI séquentiels. Cela a ainsi ouvert la porte à plusieurs perspectives :

- Les contrôleurs initialisent une communication sur un canal de sortie lorsque le protocole 4 phases est terminé sur le canal le précédant dans une séquence

de contrôle. La génération des sorties des contrôleurs est complètement séquentielle dans ce cas. Il serait intéressant d'étudier la possibilité de démarrer le protocole 4 phases sur un canal avant que le canal le précédant ait terminé son protocole. Pour cela, il faudrait utiliser d'autres types de séquenceurs, mais permettrait d'augmenter le parallélisme du circuit et de gagner sur la vitesse de réponse du contrôleur.

- L'étude de contrôleurs multicanaux en entrée et en sortie (MIMO : Multi Input Multi Output ou MISO : Multi Input Single Output) mérite une étude particulière pour garantir les propriétés QDI des circuits. Ces circuits peuvent être implémentés également par des séquenceurs. La réalisation de contrôleurs MIMO (Figure 3) permet la synthèse de contrôleurs dynamiquement configurables de type machine à état. Afin d'implémenter des contrôleurs multicanaux en entrées, il serait intéressant de réaliser des séquenceurs à trois canaux (LR, LA, O, O_ack, RR, RA).

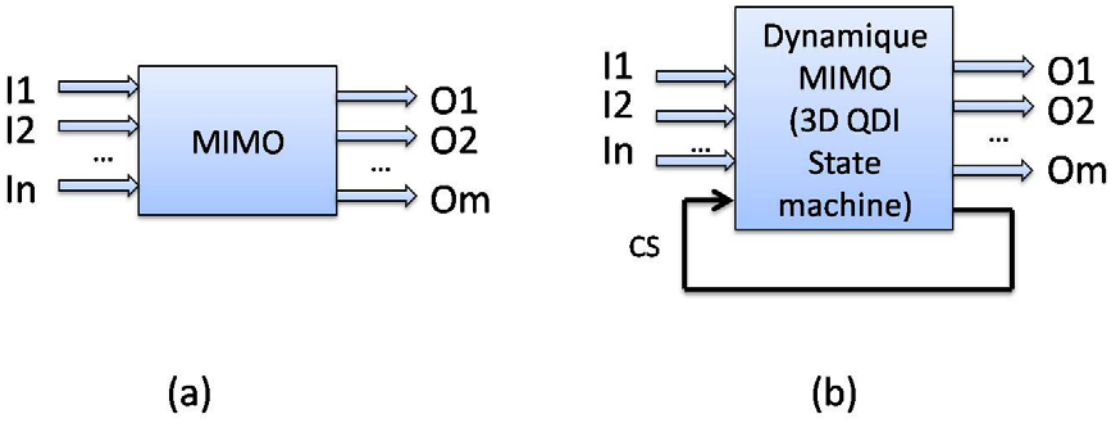


Figure 1 : Comportement complètement séquentiel des sorties du contrôleur

- La séquentialité existante dans le chemin de donnée QDI peut être exploitée pour simplifier l'architecture des contrôleurs séquentiels, l'idée est de diviser les contrôleurs en plusieurs petits contrôleurs où chacun contrôle une partie du pipeline (Figure 5). La division dans la spécification Moore-MR-CPOG en multi-contrôleurs peut être réalisée par une projection de la spécification sur un canal de sortie ou sur un ensemble de canaux de sortie.

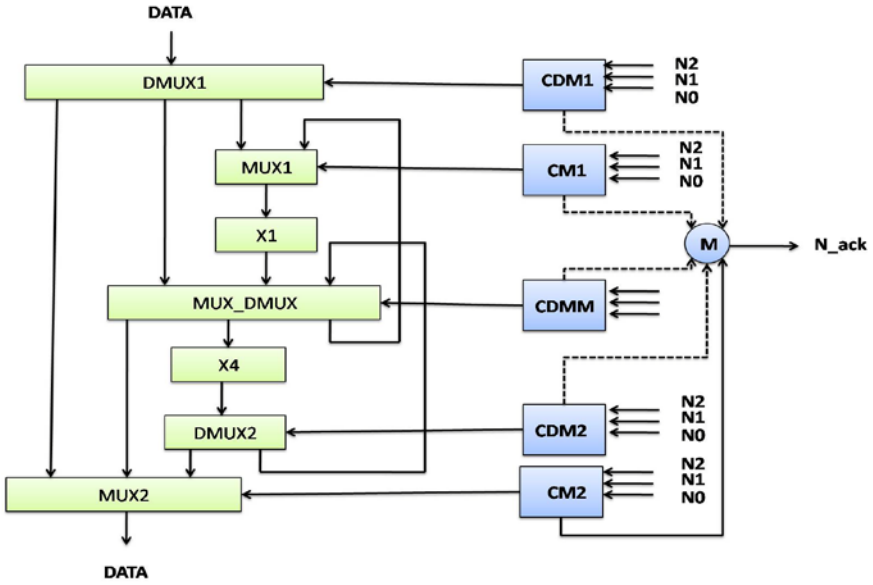


Figure 2 : Contrôleur divisé pour le registre à décalages

- Les nouvelles cellules logiques proposées peuvent aussi être appliquées dans d'autres architectures séquentielles. Par contre ces nouvelles cellules peuvent mettre les propriétés QDI du circuit en danger. Il est alors important d'ajouter des hypothèses temporelles ou d'analyser le circuit pour garantir sa robustesse.
- L'outil de vérification de modèle RAT est limité en puissance de calcul. Une étape importante est la modélisation des propriétés PSL par un diagramme de décision binaire symbolique. Il peut être intéressant d'appliquer d'autres types de modélisation comme les CBDD (Connective Binary Decision Diagrams). Ces diagrammes montrent en effet un avantage principal : ils ne souffrent pas d'une croissance exponentielle lorsque le nombre d'entrées (de signaux) et d'interconnexions logiques augmentent.

Liste de publications

Khaled ALSAYEG, Katell MORIN-ALLORY and Laurent FESQUET. RAT-based formal verification of QDI asynchronous controllers. FDL 2009, Sophia Antipolis, France, September 22 - 24, 2009. Article.

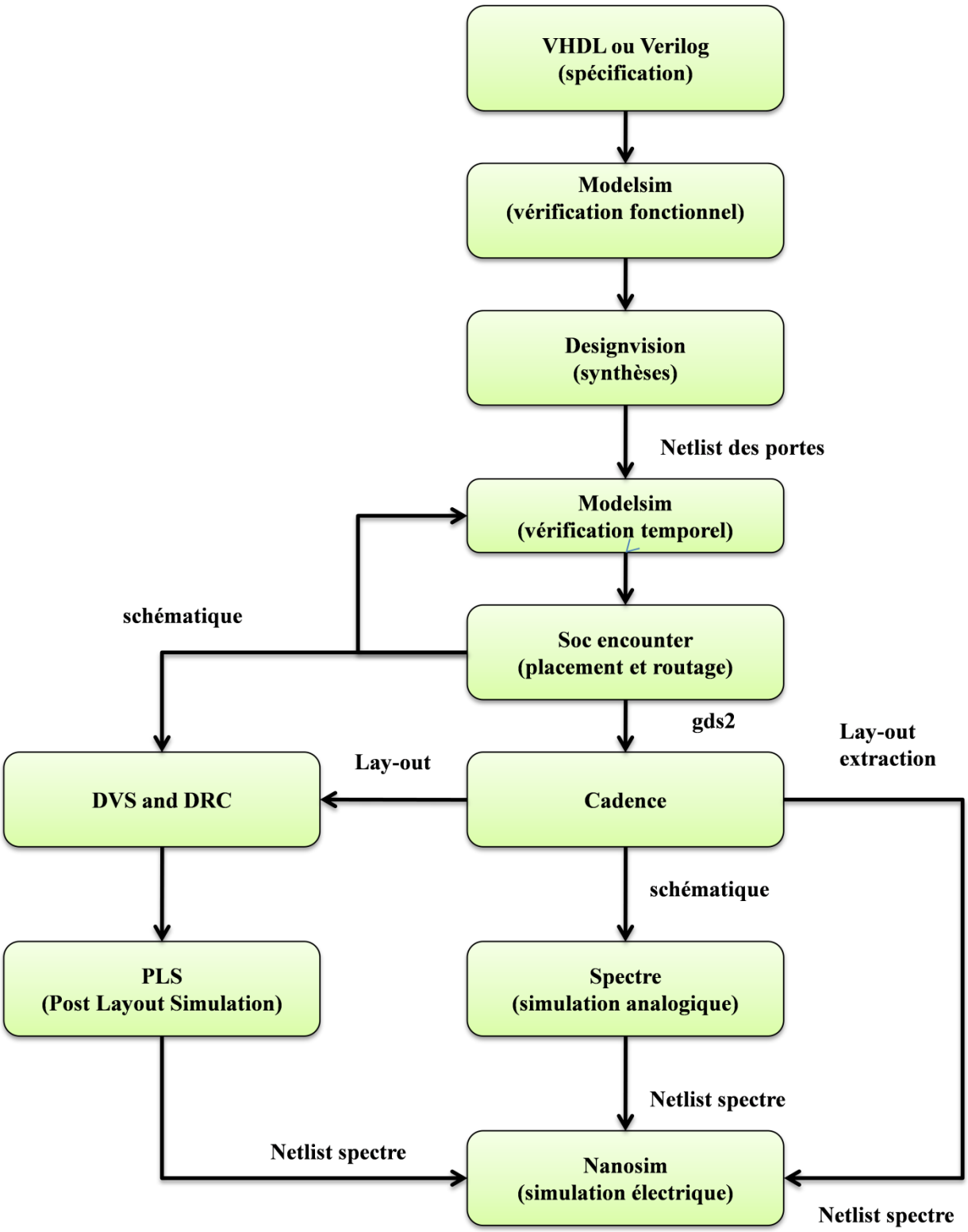
Khaled ALSAYEG, Laurent FESQUET, Gilles SICARD, David RIOS, Marc RENAUDIN. Optimizing speed and consumption of QDI controllers using direct mapping synthesis. NEWCAS 2009, June 28 to July 1, 2009, in Toulouse, France. Article.

Khaled ALSAYEG, Laurent FESQUET, Gilles SICARD, David RIOS, Marc RENAUDIN. Direct mapping of sequential QDI controllers, DATE 2009, Nice, France, 20 to 24 April 2009. Poster.

Khaled ALSAYEG, Laurent FESQUET, Gilles SICARD, David RIOS, Marc RENAUDIN. Synthesis of asynchronous QDI FSM based on optimized sequencers, ESSCIRC 2008, Edinbourg, Ecosse, 15 - 19 September 2008. Poster.

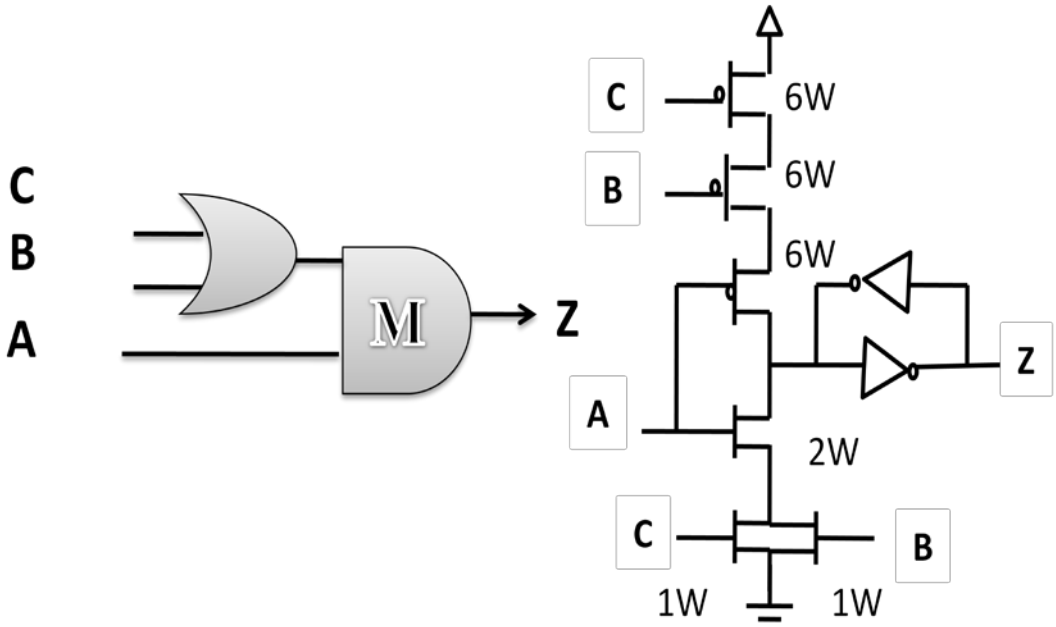
Annexe A
Simulation électrique

Flot de simulation électrique

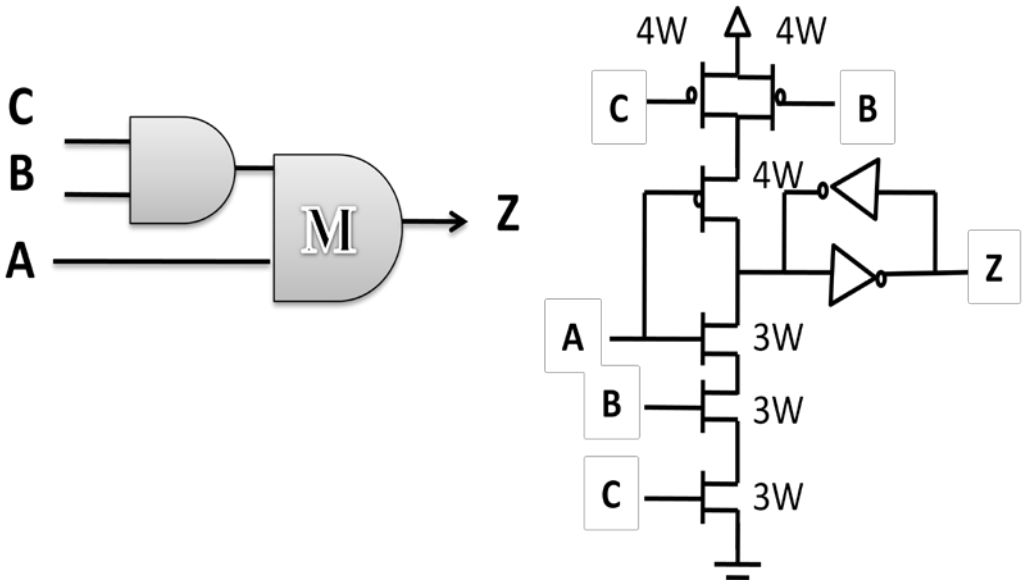


Annexe B
Nouvelles portes pour la bibliothèque TAL

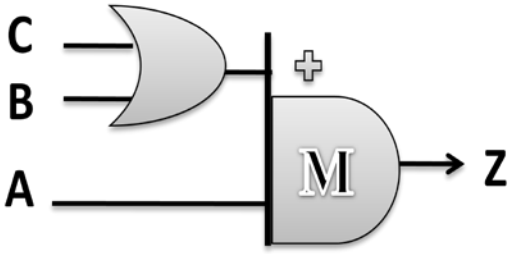
Porte	OR2M2
Fonction logique	Schématique

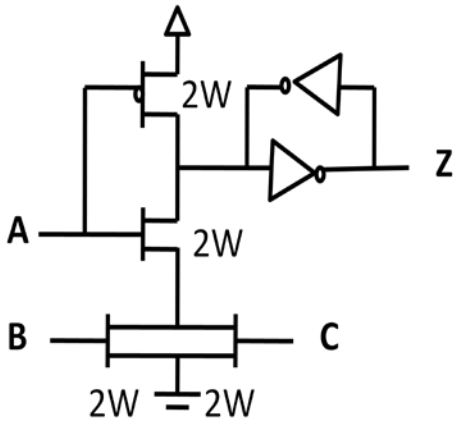


Porte	NR2M2
Fonction logique	Schématique

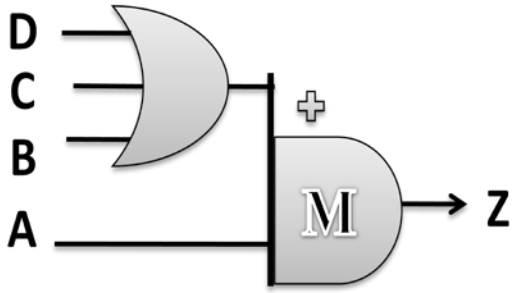


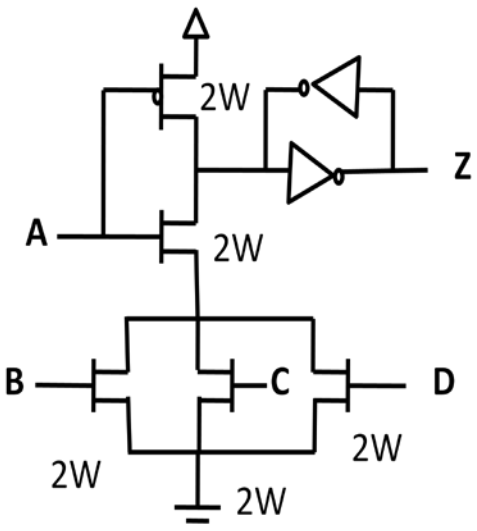
Porte	OR2M2D1P
Fonction logique	Schématique



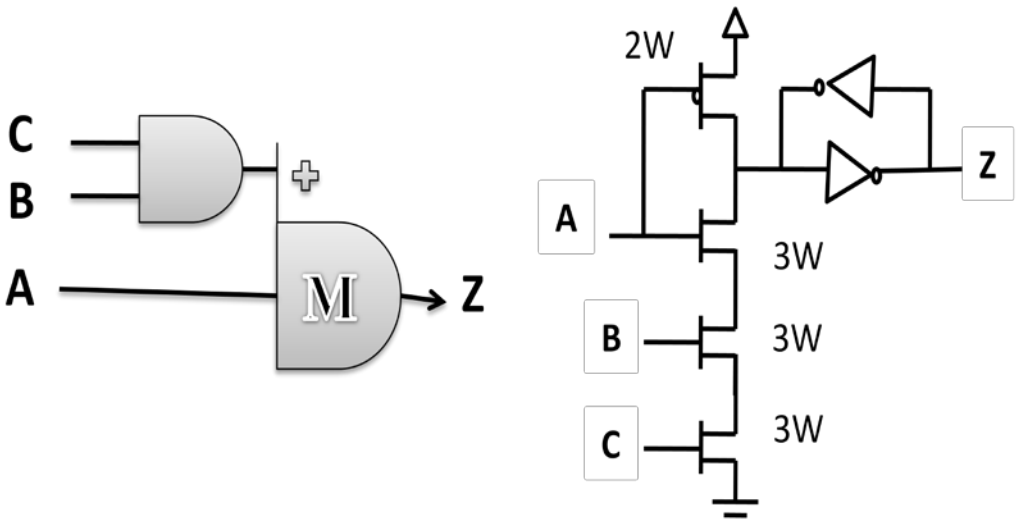


Porte	OR3M2D1P
Fonction logique	Schématique

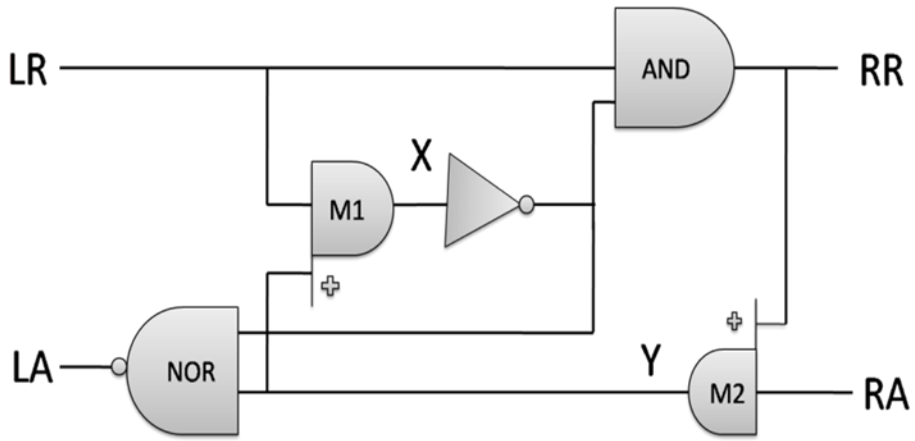




Porte	AN2M2D1P
Fonction logique	Schématique



Porte	SEQD
Fonction logique	Schématique



Annexe C
Bibliothèque des modèles PSL

Notation :

Nom_de_Portes (signaux)

R{Groupe de restrictions}

A{Groupe d'assertions}

****

M2X0 (Z, A, B)

R:{always (A & B & ! Z → eventually! Z)

&&

always (! A & ! B & Z → eventually! ! Z)

&&

always (Z → Z until_ ! A &! B)

&&

always (! Z → ! Z until_ A & B)}

A:{}

****

IVLL (Z, A)

R:{always(A & Z → eventually! ! Z)

&&

always (! A & ! Z → eventually! Z)

&&

always (! Z → ! Z until_ ! A)

&&

always (Z → Z until_ A)}

A:{}

****

AN2LL (Z, A, B)

R:{always(A & B & ! Z → eventually! Z)

&&

always((! A | ! B) & Z → eventually! ! Z)

&&

always(Z → Z until_ ! A | ! B)

&&

always(! Z → ! Z until_ A & B)}

A:{}

****

ND2LL (Z, A, B)

R:{always((! A | ! B) & ! Z → eventually! Z)

&&

always(A & B & Z → eventually! ! Z)

&&

always(Z → Z until_ A & B)

&&

always(! Z → ! Z until_ ! A | ! B)}

A:{}

****

NR2LL (Z, A, B)

R:{always((A | B) & Z → eventually! ! Z)

&&

always(! A & ! B &! Z → eventually! Z)

&&

always(! Z → ! Z until_ ! A & ! B)

&&

always(Z → Z until_ A | B)}

A:{}

****

NR2ALL (Z, A, B)

R:{always((! A | B) & Z → eventually! ! Z)

&&

always(A & ! B &! Z → eventually! Z)

&&

always(! Z → ! Z until_ A & ! B)

&&

always(Z → Z until_ ! A | B)}

A:{}

****

OR2LL (Z, A, B)

R:{always((A | B) & ! Z → eventually! Z)

&&

always (! A & ! B & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A | B)

&&

always (Z → Z until_ ! A & ! B)}

A:{}

****

OR3LL (Z, A, B, C)

R:{always((A | B | C) & ! Z → eventually! Z)

&&

always (! A & ! B & ! C & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A | B | C)

&&

always (Z → Z until_ ! A & ! B & ! C)}

A:{}

****

AO6NLL (Z, A, B, C)

R:{always(((A & B) | C) & ! Z → eventually! Z)

&&

always (((! A | ! B) & ! C) & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ (A & B) | C)

&&

always (Z → Z until_ (! A | ! B) & ! C)}

A:{}

****

AO4ALLP (Z, A, B, C, D)

R:{always(((! A | B) & (C | D)) & ! Z → eventually! Z)

&&

always (((A & ! B) | (! C & ! D)) & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ ((! A | B) & (C | D)))

&&

always (Z → Z until_ ((A & ! B) | (! C & ! D))}

A:{}

****

M2D1P (Z, A, B)

R:{always(A & B & ! Z → eventually! Z)

&&

always (! A & Z → eventually! ! Z)

&&

always (Z → Z until_ ! A)

&&

always (! Z → ! Z until_ A & B)}

A:{}

****

M2D1N (Z, A, B)

R:{always(A & ! Z → eventually! Z)

&&

always (! A & ! B & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A)

&&

always (Z → Z until_ ! A & ! B)}

A:{}

****

M3D1P1N (Z, A, B, C)

R:{always(A & B & ! Z → eventually! Z)

&&

always (! A & ! C & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A & B)

&&

always (Z → Z until_ ! A & ! C)}

A:{}

****

M3D1P (Z, A, B, C)

R:{always(A & B & C & ! Z → eventually! Z)

&&

always (! A & ! B & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A & B & C)

&&

always (Z → Z until_ ! A & ! B)}

A:{}

****

M3D1N (Z, A, B, C)

R:{always(A & B & ! Z → eventually! Z)

&&

always (! A & ! B & ! C & Z → eventually! ! Z)

&&

always (! Z → ! Z until_ A & B)

&&

always (Z → Z until_ ! A & ! B & ! C)}

A:{}

****

OR2M2D1P (Z, A, B, C)

R:{always (A & (B | C) & ! Z → eventually! Z)

&&

always (! A & Z → eventually! ! Z)

&&

always (Z → Z until_ ! A)

&&

always (! Z → ! Z until_ A & (B | C))}

A:{}

****

Sequencer_modèle_SR_1 (RR, LA, LR, RA, X)

R:{always (! RR & LR & ! RA &! X → eventually! RR)

&&

always (RR & LR & RA & X → eventually! ! RR)

&&
always (! RR → ! RR until_ LR & ! RA & ! X)
&&
always (RR → RR until_ LR & RA & X)
&&
always (! X & LR & RA → eventually! X)
&&
always (X & ! LR & ! RA → eventually! ! X)
&&
always (! X → ! X until_ LR & ! RA & ! X)
&&
always (X → X until_ LR & RA & X)
&&
always (! LA & LR & ! RA & X → eventually! LA)
&&
always (LA & ! LR & ! RA & ! X → eventually! ! LA)
&&
always (! LA → ! LA until_ LR & ! RA & X)
&&
always (LA → LA until_ ! LA & ! RA & ! X)}
A:{always (! LA & ! LR → eventually! LR)
&&
always (LA & LR → eventually! ! LR)
&&
always (! LR → ! LR until_ ! LA)
&&
always (LR → LR until_ LA)
&&
always (RR & ! RA → eventually! RA)
&&
always (! RR & RA → eventually! ! RA)
&&
always (! RA → ! RA until_ RR)
&&
always (RA → RA until_ ! RR)}

****

Sequencer_modèle_SR_2 (RR, LA, LR, RA, X)

R:{always (! RR & LR & ! X → eventually! RR)

&&

always (RR & X → eventually! ! RR)

&&

always (! RR → ! RR until_ LR & ! X)

&&

always (RR → RR until_ X)

&&

always (! X & LR & RA → eventually! X)

&&

always (X & ! LR → eventually! ! X)

&&

always (! X → ! X until_ LR & RA)

&&

always (X → X until_ ! LR)

&&

always (! LA & ! RA & X → eventually! LA)

&&

always (LA & ! X → eventually! ! LA)

&&

always (! LA → ! LA until_ ! RA & X)

&&

always (LA → LA until_ ! X)}

A:{always (! LA & ! LR → eventually! LR)

&&

always (LA & LR → eventually! ! LR)

&&

always (! LR → ! LR until_ ! LA)

&&

always (LR → LR until_ LA)

&&

always (RR & ! RA → eventually! RA)

&&

always (! RR & RA → eventually! ! RA)

&&

always (! RA → ! RA until_ RR)

&&

always (RA → RA until_ ! RR)}

****

Sequencer_Modèle_MR_3 (RR, LA, LR, RA, X)

R:{always (! RR & LR & ! RA &! X → eventually! RR)

&&

always (RR & LR & RA & X → eventually! ! RR)

&&

always (! RR → ! RR until_ LR & ! RA & ! X)

&&

always (RR → RR until_ LR & RA & X)

&&

always (! X & LR & RA → eventually! X)

&&

always (X & ! LR & ! RA → eventually! ! X)

&&

always (! X → ! X until_ LR & ! RA & ! X)

&&

always (X → X until_ LR & RA & X)

&&

always (! LA & LR & ! RA & X → eventually! LA)

&&

always (LA & ! LR & ! RA & ! X → eventually! ! LA)

&&

always (! LA → ! LA until_ LR & ! RA & X)

&&

always (LA → LA until_ ! LA & ! RA & ! X)}

A:{always (! LA & ! LR → eventually! LR)

&&

always (LA & LR → eventually! ! LR)

&&

always (! LR → ! LR until_ ! LA)

&&

always (LR → LR until_ LA)

&&

always (RR & ! RA → eventually! RA)

&&

always (! RR & RA → eventually! ! RA)

&&

always (RA → RA until_ ! RR)}

****

sequencer_ Modèle_MR_4 (RR, LA, LR, RA, XX, YY)

R:{always (! RR & LR & ! RA & ! XX & ! YY → eventually! RR)

&&

always (RR & LR & RA & XX & YY → eventually! ! RR)

&&

always (! RR → ! RR until_ LR & ! RA & ! XX & ! YY)

&&

always (RR → RR until_ LR & RA & XX & YY)

&&

always (! YY & RR & LR & RA & ! XX → eventually! YY)

&&

always (YY & ! RR & LR & ! RA & XX → eventually! ! YY)

&&

always (! YY → ! YY until_ RR & LR & RA & ! XX)

&&

always (YY → YY until_ ! RR & ! RA & LR & XX)

&&

always (! XX & RR & LR & RA & YY → eventually! XX)

&&

always (XX & ! RR & ! LR & ! RA & ! YY → eventually! ! XX)

&&

always (! XX → ! XX until_ RR & LR & RA & YY)

&&

always (XX → XX until_ ! RR & ! LR & ! RA & ! YY)

&&

always (! LA & ! RR & LR & ! RA & ! YY & XX → eventually! LA)

&&

always (LA & ! RR & ! LR & ! RA & ! YY & ! XX → eventually! ! LA)

&&

always (! LA → ! LA until_ ! RR & LR & ! RA & ! YY & XX)

&&

always (LA → LA until_ ! RR & ! LR & ! RA & ! YY & ! XX)}

A:{always (! LA & ! LR → eventually! LR)

&&

always (LA & LR → eventually! ! LR)

&&

always (! LR → ! LR until_ ! LA)

&&

always (LR → LR until_ LA)

&&

always (RR & ! RA → eventually! RA)

&&

always (! RR & RA → eventually! ! RA)

&&

always (RA → RA until_ ! RR)}

****

Double-Séquenceur (RR1, RR2, LA, LR, RA1, RA2, X1, X2)

R:{always (! RR1 & LR & ! X1 → eventually! RR1)

&&

always (RR1 & LR & X1 → eventually! ! RR1)

&&

always (! RR1 → ! RR1 until_ LR &! X1)

&&

always (RR1 → RR1 until_ LR & X1)

&&

always (! X1 & LR & RA1 → eventually! X1)

&&

always (X1 & ! LR & ! RA1 → eventually! ! X1)
&&
always (! X1 → ! X1 until_ LR & RA1)
&&
always (X1 → X1 until_ ! LR & ! RA1)
&&
always (! RR2 & ! RA1 & X1 & ! X2 → eventually! RR2)
&&
always (RR2 & ! RA1 & X1 & X2 → eventually! ! RR2)
&&
always (! RR2 → ! RR2 until_ ! RA1 & X1 & ! X2)
&&
always (RR2 → RR2 until_ ! RA1 & X1 & X2)
&&
always (! X2 & RA2 & X1 → eventually! X2)
&&
always (X2 & ! RA2 & ! X1 → eventually! ! X2)
&&
always (! X2 → ! X2 until_ RA2 & X1)
&&
always (X2 → X2 until_ ! RA2 & ! X1)
&&
always (! LA & ! RA2 & X2 → eventually! LA)
&&
always (LA & ! RA2 & ! X2 → eventually! ! LA)
&&
always (! LA → ! LA until_ ! RA2 & X2)
&&
always (LA → LA until_ ! RA2 & ! X2)}
A:{always (! LA & ! LR → eventually! LR)
&&
always (LA & LR → eventually! ! LR)
&&
always (! LR → ! LR until_ ! LA)
&&

always (LR → LR until_ LA)

&&

always (RR1 & ! RA1 → eventually! RA1)

&&

always (! RR1 & RA1 → eventually! ! RA1)

&&

always (! RA1 → ! RA1 until_ RR1)

&&

always (RA1 → RA1 until_ ! RR1)

&&

always (RR2 & ! RA2 → eventually! RA2)

&&

always (! RR2 & RA2 → eventually! ! RA2)

&&

always (! RA2 → ! RA2 until_ RR2)

&&

Always (RA2 → RA2 until_ ! RR2)}

****

ENV_A_SR (X, A)

R:{always (! A & ! X → eventually! X)

&&

always (A & X → eventually! ! X)

&&

always (! X → ! X until_ ! A)

&&

always (X → X until_ A)}

A:{always (A → A until_ ! X)

&&

always (! A → ! A until_ X)}

****

ENV_A_DR (X, Y, A)

R:{always (! A & ! X & ! Y → eventually! X | Y)

&&

always (A & X → eventually! ! X)

&&

always (A & Y → eventually! ! Y)

&&

always (! X → ! X until_ ! A)

&&

always (X → X until_ A)

&&

always (! Y → ! Y until_ ! A)

&&

always (Y → Y until_ A)

&&

never! (R1 & Y)

}

A:{always (A → A until_ ! X & ! Y)

&&

always (! A → ! A until_ X | Y)}

****

ENV_A_3R (X, Y, Z, A)

R:{

always (! A & ! X → eventually! X)

&&

always (! X → ! X until_ ! A)

&&

always (X → X until_ A)

&&

always (! Y)

&&

always (! Z)

}

A:{always (A → A until_ ! X & ! Y & ! Z)

&&

always (! A → ! A until_ X | Y | Z)}

****

ENV_P_SR (A, X)

R:{always (X & ! A → eventually! A)

&&

always (! X & A → eventually! ! A)

&&

always (! A → ! A until_ X)

&&

always (A → A until_ ! X)}

A:{always (X → X until_ A)

&&

always (! X → ! X until_ ! A)}

****

ENV_P_DR(A, X, Y)

R:{always ((X | Y) & ! A → eventually! A)

&&

always (! X & ! Y & A → eventually! ! A)

&&

always (! A → ! A until_ X | Y)

&&

always (A → A until_ ! X & ! Y)}

A:{always (X → X until_ A)

&&

always (! X → ! X until_ ! A)

&&

always (Y → Y until_ A)

&&

always (! Y → ! Y until_ ! A)}

****

ENV_P_3R (A, X, Y, Z)

R:{always ((X | Y | Z) & ! A → eventually! A)

&&

always (! X & ! Y & ! Z & A → eventually! ! A)

&&

always (! A → ! A until_ X | Y | Z)

&&

always (A → A until_ ! X & ! Y & ! Z)}

A:{always (X → X until_ A)

&&

always (! X → ! X until_ ! A)

&&

always (Y → Y until_ A)

&&

always (! Y → ! Y until_ ! A)

&&

always (Z → Z until_ A)

&&

always (! Z → ! Z until_ ! A)}

****

ENV_P_4R (A, X, Y, Z, W)

R:{always ((X | Y | Z | W) & ! A → eventually! A)

&&

always (! X & ! Y & ! Z & ! W & A → eventually! ! A)

&&

always (! A → ! A until_ X | Y | Z | W)

&&

always (A → A until_ ! X & ! Y & ! Z & ! W)}

A:{always (X → X until_ A)

&&

always (! X → ! X until_ ! A)

&&

always (Y → Y until_ A)

&&

always (! Y → ! Y until_ ! A)

&&

always (Z → Z until_ A)

&&

always (! Z → ! Z until_ ! A)

&&

always (W → W until_ A)

&&

always (! W → ! W until_ ! W)}

****

4PH_A_SR(X, A)

R:{}

A:{always (A → A until_ ! X)

&&

always (! A → ! A until_ X)}

****

4PH_A_DR(X, A)

R:{}

A:{always (A → A until_ ! X)

&&

always (! A → ! A until_ X)}

****

4PH_P_SR(A, X)

R:{}

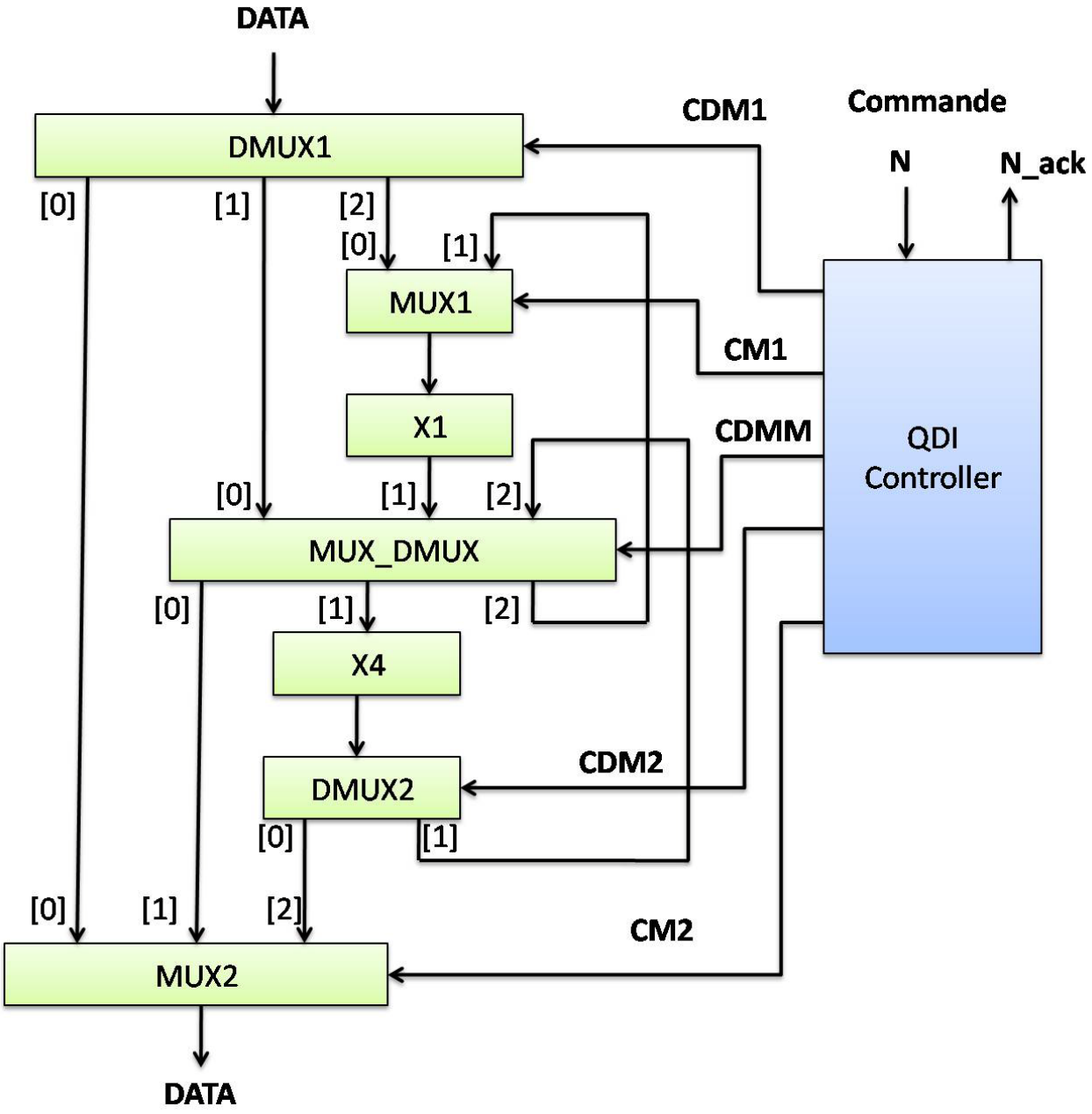
A:{always (X → X until_ A)

&&

always (! X → ! X until_ ! A)}

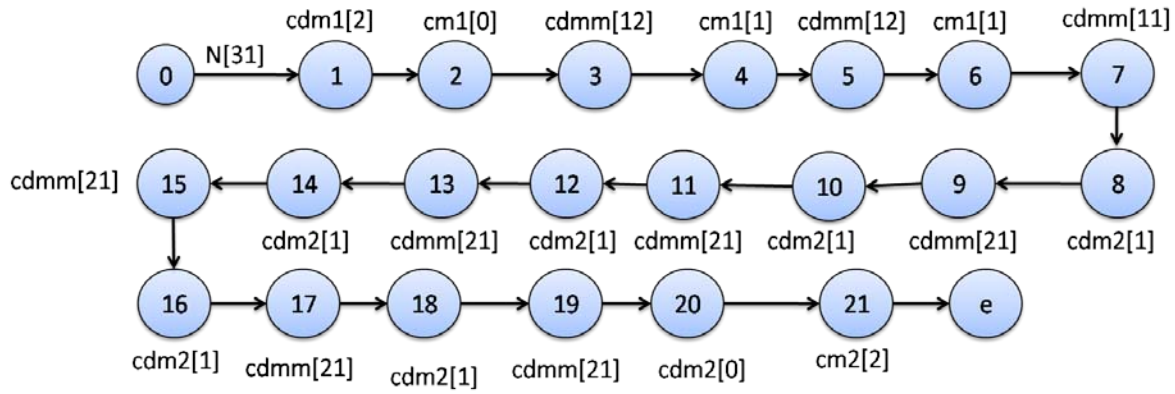
Annexe D
Implémentation
(Contrôleur séquentiel QDI pour un registre à décalage)

Schématique :

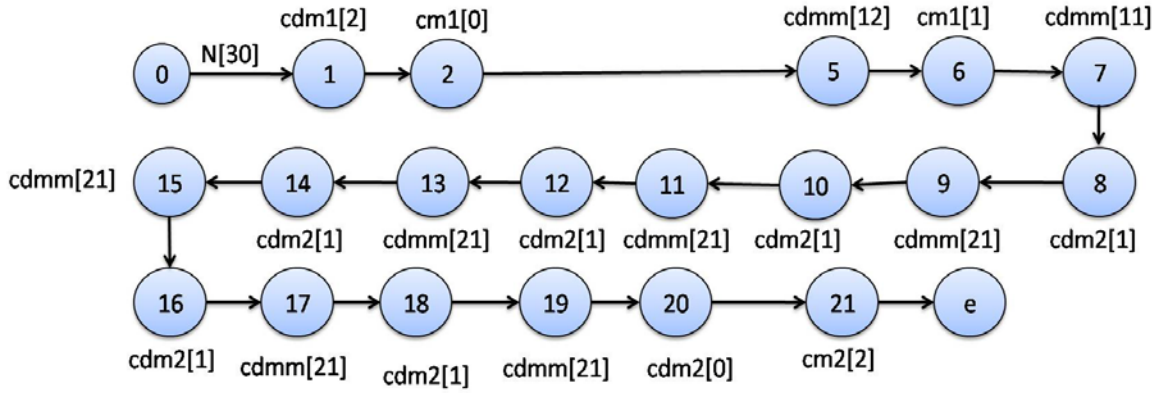


Description du contrôleur par Moore_MR_CPOG :

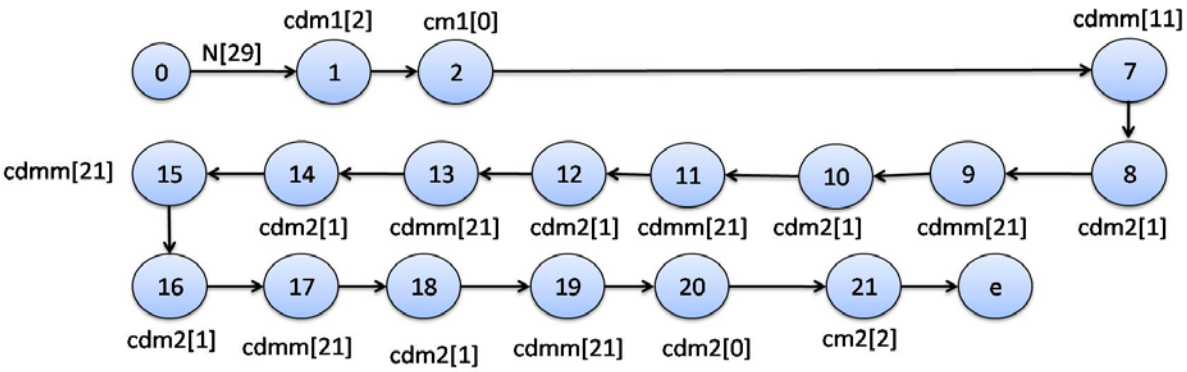
Séquence de contrôleurs correspondants à un décalage de 31 bits :



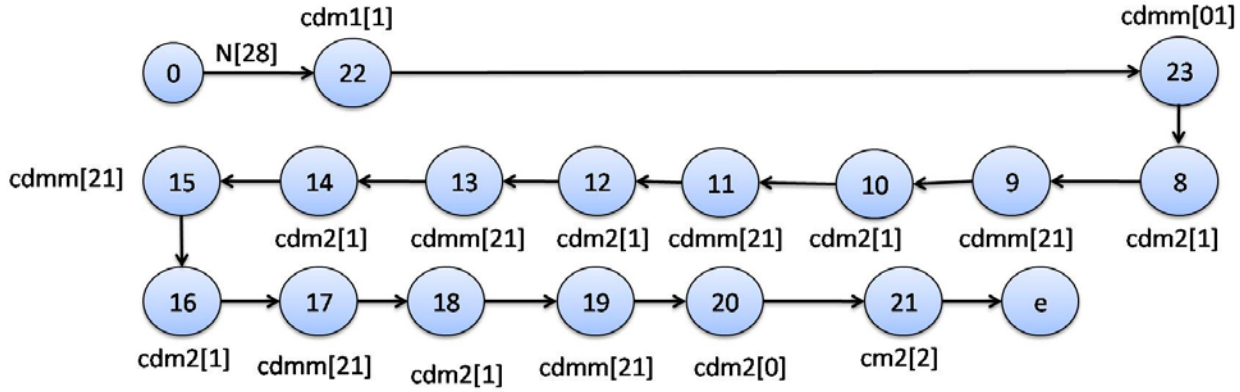
Séquence de contrôleurs correspondants à un décalage de 30 bits :



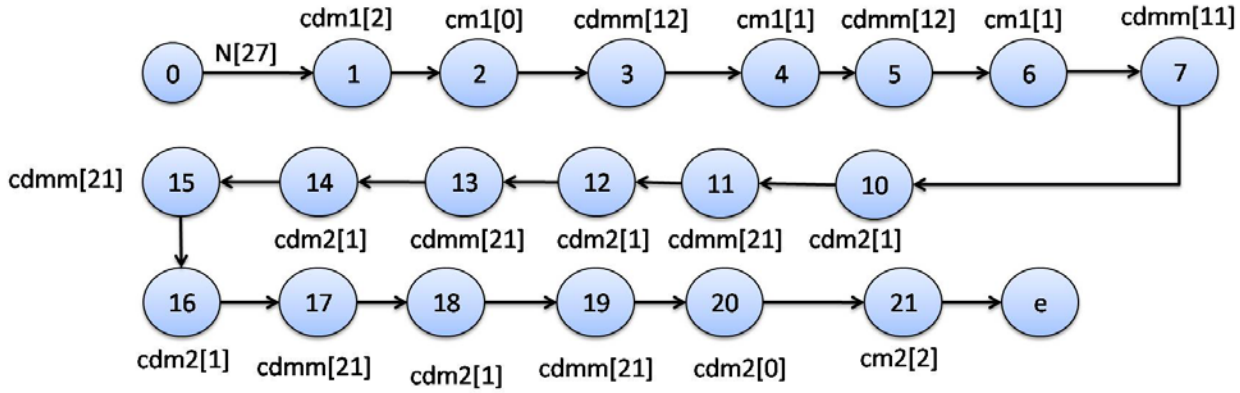
Séquence de contrôleurs correspondants à un décalage de 29 bits :



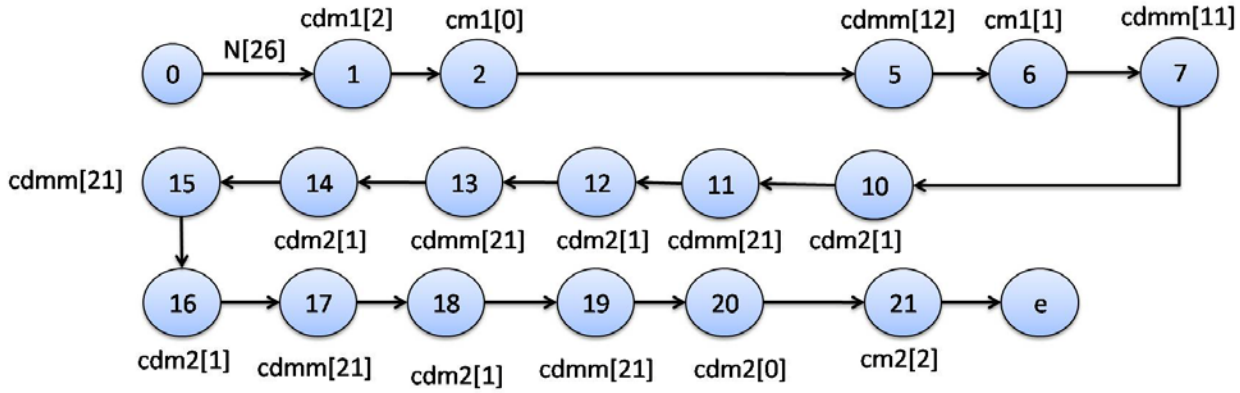
Séquence de contrôleurs correspondants à un décalage de 28 bits :



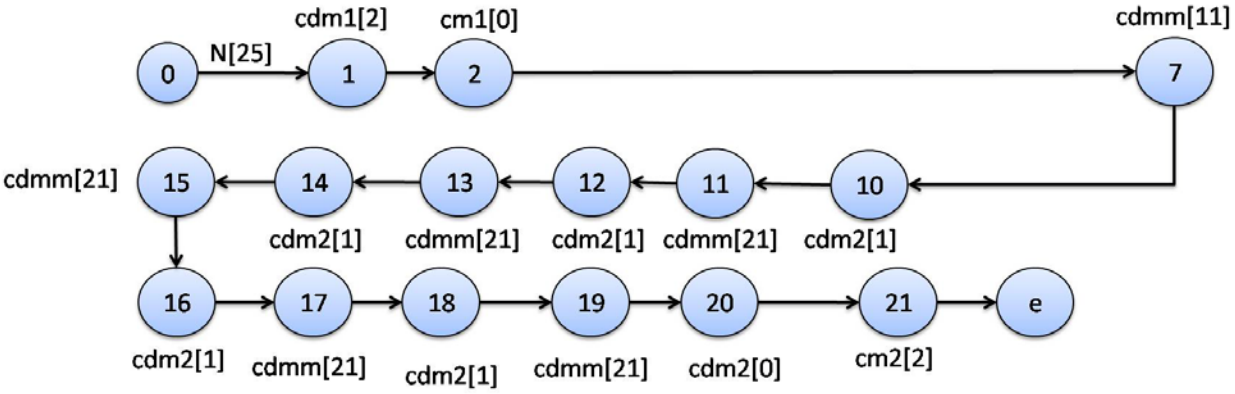
Séquence de contrôleurs correspondants à un décalage de 27 bits :



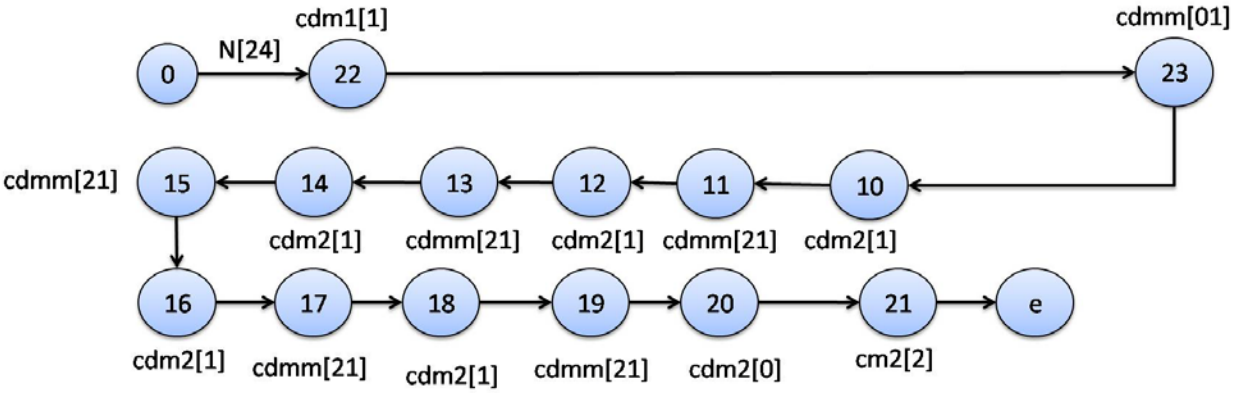
Séquence de contrôleurs correspondants à un décalage de 26 bits :



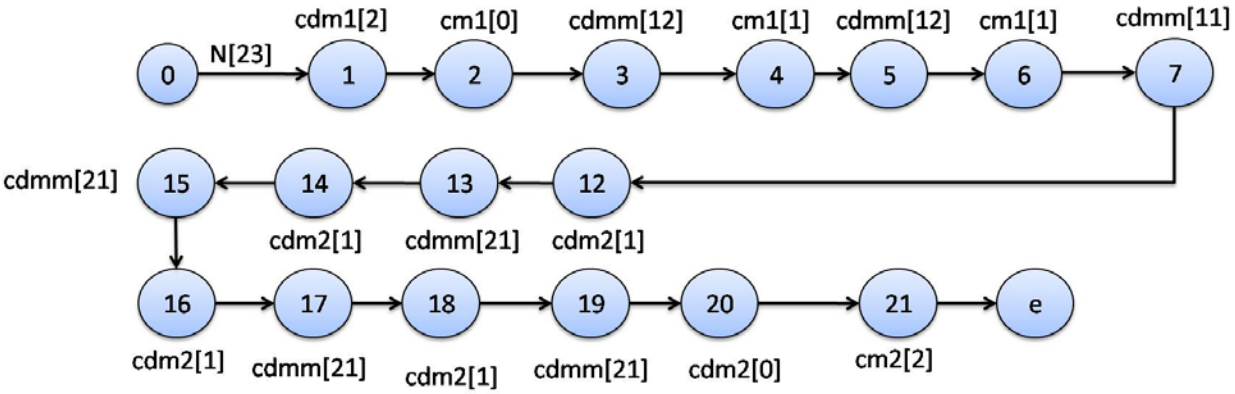
Séquence de contrôleurs correspondants à un décalage de 25 bits :



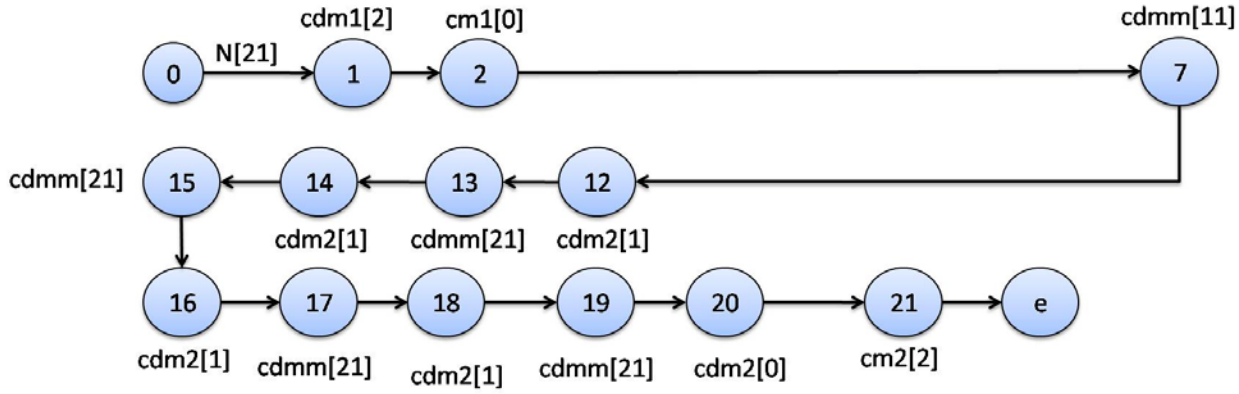
Séquence de contrôleurs correspondants à un décalage de 24 bits :



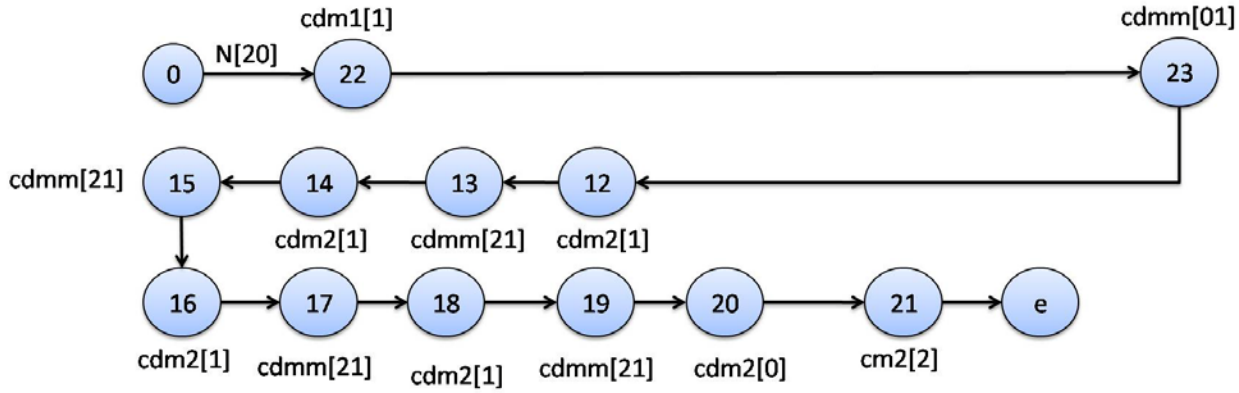
Séquence de contrôleurs correspondants à un décalage de 23 bits :



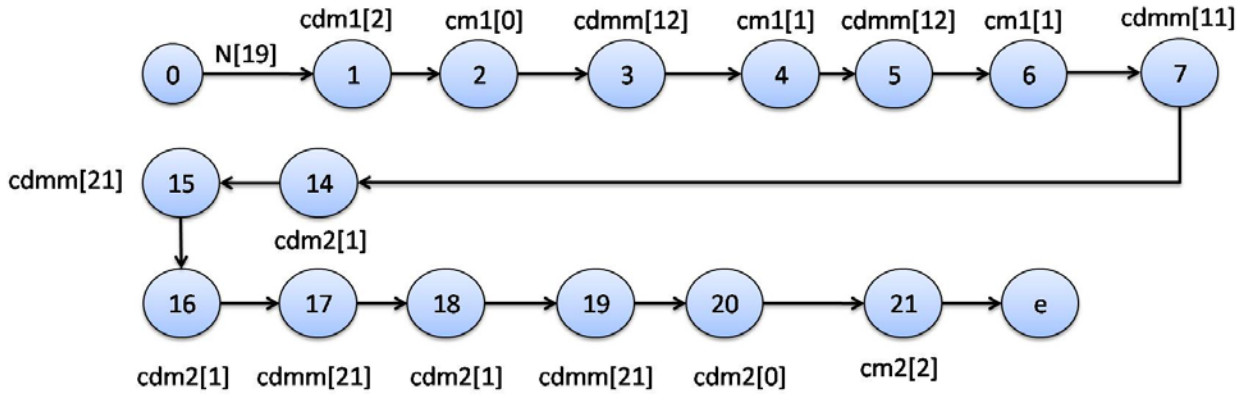
Séquence de contrôleurs correspondants à un décalage de 22 bits :



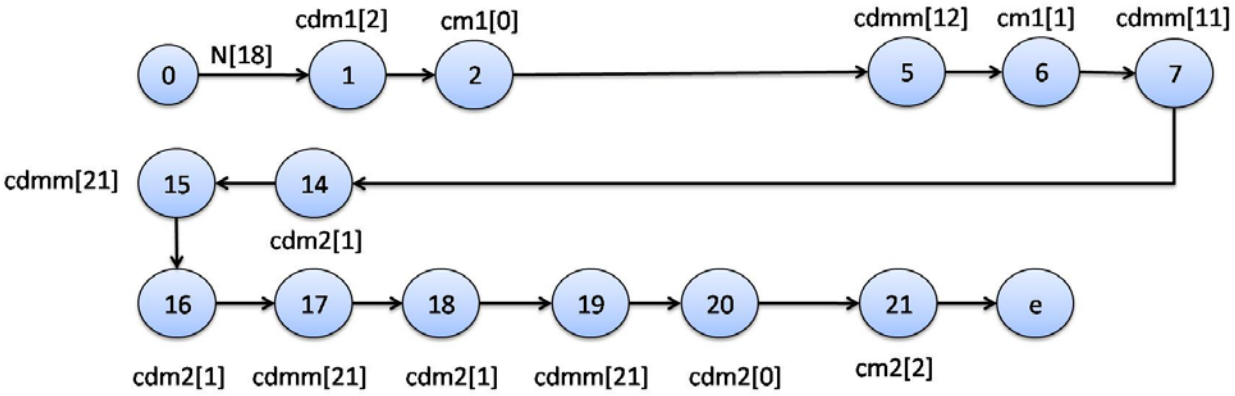
Séquence de contrôleurs correspondants à un décalage de 21 bits :



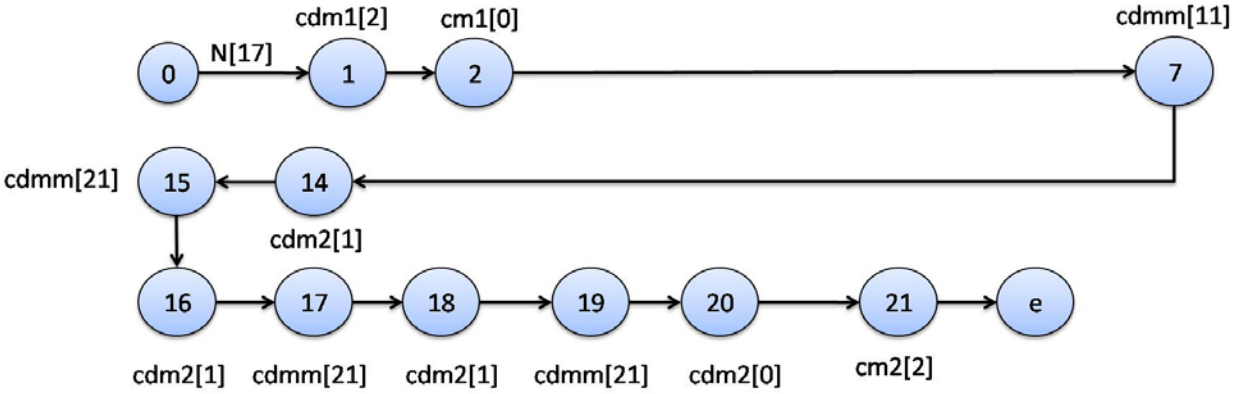
Séquence de contrôleurs correspondants à un décalage de 20 bits :



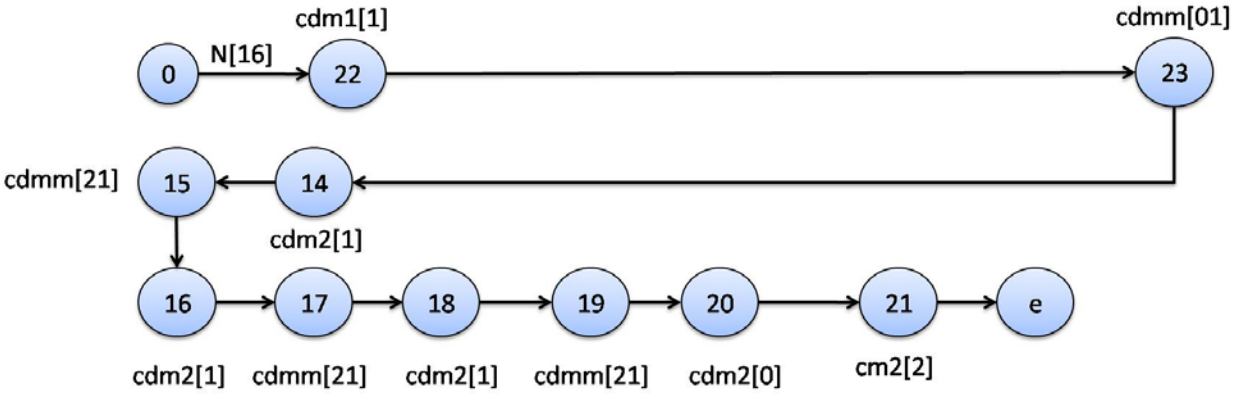
Séquence de contrôleurs correspondants à un décalage de 19 bits :



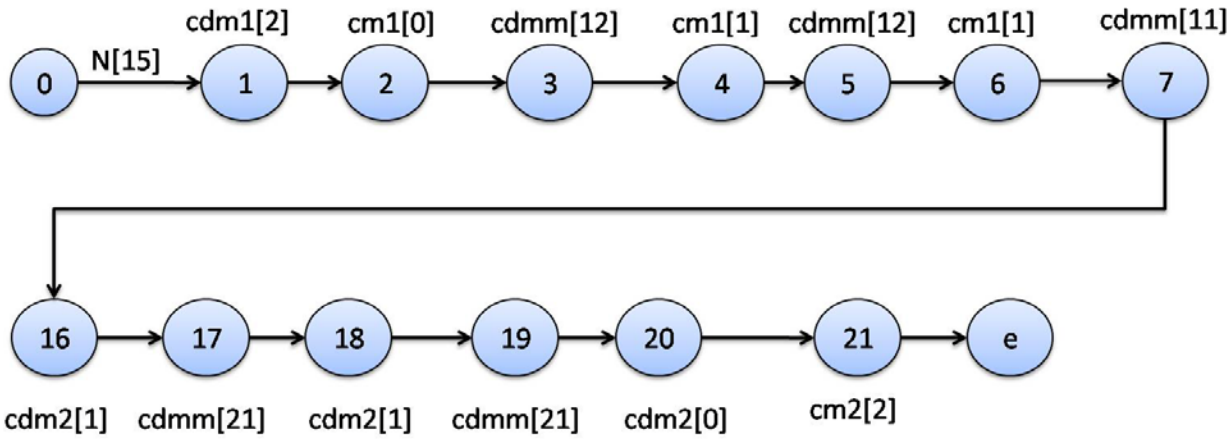
Séquence de contrôleurs correspondants à un décalage de 18 bits :



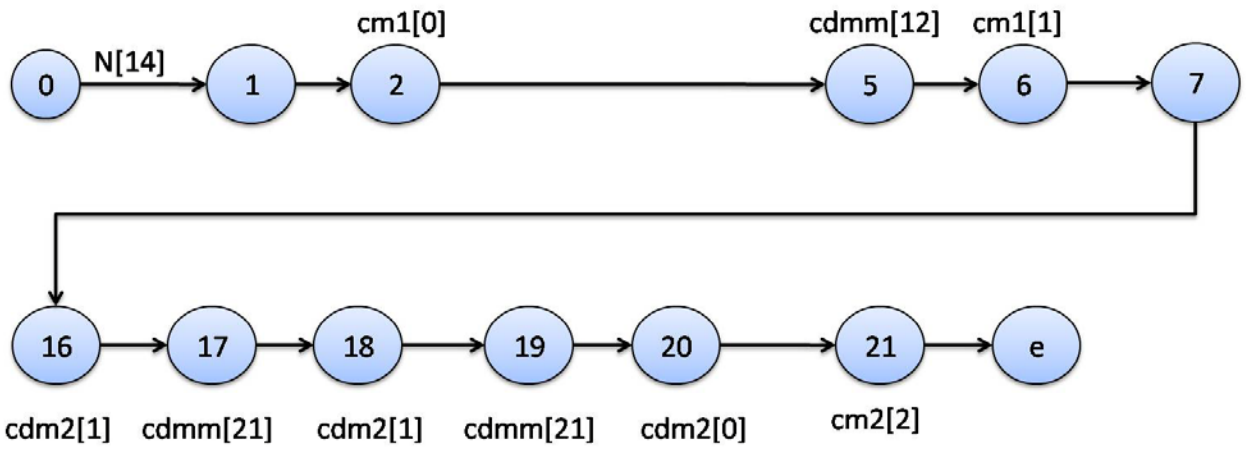
Séquence de contrôleurs correspondants à un décalage de 17 bits :



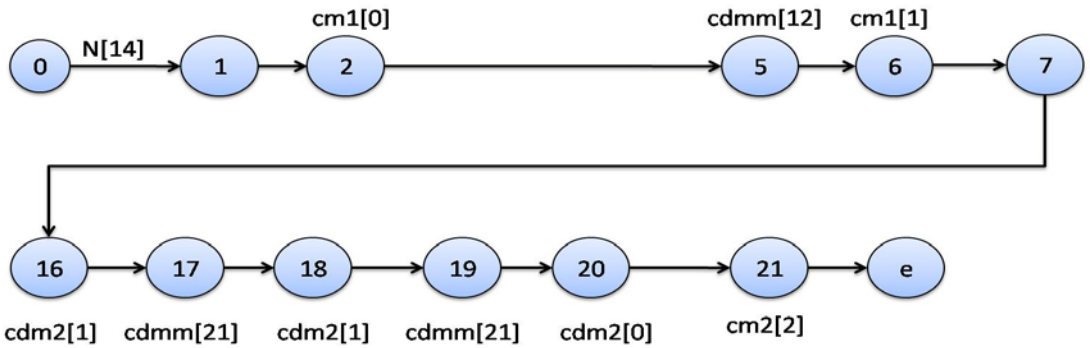
Séquence de contrôleurs correspondants à un décalage de 16 bits :



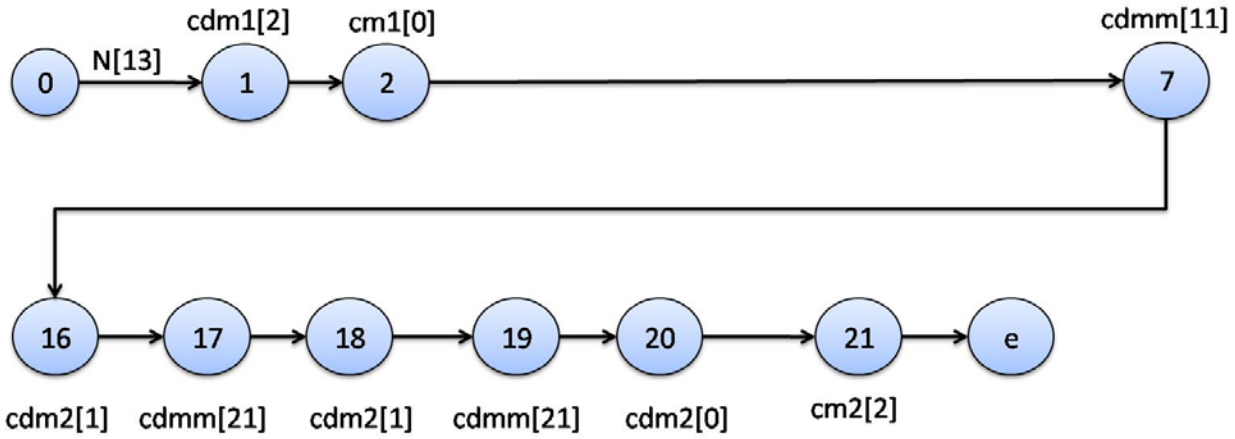
Séquence de contrôleurs correspondants à un décalage de 15 bits :



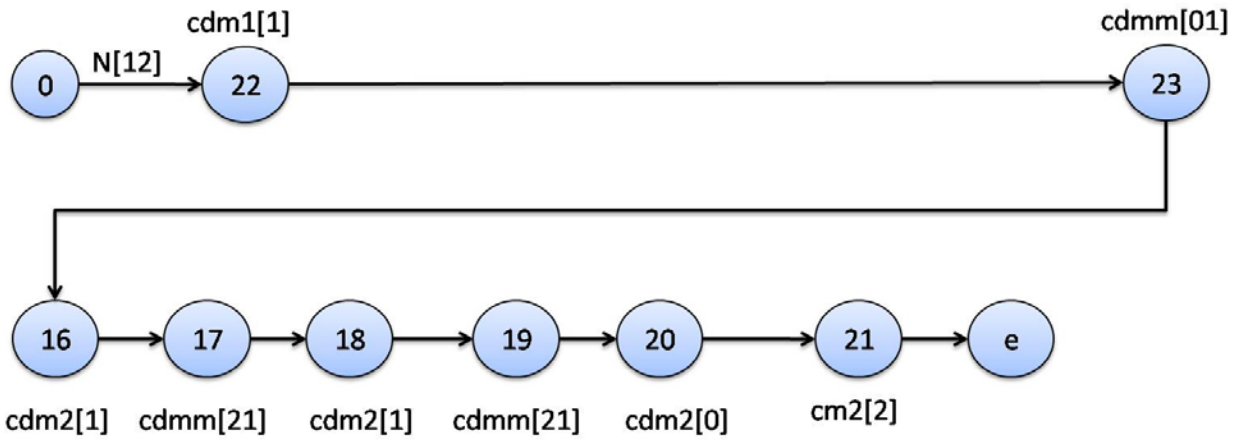
Séquence de contrôleurs correspondants à un décalage de 14 bits :



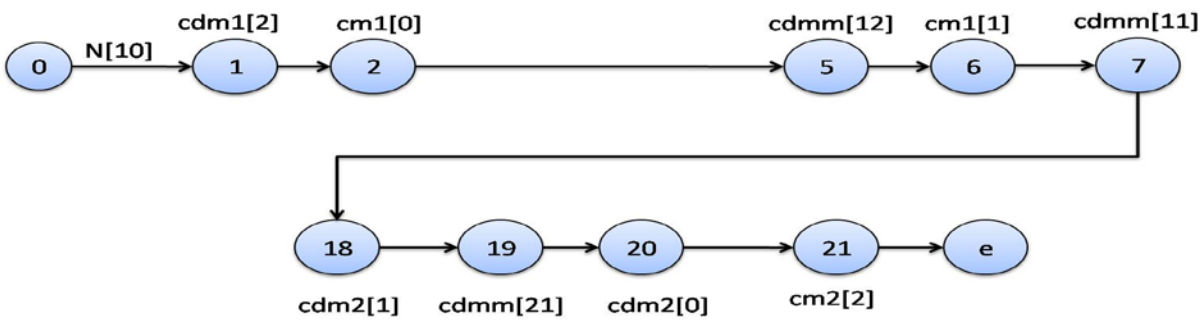
Séquence de contrôleurs correspondants à un décalage de 13 bits :



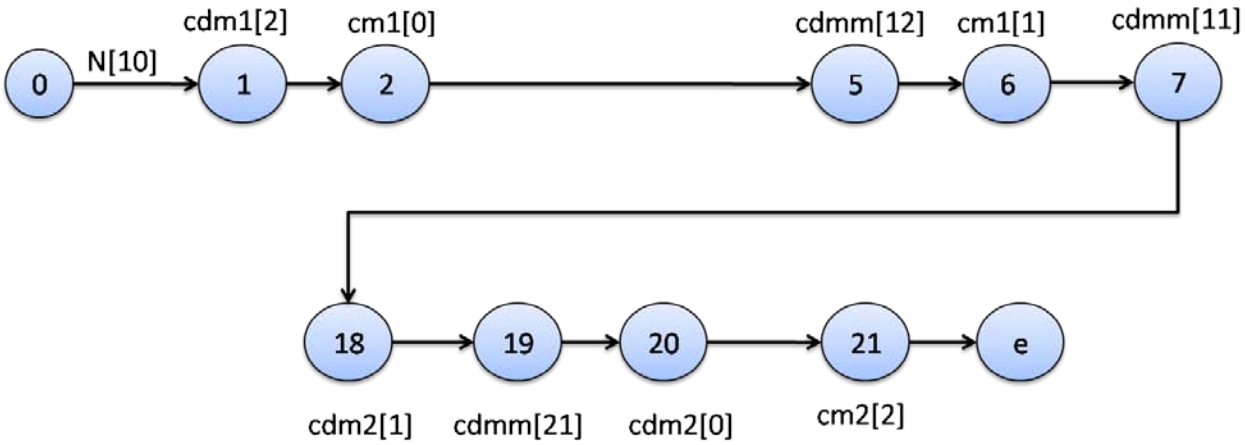
Séquence de contrôleurs correspondants à un décalage de 12 bits :



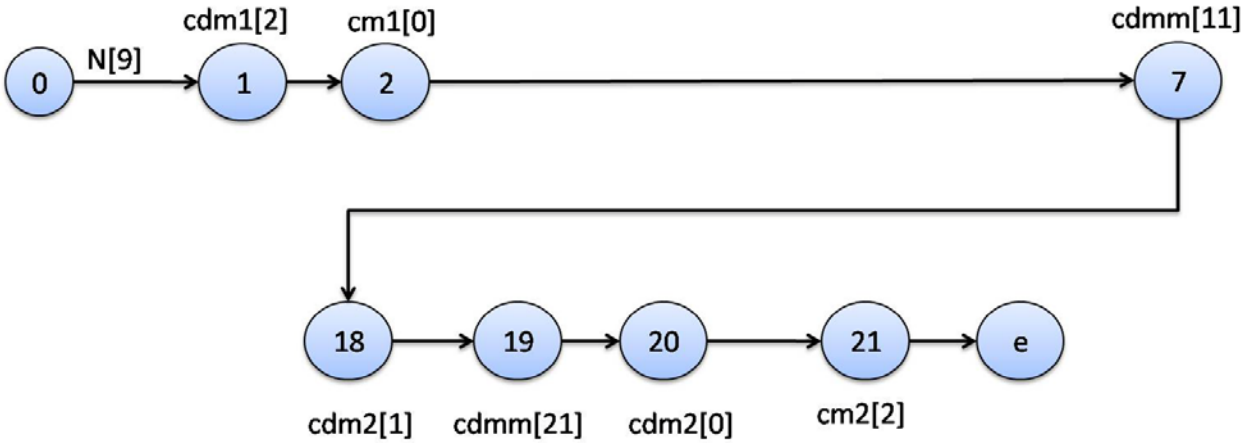
Séquence de contrôleurs correspondants à un décalage de 11 bits :



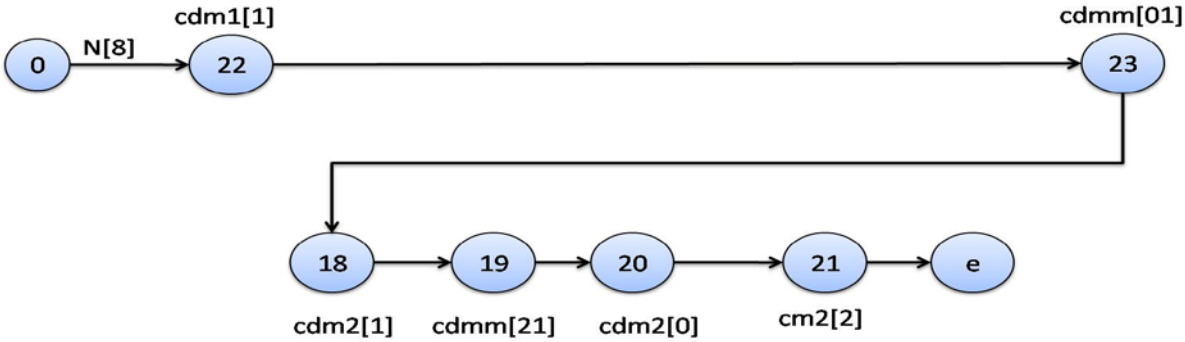
Séquence de contrôleurs correspondants à un décalage de 10 bits :



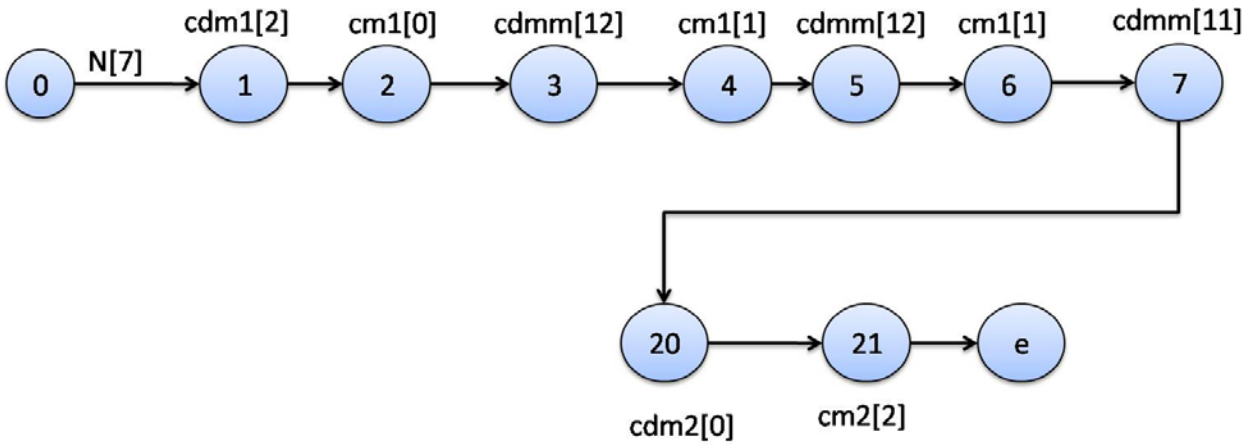
Séquence de contrôleurs correspondants à un décalage de 9 bits :



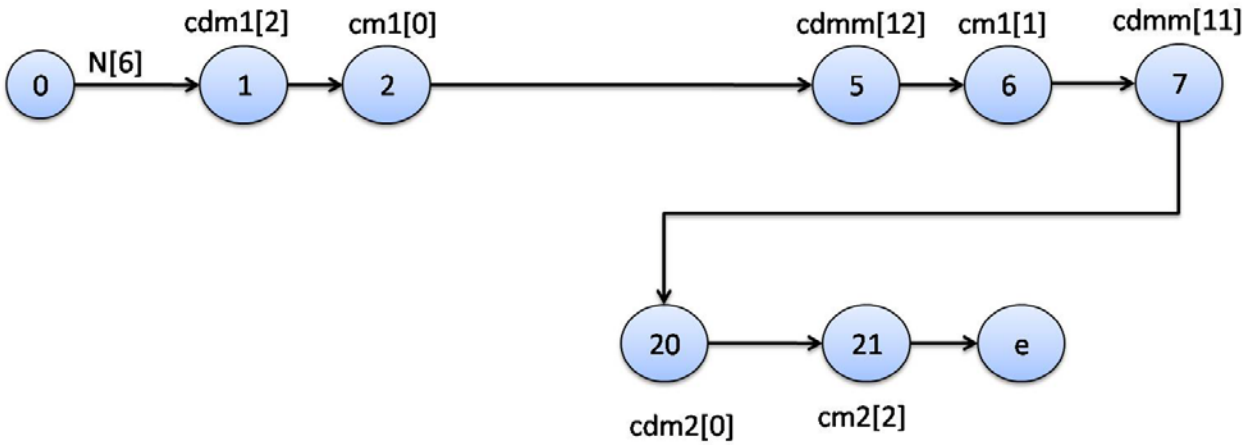
Séquence de contrôleurs correspondants à un décalage de 8 bits :



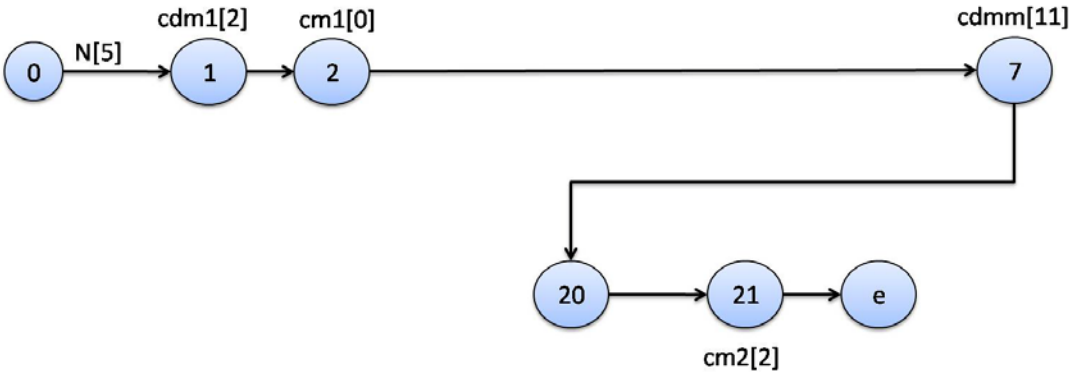
Séquence de contrôleurs correspondants à un décalage de 7 bits :



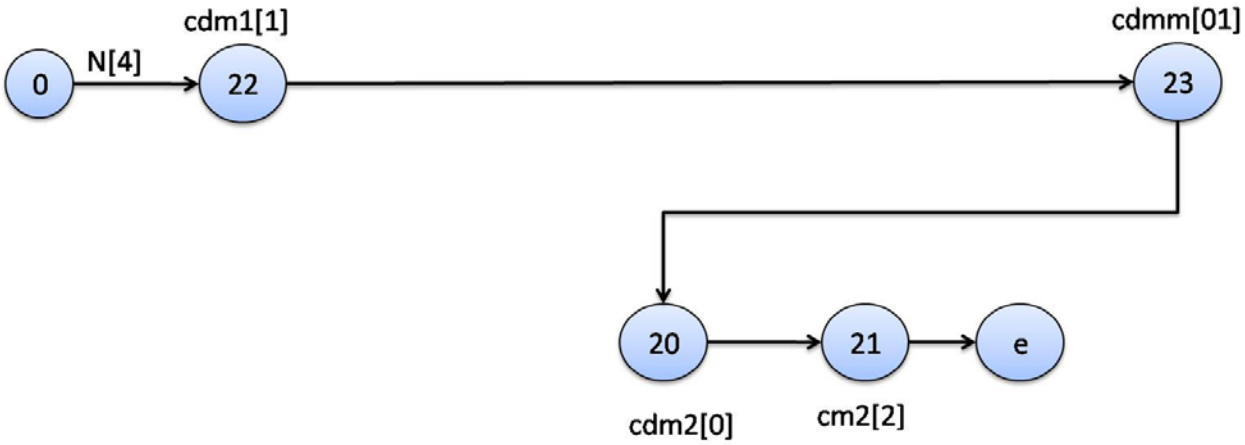
Séquence de contrôleurs correspondants à un décalage de 6 bits :



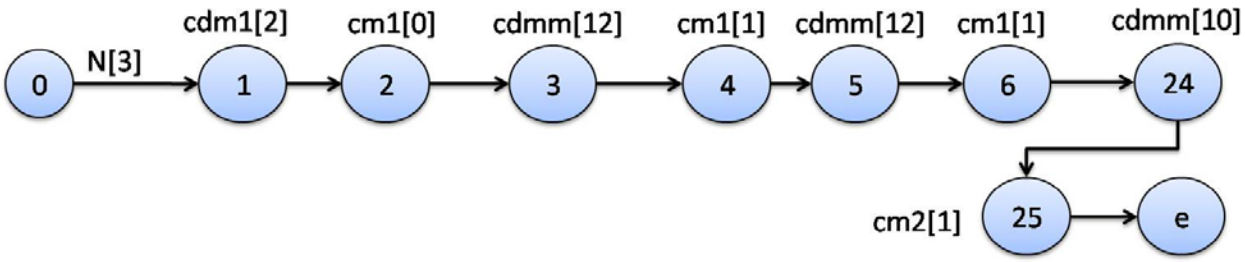
Séquence de contrôleurs correspondants à un décalage de 5 bits :



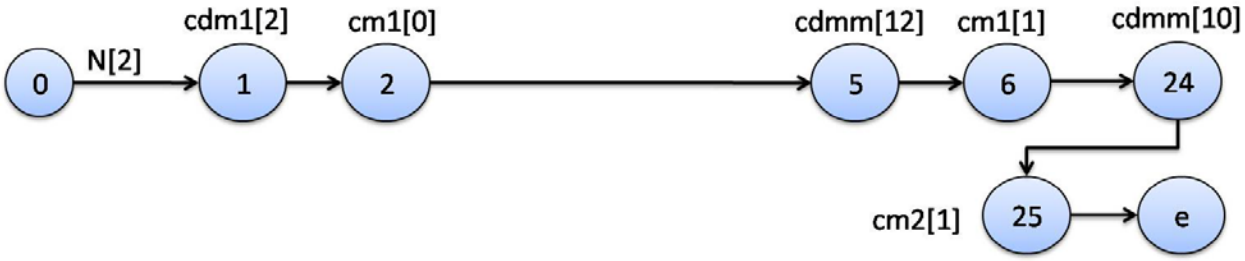
Séquence de contrôleurs correspondants à un décalage de 4 bits :



Séquence de contrôleurs correspondants à un décalage de 3 bits :

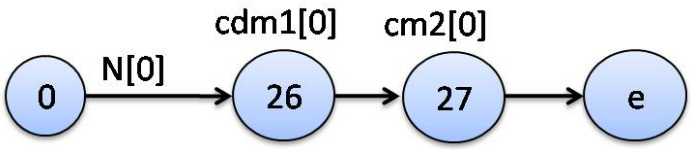


Séquence de contrôleurs correspondants à un décalage de 2 bits :



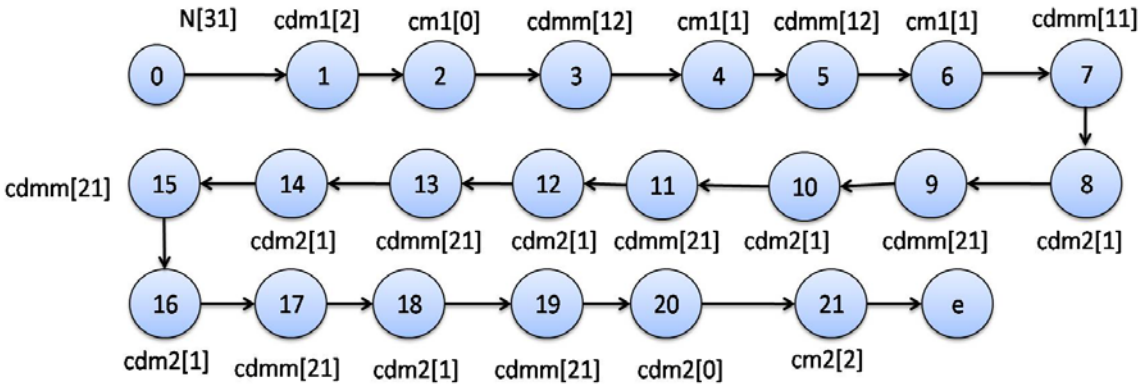
Séquence de contrôleurs correspondants à un décalage de 1 bits :

Séquence de contrôleurs correspondants à un décalage de 0 bits :

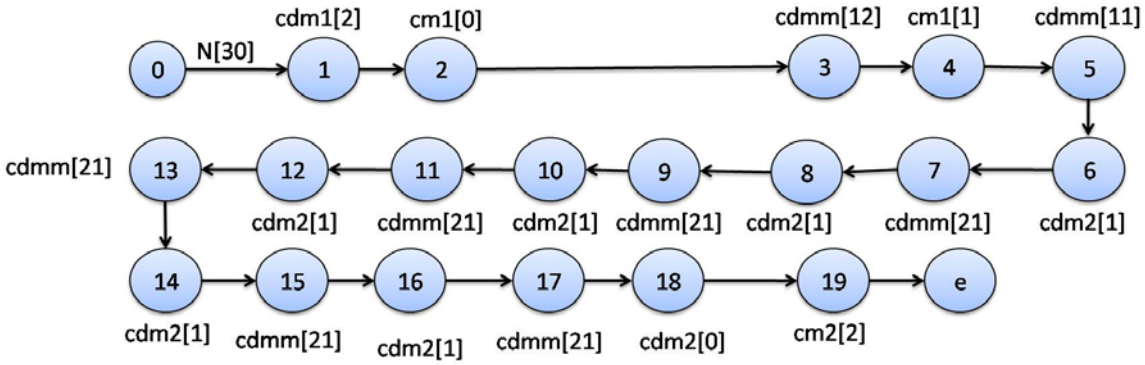


Description du contrôleur par Mealy_MR_CPOG :

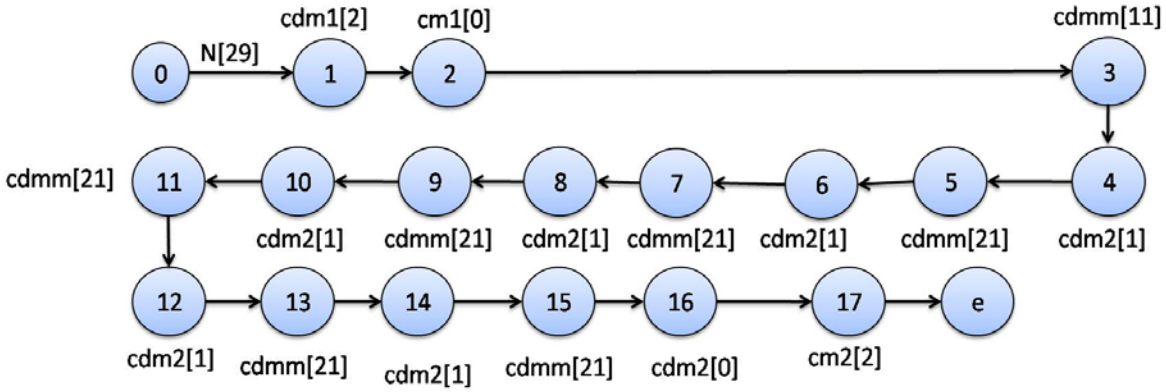
Séquence Mealy de contrôleurs correspondants à un décalage de 31 bits :



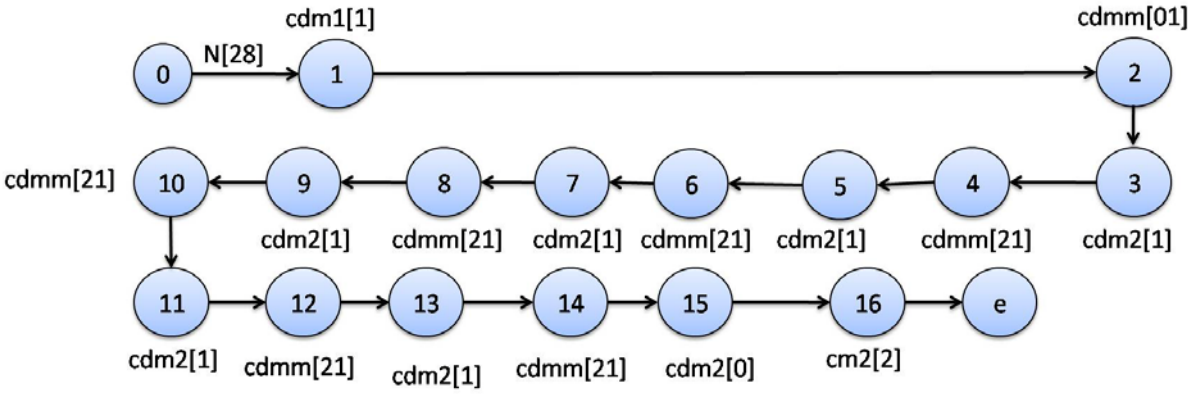
Séquence Mealy de contrôleurs correspondants à un décalage de 30 bits :



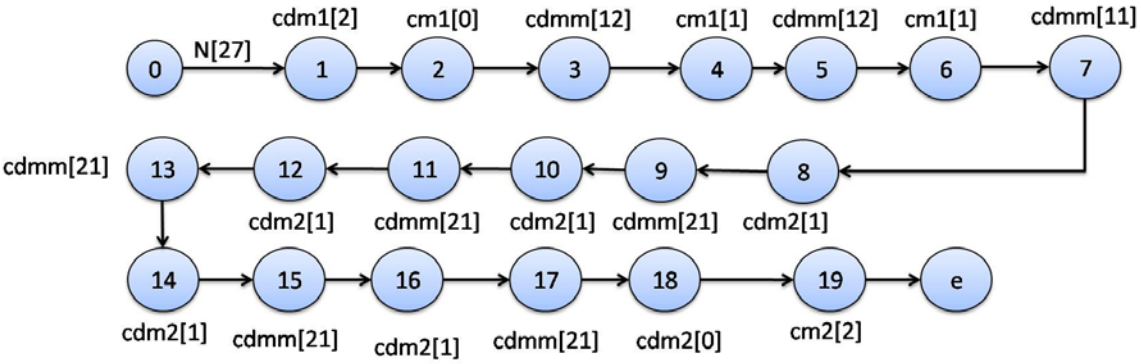
Séquence Mealy de contrôleurs correspondants à un décalage de 29 bits :



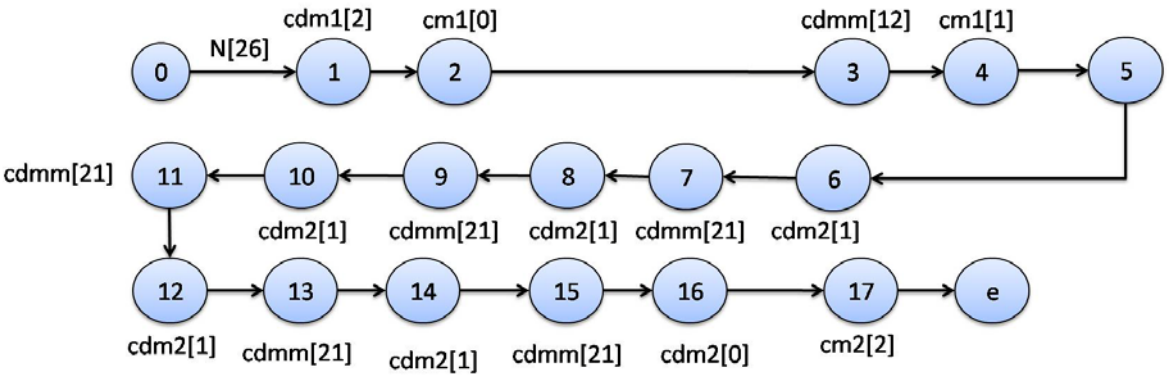
Séquence Mealy de contrôleurs correspondants à un décalage de 28 bits :



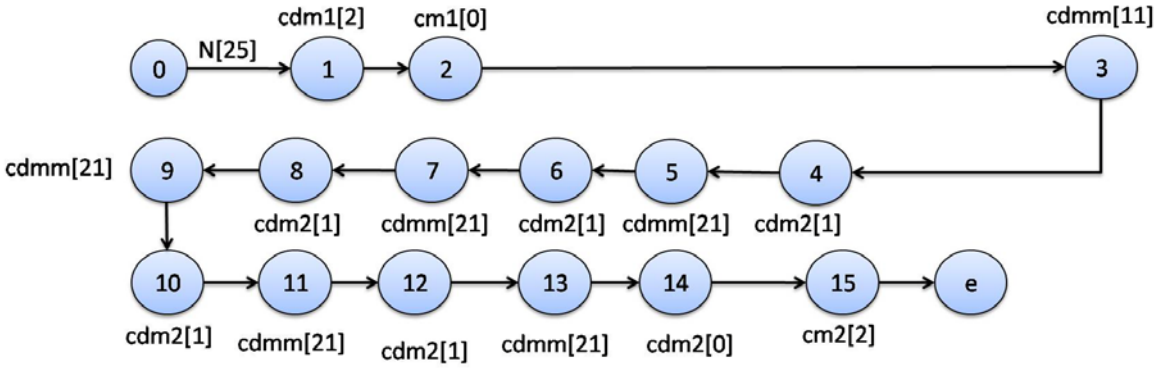
Séquence Mealy de contrôleurs correspondants à un décalage de 27 bits :



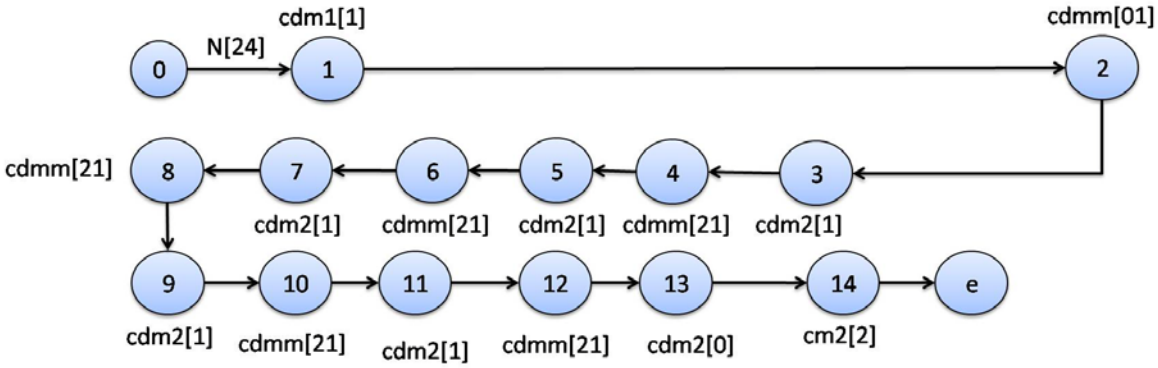
Séquence Mealy de contrôleurs correspondants à un décalage de 26 bits :



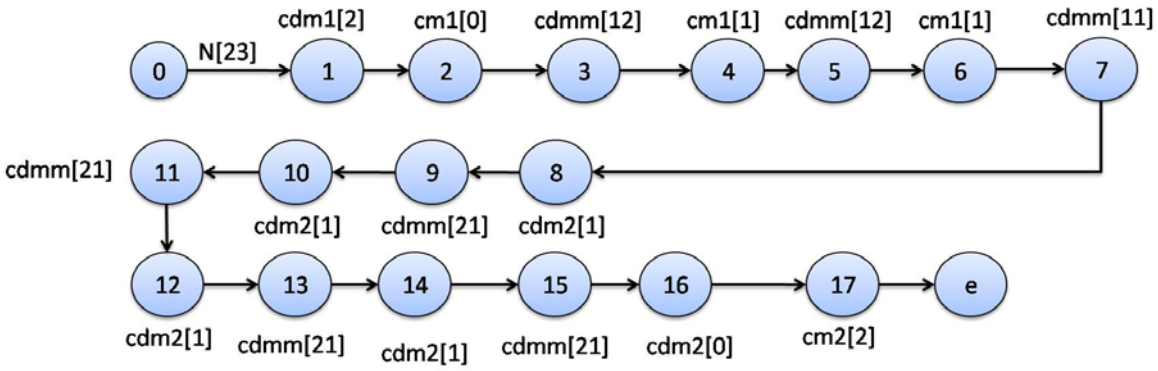
Séquence Mealy de contrôleurs correspondants à un décalage de 25 bits :



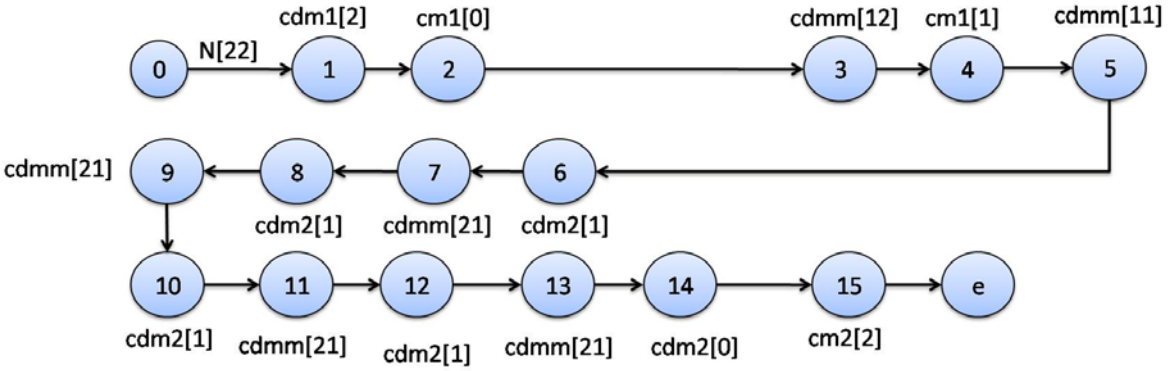
Séquence Mealy de contrôleurs correspondants à un décalage de 24 bits :



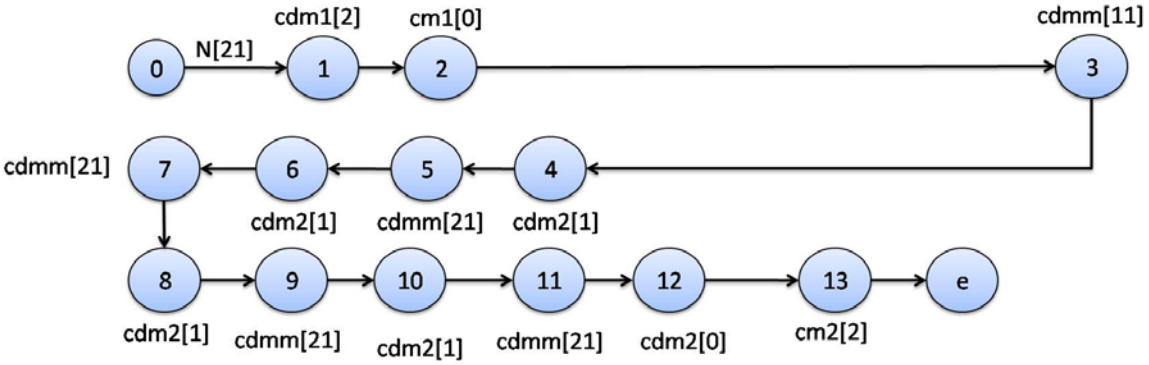
Séquence Mealy de contrôleurs correspondants à un décalage de 23 bits :



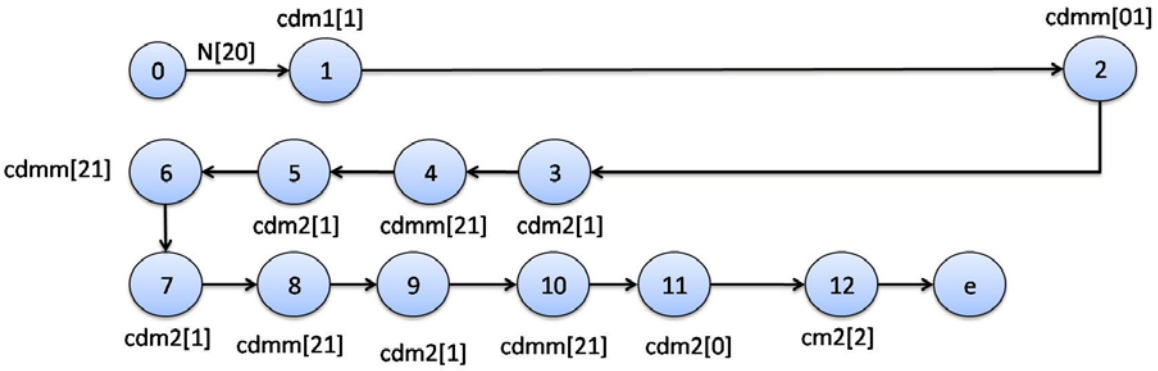
Séquence Mealy de contrôleurs correspondants à un décalage de 22 bits :



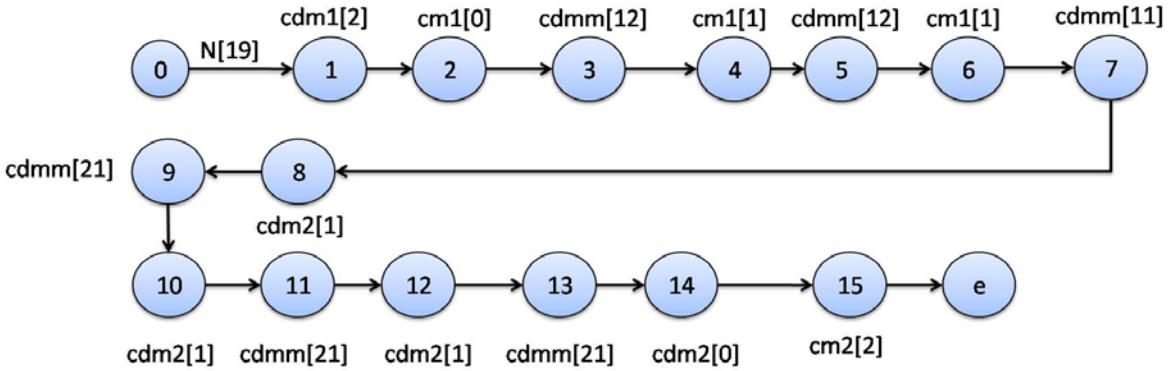
Séquence Mealy de contrôleurs correspondants à un décalage de 21 bits :



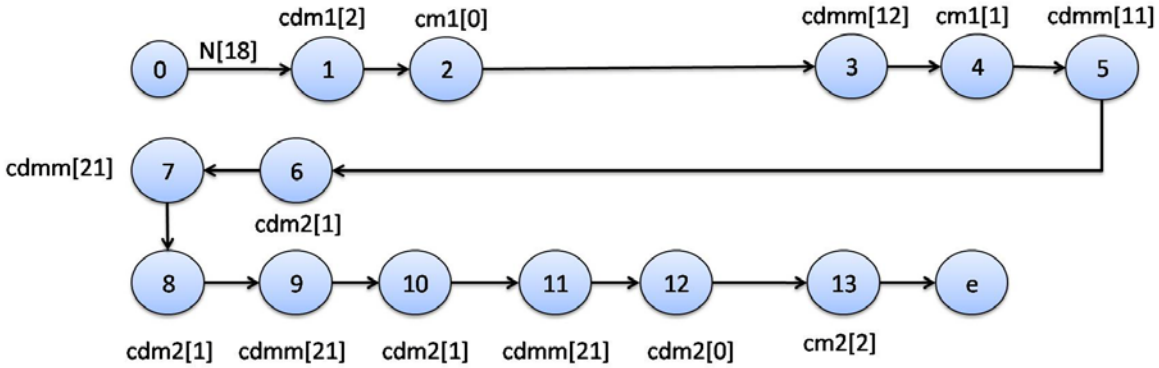
Séquence Mealy de contrôleurs correspondants à un décalage de 20 bits :



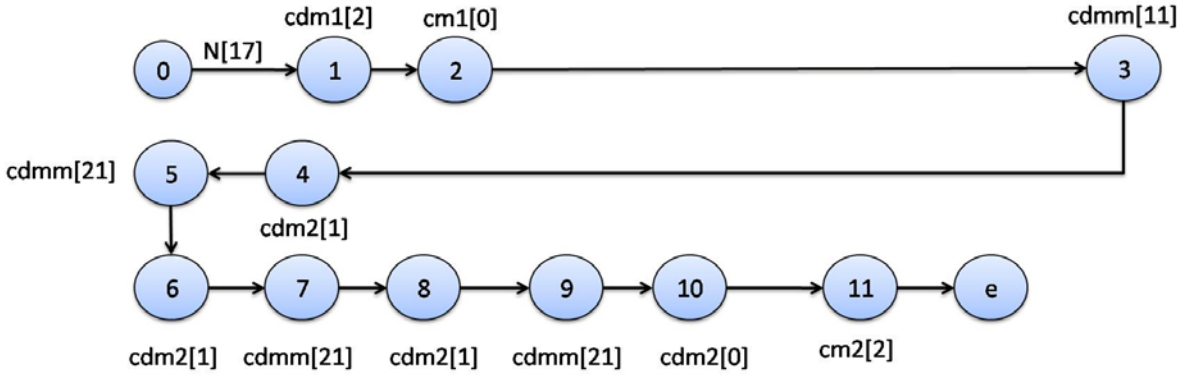
Séquence Mealy de contrôleurs correspondants à un décalage de 19 bits :



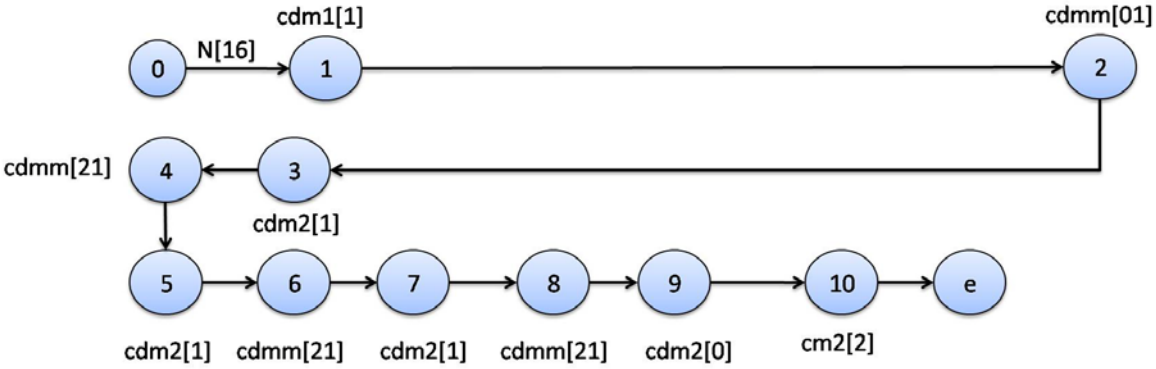
Séquence Mealy de contrôleurs correspondants à un décalage de 18 bits :



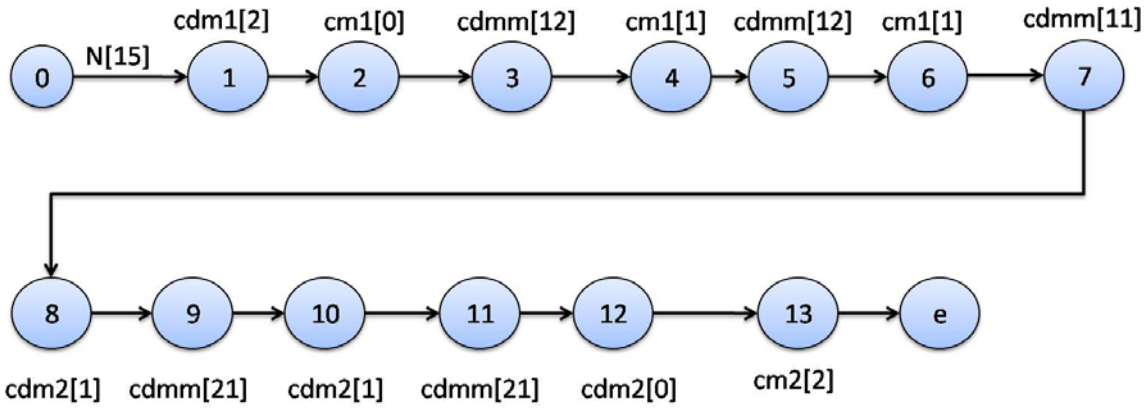
Séquence Mealy de contrôleurs correspondants à un décalage de 17 bits :



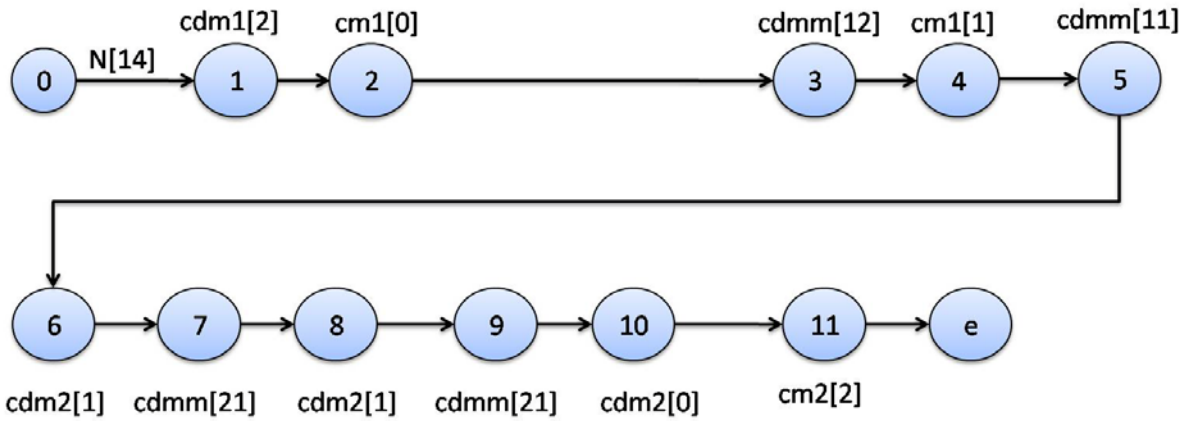
Séquence Mealy de contrôleurs correspondants à un décalage de 16 bits :



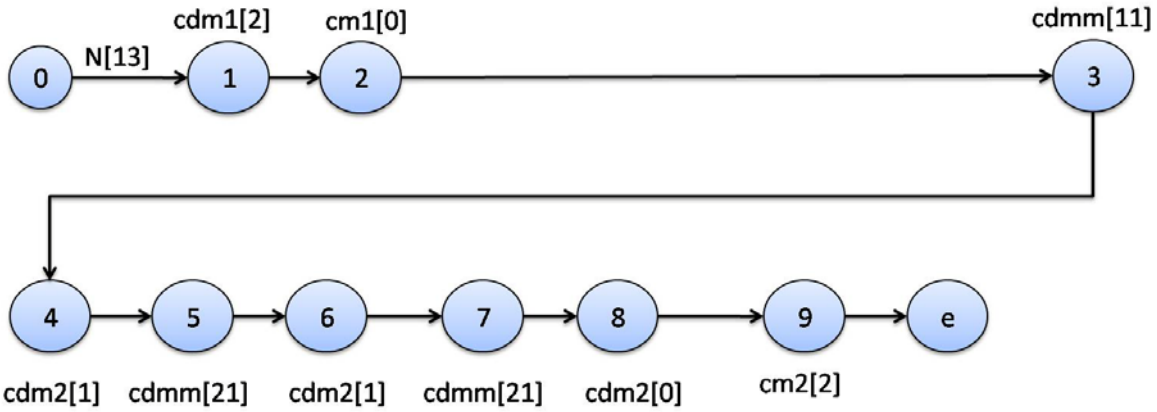
Séquence Mealy de contrôleurs correspondants à un décalage de 15 bits :



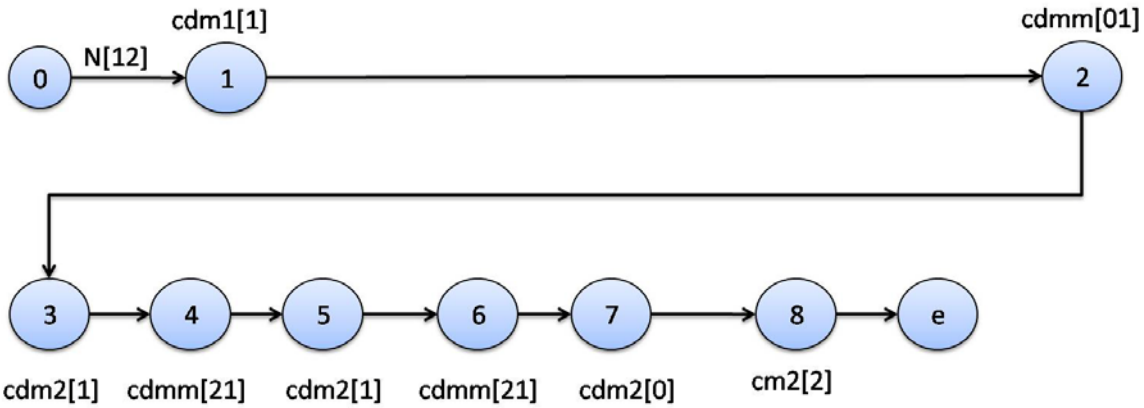
Séquence Mealy de contrôleurs correspondants à un décalage de 14 bits :



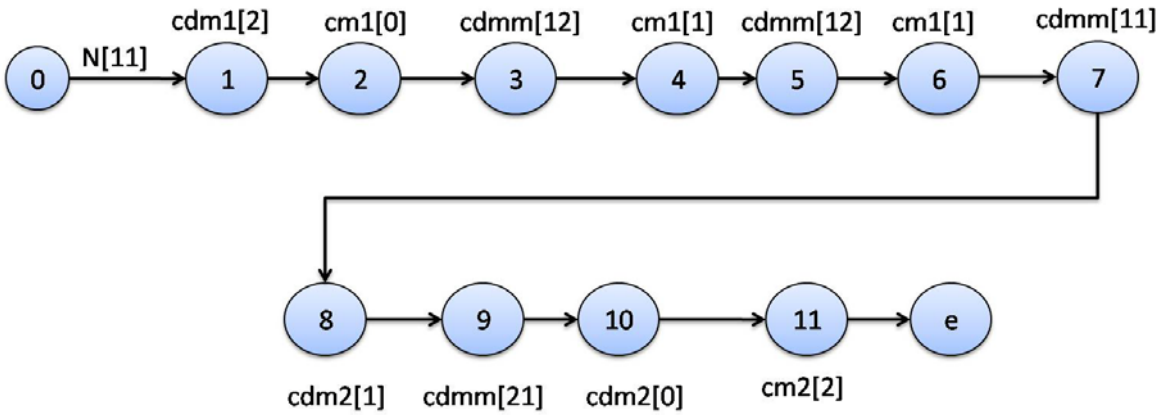
Séquence Mealy de contrôleurs correspondants à un décalage de 13 bits :



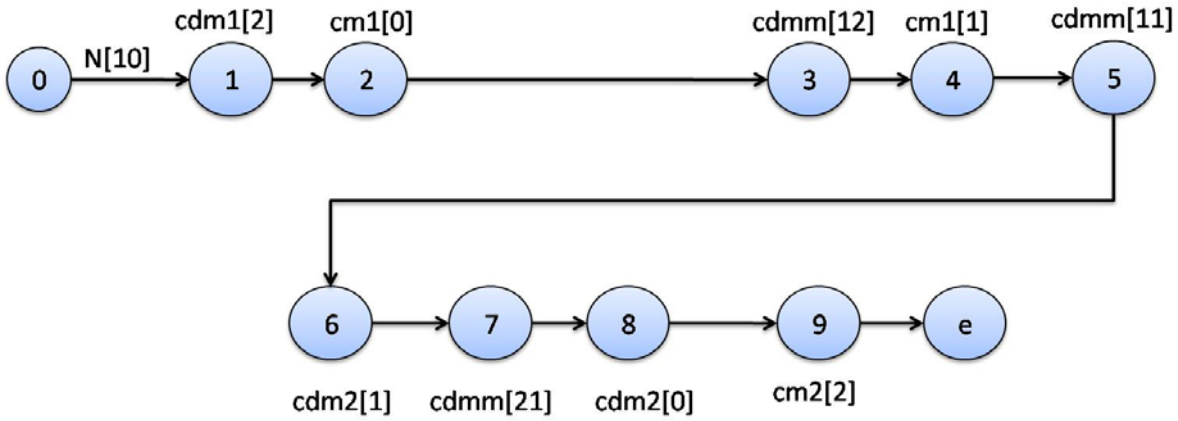
Séquence Mealy de contrôleurs correspondants à un décalage de 12 bits :



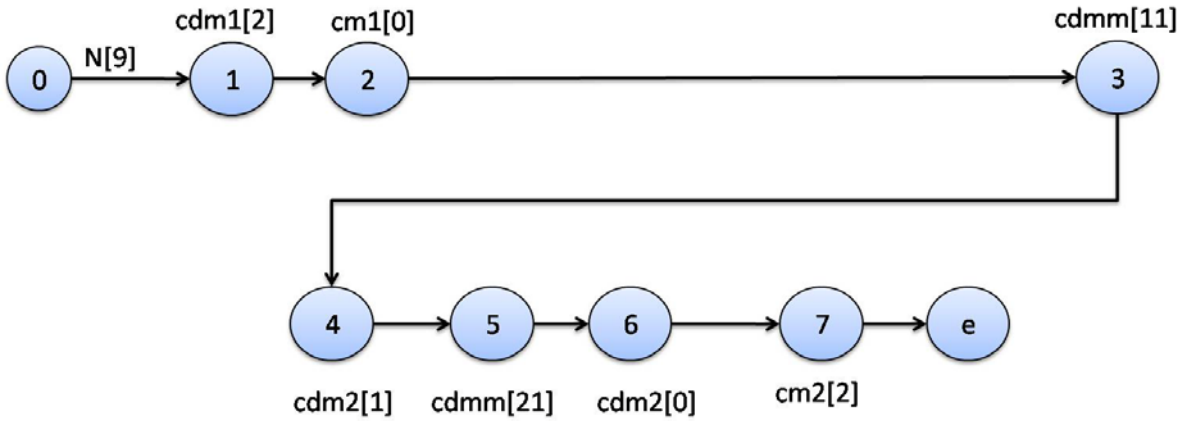
Séquence Mealy de contrôleurs correspondants à un décalage de 11 bits :



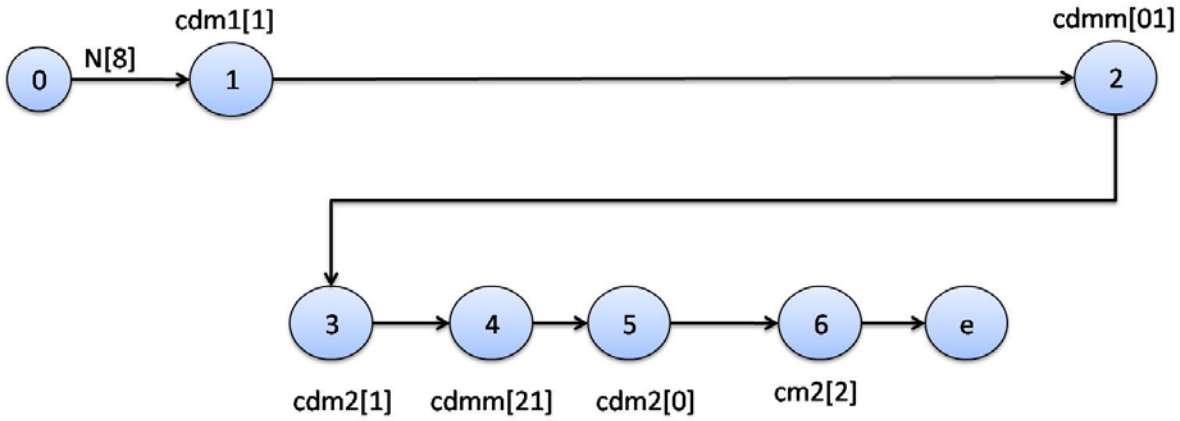
Séquence Mealy de contrôleurs correspondants à un décalage de 10 bits :



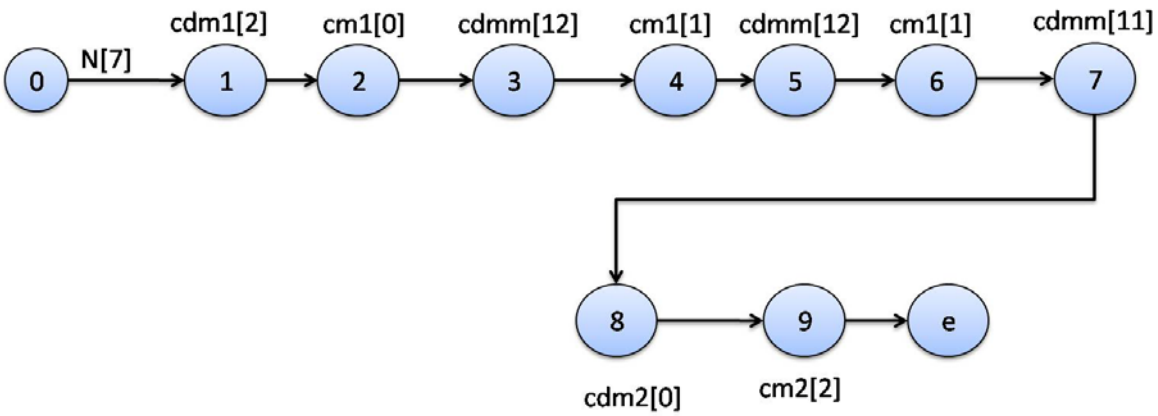
Séquence Mealy de contrôleurs correspondants à un décalage de 9 bits :



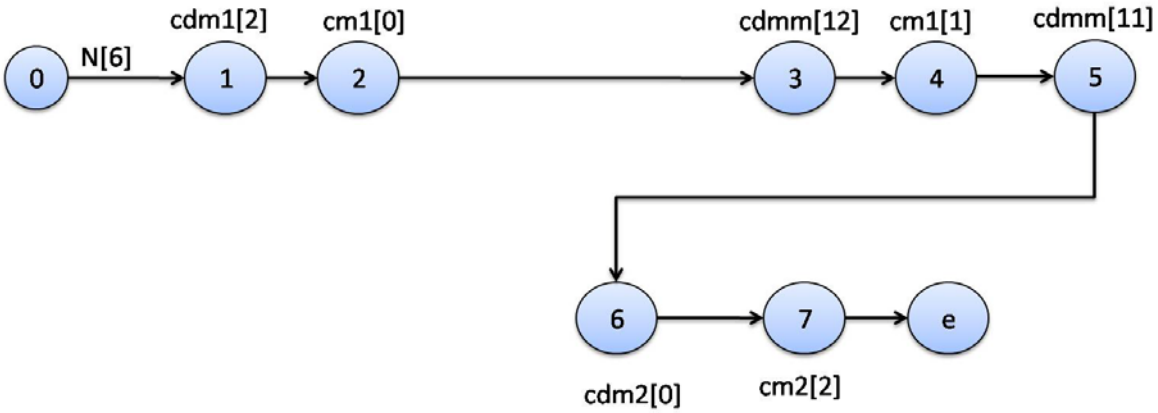
Séquence Mealy de contrôleurs correspondants à un décalage de 8 bits :



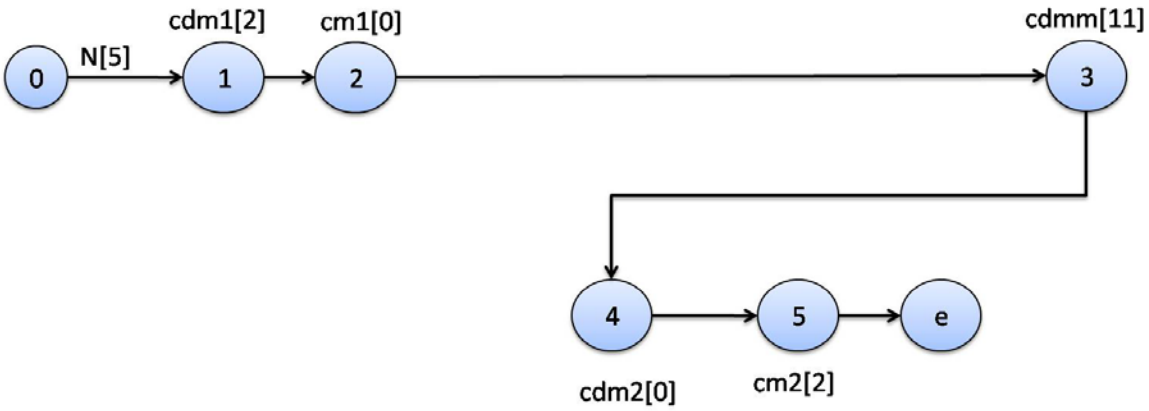
Séquence Mealy de contrôleurs correspondants à un décalage de 7 bits :



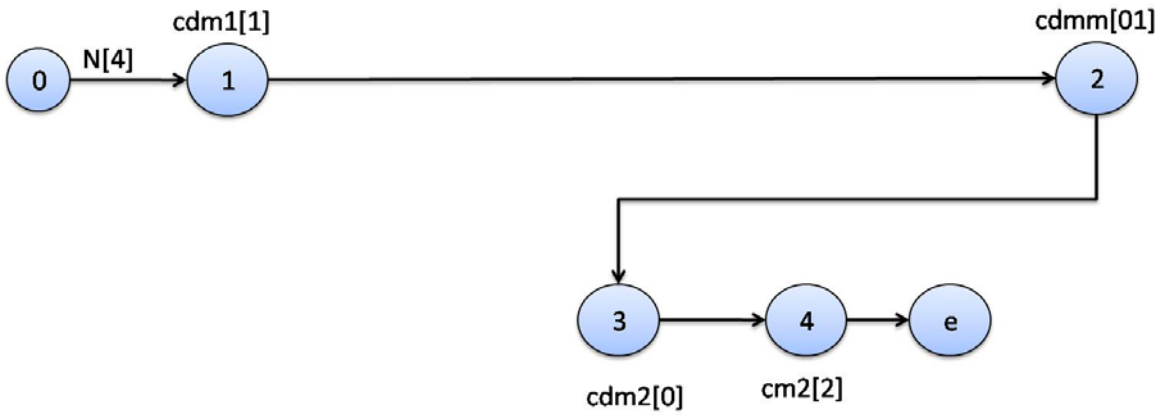
Séquence Mealy de contrôleurs correspondants à un décalage de 6 bits :



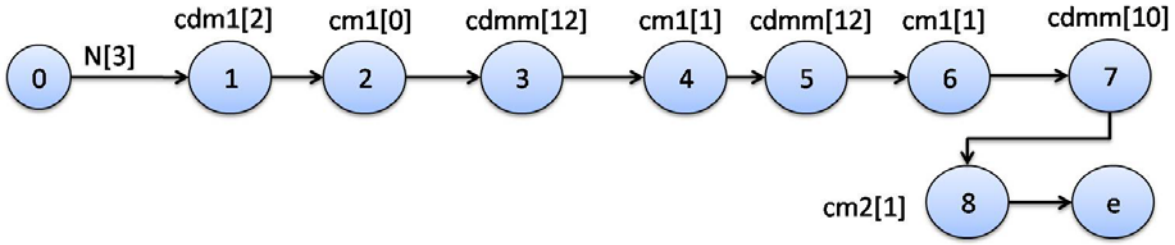
Séquence Mealy de contrôleurs correspondants à un décalage de 5 bits :



Séquence Mealy de contrôleurs correspondants à un décalage de 4 bits :



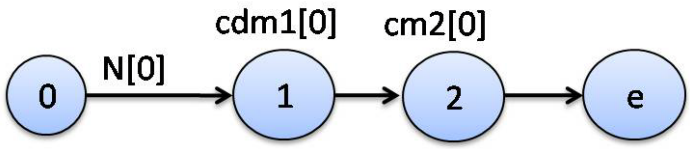
Séquence Mealy de contrôleurs correspondants à un décalage de 3 bits :



Séquence Mealy de contrôleurs correspondants à un décalage de 2 bits :

Séquence Mealy de contrôleurs correspondants à un décalage de 1 bit :

Séquence Mealy de contrôleurs correspondants à un décalage de 0 bit :


cdm1[0]
cm2[0]
N[0]
0
1
2
e

RESUME
L'étude des circuits asynchrones est un secteur dans lequel de nombreuses recherches ont été effectuées ces dernières années. Les circuits asynchrones ont démontré plusieurs caractéristiques intéressantes comme la robustesse, l'extensibilité, la faible consommation ou le faible rayonnement électromagnétique. Parmi les différentes classes de circuits asynchrones, les circuits quasi-insensibles aux délais (QDI) ont montré des caractéristiques extrêmement intéressantes en termes de faible consommation et de robustesse aux variations PVT (Process, Voltage, Temperature). L'usage de ces circuits est notamment bien adapté aux applications fonctionnant dans un environnement sévère et pour lesquelles la consommation est un critère primordial. Les travaux de cette thèse s'inscrivent dans ce cadre et visent la conception et la synthèse de machines à états asynchrones (QDI) faiblement consommantes. Une méthode de synthèse dédiée à des contrôleurs asynchrones à faible consommation a donc été développée. Cette technique s'est montrée particulièrement efficace pour synthétiser les contrôleurs de grande taille. La méthode s'appuie sur une modélisation appropriée des contrôleurs et une technique de synthèse dirigée par la syntaxe utilisant des composants spécifiques appelés séquenceurs. Les circuits obtenus ont été vérifiés formellement afin de s'assurer de leurs propriétés en termes de robustesse et de correction fonctionnelle. A cette occasion, une méthode de vérification formelle a été mise en place pour valider les contrôleurs d'une part, et plus généralement, n'importe quel circuit asynchrone d'autre part. Cette technique fait appel à une modélisation hiérarchique des circuits asynchrones en PSL et à un outil de vérification formelle (RAT).

MOTS-CLES
Circuits asynchrones, machines à états, synthèse de contrôleurs, flot de conception, circuits quasi insensibles aux délais, vérification formelle, conception faible consommation, bibliothèque des composants, cellules standards, outils CAO, théorie des graphes, algorithmes d'optimisation, projection technologique, logique temporelle, séquenceurs.

TITLE

SYNTHESIS OF LOW POWER AND CORRECT PROVEN QDI ASYNCHRONOUS SEQUENTIAL CONTROLLERS

ABSTRACT
The study of asynchronous circuits is an area where much research has been conducted in recent years. Asynchronous circuits have shown several interesting features like robustness, scalability, low consumption or low electromagnetic radiation. Among the different classes of asynchronous circuits, Quasi Delay Insensitive circuits (QDI) showed very interesting characteristics in terms of low power consumption and robustness to variations of PVT (Process, Voltage, and Temperature). The use of these circuits is particularly well suited for applications operating in a critical environment and for which consumption is paramount. In this framework, the work of this thesis aims the low power consumption design and synthesis of asynchronous state machines (QDI). A method for synthesizing low-consumption asynchronous sequential controllers has been developed. The method relies on an adequate modeling of controllers and a direct mapping synthesis technique using specific components called sequencers. This technique is suitable for synthesizing large controllers. The circuits obtained are formally verified to ensure their properties in terms of robustness and are proved functionally correct. Thereby, a formal verification method has been implemented to validate the sequential controllers on the one hand, and more generally, any other asynchronous circuit. This technique uses a hierarchical model of asynchronous circuits in PSL and a formal verification tool called RAT.

INTITULE ET ADRESSE DU LABORATOIRE
Laboratoire TIMA, 46, avenue Félix Viallet, 38031 Grenoble Cedex, France

ISBN : 978-2-84813-155-9