Flot de conception pour circuits asynchrones : de la HLS à l'implémentation en FDSOI

Yoan Decoudu

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano-Électronique et Nano-Technologies (NENT)**

Arrêtée ministériel : 25 mai 2016

Présentée par

Yoan DECOUDU

Thèse dirigée par **Laurent FESQUET**
et codirigéee par **Katell MORIN-ALLORY**

préparée au sein du **Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés (TIMA)**
dans **l'École Doctorale Électronique, Électrotechnique, Automatique et Traitement du Signal (EEATS)**

Flot de conception pour circuits asynchrones : de la HLS à l'implémentation en FDSOI

Thèse soutenue publiquement le **23 septembre 2022**,
devant le jury composé de :

Laurent FESQUET
Maître de Conférences, Université Grenoble Alpes, Directeur de thèse

Katell MORIN-ALLORY
Maître de Conférences, Université Grenoble Alpes, Co-Directrice de thèse

Habib MEHREZ
Professeur des universités, Sorbonne Université, Président

Giovanni DE MICHELI
Professeur, École polytechnique fédérale de Lausanne, Rapporteur

Arnaud VIRAZEL
Professeur des universités, Université de Montpellier, Rapporteur

Frédéric ROUSSEAU
Professeur des universités, Université Grenoble Alpes, Examinateur

Remerciements

Cette thèse a été une grande aventure, dont l'origine commence un peu par hasard au détour d'un projet en école. Je tiens à remercier tous ceux qui de près ou de loin m'ont aidé durant ce voyage passionnant !

Un grand merci à Laurent et Katell, pour m'avoir donné l'opportunité de découvrir l'asynchrone et de réaliser cette thèse. Merci à tous les deux pour m'avoir guidé, formé, poussé pendant ces 4 années intenses. Merci aussi pour les grandes discussions et débats qui ont ponctué cette thèse ! Cela aura été un plaisir de travailler avec vous.

Merci à tous les collègues de l'équipe CDSI pour l'accueil, l'ambiance, les pauses, les bars et les grandes discussions. Merci aux compagnons de route, Grégoire, Ricardo, Mohamed, Rodrigo, Olivier, Florent et à ceux de passage, Nils et Mehdi (et ceux que j'ai probablement oubliés). Et aussi merci aux collègues plus récents, Rosalie, Xavier, Marco, Damiano, Cristiano (et bon courage pour la suite !). Je n'oublie pas non plus les chercheurs, Sylvain (tu comptes quasiment comme un chercheur), Stéphane, Skandar et Martial, merci pour les conseils et les retours. Je tiens aussi à remercier les membres du CIME, notamment Abdelhamid, Mohamed et Robin, que j'ai souvent sollicité.

Merci également à mes amis, de Grenoble et d'ailleurs. Et notamment pour les copains de Bourgogne, qui m'ont vu disparaître de la circulation pendant quelques années et qui ne m'ont pas oublié pour autant.

Enfin, merci à ma famille qui m'a toujours soutenu dans tout mon parcours de vie. Vous m'avez soutenu dans les moments difficiles, changer les idées quand il fallait que je décroche. Donc papa, maman, Prunelle, Allan et Clovis, merci pour tout.

Table des matières

Remerciements i
Table des matières iii
Table des figures v
Liste des tableaux vii

Introduction **1**

I Architecture asynchrone **5**

1 État de l'art **7**
1.1 Introduction 8
1.2 Principes des circuits asynchrones 8
1.2.1 Protocole de communication 8
1.2.2 Classes des circuits asynchrones 11
1.2.3 Avantages de l'asynchrone 14
1.3 Les circuits à données groupées 16
1.3.1 Modélisation des circuits asynchrones 16
1.3.2 Méthodes de conception des circuits asynchrones 18
1.3.3 Vérification des contraintes temporelles 21
1.4 Synthèse de haut niveau 24
1.4.1 Principe de la HLS synchrone 25
1.4.2 HLS asynchrone 25
1.5 Conclusion 27

2 Désynchronisation de circuits **29**
2.1 Introduction 30
2.2 Structure du circuit 30
2.2.1 Définition structurelle 31
2.2.2 Chemin de données 33
2.2.3 Chemin de contrôle 33
2.3 Méthode de désynchronisation 35
2.3.1 FSM asynchrone 36
2.3.2 Preuve de correction 40
2.3.3 Contraintes temporelles 44
2.4 Application à Catapult HLS 46
2.4.1 Architecture des circuits 46
2.4.2 Génération des contraintes 47

2.5 Conclusion 48

3 Évaluation de la méthode 51
3.1 Introduction 52
3.2 Étude comparative avec les autres méthodes 52
3.2.1 Description du circuit 52
3.2.2 Résultats de la désynchronisation 55
3.3 Application à la HLS 57
3.3.1 Architecture du prototype 57
3.3.2 Résultats 59
3.4 Conclusion 61

II Stratégie de polarisation de substrat 63

4 Technologie FDSOI 65
4.1 Introduction 66
4.2 Limitations des technologies classiques 66
4.3 Alternatives au Bulk CMOS 68
4.3.1 FinFET 68
4.3.2 FDSOI 68
4.4 Caractéristiques des technologies FDSOI 69
4.4.1 Isolation du substrat 69
4.4.2 Polarisation du substrat 70
4.5 Méthodes de polarisation de substrat 71
4.6 Conclusion 72

5 Stratégie de polarisation de substrat 75
5.1 Introduction 76
5.2 Définition de la stratégie 76
5.2.1 Principe de la stratégie 78
5.2.2 Éstimation de l'efficacité 81
5.3 Implémentation de notre stratégie 83
5.4 Évaluation de la méthode 86
5.4.1 Simulations numériques 86
5.4.2 Conception d'un prototype 88
5.5 Conclusion 90

Conclusion 93

Bibliographie I

Publications de l'auteur XI

Liste des acronymes XIII

A Prototype ASIC 65 nm XV

Table des figures

1 Flot de conception développé dans cette thèse. 3

1.1 Principe de communication entre deux systèmes. 9
1.2 Architecture d'un circuit synchrone. 9
1.3 Principe de communication à données groupées. 9
1.4 Protocole de communication (a) 4 phases et (b) 2 phases. 10
1.5 Principe de communication avec un encodage double rail. 10
1.6 Symbole et table de vérité d'une porte de Muller. 11
1.7 Classification des types de circuits asynchrones (adaptée de [6]) 11
1.8 Fourche isochronique dans les circuits QDI. 12
1.9 Architecture d'une machine Burst-Mode. 13
1.10 Architecture d'un circuit à données groupées. 14
1.11 Exemple d'un STG pour le protocole WCHB. 17
1.12 Composants de synchronisation [20] (a) totale et (b) sélective. 17
1.13 Principe des protocoles 4 phases précoces et tardifs. 18
1.14 Classification des méthodes de désynchronisation. 21
1.15 Contraintes temporelles min/max dans un circuit à données groupées. . . 23
1.16 Méthode LCS dans un circuit à données groupées. 24

2.1 Flot de désynchronisation. 31
2.2 Architecture d'un circuit synchrone. 31
2.3 Définition des sommets r et v d'un graphe orienté G. 32
2.4 Exemple d'une FSM issue d'une synthèse HLS. 34
2.5 Implémentation d'une FSM synchrone avec un encodage one-hot. 35
2.6 Circuit désynchronisé avec son chemin de données et une AFSM. 36
2.7 Implémentation d'une FSM désynchronisée. 36
2.8 Modèle de contrôleur. 37
2.9 Protocole *late-capture*. 37
2.10 Architecture d'un démultiplexeur. 38
2.11 Architecture d'un merge. 38
2.12 Implémentation des blocs V_{ready} et V_{valid}. 39
2.13 Génération d'un signal or^{i}_{out}. 39
2.14 Exemple d'un bloc Λ'_{en} avec le chronogramme associé. 40
2.15 Chronogrammes des circuits (a) synchrone et (b) asynchrone. 41
2.16 Chronogrammes des séquences d'états des circuits (a) synchrone et (b) asynchrone. 42
2.17 Architecture d'un circuit synchrone produit par Catapult. 46
2.18 Architecture du circuit désynchronisé. 47
2.19 Générateur de contraintes temporelles. 48

3.1 Algorithme de l'AES avec une clé de 128 bits. 53
3.2 Schéma bloc de l'architecture du circuit. 54
3.3 FSM du circuit AES. 54
3.4 AFSM du circuit désynchronisé. 55
3.5 Layout du prototype en 65 nm. 58
3.6 Caractérisation du prototype suivant la tension d'alimentation pour (a) le temps de calcul et (b) la puissance moyenne consommée. 60
3.7 Mesure de la consommation d'énergie suivant la tension d'alimentation. . 60

4.1 Schéma d'un transistor NMOS en technologie bulk CMOS. 67
4.2 Apparition des SCE avec la diminution de la taille du canal. 67
4.3 Schéma d'un transistor FinFET. 68
4.4 Transistors NMOS en technologie bulk CMOS (a) et FDSOI (b). 69
4.5 Schéma des transistor NMOS et PMOS en technologie FDSOI. 69
4.6 Configurations possibles pour des transistors FDSOI. 71
4.7 Architecture d'un système utilisant une polarisation de substrat dynamique. 72
4.8 Principe de la méthode de polarisation de substrat pour un circuit linéaire. 72

5.1 Flot implémentant la polarisation de substrat dynamique. 77
5.2 Principe de la stratégie de polarisation dynamique du substrat. 79
5.3 Implémentation du détecteur d'activité. 79
5.4 Architecture d'un BBG. 80
5.5 Exemple d'un circuit avec trois sous-blocs. 81
5.6 Exemple d'un circuit avec trois sous-blocs. 81
5.7 Variation du nombre de BBDs actifs dans le système (adapté de [114]). . . 82
5.8 Flot implémentant la polarisation de substrat dynamique. 84
5.9 Vue en coupe de l'isolation de deux BBDs. 85
5.10 Partitionnement des BBDs lors de l'implémentation physique d'un circuit. 85
5.11 Circuit linéaire avec cinq AES. 86
5.12 Puissance moyenne normalisée en fonction du taux d'activité. 88
5.13 Exemple d'une couche de *max pooling*. 89
5.14 Architecture du CNN. 90

A.1 Architecture des circuits AES asynchrones au sein du prototype. XVI

Liste des tableaux

2.1 Contraintes temporelles sur les registres (signaux Λ^i_{en}) 45
2.2 Contraintes temporelles sur la logique combinatoire (signaux Λ^i_{comb}) . . . 45

3.1 Résultat en surface et impact de la désynchronisation 55
3.2 Caractéristiques des circuits synchrone et désynchronisés 56
3.3 Simulation du temps de calcul et consommation du prototype 59

5.1 Impact de la désynchronisation sur l'AES 87

A.1 Pins d'entrées et sorties de la partie AES du prototype XVII

Introduction

Contexte et motivations

Durant ces deux dernières décennies, notre société a vu un nombre de plus en plus important de systèmes électroniques apparaître. La dernière manifestation de cette tendance est l'émergence de l'Internet des Objets, ou *Internet of Things* (IoT). Celui-ci est surtout composé de systèmes autonomes fonctionnant avec ou sans batterie. En outre, la demande globale en énergie électrique ne faisant que croître, il devient nécessaire y compris pour les équipements électroniques de devenir plus efficients en énergie. Cette nécessité va bien au-delà des besoins de l'IoT. Par exemple, l'amélioration de l'efficacité énergétique des processeurs utilisés dans les *data centers* est un sujet de recherche d'actualité. Par conséquent, l'industrie microélectronique s'est emparée de ce sujet et travaille intensivement à la réduction de la consommation de ces circuits intégrés. La réduction de la consommation était autrefois surtout apportée par la réduction de la taille des transistors, qui réduisait de manière quadratique leur consommation [1]. Mais, avec les noeuds technologiques récents pour lesquels la tension ne diminue plus, le gain en efficacité énergétique n'existe plus et l'on observe une densité de puissance qui augmente. Ainsi, un grand nombre de méthodes ont été mises en place pour réduire la consommation comme le changement dynamique de la tension d'alimentation (le *Dynamic Voltage Scaling* ou DVS), sa désactivation ou bien la polarisation du substrat dans des technologies comme le *Fully Depleted Silicon on Insulator* (FDSOI).

Aujourd'hui, la majorité des circuits intégrés sont en grande partie sur des circuits numériques synchrones. Ceux-ci possèdent un signal global, l'horloge, pour synchroniser l'ensemble du circuit. Pour fonctionner correctement, ce moment d'activation doit être exactement le même dans tout le circuit et être à une fréquence précise. L'activation globale du circuit crée une consommation inutile lorsque le circuit ou une partie de celui-ci n'est pas utilisé. Cette consommation peut alors être supprimée par des moyens additionnels comme le *clock gating* ou le DVS. De plus, les variations des procédés de fabrication dans les noeuds technologiques récents nécessitent d'ajouter des marges de sécurité vis-à-vis du signal d'horloge pour assurer un bon fonctionnement. Ces marges limitent les performances des circuits, que ce soit en vitesse ou en consommation. Dans ce contexte, il est indispensable de développer de nouvelles méthodes pour la gestion de la consommation et de se tourner vers des architectures alternatives pour les circuits intégrés numériques.

Une solution déjà expérimentée dans le milieu de la recherche et dans quelques entreprises est la logique asynchrone. Les circuits asynchrones, contrairement à leurs homologues synchrones, utilisent une synchronisation locale pour fonctionner. Sans la contrainte d'un signal global d'horloge, il est possible de concevoir des circuits avec une plus grande liberté. Ainsi, une grande variété d'architectures asynchrones a été développée au cours du temps. Les circuits asynchrones sont connus pour présenter de nombreux avantages. En

effet, toute consommation dynamique superflue est supprimée, ils sont plus robustes, ont un rayonnement électromagnétique plus faible, sont généralement plus rapides et offrent un vrai potentiel pour la sécurité matérielle. Ils sont donc de parfaits candidats pour les problématiques actuelles de l'industrie microélectronique. Les recherches sur la logique asynchrone ont débuté dès les années 50. Dans les années 80 et 90, des laboratoires de recherche ont commencé à concevoir des processeurs asynchrones dont le premier en 1989 par l'équipe du Professeur Alain Martin à Caltech, suivi par le processeur AMULET de l'université de Manchester et un MISP R3000 asynchrone à Caltech. Les années 2000 ont vu le premier processeur asynchrone européen, ASPRO, conçu au laboratoire TIMA, ainsi que les microcontrôleurs asynchrones MICA (TIMA) et le 8051 asynchrone de la société Philips Semiconductor. Enfin, nous avons assisté à l'éclosion d'entreprises spécialisées dans le domaine de la conception de circuits asynchrones. Handshake Solutions et Theseus Logic ont offert des solutions pour la synthèse des circuits asynchrones. Tiempo se concentre sur des circuits asynchrones robustes et sécurisés. Achronix a développé les premières FPGA très rapides et asynchrones. Pour sa part, ChronosTech propose des systèmes de réseaux sur puce, ou *Network-on-Chip*. Plus récemment, IBM et Intel ont respectivement conçu TrueNorth [2] et Loihi [3], des réseaux de neurones à impulsions asynchrones. Les circuits asynchrones ne sont donc plus une approche inconnue.

Malgré les avantages cités et les avancées de la recherche, les circuits asynchrones n'ont pas réussi à s'implanter durablement et largement dans l'industrie. Ceci peut s'expliquer par plusieurs raisons. D'une part, l'industrie a mis en place un environnement complet d'outils de Conception Assistée par Ordinateur (CAO) pour concevoir les circuits synchrones, couplé à des langages de description matériel spécifiques. De plus, les ingénieurs sont formés à ces outils ainsi qu'à l'architecture synchrone. Le domaine asynchrone a de son côté développé des langages et outils spécifiques pour concevoir au mieux les circuits asynchrones. Mais cet environnement de conception parallèle ajoute des contraintes. Une entreprise voulant concevoir des circuits asynchrones devra donc acheter de nouveaux outils de CAO et former ses ingénieurs, voire développer ses propres logiciels. Cela représente un investissement important qui peut décourager les industriels de choisir l'approche asynchrone. D'autre part, il est possible d'améliorer les performances des circuits synchrones sans recourir à une architecture complètement différente. Par exemple, il est tout à fait possible d'éteindre l'horloge lorsqu'un bloc n'est pas utilisé et de mettre en veille un circuit synchrone. Le passage du synchrone à l'asynchrone semble par conséquent bien souvent trop compliqué pour les gains espérés.

Mais le contexte commence à changer. Les améliorations en consommation demandées par l'IoT poussent à revoir l'architecture des circuits intégrés pour les optimiser en énergie. L'investissement à consentir pour passer à l'asynchrone ne paraît plus si coûteux désormais. Il faut donc rendre plus accessible la conception des circuits asynchrones au monde industriel. Dans cette optique, une pratique intéressante serait d'intégrer leur conception au sein des flots de conception et outils de CAO déjà existants.

Contributions

Pour favoriser l'adoption de la conception asynchrone par les ingénieurs "synchrones" et ainsi permettre une diffusion plus large de ces circuits dans l'industrie, nous proposons d'exploiter un flot de conception partant d'une description algorithmique. Les difficultés de la conception asynchrone que pourrait rencontrer les non-initiés sont par conséquent cachées. Ensuite, nous nous reposons totalement sur les outils de CAO commerciaux uti-

lisés dans l'industrie, et prévus pour de la logique synchrone. En exploitant une synthèse de haut niveau (HLS), nous générons des circuits synchrones qu'il est ensuite possible de convertir automatiquement en circuits asynchrones grâce à un petit programme. La suite de la conception est effectuée de façon standard avec des outils de CAO commerciaux. Ainsi la conception asynchrone devient compatible avec les flots synchrones habituels de l'industrie. Pour améliorer l'efficacité énergétique des circuits et montrer la pertinence des circuits asynchrones dans le développement de nouvelles méthodes de gestion de la consommation, nous couplons les circuits asynchrones générés par HLS avec une méthode dynamique de polarisation de substrat en FDSOI. Celle-ci permet de changer les paramètres physiques des transistors durant le fonctionnement du circuit. Nous utilisons cette possibilité pour améliorer l'efficacité énergétique des circuits, en augmentant la vitesse des transistors lors des calculs et en économisant l'énergie lors des phases de veille. Afin d'implémenter cette approche, il est nécessaire de respecter des règles précises afin de garantir l'implémentation physique du circuit. Le flot de conception complet est présenté en figure 1.

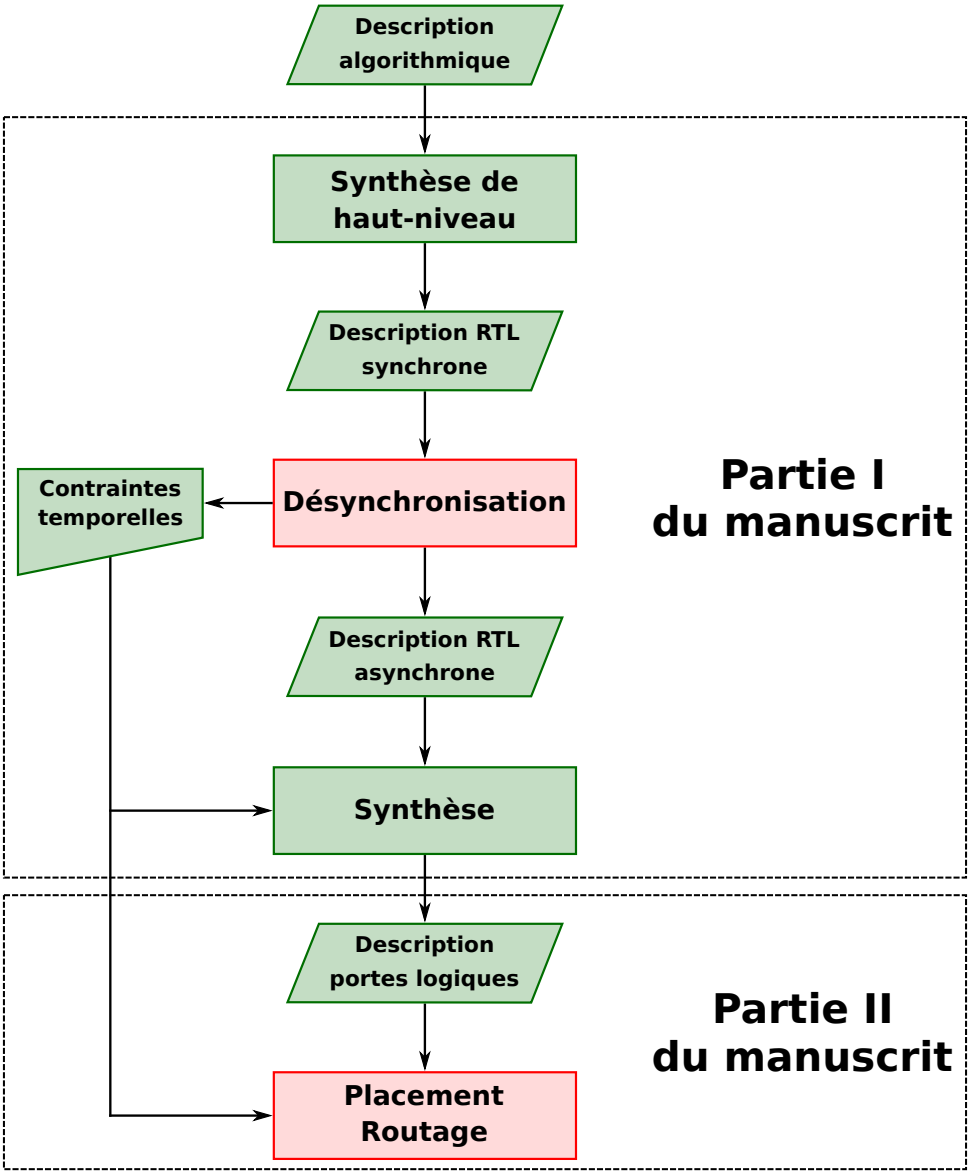


FIGURE 1 – *Flot de conception développé dans cette thèse.*

Le flot de conception est très semblable à un flot classique, à deux exceptions près : les phases de désynchronisation et de Placement/Routage. La phase de désynchronisation convertit le circuit synchrone en circuit asynchrone en remplaçant la partie contrôle, qui est une machine à états finis, par un circuit de contrôle asynchrone équivalent. Les circuits obtenus sont similaires aux circuits d'origine, mais sont plus robustes. Cette méthode, dont une preuve de concept avait été réalisée dans des travaux antérieurs, a été modélisée dans ce travail de thèse en vue d'en réaliser la preuve formelle. Cette dernière a été effectuée manuellement. De plus, nous avons intégré notre méthode à l'outil industriel Catapult HLS de Siemens EDA pour prouver sa faisabilité dans un cadre industriel. Notre flot de conception permet aussi de rendre la conception des circuits asynchrones aussi accessible que leurs homologues synchrones en exploitant les outils de CAO classiques. Tous ces résultats ont été validés par la conception, la fabrication et le test d'un prototype.

La phase de Placement/Routage permet ensuite d'implémenter la méthode de polarisation du substrat. Cette dernière phase permet de polariser avec un grain fin des sous-systèmes du circuit dès lors qu'une activité est présente. Cette approche favorise une meilleure efficacité énergétique du circuit. Grâce à l'utilisation de circuits asynchrones, le système de contrôle de la tension polarisation peut être également simplifier par rapport aux méthodes classiques. Nous utilisons alors de simples translateurs de tension contrôlés par un petit détecteur d'activité, qui sont intégrés au flot de conception numérique. Dans cette thèse, la méthode a été appliquée aux circuits asynchrones désynchronisés, et le modèle associé est présenté. L'implémentation physique de la méthode est en très grande partie automatisée.

Plan du manuscrit

Ce manuscrit de thèse est divisé en deux grandes parties. La première se concentre sur la partie conception asynchrone de notre travail, qui correspond à la première partie du flot développé. Le chapitre 1 décrit l'état de l'art des circuits asynchrones, ainsi que des méthodes de conception existantes. Le chapitre 2 présente notre méthode de génération automatique de circuits asynchrones à partir de circuits synchrones. Cette méthode a été intégrée au sein d'un flot de haut niveau reposant sur les outils commerciaux standards. Enfin, les circuits résultants de notre méthode de désynchronisation sont étudiés en termes de surface, vitesse et consommation dans le chapitre 3. Un prototype a aussi été fabriqué pour valider le flot de conception.

La deuxième partie du manuscrit développe la méthode de polarisation dynamique du substrat en FDSOI. Le chapitre 4 donne une présentation de la technologie FDSOI et ses principales caractéristiques, notamment la polarisation de substrat. Notre méthode de polarisation de substrat est définie au chapitre 5, avec son principe, son modèle et son flot d'implémentation physique. De premières simulations numériques ont aussi été réalisées pour donner un aperçu des gains potentiels de la méthode. Enfin nous concluons que les travaux menés permettent à la logique asynchrone de franchir la frontière entre les mondes académique et industriel, permettant ainsi le développement de nouvelles méthodes de gestion de la consommation.

Première partie

Architecture asynchrone

1 État de l'art

Sommaire

1.1 Introduction 8
1.2 Principes des circuits asynchrones 8
1.2.1 Protocole de communication 8
1.2.2 Classes des circuits asynchrones 11
1.2.3 Avantages de l'asynchrone 14
1.3 Les circuits à données groupées 16
1.3.1 Modélisation des circuits asynchrones 16
1.3.2 Méthodes de conception des circuits asynchrones 18
1.3.3 Vérification des contraintes temporelles 21
1.4 Synthèse de haut niveau 24
1.4.1 Principe de la HLS synchrone 25
1.4.2 HLS asynchrone 25
1.5 Conclusion 27

1.1 Introduction

Les circuits synchrones possèdent tous une synchronisation globale. En effet, les éléments mémorisants d'un circuit synchrone sont tous synchronisés par un unique signal d'horloge. Ce signal détermine par la suite la fréquence du circuit. Ce principe, très largement utilisé dans l'industrie, est simple et efficace mais les contraintes sur les signaux d'horloge sont de plus en plus importantes. En effet, l'activation de l'ensemble des éléments d'un circuit à intervalle de temps régulier présente des pics de consommation importants [4]. De même, les variations de fabrication dans les derniers noeuds technologiques rendent de plus en plus compliquées l'activation uniforme de l'horloge dans le circuit [5], limitant les performances. Pour dépasser ces limitations, il est nécessaire de rechercher de nouvelles solutions architecturales.

À l'inverse, les circuits asynchrones utilisent une synchronisation locale pour fonctionner. Sans signal d'horloge, les composants du circuit communiquent alors les uns avec les autres à l'aide de protocoles de communication de type "poignée de main". Pour être fonctionnel, un circuit asynchrone repose par conséquent sur des hypothèses temporelles locales et son fonctionnement est par nature événementiel : seuls les composants nécessaires sont activés. Ce type de circuit possède donc une consommation réduite et une robustesse accrue aux conditions liées à l'environnement ainsi qu'aux variations induites lors de la fabrication. Ces caractéristiques [6] sont prometteuses pour résoudre les problématiques actuelles des circuits synchrones [7, 8], ainsi que pour développer de nouvelles méthodes événementielles de gestion de la consommation.

Mais le domaine des circuits asynchrones est vaste car il existe une grande liberté dans la conception de ces derniers. Nous présenterons donc dans ce chapitre les concepts de base des circuits asynchrones, ainsi que les principales classes de circuits asynchrones existantes. Nous verrons aussi les avantages que peuvent apporter ces circuits, qui restent aujourd'hui pour le moins non-conventionnels. Nous rentrerons plus en détail sur les méthodes de modélisation et de conception de ces circuits, et notamment dans l'environnement des logiciels industriels.

1.2 Principes des circuits asynchrones

1.2.1 Protocole de communication

Dans un circuit numérique, les différents systèmes du circuit réalisent des opérations sur les données et s'échangent ces données entre eux. Pour que cette communication s'effectue correctement, il est nécessaire d'avoir une méthode de synchronisation entre les systèmes du circuit. La figure 1.1 présente le schéma de principe d'une communication entre deux systèmes d'un circuit, dont un joue le rôle d'émetteur de données et l'autre sert de récepteur. À l'intérieur de ces systèmes, il existe deux types de composants : des portes logiques combinatoires, qui calculent de nouvelles données, et des éléments séquentiels pour enregistrer des données, principalement des registres ou des verrous. La synchronisation va permettre d'indiquer au récepteur à quel moment les données qui lui sont envoyées sont valides et donc quand activer ses composants séquentiels.

Une manière simple de synchroniser les systèmes d'un circuit est d'utiliser un signal

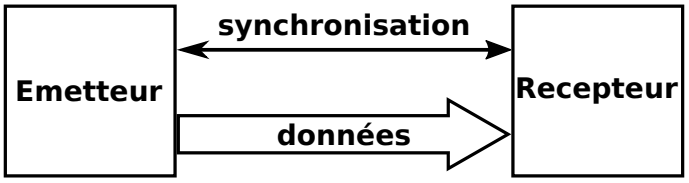


FIGURE 1.1 – *Principe de communication entre deux systèmes.*

de synchronisation externe global : une horloge. Tous les composants séquentiels du circuit sont donc activés au même moment par l'horloge à intervalle de temps régulier. On obtient alors un circuit synchrone, comme en figure 1.2. Par leur simplicité de conception, les circuits synchrones se sont imposés dans l'industrie et représentent la quasi-totalité des circuits produits dans le monde. La vitesse du circuit est alors déterminée par la fréquence d'horloge, qui dépend elle-même du chemin le plus long entre deux registres dans le circuit. Il s'agit donc d'une approche pire cas.

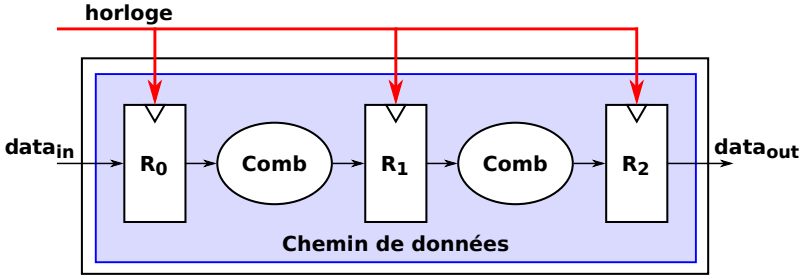


FIGURE 1.2 – *Architecture d'un circuit synchrone.*

Néanmoins, il existe d'autres façons de synchroniser un circuit. En effet, la validité des données peut être groupée au sein d'une communication. Au lieu d'un signal global, les différents systèmes du circuit communiquent localement à l'aide de signaux de requête et d'acquittement, dont le principe est présenté en figure 1.3. Lors d'une communication, l'émetteur va envoyer un signal de requête en même temps que l'envoi des données. De son côté, le récepteur n'accepte des données que lorsqu'il reçoit le signal de requête. Après l'enregistrement des données et pour indiquer la bonne réception des données à l'émetteur, il envoie à son tour un signal d'acquittement. On appelle ce type de design des circuits à données groupées.

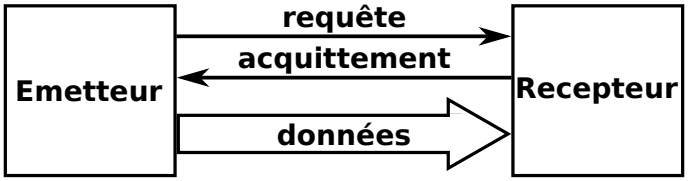
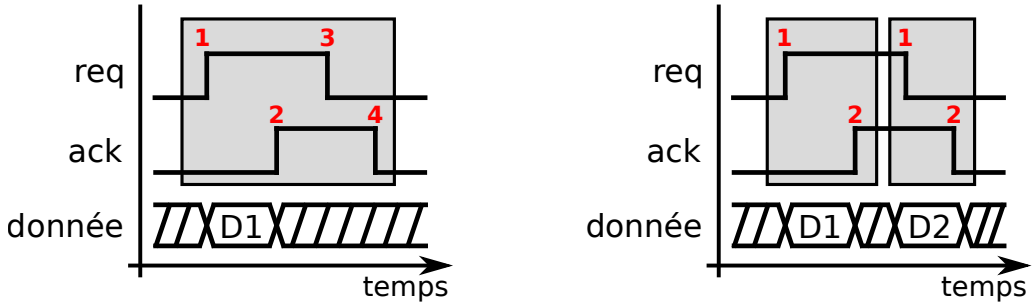


FIGURE 1.3 – *Principe de communication à données groupées.*

Pour assurer la bonne transmission des données, un protocole de communication doit être défini entre l'émetteur et le récepteur. Ces protocoles, que l'on appelle de *handshake* ou poignée de main, sont très variés mais il en existe deux grandes familles présentées figure 1.4 : les protocoles 4 phases et 2 phases. Les protocoles 4 phases sont les plus simples à implémenter et sont présentés figure 1.4a. Les deux premières phases sont utilisées pour indiquer l'envoi et la réception des données puis les phases 3 et 4 permettent de remettre



(a) Protocole 4 phases. *(b) Protocole 2 phases.*

FIGURE 1.4 – *Protocole de communication (a) 4 phases et (b) 2 phases.*

à zéro tous les signaux. Dans un protocole de type 2 phases comme en figure 1.4b, il n'y a pas de retour à zéro des signaux avant une nouvelle communication. C'est donc une nouvelle transition, et non un niveau logique qui indique une nouvelle donnée. Ces protocoles, nécessitant une implémentation plus complexe, sont aussi plus rapides.

Il est aussi tout à fait possible d'intégrer la requête directement dans les données envoyées avec le protocole de communication associé. On parlera alors de validité intégrée aux données. Mais dans ce cas, il n'est plus possible d'utiliser l'encodage classique des données. En effet, il existe habituellement un fil mis à zéro ou à un pour représenter la valeur binaire d'un signal logique. Une autre catégorie de codage, appelée M parmi N, permet de représenter M signaux avec N fils. Ce type d'encodage est moins usité et se traduit souvent par un circuit plus complexe et plus coûteux. Il est le plus souvent employé sous la forme d'un double rail, qui correspond à un codage de 1 parmi 2. L'architecture obtenue est présentée en figure 1.5.

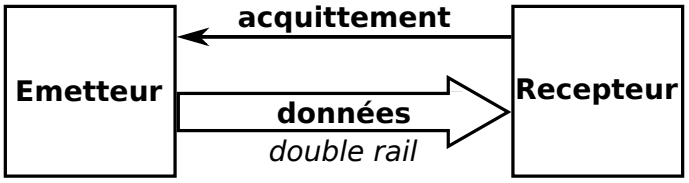


FIGURE 1.5 – *Principe de communication avec un encodage double rail.*

Il est là encore nécessaire d'utiliser un protocole de communication, mais cette fois-ci l'encodage des données permet de le définir. Dans le cas du double rail par exemple, un encodage à trois états permet de réaliser un protocole 4 phases. Tandis que les codes '01' et '10' représentent respectivement un 1 et un 0 logique, le code '00' correspond au retour à zéro du protocole. Pour les protocoles 2 phases, un encodage sur quatre états sera utilisé.

Afin d'implémenter matériellement ces protocoles de communication, il est nécessaire de pouvoir synchroniser des signaux entre eux. Pour réaliser cette fonction, une nouvelle porte séquentielle est introduite. Cette porte logique est la porte de Muller, ou *C-element*, dont le symbole et la table de vérité sont présentés en figure 1.6. La sortie de la porte de Muller prend la valeur des entrées lorsque celles-ci sont identiques, et enregistre la précédente valeur lorsqu'elles sont différentes. Comme ce type de porte n'existe pas dans les librairies standards habituelles, elle doit être conçue soit avec des portes logiques existantes soit comme une cellule standard additionnelle.

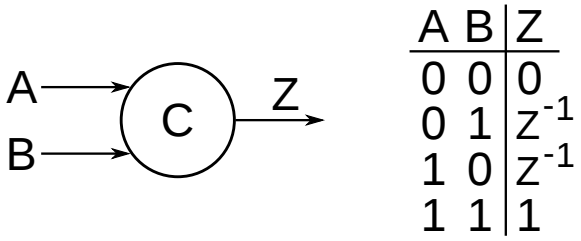


FIGURE 1.6 – *Symbole et table de vérité d'une porte de Muller.*

1.2.2 Classes des circuits asynchrones

Il existe un grand nombre de type de circuits asynchrones dans la littérature, très différents les uns des autres. Suivant le choix du protocole de communication ou de l'encodage de données, leurs implémentations diffèrent. Il est à noter qu'en fonction des hypothèses temporelles, les circuits ont des propriétés différentes et notamment seront plus au moins robustes aux variations des procédés de fabrication, de la tension et de la température (PVT). Une classification comme en figure 1.7 des principales classes de circuits peut alors en être déduite. Les architectures asynchrones sont classées suivant leurs hypothèses temporelles et le type de validité des données utilisé, du plus local au plus global.

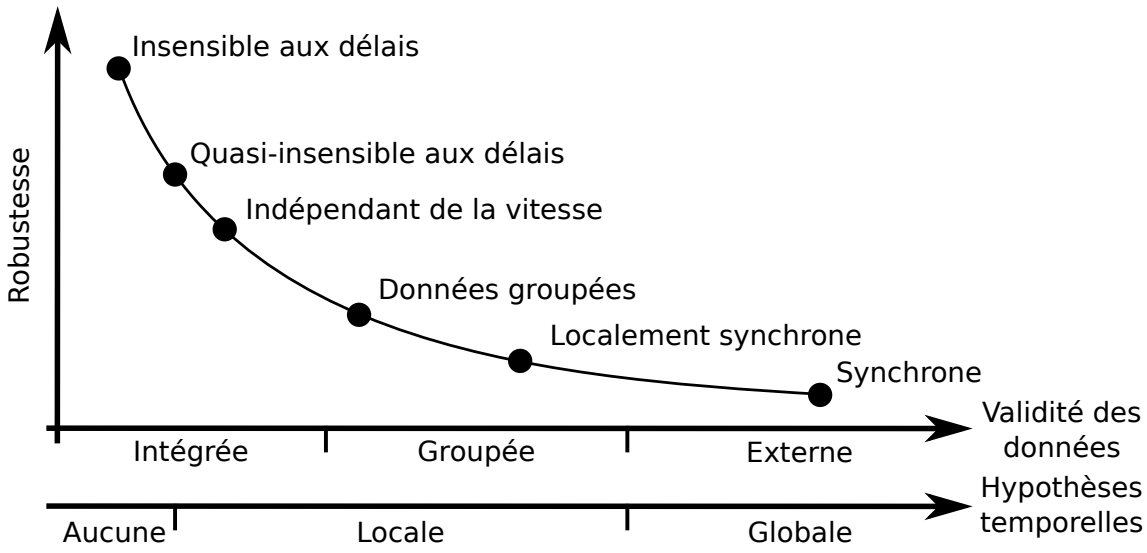


FIGURE 1.7 – *Classification des types de circuits asynchrones (adaptée de [6])*

À l'extrême gauche du graphique, les circuits insensibles aux délais, ou *Delay Insensitive* (DI), n'ont aucune hypothèse temporelle et sont les circuits les plus robustes existants. Ensuite, des hypothèses temporelles sont progressivement ajoutées très localement avec les circuits quasi-insensibles aux délais puis les circuits indépendants à la vitesse. Les circuits à données groupées et les machines localement synchrones reposent sur des signaux de synchronisation et utilisent des contraintes temporelles plus fortes, quoique toujours locales. Pour finir, les circuits synchrones utilisent un signal d'horloge externe pour indiquer la validité des données, qui détermine la seule hypothèse temporelle globale du circuit suivant la fréquence d'horloge choisie.

Circuits insensibles aux délais

Les circuits DI ne nécessitent aucune contrainte temporelle pour fonctionner correctement [9]. Ce sont les circuits les plus robustes que l'on peut concevoir. Néanmoins, les contraintes de conception de ces circuits les rendent en pratique inutilisables car les deux seules portes logiques acceptées sont les inverseurs et les portes de Muller [10].

Circuits quasi-insensibles aux délais

Une classe proche des circuits précédents correspond aux circuits quasi-insensibles aux délais, ou *Quasi-Delay Insensitive* (QDI). Ces circuits reposent sur une seule hypothèse temporelle locale : la fourche isochronique. Pour que les circuits QDI soient fonctionnels, elle suppose, comme présentée en figure 1.8, que certaines fourches présentent des délais égaux sur leurs branches. Cette hypothèse temporelle assez simple permet d'obtenir des circuits très robustes et bien plus pratiques à concevoir que leurs équivalents DI [10].

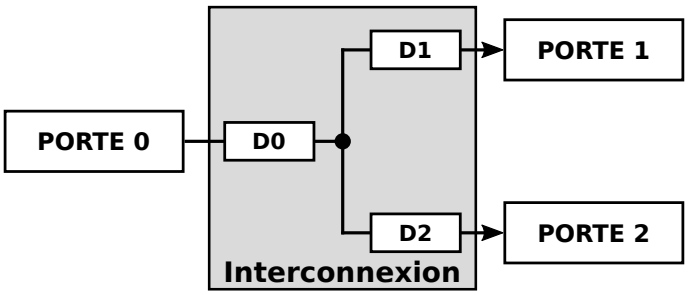


FIGURE 1.8 – *Fourche isochronique dans les circuits QDI.*

Les circuits QDI ont été largement étudiés au fil des ans, ce qui a amené un grand nombre d'implémentations. Par exemple, l'implémentation *Weak Contention Half Buffer* (WCHB) synthétisée à partir de la méthode *Delay Insensitive Min-terms Synthesis* (DIMS) est simple à concevoir mais présente une grande surface, tandis que la logique *Null Convention Logic* (NCL) [11] peut être généré à partir des outils de CAO classiques et se concentre la basse consommation [12]. Toutes ces implémentations ont en commun une grande robustesse et une sécurité améliorée, au prix d'une plus grande surface comparée à un circuit synchrone.

Circuits indépendants de la vitesse

Les circuits indépendants de la vitesse, ou *Speed Independent* (SI) sont très similaires aux circuits QDI et sont issus d'un modèle développé par Muller [13]. Ce modèle suppose que les délais des fils sont négligeables mais que les délais des composants sont non bornés. Par conséquent, toutes les fourches sont isochrones. Les circuits QDI sont toutefois préférés aux circuits SI, car un modèle avec des délais d'interconnexion négligeables n'est plus pertinent au vu des technologies CMOS actuelles.

Machines localement synchrones

Ces circuits, historiquement étudiés par Huffman [14], sont des machines à états finis asynchrones (AFSM). Une implémentation efficace, qui permet la prise en compte

d'un groupe d'entrées pour chaque nouvel état, appelée une machine Burst-mode, a été présentée formellement par Nowick *et al.* [15] dont un schéma de principe est visible en figure 1.9.

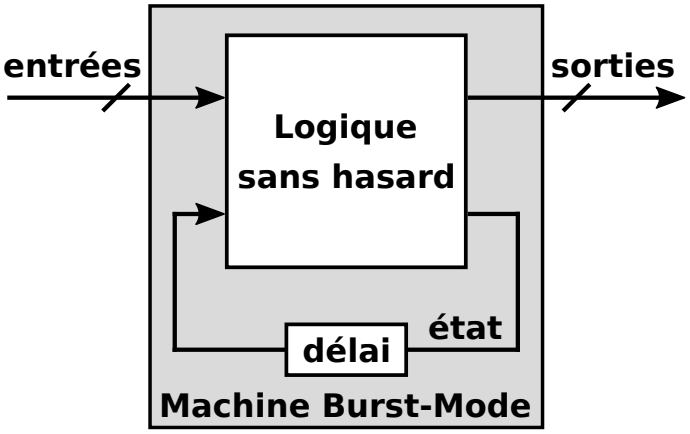


FIGURE 1.9 – *Architecture d'une machine Burst-Mode.*

Ces circuits reposent sur une horloge locale interne générée par de la logique combinatoire. En revanche, il existe plusieurs contraintes dans ces circuits. Premièrement, la logique combinatoire doit être sans aléa, c'est-à-dire exempte de *glitches*. Ensuite, des délais doivent être présents sur le rebouclage de l'AFSM afin que les données d'états ne soient pas effacées. Et enfin, aucun changement sur les entrées ne doit survenir pendant le calcul d'un nouvel état. Pour ce dernier point, la spécification *Extended Burst-Mode* (XBM) [16, 17] élargit les possibilités de changement des entrées du système.

Toutes ces contraintes rendent la conception des machines Burst-Mode ardue, mais des outils ont été développés pour synthétiser une logique sans aléa, comme par exemple les logiciels de synthèse 3D [17] ou minimalist [18].

Circuits à données groupées

Tout d'abord présentés sous la forme du *micropipeline* par Ivan Sutherland [19], les circuits à données groupées sont très semblables aux circuits synchrones. Alors que leurs chemins de données sont similaires à leurs homologues synchrones, la synchronisation des éléments mémorisants du circuit n'est plus réalisée par un signal global d'horloge mais par un circuit de contrôle [20] (voir figure 1.10). Ce circuit de contrôle est composé de contrôleurs locaux communiquant entre eux à l'aide d'un protocole de communication.

Comme présenté en figure 1.10, chaque contrôleur $ctrl_i$ est associé à un banc de registres R_i. Une nouvelle donnée entrante $data_{in}$ est annoncée au circuit par le signal de communication req_{in} qui active le premier contrôleur. Quand un contrôleur est actif, il déclenche l'activation du banc de registres associé et lance une communication avec le contrôleur suivant. Afin que le circuit soit fonctionnel, il faut que l'activation des registres se produise après le calcul et la stabilisation des données dans la partie combinatoire. Pour respecter cette contrainte temporelle, il est donc nécessaire d'insérer des blocs de délais sur les lignes de requête dans le circuit de contrôle.

De nombreuses variantes d'implémentations et de protocoles ont été développées pour les circuits à données groupées. De plus, leur proximité avec les circuits synchrones les rend plus simples à concevoir avec des outils de CAO standards. C'est pourquoi nous uti-

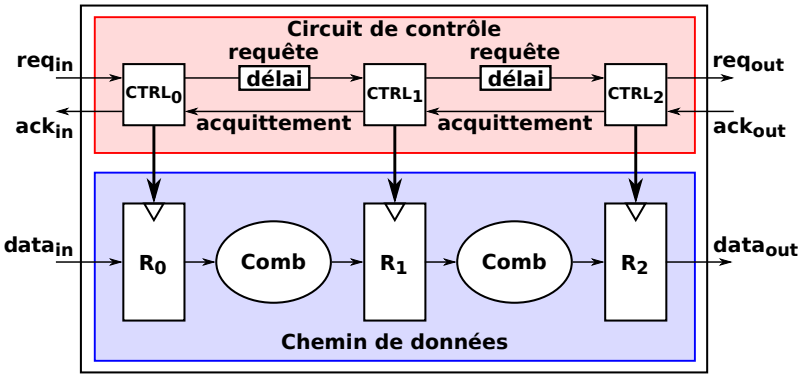


FIGURE 1.10 – *Architecture d'un circuit à données groupées.*

liserons ces circuits dans ce travail de thèse. Toutes leurs caractéristiques seront détaillées dans la section 1.3.

1.2.3 Avantages de l'asynchrone

Les circuits asynchrones, de par leur synchronisation locale, possèdent un certain nombre d'avantages comparés à leurs homologues synchrones. Si les circuits synchrones se sont imposés dans l'industrie microélectronique grâce à la simplicité de leur conception, des limites commencent à apparaître du fait de leur synchronisation globale. Les contraintes imposées sur le signal d'horloge en termes de performance et de consommation sont telles que, désormais, de nouvelles approches ont été mises en place, comme les systèmes Globalement Asynchrones Localement Synchrones ou *Globally Asynchronous Locally Synchronous* (GALS).

Consommation

Le paradigme asynchrone repose sur la synchronisation locale des éléments du circuits. Un composant n'est donc activé que lorsque cela est nécessaire. Les architectures asynchrones réduisent donc intrinsèquement la consommation dynamique du circuit [12, 21]. De plus, les circuits asynchrones, dont le comportement est événementiel, sont automatiquement mis en veille lors de l'attente d'une nouvelle donnée à traiter. Ils n'ont donc pas besoin d'un système additionnel de mise en veille, réduisant de fait la consommation.

Il est aussi à noter que des améliorations au niveau système sont aussi présentes. En effet, un circuit synchrone nécessite un générateur externe d'horloge. Ce générateur d'horloge, généralement composé d'un oscillateur et d'une *Phase Locked Loop* (PLL), va consommer à moins d'être désactivé lors des moments de veille. Sa suppression est un réel avantage pour l'asynchrone, même si ce phénomène est rarement discuté dans la littérature.

Robustesse

En fonction des variations PVT, les délais des portes logiques dans un circuit sont variables. Les générateurs d'horloge procurent un signal à une fréquence précise qui ne prend pas en compte ces variations. Pour résoudre ce problème, des marges sont ajoutées

lors de la conception des circuits pour tenir compte des variations de fréquence de l'horloge d'une part et des délais dans le circuit d'autre part. Ainsi, on garantit que les circuits synchrones fonctionnent sur des plages PVT précises mais au détriment des performances globales.

Les circuits asynchrones de leur côté sont bien plus robustes. Les circuits QDI ne font aucune hypothèse temporelle sur les délais de propagation des portes logiques et des interconnexions hormis quelques fourches dites isochrones. Ils sont par conséquent extrêmement robustes [22] : ils sont fonctionnels tant que les portes logiques restent elles-mêmes fonctionnelles et vont à la vitesse maximale permise par les conditions PVT. Les circuits à données groupées possèdent quant à eux des hypothèses temporelles locales et sont donc moins robustes que les circuits QDI. Néanmoins, les lignes de délais qui assurent ces hypothèses sont intégrées directement en portes logiques sur le circuit. Les délais vont donc varier en même temps et de la même manière que les chemins de données associés, apportant ainsi une amélioration de la robustesse comparée à celle des circuits synchrones.

Modularité

Avec une synchronisation locale entre les sous-systèmes reposant sur un protocole de communication, les circuits asynchrones possèdent une modularité supérieure à leurs homologues synchrones. En effet, chaque sous-système fonctionne à sa propre vitesse et ne prend pas en compte les contraintes temporelles des autres blocs, ce qui facilite l'assemblage des différents composants d'un circuit. En effet, une validation globale des hypothèses temporelles n'est pas nécessaire. Pour un circuit synchrone, la fréquence de fonctionnement est déterminée par le système le plus lent et peut donc amener de nouvelles itérations de conception pour améliorer les performances. Pour éviter des contraintes trop lourdes sur le signal d'horloge, les systèmes GALS sont apparus ces dernières années, avec un signal d'horloge dédié à chaque bloc. Toutefois, dans ce cas, les interfaces entre les différents domaines d'horloge sont complexes et pas complètement sûres (risque de métastabilité). Cette difficulté est inexistante avec les circuits asynchrones.

Faibles émissions électromagnétiques

Dans un circuit synchrone, comme l'horloge active globalement le circuit, un fort rayonnement électromagnétique apparaît couplé à la fréquence d'horloge. Ce rayonnement peut poser problème, notamment avec la partie analogique d'un circuit [23], cette dernière, étant assez sensible au bruit environnant, est par conséquent soumise à des normes spécifiques. Dans le cas asynchrone, les activations de composants sont réalisées localement, distribuées spatialement et temporellement au sein du circuit. Le rayonnement électromagnétique est donc diminué [24], de même que le spectre électromagnétique sera plus étalé. Il est même possible de mettre en place des méthodes de contrôle du spectre dès la conception [25], ce qui se révèle très compliqué, voire impossible, pour un circuit synchrone.

Malgré tous ces avantages, les systèmes asynchrones n'ont pas réellement réussi à émerger dans l'industrie même si cela est annoncé depuis de nombreuses années par l'*International Technology Roadmap for Semiconductors*. La question est donc d'identifier les raisons de ces difficultés d'adoption par l'industrie. Plusieurs problématiques

peuvent expliquer ce phénomène. Tout d'abord, les circuits asynchrones sont difficiles à concevoir. Ensuite, les circuits synchrones ont été grandement améliorés au fil des années et ont réduit la différence avec l'asynchrone sur un certain nombre de paramètres. Par exemple, le *clock gating* permet de réduire la consommation, les horloges peuvent maintenant être intégrées en portes logiques pour augmenter la robustesse [26] et les GALS diminuent les contraintes sur les signaux d'horloge.

Une grande difficulté du domaine asynchrone est donc la conception des circuits asynchrones, notamment avec des outils de CAO classiques. En effet, dans un contexte de coût productions importants ainsi que la nécessité d'un *time to market* plus court possible, les entreprises sont réticentes à utiliser de nouveaux outils ou langages. Dans cette optique, nous nous intéresserons dans ce travail à une classe spécifique de circuits asynchrone : les circuits à données groupées. En effet, ceux-ci sont les circuits plus proches des circuits synchrones tout en gardant la majorité des avantages du paradigme asynchrone. La section suivante se concentrera donc sur ces circuits asynchrones ainsi qu'à leur méthode de conception.

1.3 Les circuits à données groupées

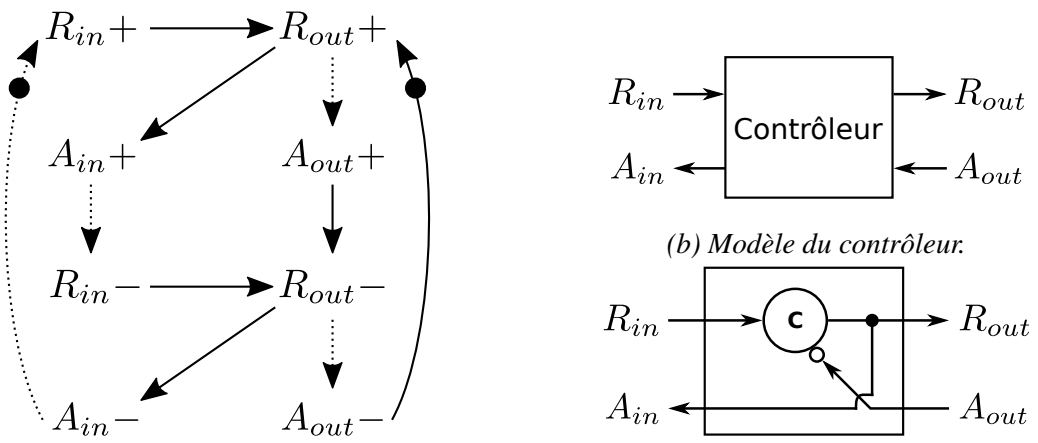
1.3.1 Modélisation des circuits asynchrones

Le circuit de contrôle d'un circuit à données groupées repose sur des événements de signaux, dont l'apparition est intrinsèquement concurrente. Un moyen pratique et efficace de modéliser les protocoles de communication est alors d'utiliser des réseaux de Petri [27].

Un modèle populaire pour représenter le comportement asynchrone d'un circuit asynchrone est le graphe de transitions de signaux ou *State Transition Graph* [28] (STG), qui est un réseau de Petri montrant toutes les transitions existantes. Pour tout signal *Sig*, les transitions $Sig+$ et $Sig-$ représentent respectivement les fronts montants et descendants de *Sig*. Ces transitions sont reliées à des places. Afin d'améliorer la lisibilité du modèle, les places ne sont pas représentées et sont remplacées par des flèches. Les flèches en pointillées modélisent l'environnement du contrôleur. Un jeton placé sur la flèche signifie que la place est remplie. La transition est effectuée quand chaque place connectée à la transition est remplie.

Pour illustrer le fonctionnement de ce modèle, la figure 1.11a présente le STG du protocole WCHB. L'environnement déclenche le contrôleur en générant un front montant sur le signal d'entrée R_{in}. Cette transition génère à son tour un front montant sur le signal de sortie R_{out}. Il est à noter qu'il n'y a pas d'hypothèse temporelle. Par conséquent, l'événement peut apparaître après un délai indéfini. Ensuite, deux nouvelles transitions vont se produire : un front montant sur A_{in} en interne du contrôleur et un autre sur A_{out} en passant par l'environnement. Les deux événements apparaissent de manière concurrente et il n'y a donc aucune question d'ordre entre les deux. Les transitions continuent à s'enchaîner de la même manière jusqu'au retour à l'état initial du contrôleur.

Le STG peut ensuite être synthétisé matériellement avec des outils comme *Petrify* [29]. Dans le cas de notre exemple avec le protocole WCHB, *Petrify* implémente le STG sous la forme d'une porte de Muller et d'un inverseur (voir figure 1.11c). Si l'implémentation de



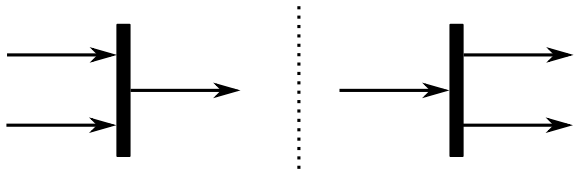
(a) STG du protocole WCHB.

(b) Modèle du contrôleur.

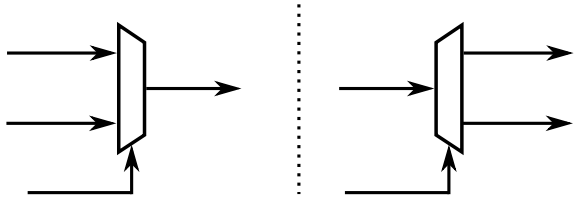
(c) Implémentation du contrôleur.

FIGURE 1.11 – *Exemple d'un STG pour le protocole WCHB.*

ce STG est simple, des STGs plus complexes deviennent difficiles à synthétiser à cause de l'explosion du nombre d'états et du temps de vérification [30] lors de la génération de l'implémentation. Cette méthode est donc peu adaptée pour un circuit de contrôle entier. Par conséquent, la conception d'un circuit asynchrone s'appuie la plupart du temps sur l'assemblage de primitives. Dans ce cadre, les contrôleurs sont construits à partir du STG d'un protocole et contrôlent les registres du chemin de données. Des composants additionnels compatibles avec ce protocole sont utilisés pour synchroniser les contrôleurs entre eux dans le cas d'un circuit non-linéaire, comme par exemple ceux présentés en figure 1.12.



(a) Composants join (à gauche) et fork (à droite).



(b) Composants multiplexeur (à gauche) et demultiplexeur (à droite).

FIGURE 1.12 – *Composants de synchronisation [20] (a) totale et (b) sélective.*

Dans le cadre des circuits à données groupées, il existe un grand nombre de protocoles offrant différents avantages. Tout d'abord, les protocoles 2 phases visent la performance car ils ne possèdent pas de phase de retour à zéro. On pourra citer le micropipeline [19], le GasP [31], le mousetrap [32] et enfin Click [33]. Ce dernier protocole a aussi l'avantage de ne pas utiliser de porte de Muller pour améliorer l'intégration au flot de conception conventionnel et la testabilité des circuits asynchrones.

Si les protocoles 2 phases sont performants, ils impliquent aussi une conception du circuit plus complexe. Pour faciliter cette conception, l'utilisation des protocoles 4 phases

est souvent préférée. La plupart de ces protocoles 4 phases activent les registres lors du front montant de la requête, que l'on appelle des protocoles précoces [34]. Parmi eux, les protocoles découplent plus ou moins les communications d'entrée et de sortie d'un contrôleur [35], ce qui détermine la performance du contrôleur en fonction de sa complexité. Un très grand nombre de ces protocoles ont été développés au cours des dernières décennies et on dénombre aujourd'hui plus d'une centaine de protocoles [36, 37]. La figure 1.13 présente le chronogramme d'une communication entre deux contrôleurs. Afin de respecter les contraintes temporelles, il est nécessaire d'avoir une ligne de délais δ sur le signal requête qui soit plus grande que le chemin critique du chemin de données. On appelle temps de cycle le temps nécessaire à une communication pour être complète, temps avant lequel une nouvelle communication ne peut s'établir. Dans le cas des protocoles précoces, le temps de cycle du protocole équivaut à plus de deux fois le délai de la ligne de requête car le front descendant est aussi propagé dans le délai δ lors de la phase de retour à zéro. Cela mène à une importante diminution de vitesse comparée à un circuit synchrone si le chemin de données est conservé en l'état.

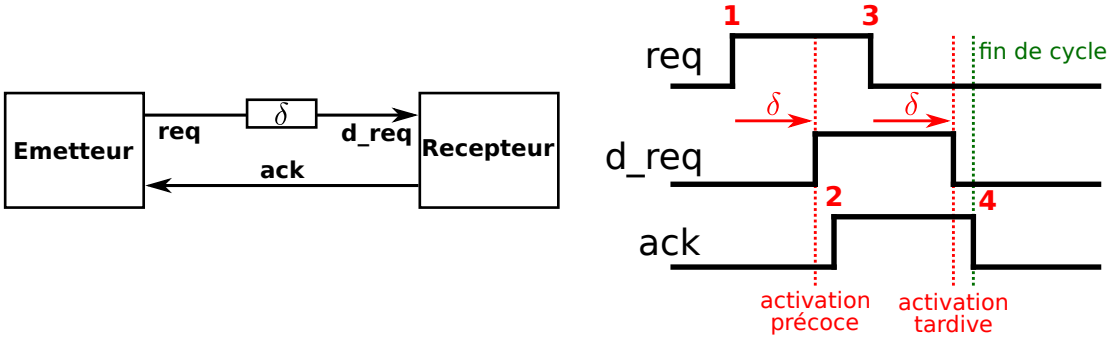


FIGURE 1.13 – *Principe des protocoles 4 phases précoces et tardifs.*

Afin d'être plus efficace, il est préférable d'utiliser un protocole activant les registres sur le front descendant de la requête, appelé protocole 4 phases tardif [34]. Dans ce cas, les fronts montants et descendants du signal de requête sont propagés dans la ligne de délais avant l'activation des registres, comme présenté en figure 1.13. De cette manière, le bloc de délai est réduit à seulement la moitié de celui requis pour un protocole classique à 4 phases. Ces derniers possèdent deux avantages : la vitesse redevient comparable à celle des circuits synchrones et la taille de la ligne de délai est diminuée par 2 ainsi que sa consommation.

Contrairement aux protocoles 4 phases précoces, les protocoles tardifs restent encore peu étudiés dans la littérature. Seulement quatre ont été développés : les protocoles Burst-Mode [38], Early-Ack [39], Late-Capture et Maximus [40]. Tous possèdent des performances similaires à l'exception du protocole Maximus. Ce dernier permet un temps de cycle minimal au prix d'une plus grande surface.

1.3.2 Méthodes de conception des circuits asynchrones

Afin de concevoir les circuits asynchrones, il est nécessaire d'avoir des langages HDLs et les outils de CAO associés. Les langages HDLs classiques, le VHDL et le verilog, sont peu adaptés pour décrire les circuits asynchrones. Un certain nombre de travaux visent tout de même l'utilisation des outils de CAO standards pour éviter le surcoût du développement de nouveaux outils. Une meilleure approche serait l'usage d'une description

au niveau transactionnel permettant d'abstraire le protocole de communication et de représenter la concurrence dans les circuits asynchrones ciblés. Dans ce cas, cela implique d'utiliser des langages adaptés à ce niveau de description ainsi que des logiciels dédiés.

Traduction dirigée par la syntaxe

Une approche pour la conception des circuits asynchrones est le développement de HDLs spécifiques. Ces langages, de plus haut niveau que le VHDL et le verilog, permettent de représenter la concurrence des opérations asynchrones et de traduire la description en macro-composants matériels issus de librairies préexistantes. Dans ce contexte, les outils de synthèse TiDE et Balsa [41] utilisent des composants *handshake* [42]. L'outil TAST [43] utilise quant à lui le langage CHP [44]. Tout comme pour les langages utilisés pour les circuits synchrones, les circuits générés par ces méthodes dépendent fortement de la description, par conséquent les concepteurs doivent avoir de bonnes connaissances sur le langage et l'architecture souhaitée.

Désynchronisation de circuits

Il est assez compliqué d'utiliser les outils commerciaux standards pour concevoir des circuits QDI du fait de leur architecture assez éloignée du synchrone. Néanmoins Ligthart *et al.* [45] ou plus récemment les travaux en [46,47] ont mis en place des flots de conception avec les circuits QDI de type NCL. La difficulté réside dans la façon d'écrire le RTL pour représenter correctement le circuit QDI, puis de transformer la sortie de l'outil de synthèse pour obtenir le circuit voulu.

Les circuits synchrones et asynchrones à données groupées ont en revanche des architectures très proches. En effet, ils partagent un chemin de données similaire et seul l'élément de synchronisation diffère avec un signal d'horloge pour les uns et un circuit de contrôle pour les autres. Il est donc possible d'obtenir très rapidement un circuit asynchrone par la conversion d'un circuit synchrone par le biais d'une transformation appelée désynchronisation. L'objectif des méthodes de désynchronisation est de diminuer fortement le temps de conception des circuits asynchrones et de permettre à des ingénieurs non formés au domaine asynchrone de participer au développement de ces circuits.

Désynchronisation classique Une méthode assez simple et directe de désynchroniser un circuit synchrone est de remplacer son signal d'horloge par un circuit de contrôle. Néanmoins, cette transformation nécessite de garantir le même comportement par rapport au circuit d'origine.

Cortadella *et al.* [48] ont été les premiers à mettre en place un flot de désynchronisation complet. Dans celui-ci, chaque registre du circuit est converti en une paire de *latches* maître/esclave et ensuite le circuit de contrôle est construit en définissant un contrôleur asynchrone par *latch*. De plus, leurs travaux présentent les différents protocoles [49] compatibles avec la désynchronisation (même si des travaux récents [50] montrent qu'un de ces protocoles n'est finalement pas correct).

La méthode CAR [51] conserve quant à elle les bascules du circuit, tandis que le circuit de contrôle est composé de petits contrôleurs afin de réduire l'augmentation de surface. La classification de [49] ne contient pas ce protocole mais la bonne opération du circuit est garantie en ajoutant des registres tampons dans le chemin de données. Quelques travaux [52,53] utilisent des protocoles 2 phases pour la conception du circuit de contrôle.

Cependant les seuls circuits étudiés sont des pipelines linéaires, ce qui limite fortement la portée de leur application.

Si les méthodes précédentes attribuent un contrôleur par groupe de registres, certains flots de désynchronisation utilisent une granularité moins fine. Srivastava *et al* [54] divisent le circuit en plusieurs régions de désynchronisation et chacune d'entre elles est dotée d'une paire de contrôleurs. De plus, les lignes de délais sont dépendantes des opérations et vont changer de valeur selon l'activité du chemin de données. Cette méthode améliore grandement la vitesse du circuit tout en diminuant l'augmentation de surface. Néanmoins, cette méthode n'est pas automatique.

Le flot open-source Edge [55] est l'approche la plus haut niveau et utilise seulement deux contrôleurs rebouclés. Un des contrôleurs active les *latches* maîtres tandis que le second contrôle les *latches* esclaves. Même si la cohérence entre les chemins de contrôle et de données est perdue, cette méthode réduit significativement l'augmentation de surface et continue d'apporter une plus grande robustesse aux variations PVT due au paradigme asynchrone.

Désynchronisation du chemin de contrôle Dans la section précédente, toutes les méthodes de désynchronisation présentées replacent uniquement l'arbre d'horloge par un circuit de contrôle, même si des changements mineurs peuvent être faits dans le chemin de données d'origine. Pourtant, ce n'est pas la seule façon d'obtenir un circuit désynchronisé. Une désynchronisation du chemin de contrôle supprime naturellement l'horloge. Mais ce type de méthode va aussi remplacer le chemin de contrôle du circuit, généralement une machine à états finis (FSM), pour mettre un circuit asynchrone à la place. Par conséquent, cette FSM asynchrone agit comme un circuit de contrôle et génère tous les signaux nécessaires pour le chemin de données.

Les AFSMs sont très bien étudiées et de nombreuses implémentations existent dans la littérature. Deux grandes familles sont présentées :

— Les AFSMs localement synchrones [15,16] déjà présentées précédemment. Ce type d'AFSM est très bien formalisé et l'implémentation associée est compacte. Néanmoins, l'utilisation de logique sans aléa peut être une limite car les outils de CAO ne supportent pas ce type de logique.

— Les AFSMs distribuées où chaque état est représenté par un élément mémorisant [56, 57]. Ces architectures utilisant l'encodage *one-hot* sont moins compactes que leurs homologues localement synchrones mais sont plus modulaires et plus faciles à implémenter.

Les AFSMs ne sont pas souvent utilisées pour gérer un chemin de données synchrone. À notre connaissance, seuls quelques travaux expérimentent cette architecture et peuvent être catégorisés suivant la famille d'AFSM choisie.

Certains travaux [58] utilisent une AFSM localement synchrone en remplacement de la FSM synchrone d'origine. Ils ciblent l'amélioration de la latence des circuits sur FPGA. Cependant, l'AFSM doit respecter les spécifications XBM et la transformation de synchrone à asynchrone nécessite d'importants post-traitements de la FSM initiale pour être conforme à ses spécifications. De plus, le chemin de données et l'AFSM doivent être compatibles l'un avec l'autre. Ce dernier point est effectué soit en modifiant le chemin de données soit en utilisant une version altérée des spécifications XBM.

Dans d'autres travaux, l'AFSM est implémentée avec une architecture distribuée. Iizuka *et al* [59] représentent chaque état à l'aide d'un module Q [60]. Tandis qu'un en-

semble de couches logicielles a été développé pour faciliter la conception de l'AFSM, il existe en revanche très peu de détails sur la méthodologie pour déterminer comment faire la conversion entre synchrone et asynchrone, ni sur la vérification de cette transformation. Enfin, Simatic *et al.* [61] utilisent quant à eux des contrôleurs asynchrones plus standards. Cette désynchronisation est associée à un outil de HLS académique, AUGH [62], pour concevoir les circuits plus rapidement.

Toutes les méthodes présentées dans cette partie peuvent être classées suivant la granularité utilisée dans le circuit de contrôle. De plus, ces méthodes sont intégrées dans des flots de conception plus ou moins automatiques afin de faciliter la génération des circuits asynchrones. Les principaux travaux de désynchronisation sont catégorisés suivant ces deux paramètres dans la figure 1.14.

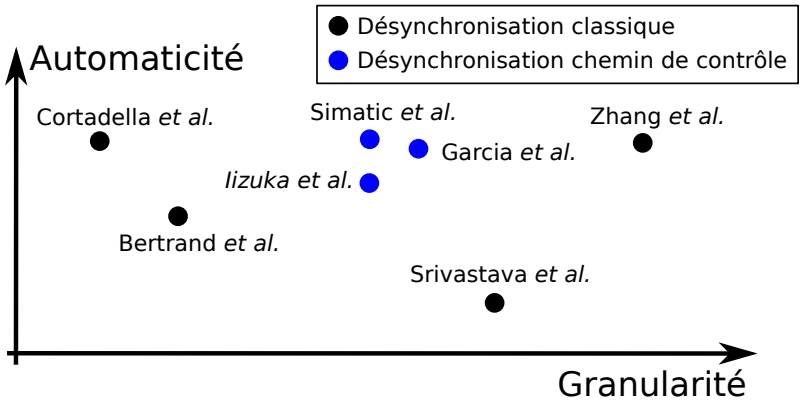


FIGURE 1.14 – *Classification des méthodes de désynchronisation.*

1.3.3 Vérification des contraintes temporelles

Pour valider le bon fonctionnement d'un circuit asynchrone, il est nécessaire de s'assurer que les contraintes temporelles locales sont respectées. Pour ce faire, il faut pouvoir calculer les différents délais au sein d'un circuit asynchrone de manière précise. Pour atteindre ce but, deux grandes tendances existent dans le domaine asynchrone.

Expression des contraintes temporelles

Dans les circuits à données groupées, les contraintes temporelles s'établissent entre chaque paire des contrôleurs du circuit de contrôle. Ces contraintes correspondent à la relation temporelle entre des chemins appartenant au chemin de données et au circuit de contrôle. On parle alors de contrainte temporelle relative, ou *Relative Timing Constraint* (RTC) [63]. En effet, elles reposent sur le fait que le chemin suivant la ligne de requête jusqu'à l'activation du prochain banc de registres est plus lent que le chemin de la logique combinatoire où sont calculées les données. Ces chemins possèdent un point de divergence, noté *pod*, qui sert de point de départ à la contrainte ainsi que deux points de convergence au niveau de chemin de données et de contrôle, respectivement poc_{data} et poc_{ctrl}. Une RTC s'écrit de la manière suivante :

$$pod \mapsto poc_{data} \prec poc_{ctrl}$$

Cette équation signifie qu'un événement se propageant de *pod* à poc_{ctrl} doit mettre un délai supérieur ou égal à la propagation d'un événement de *pod* à poc_{data}. Afin que la mémorisation des données soit correctement réalisée dans les éléments séquentiels, il est nécessaire que les données soient stables avant l'activation du registre et qu'elles restent stables un peu après son activation. Des marges doivent donc être ajoutées pour respecter ces délais supplémentaires. Ces temps sont respectivement les temps d'établissement et de maintien, plus communément appelés temps de *setup* et de *hold*, et sont des paramètres définis par les librairies de cellules standards.

Développement d'outils spécifiques

Afin de calculer les contraintes temporelles, et notamment les temps de cycle entre deux contrôleurs asynchrones, il est indispensable de manipuler des graphes de délais cycliques. Dans ce cadre, il est possible de calculer le temps exact d'arrivée d'un événement [64]. Par conséquent, plusieurs outils d'analyse temporelle ont été développés récemment [65, 66]. Ces deux approches sont très proches, et diffèrent seulement sur le format des fichiers utilisés : Xiromeritis *et al.* [65] exploitent les fichiers standards des flots de conception synchrones tandis que Hua *et al.* [66] utilisent des formats spécifiques.

Utilisation des outils commerciaux

Afin de concevoir un circuit asynchrone mais sans utiliser des outils spécifiques, qui peuvent ajouter des surcoûts de licence et de formation, une autre idée est d'utiliser les outils commerciaux. Ces outils de CAO sont bien connus des ingénieurs et ont été optimisés depuis de nombreuses années. En revanche, ils sont développés initialement pour concevoir des circuits synchrones.

Les circuits à données groupées sont la classe de circuits asynchrones la plus proche des circuits synchrones. Une fois le code RTL écrit, les questions majeures qui se posent sont celles de l'application des contraintes temporelles locales sur les chemins de données et de leurs vérifications. Dans les outils de CAO classiques, la vérification repose sur l'analyse temporelle statique ou *Static Timing Analysis* (STA). Cette analyse vérifie que chaque chemin combinatoire entre deux registres possède un temps inférieur à la période de l'horloge.

Dans le cadre des circuits asynchrones, plusieurs problèmes apparaissent lors de l'application de la STA. Il existe deux grandes difficultés :

— D'une part, les boucles combinatoires ne sont pas acceptées par les outils de STA. Or, les circuits de contrôle possèdent un grand nombre de ces boucles, qu'elles soient internes aux contrôleurs ou architecturales lorsque le circuit est non linéaire. Il faut donc trouver un moyen de les couper efficacement.

— D'autre part, et c'est là le problème majeur, la STA repose sur un signal global pour vérifier tous les chemins présents dans le chemin de données. Il est donc nécessaire de définir des chemins locaux pour vérifier que les délais du circuit de contrôle sont adaptés au chemin de données associé.

Ces problèmes doivent impérativement être résolus pour effectuer correctement une STA sur un circuit à données groupées. Plusieurs méthodes ont été développées pour réaliser cette tâche, que nous allons présenter.

Méthode min/max Cette méthode repose sur l'utilisation de commandes non conventionnelles pour contraindre les circuits asynchrones et représenter les RTCs. Les boucles combinatoires sont coupées en désactivant les arcs temporels à l'aide de la commande *set_disable_timing*. Ensuite, les hypothèses temporelles sont appliquées, comme montrées en figure 1.15, en contraignant à la fois les chemins de données et de contrôle avec les commandes *set_min_delay* et *set_max_delay*. Un délai maximal est défini sur la logique combinatoire du chemin de données pour contraindre les performances du circuit et ce dernier détermine le délai minimal à appliquer sur la ligne de requête associée au circuit de contrôle pour garantir le bon fonctionnement du circuit.

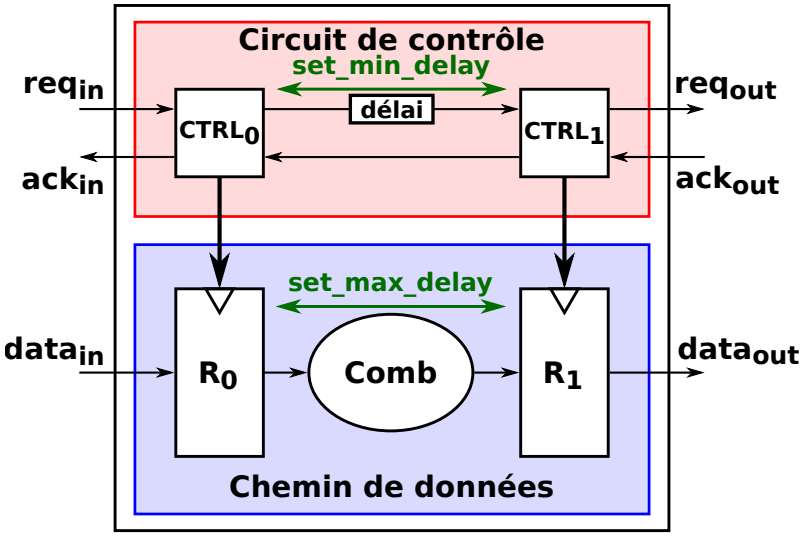


FIGURE 1.15 – *Contraintes temporelles min/max dans un circuit à données groupées.*

Plusieurs travaux ont réussi à intégrer cette méthode dans un flot numérique global [67–69] avec quelques différences entre eux, comme l'utilisation d'horloges virtuelles à la place dans le chemin de données dans [68]. De plus, pour s'assurer de la bonne génération des contraintes, Stevens *et al.* [67] proposent une méthode formelle pour adapter les hypothèses temporelles au format des outils de CAO.

Pourtant, cette méthode possède plusieurs désavantages. Tout d'abord, l'ensemble des chemins n'est pas vérifié tout comme les temps de *setup* et *hold*, mais seulement la partie combinatoire et la ligne de requête. Il est donc toujours possible d'avoir des violations si une marge assez importante n'est pas prise sur la ligne de requête. De plus, cette méthode est dépendante des conditions PVT, et nécessite donc une nouvelle génération des contraintes pour chaque condition (*corner*) étudiée. Ceci peut devenir complexe dans les technologies CMOS avancées qui possèdent un grand nombre de conditions PVT à vérifier.

Méthode LCS Une méthode récente proposée par Gimenez *et al.* [70], appelée *Local Clock Sets* (LCS), exploite les horloges pour réaliser des STAs de manière locale dans un circuit à données groupées.

Afin de désactiver les boucles combinatoires du circuit de contrôle, des horloges maîtres sont déclarées au sein de chaque contrôleur (les losanges verts sur la figure 1.16). Ensuite, les chemins nécessaires à l'outil de CAO pour réaliser la STA doivent être construits : le chemin de lancement et de capture représente respectivement la propagation des données dans le chemin de données et le chemin d'activation des registres. Le chemin de

lancement est généré directement par l'horloge maître, représenté par la flèche verte sur la figure 1.16 partant du contrôleur $ctrl_0$ allant jusqu'au registre R_1. Le chemin de capture, qui passe lui par le circuit de contrôle, doit être propagé à travers l'horloge maître du contrôleur $ctrl_1$. Pour cela, la méthode LCS utilise une horloge générée, représentée par la flèche en pointillé. De manière plus générale, le chemin d'horloge générée est défini pour construire le chemin jusqu'au banc de registre. De cette manière et contrairement à la méthode précédente, les contraintes temporelles sont vérifiées sur l'ensemble du chemin et l'outil peut prendre en compte les temps de *setup* et *hold* des registres. Par ailleurs, la méthode LCS est indépendante des conditions PVT, tout comme dans le cas synchrone, et peut aussi être utilisée pour contraindre le chemin de données suivant une fréquence cible.

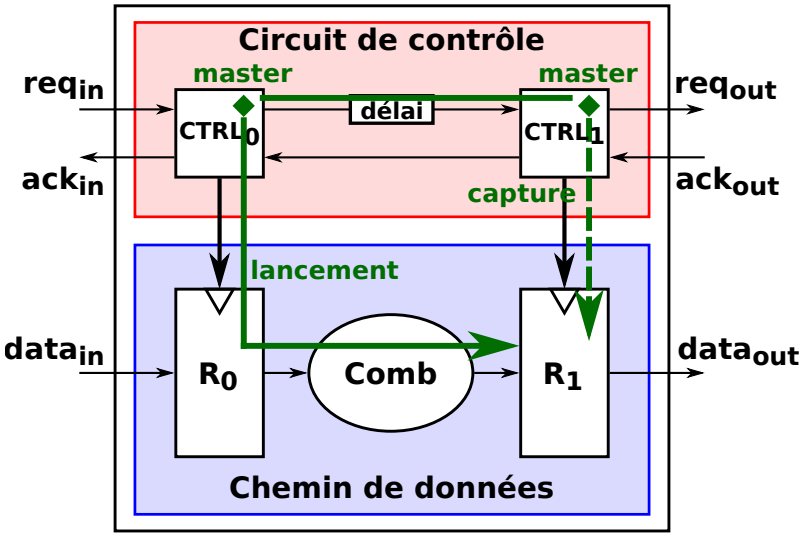


FIGURE 1.16 – *Méthode LCS dans un circuit à données groupées.*

Pour conclure, cette méthode prometteuse permet de profiter de toutes les avancées faites dans le domaine synchrone au niveau STA dans le domaine asynchrone. Les règles de construction des LCS peuvent être extraites d'un STG et peuvent s'adapter à n'importe quel type de contraintes temporelles ou d'implémentations [71]. Il reste en revanche à intégrer cette méthode dans un flot de conception global pour la rendre automatique car l'écriture des LCS pour un circuit peut être fastidieuse, ainsi qu'à assurer de manière formelle la génération des contraintes temporelles comme dans [67].

1.4 Synthèse de haut niveau

Les outils de HLS prennent en entrée la description haut niveau d'un circuit sans information d'architecture ou de délais [72]. Cette description est synthétisée en un code RTL. De plus, la HLS est capable d'effectuer une exploration des architectures possibles afin d'optimiser le circuit suivant des contraintes spécifiques. Dans le contexte de l'asynchrone, la HLS est une voie prometteuse pour générer rapidement des circuits habituellement considérés comme difficile à concevoir.

1.4.1 Principe de la HLS synchrone

La HLS s'est imposée ces dernières années pour concevoir des circuits complexes rapidement [73] dans des domaines où les algorithmes changent régulièrement, et où l'optimisation des algorithmes est prioritaire. Un grand nombre d'outils de HLS ont été développés ces dernières années dans cette optique, qu'ils soient industriels comme Catapult HLS [74] ou Vivado HLS, ou bien académiques comme LegUp [75] et AUGH [62].

Tout d'abord, la description haut niveau, généralement un sous-ensemble de C++ ou en SytsemC, doit être analysée pour en extraire le graphe des flots de contrôle et données (CDFG). Ce graphe contient l'ensemble des opérations réalisées dans le programme ainsi que leurs dépendances. Une fois que le CDFG du programme a été construit, la HLS suit trois étapes principales [76] pour générer le code RTL d'un circuit :

— L'allocation définit le nombre d'opérateurs qui seront disponibles dans le circuit. Ces opérateurs correspondent à des opérateurs arithmétiques et logiques ou bien à des mémoires.

— L'ordonnancement définit les opérations qui seront exécutées à chaque cycle d'horloge, en respectant les relations de dépendance de données. Cet ordonnancement est réalisé par rapport aux contraintes architecturales ou temporelles données par l'utilisateur. C'est cette étape qui génère la FSM du circuit.

— L'affectation attribue à chaque opérateur une opération pour chaque cycle de calcul. Cette dernière étape définit donc les interconnexions entre les différents composants du chemin de données.

Les différentes étapes présentées sont effectuées les unes après les autres, car leur exécution simultanée serait trop longue. Il est possible de réaliser l'allocation et l'ordonnancement dans un ordre différent suivant l'objectif souhaité, par exemple maximiser la latence du circuit sous une contrainte de taille (pour l'ordre présenté ici). Les circuits obtenus par les outils de HLS présentent une division claire entre le chemin de données, qui traite les données, et le chemin de contrôle qui contrôle les opérations du circuit.

1.4.2 HLS asynchrone

Dans le domaine asynchrone, plusieurs outils et méthodes de HLS ont été développés [77]. Effectivement, la génération de circuits asynchrones à partir de description haut niveau peut aider les concepteurs synchrones à pallier à leur manque de compétences dans le domaine asynchrone en masquant certaines spécificités architecturales. Ces méthodes de HLS asynchrones peuvent être classées en trois grandes familles.

Décomposition en processus pipelinés

Plusieurs outils de HLS asynchrones vont utiliser une méthode d'apparence simple pour générer un circuit : le graphe de données est extrait de la description algorithmique haut niveau, puis optimisé pour être indépendant de la syntaxe. Chaque opération du graphe est alors implémentée en un bloc matériel. Les circuits générés sont fortement pipelinés et possèdent de forts débits. La décomposition dirigée par les données [78], ou *Data-Driven Decomposition*, ainsi que le flot de Venkataramani *et al.* [79] utilisent tous deux cette stratégie, quoique qu'avec des formes d'affectation différentes (respectivement dynamique et statique [80]). Plus récemment, l'outil Fluid [81] génère des circuits

à données groupés à partir d'un code C tout en supportant des graphes de données plus complexes que les travaux précédents. Si ces outils permettent de concevoir des circuits très performants, ils sont moins adaptés pour obtenir des circuits à faible consommation.

Traduction dirigée par la syntaxe

Ces solutions proposent une surcouche comportementale à certains langages HDLs développés dans le domaine asynchrone, plus particulièrement pour Haste. Nielsen *et al.* [82, 83] utilisent les étapes classiques de la HLS synchrone pour générer, à partir d'une description comportementale en Haste, une description matérielle Haste qui peut être synthétisée avec l'outil TiDE. Le circuit obtenu, décomposé en FSMs et chemins de données, est optimisé en taille et vitesse suivant les contraintes de l'utilisateur. Ces descriptions haut niveau permettent de faciliter la conception avec ces langages, même si leur apprentissage reste une barrière. Pour atténuer ce problème, des optimisations additionnelles existent comme celles d'Hansen [84] ou de Tranchero *et al.* [85] qui ont décidé d'utiliser un langage plus répandu, Simulink.

Flots basés sur l'ordonnancement

Une autre voie explorée dans la HLS asynchrone est très similaire au domaine synchrone et se base sur les mêmes étapes : l'allocation, l'ordonnancement et l'affectation. En revanche, cette méthode soulève plusieurs problèmes lorsqu'elle est appliquée pour des circuits asynchrones.

Tout d'abord, cette stratégie utilise très majoritairement une décomposition des circuits en chemins de données et en circuits de contrôle, la FSM. A priori, cette décomposition n'est pas naturelle pour les circuits asynchrones, qui reposent sur des synchronisations locales entre sous-blocs du circuit. Mais les quelques travaux [86, 87] qui ont tenté d'appliquer un circuit de contrôle distribué sur les circuits obtenus montrent des résultats assez mauvais pour des circuits à données groupées avec notamment une augmentation conséquente de la surface et des circuits de contrôle complexes. C'est pourquoi les solutions proposées dans la littérature utilisent des circuits de contrôle centralisés, c'est-à-dire des AFSMs, avec de bien meilleurs résultats.

L'autre problématique correspond à l'algorithme d'ordonnancement. En effet, les algorithmes classiques d'ordonnancement, comme utilisé dans [82], utilise une discrétisation constante du temps. Ce paradigme, adapté aux circuits synchrones, l'est beaucoup moins pour les circuits asynchrones et peut mener à des ordonnancements sous-optimaux des opérations. Pour avoir des circuits asynchrones correctement optimisés, il est par conséquent nécessaire de développer de nouveaux algorithmes d'ordonnancement.

Quelques algorithmes originaux ont été proposés au cours des dernières années [86, 88, 89]. Ces algorithmes sont parfois implémentés dans des outils [88, 90] prouvant la faisabilité de ces méthodes, mais il manque encore des études comparatives sur ceux-ci pour caractériser les circuits obtenus.

1.5 Conclusion

Si les circuits asynchrones sont étudiés depuis longtemps et qu'une grande variété de classes a été développée, leur conception reste encore pour l'heure loin d'être triviale ce qui freine leur déploiement dans l'industrie. En effet, concevoir des circuits asynchrones conséquents induit l'utilisation de flots loin d'être standards, avec des logiciels de CAO et des langages matériels spécifiques. Les quelques solutions utilisant des logiciels de CAO standards reposent sur des commandes qu'il est difficile d'utiliser et de passer à l'échelle pour des circuits complexes.

Nous souhaitons dans cette thèse aborder la question de la conception des circuits asynchrones et la rendre plus propice à une large acceptation dans le domaine industriel. Pour ce faire, nous utiliserons uniquement des outils de CAO de l'écosystème industriel, et donc synchrone. De plus, pour faciliter la conception de grands circuits asynchrones, il est nécessaire de mettre en place une méthode rapide et haut niveau pour développer des circuits asynchrones. Les circuits à données groupées représentent une classe asynchrone prometteuse pour remplir ces deux objectifs.

2
Désynchronisation de circuits

Sommaire

2.1 Introduction 30
2.2 Structure du circuit 30
2.2.1 Définition structurelle 31
2.2.2 Chemin de données 33
2.2.3 Chemin de contrôle 33
2.3 Méthode de désynchronisation 35
2.3.1 FSM asynchrone 36
2.3.2 Preuve de correction 40
2.3.3 Contraintes temporelles 44
2.4 Application à Catapult HLS 46
2.4.1 Architecture des circuits 46
2.4.2 Génération des contraintes 47
2.5 Conclusion 48

2.1 Introduction

Le flot de conception de ce travail de thèse permet d'obtenir rapidement et facilement un circuit asynchrone. Notre choix s'est donc porté naturellement sur la synthèse de haut niveau. Néanmoins, utiliser un outil de HLS spécifique se traduit par un surcoût en termes de développement logiciel ainsi qu'une formation spécifique des ingénieurs. Pour éviter ces surcoûts, nous avons choisi de travailler à partir d'un outil de HLS synchrone utilisé dans l'industrie, Catapult HLS de Siemens EDA, puis de désynchroniser le circuit obtenu. Ceci nous permet d'exploiter la puissance d'un outil industriel pour la génération de circuits asynchrones ainsi que d'éviter l'apprentissage d'un nouveau langage pour les concepteurs synchrones.

Dans ce cadre, les circuits à données groupées sont de très bons candidats pour être générés à partir de circuits synchrones car les deux types de circuits possèdent une grande proximité. Parmi les méthodes de désynchronisation présentées dans le chapitre précédent en section 1.3.2, nous souhaitons développer une méthode automatique avec une granularité moyenne (voir figure 1.14). Ce choix permet de limiter l'augmentation de surface du circuit tout en bénéficiant de la robustesse des circuits asynchrones. De plus, les circuits issus des outils de HLS possèdent une structure bien définie avec un chemin de données contrôlé par une FSM. Les méthodes de désynchronisation ciblant le chemin de contrôle sont donc des candidates idéales.

Par conséquent, nous allons reprendre la méthode présentée en [61]. Celle-ci remplace la FSM d'origine par une AFSM utilisant un encodage *one-hot*, où chaque état de la FSM est représenté par un contrôleur classique des circuits à données groupées. Une preuve de concept de cette méthode a été réalisée sur l'outil académique de HLS AUGH [62] mais il reste à démontrer la validité de la conversion. Notre travail propose donc une formalisation du circuit et de la méthode en vue d'une preuve formelle, dont la présentation est faite succinctement.

L'intégration de la méthode de désynchronisation à l'outil industriel Catapult HLS a mené au flot présenté en figure 2.1, prouvant ainsi la faisabilité de notre approche dans un cadre industriel qui se veut plus complet et plus complexe. La méthode repose sur une représentation intermédiaire sous forme de graphe d'états. L'extraction de la FSM synchrone ainsi que sa conversion en AFSM sont réalisées au niveau RTL pour faciliter la vérification de la fonctionnalité du circuit désynchronisé. De plus, afin de faciliter la conception des circuits asynchrones dans les étapes telles que la synthèse et le Placement/Routage (PnR), les contraintes temporelles du circuit asynchrone sont générées suivant la méthode LCS à partir du graphe d'états. Il s'agit à notre connaissance de la première intégration de la méthode LCS à un flot de conception asynchrone.

2.2 Structure du circuit

Les circuits générés par les outils de HLS ont généralement une structure stricte : ils contiennent un chemin de données et un chemin de contrôle, correspondant à une FSM. Nous allons maintenant modéliser ces deux parties.

Soit $\mathscr{C}$ un circuit synchrone (voir figure 2.2). Celui-ci interagit avec un environnement $\mathscr{E}$ à travers les signaux I, O, *ready* et *valid*. Les signaux *ready* et *valid* sont des signaux

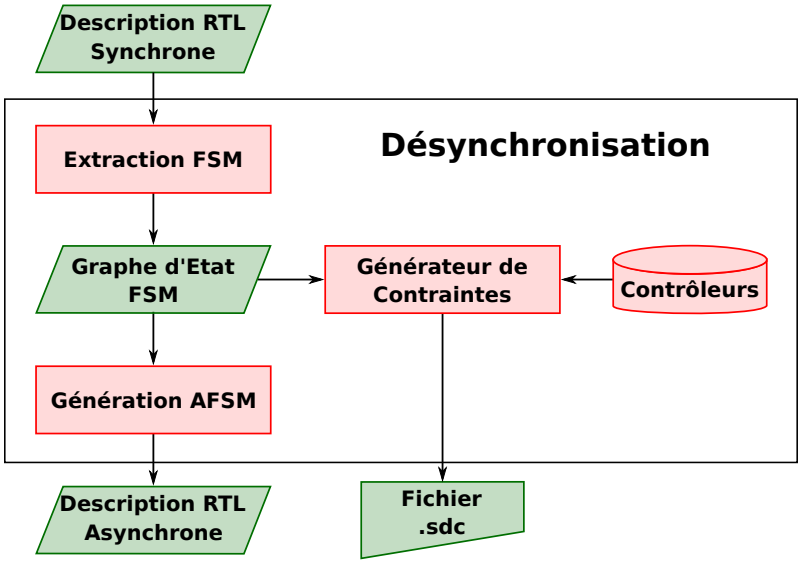


FIGURE 2.1 – *Flot de désynchronisation.*

de contrôle, et I et O des signaux de données. L'entrée I est uniquement lue par le circuit lorsque le signal d'entrée *ready* passe à un. De même, les données de sortie O sont valides quand le signal de sortie *valid* est actif.

Le signal d'horloge est noté *clock* (en vert sur la figure 2.2). Le circuit $\mathscr{C}$ est décomposé en deux blocs : un bloc de chemin de contrôle *FSM* (en rouge), et le bloc du chemin de données *DP* (en bleu). Ces deux blocs communiquent par les signaux Λ et Δ. Les signaux générés par la FSM, Λ_{en} et Λ_{comb}, contrôlent le chemin de données. Les signaux Δ_{in} issus du chemin de données permettent de choisir les prochains états de la FSM.

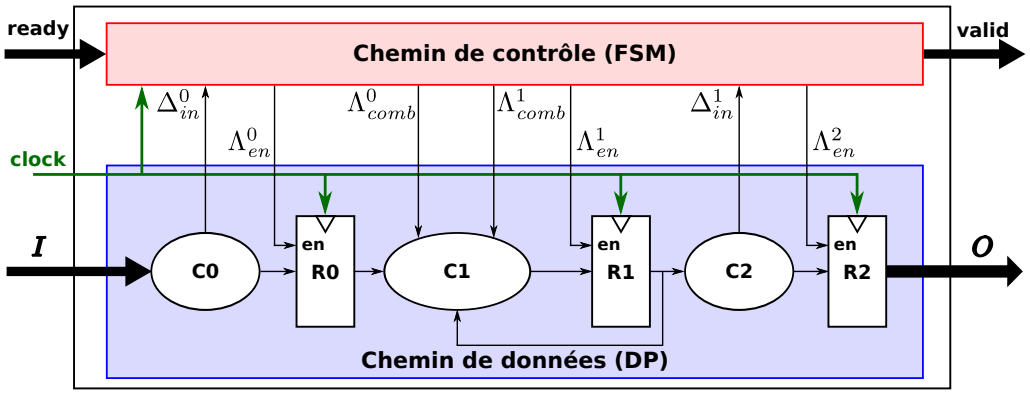


FIGURE 2.2 – *Architecture d'un circuit synchrone.*

2.2.1 Définition structurelle

Chaque bloc composant le circuit est défini par un graphe orienté $G = (V, E)$ où les noeuds de V correspondent soit à un registre, soit à une fonction combinatoire et les arcs de E correspondent à des signaux. Plus précisément :

— L'ensemble des noeuds V est partitionné en deux ensembles V_{comb} et R. Ces derniers représentent respectivement les blocs combinatoires et séquentiels du circuit et sont

définis de la manière suivante :

$$\begin{aligned} V &= V_{comb} \cup R \\ V_{comb} &\cap R = \emptyset \end{aligned} \tag{2.1}$$

Nous notons V^* l'extension de l'ensemble V à l'environnement du circuit et à l'autre bloc du circuit, soit *DP* soit *FSM* suivant le cas. Dans ce cadre, les ensembles $\mathscr{E}$ et $\mathscr{C} \setminus \{V\}$ sont alors vus comme des singletons :

$$V^* = V \cup \{\mathscr{E}, \mathscr{C} \setminus \{V\}\} \tag{2.2}$$

— L'ensemble des arcs du graphe $E \subseteq V^* \times V^*$ se partitionne en trois sous-ensembles :
  — Les *arcs de données* relient les fonctions combinatoires et les registres. Toutes les combinaisons de sommet sont possibles dans l'ensemble $V \times V$.
  — Les *arcs inter-blocs* connectent les blocs *DP* et *FSM*.
  — Les *arcs primaires* connectent l'environnement $\mathscr{E}$ au reste du graphe.

La figure 2.3 présente les caractéristiques de chaque type de sommets. Soit r^i le i^{ieme} sommet de R. Celui-ci a quatre arcs entrants notés r^i_{clk}, r^i_{rst}, r^i_{en}, r^i_{in} et un arc sortant noté r^i_{out}. Les arcs r^i_{clk} et r^i_{rst} sont des *arcs primaires* qui correspondent au signal d'horloge *clock* et de *reset* [1]. L'arc r^i_{en} représente le signal *enable* d'un registre et r^i_{in} et r^i_{out} sont respectivement les données entrantes et sortantes. Par abus de langage, nous utiliserons dans la suite de ce travail les notations R_{in} et R_{out} pour parler des vecteurs formés par les arcs respectivement entrants et sortants de l'ensemble des sommets appartenant à R. Pour un sommet v^i appartenant à V_{comb}, les arcs entrants et sortants sont respectivement notés $v^i_{in,j}$ et $v^i_{out,j}$ (où j est l'indice des entrées ou des sorties selon le cas).

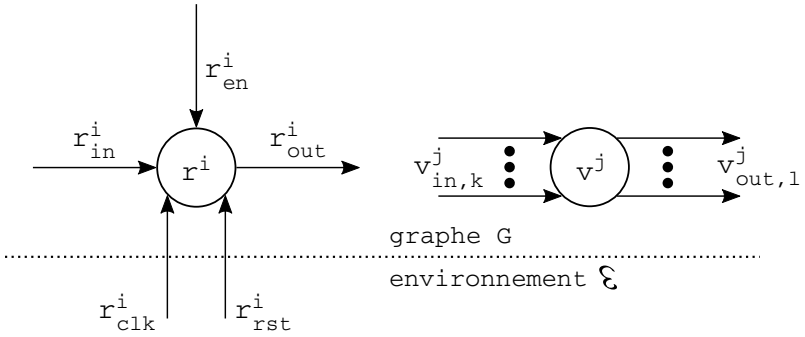


FIGURE 2.3 – *Définition des sommets r et v d'un graphe orienté G.*

Soit P un ensemble quelconque de signaux : $s(t)$ est défini comme étant la valeur booléenne du signal s dans P au moment t. Dans un circuit synchrone, le temps t correspond au nombre de fronts montants sur le signal d'horloge. En revanche, dans un circuit asynchrone, t correspond au nombre d'évènements sur un signal de contrôle.

La sémantique du circuit est définie par chacun des arcs sortants. Soit r^i le i^{ieme} sommet appartenant à R, alors la sémantique de r^i_{out} est donnée par l'équation suivante :

1. Pour une question de simplicité, nous supposerons que le signal de *reset* est asynchrone.

$$\begin{aligned} r^i_{out}(t) &= r^i_{in}(t-1) \text{ si } r^i_{en}(t) = 1 \\ & \quad r^i_{out}(t-1) \text{ sinon} \end{aligned} \tag{2.3}$$

Si v^i est le i^{ieme} sommet appartenant à V_{comb}, alors la sémantique d'un arc sortant $v^i_{out,j}$ est donnée par la fonction combinatoire f_j selon :

$$v^i_{out,j}(t) = f_j(v^i_{in,k}(t), \dots) \tag{2.4}$$

Pour le moment, toutes les portes logiques du circuit sont considérées comme parfaites et sans délai. De plus, nous ne ferons pas de distinction par la suite entre les sommets du graphe et les blocs combinatoires ou séquentiels, ainsi qu'entre les arcs et les signaux. Maintenant que le modèle structurel a été défini pour un bloc quelconque, nous allons appliquer ce modèle de graphe à chacun des blocs spécifiques du circuit, *DP* et *FSM*.

2.2.2 Chemin de données

Pour chaque sommet r^i du bloc chemin de données *DP*, l'arc entrant r^i_{en} est un *arc inter-blocs* venant de la FSM, correspondant aux signaux Λ_{en} sur la figure 2.2. Les arcs r^i_{in} et r^i_{out} sont quant à eux des *arcs de données*. Pour les sommets v^i de *DP*, les signaux entrants et sortants $v^i_{in,j}$ et $v^i_{out,j}$ peuvent être indistinctement des *arcs de données* ou des *arcs inter-blocs*. Si ce sont des *arcs inter-blocs*, ils correspondent aux signaux Λ_{comb} pour les arcs d'entrées et Δ_{in} pour les sorties.

2.2.3 Chemin de contrôle

Modèle du chemin de contrôle

Une machine de Moore à états finis est définie par le 6-uplet $FSM = (S, s_0, I_F, O_F, \delta, \lambda)$ où :

- S est un ensemble de n états. L'état s_i se réfère au i^{ieme} état de la FSM.
- s_0 correspond à l'état initial de la FSM.
- I_F est l'ensemble des signaux d'entrée de la FSM.
- O_F est l'ensemble des signaux de sortie de la FSM.
- $\delta : I_F \times S \to S$ la fonction de transition d'états de la FSM (dépendant de l'état courant et des valeurs d'entrées).
- $\lambda : S \to O_F$ est la fonction de calcul des sorties de la FSM suivant l'état courant.

Nous nous concentrons sur une catégorie de circuits où les entrées I du chemin de données *DP* sont capturées dans l'état initial (où *ready* vaut '1') et les sorties O de *DP* sont valides dans le dernier état (où *valid* vaut '1'). Par conséquent, nous supposons que toutes les FSMs possèdent des états spécifiques, appelés *wait* et *end*, tel que :

$$\begin{aligned} \delta(end, x) &= wait, \forall x \in I_F \\ \delta(wait, \neg ready) &= wait \\ \lambda(end) &= valid \\ \lambda(s) &= \neg valid, \forall s \in S \wedge s \neq end \end{aligned}$$

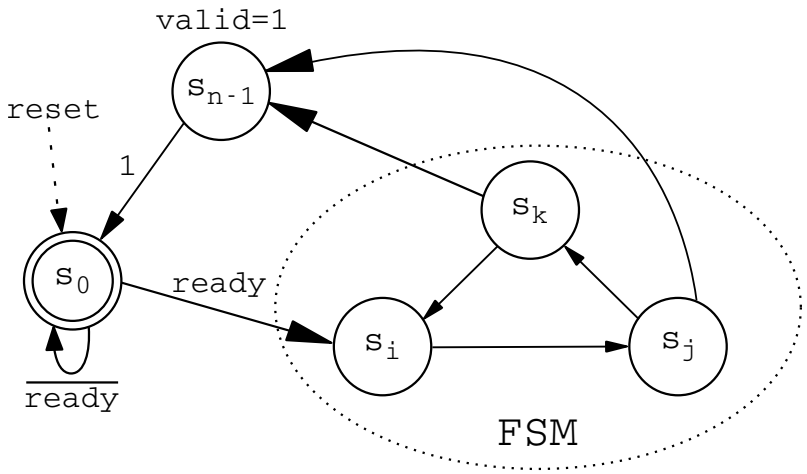


FIGURE 2.4 – *Exemple d'une FSM issue d'une synthèse HLS.*

La figure 2.4 présente un telle FSM. Celle-ci est composée de n états $\{s_0, \ldots, s_{n-1}\}$. L'état s_0 correspond à l'état initial *wait*. Lorsque le signal *ready* est au niveau haut, les entrées sont récupérées et le circuit commence son calcul. À la fin du calcul, l'état s_{n-1}, qui représente l'état *end*, est activé pour un cycle et le signal de sortie *valid* passe à '1'.

Synthèse synchrone

Afin de faciliter l'étape de désynchronisation, nous supposerons que l'encodage de la FSM est en *one-hot*. Cette hypothèse permet de simplifier la modélisation de notre méthode mais ne restreint pas le champ de notre travail : il est en effet toujours possible de convertir l'encodage d'une FSM en encodage *one-hot*. La structure matérielle de la FSM est définie par un graphe orienté (V_F, E_F) (voir la figure 2.5). Chaque sommet représente un bloc matériel. L'ensemble de ces sommets est partitionné de 5 sous-ensembles : R, Δ, V_{demux}, Λ_{en}, Λ_{comb}. Ceux-ci sont définis comme suit :

— **Bloc R :** Ce bloc correspond au banc de registres de la FSM. Il contient n registres, où r^i est le i^{ieme} registre de R. Ces registres représentent l'implémentation matérielle de l'ensemble S et représentent toutes les éléments appartenant à celui-ci. Comme nous utilisons un encodage *one-hot*, la sémantique satisfait donc la propriété suivante :

Propriété 2.1. *À tout instant t, un seul registre r de R à la valeur '1' et tous les autres auront la valeur '0'.*

$$\begin{aligned} \forall t, \quad & \exists i, r^i_{out}(t) = 1 \quad \wedge \quad \forall j \neq i, r^j_{out}(t) = 0 \\ \wedge \quad & \exists k, r^k_{in}(t) = 1 \quad \wedge \quad \forall l \neq k, r^l_{in}(t) = 0 \end{aligned}$$

L'indice i est donné par la valeur de $R_{out}(t)$. L'indice k est calculé suivant les valeurs de i et de la fonction δ.

Le bloc R est relié aux blocs Δ, Λ_{en} et Λ_{comb} grâce au vecteur R_{out}. Les entrées de R, le vecteur R_{in}, sont déterminées par le bloc V_{demux}.

— **Blocs Δ et V_{demux} :** La combinaison de ces deux sommets implémente la fonction δ. En d'autres termes, Δ calcule la valeur du prochain état de la FSM en binaire

et V_{demux} va convertir cette dernière en valeur unaire et générer le prochain vecteur d'état de la FSM. Le sommet V_{demux} possède n arcs sortants, connectés à chacun des registres de R.

— **Blocs Λ_{en} et Λ_{comb} :** Ces deux sommets implémentent la fonction λ qui calcule les sorties de la FSM. Celle-ci contient deux sous-fonctions : le bloc Λ_{comb} dont les sorties sont connectées aux entrées combinatoires du chemin de données, et le bloc Λ_{en} qui est connecté aux ports *enable* du chemin de données. Etant donné que l'encodage de la FSM est en *one-hot*, chaque sortie est une disjonction des registres r^j_{out} du bloc R :

$$\begin{aligned} \Lambda^i_{comb} &= \Sigma_{j \in S_{i,comb}} r^j_{out} \\ \Lambda^i_{en} &= \Sigma_{j \in S_{i,en}} r^j_{out} \end{aligned} \tag{2.5}$$

où $S_{i,en}$ et $S_{i,en}$ sont des sous-ensembles d'indice dépendant de i.

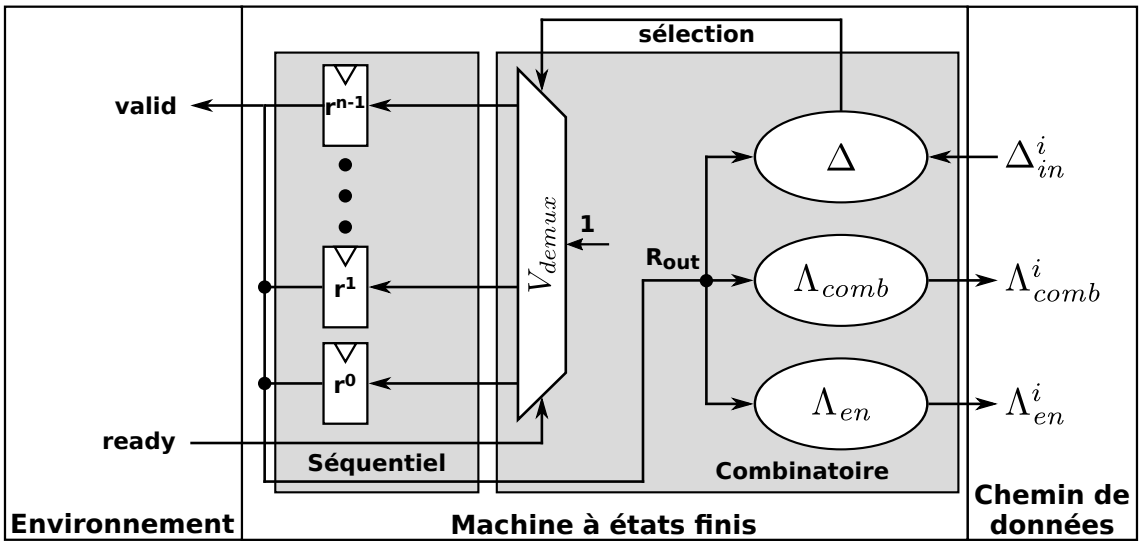


FIGURE 2.5 – *Implémentation d'une FSM synchrone avec un encodage one-hot.*

2.3 Méthode de désynchronisation

Désynchroniser un circuit signifie remplacer la synchronisation globale de ce circuit (l'horloge) par un mécanisme de synchronisation locale, c'est-à-dire un ensemble de contrôleurs distribués localement. La méthode proposée est très simple car seulement la FSM est modifiée et convertie en une AFSM. Tout comme la FSM originale, cette AFSM génère les signaux qui sont connectés au chemin de données. Mais en plus, elle génère aussi les signaux de synchronisation locaux qui sont connectés aux signaux d'horloge des registres dans le chemin de données (voir la figure 2.6).

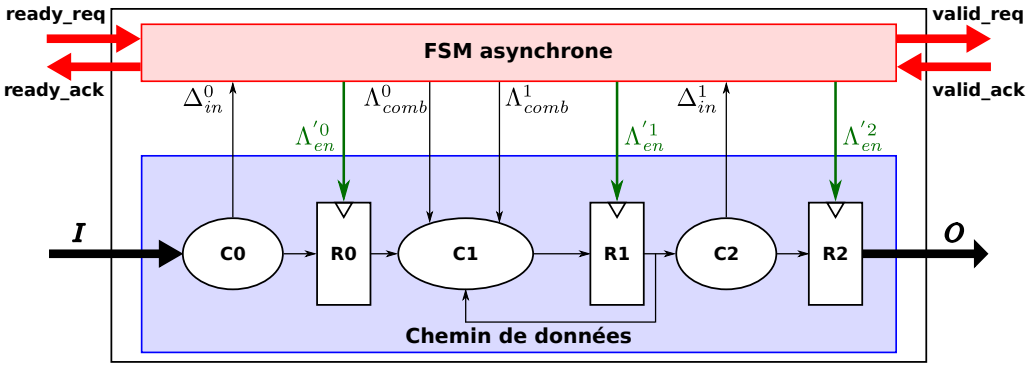


FIGURE 2.6 – *Circuit désynchronisé avec son chemin de données et une AFSM.*

2.3.1 FSM asynchrone

À partir de notre modèle de FSM, nous souhaitons synthétiser une FSM asynchrone équivalente à la FSM d'origine. La structure matérielle de notre AFSM est un graphe orienté (V'_F, E'_F) (voir figure 2.7). Ce dernier est directement dérivé de (V_F, E_F). L'ensemble des sommets est partitionné en neuf sous-ensembles C, V'_{demux}, V_{merge}, $Cint$, Δ, Λ'_{en}, Λ_{comb}, V_{ready} et V_{valid}.

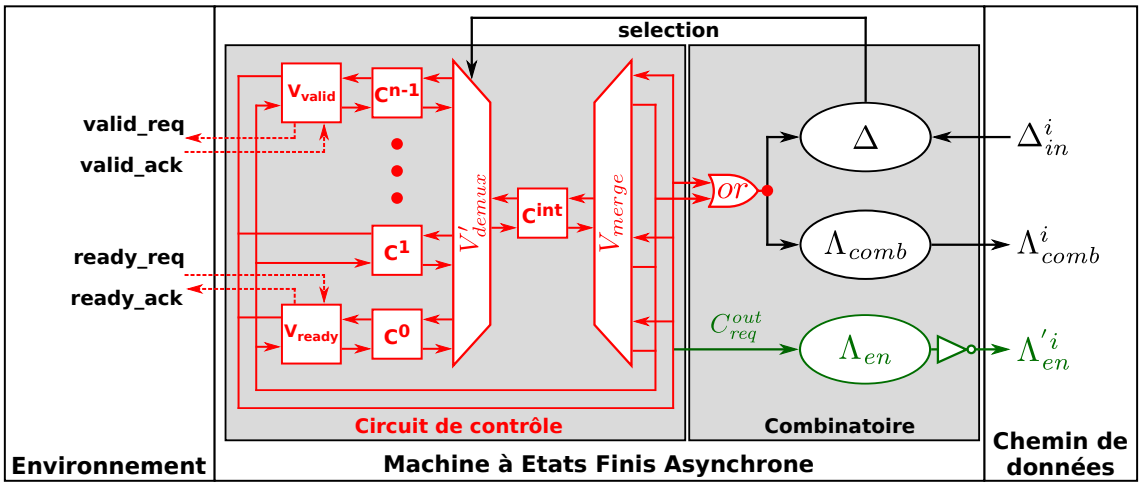


FIGURE 2.7 – *Implémentation d'une FSM désynchronisée.*

Contrôleurs locaux C

Dans un circuit à données groupées, l'élément de base du circuit de contrôle est le contrôleur, qui synchronise localement le circuit. Un contrôleur est un composant matériel avec des interfaces d'entrée et de sortie comme présenté sur la figure 2.8. Soit c^i le i^{ieme} contrôleur de C. Chaque canal de communication est composé de deux signaux, un signal d'entrée et un de sortie, notés entre parenthèses sous la forme (*req*,*ack*). L'interface d'entrée correspond à $(c^{i,req}_{in}, c^{i,ack}_{in})$, où $c^{i,req}_{in}$ est un signal d'entrée et $c^{i,ack}_{in}$ un signal de sortie. De manière analogue, l'interface de sortie est définie par $(c^{i,req}_{out}, c^{i,ack}_{out})$ où les signaux $c^{i,req}_{out}$ et $c^{i,ack}_{out}$ sont respectivement les signaux de sortie et d'entrée. Tout comme pour les registres dans la section précédente, nous définissons C_{in} et C_{out} pour parler des

vecteurs composés des interfaces respectivement entrantes et sortantes de l'ensemble des contrôleurs appartenant à C.

Ces contrôleurs communiquent grâce à un protocole à 4 phases. Les interactions entre les interfaces d'entrée et de sortie peuvent être définies par différents protocoles (i.e. WCHB, *late-capture*, ...). Ces derniers peuvent être décrits selon un STG [28] comme expliqué en section 1.3.1.

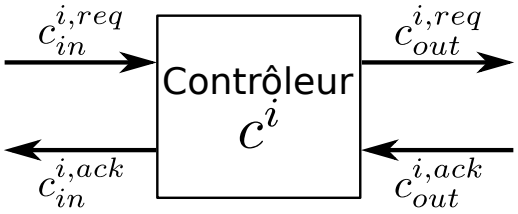
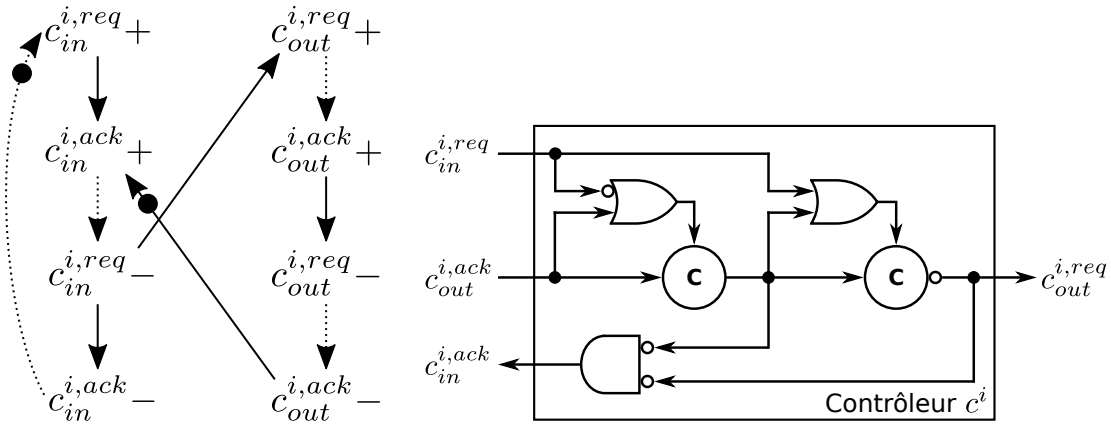


FIGURE 2.8 – *Modèle de contrôleur.*

Dans ce travail, nous utiliserons le protocole *late-capture* [40] dont le STG est décrit figure 2.9a. Dans ce protocole dit tardif, les fronts montants et descendants du signal de requête sont propagés dans la ligne de délais avant l'activation des registres. De cette manière, le bloc de délai est seulement la moitié de celui requis pour un protocole classique à 4 phases. De plus, il découple le protocole d'entrée et de sortie du contrôleur pour augmenter la vitesse de la communication. Par conséquent, le protocole *late-capture* attend la désactivation de la requête $c_{in}^{i,req}$ avant d'activer la requête de sortie $c_{out}^{i,req}$. La figure 2.9b présente l'implémentation du contrôleur qui découle du STG.



(a) STG du late-capture. *(b) Implémentation du contrôleur.*

FIGURE 2.9 – *Protocole late-capture.*

Blocs V_{merge} et V'_{demux}

Ces deux blocs sont implémentés suivant le protocole *late-capture*. Les figures 2.10 et 2.11 présentent leur architecture. Le bloc V_{merge} combine l'ensemble des signaux de requête d'entrée en une seule requête. Lorsque ce composant reçoit un signal d'acquittement, il transfère ce signal au canal avec la requête activée. Le bloc V'_{demux} est quant à lui assez similaire au composant dans l'architecture synchrone. Ce bloc convertit le signal de sélection binaire en un signal unaire. Ensuite, le chemin sélectionné est activé tout en respectant le protocole *late-capture*.

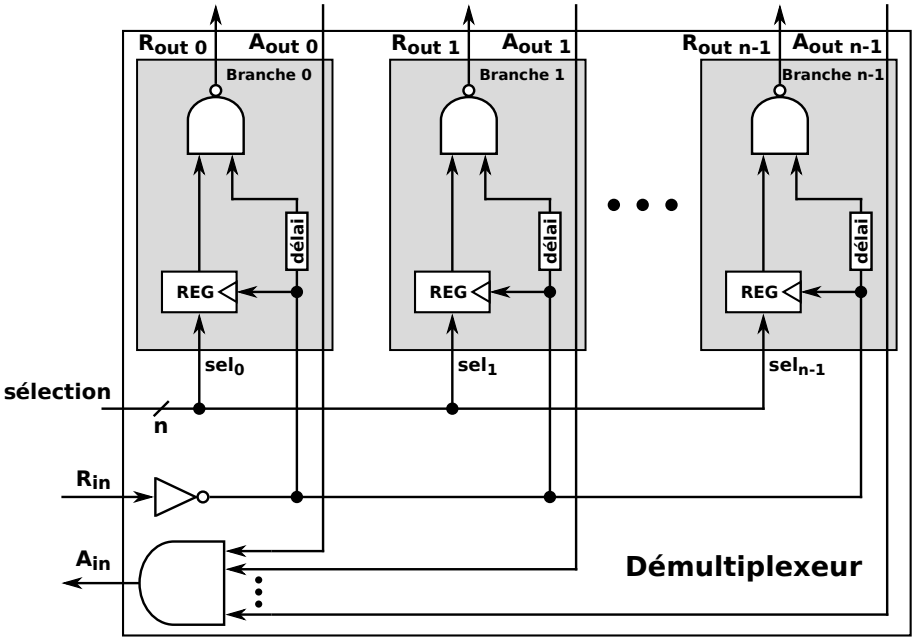


FIGURE 2.10 – *Architecture d'un démultiplexeur.*

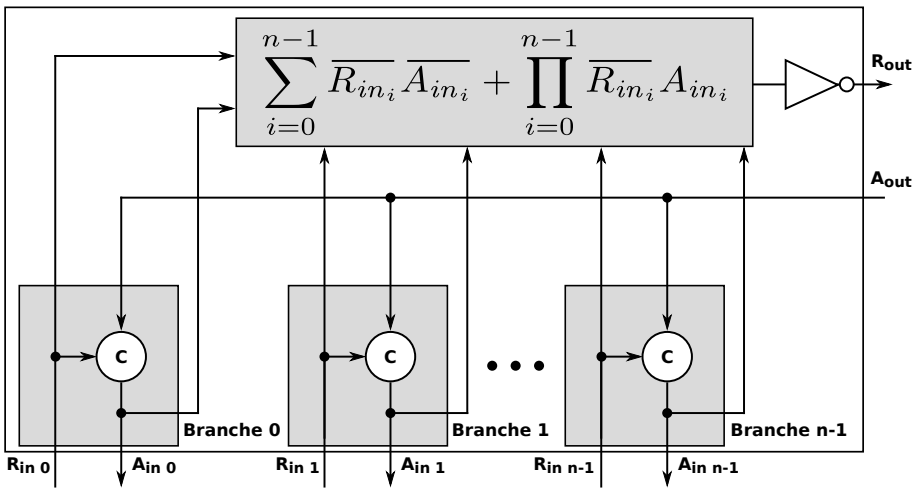
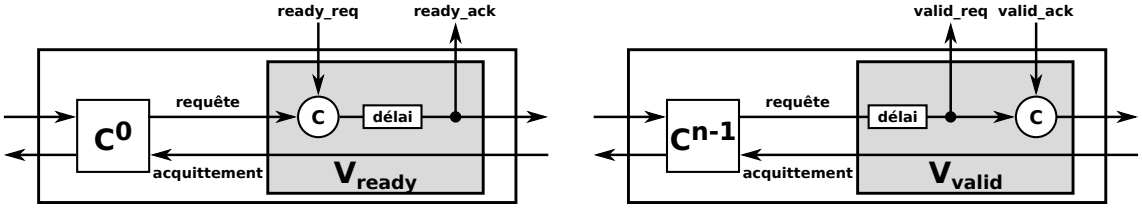


FIGURE 2.11 – *Architecture d'un merge.*

Blocs V_{ready} et V_{valid}

Les deux blocs V_{ready} et V_{valid} sont ajoutés pour démarrer le circuit et le mettre en veille. La combinaison de ces blocs implémente le protocole *late-capture* entre la FSM et les signaux *ready* et *valid*. Leur implémentation, présentée figure 2.12, est très simple : il suffit d'une porte de Muller sur le signal de requête pour synchroniser l'avancée des états de la FSM et la réponse de l'environnement (respectivement les signaux *ready_req* et *valid_ack* pour les blocs V_{ready} et V_{valid}).



(a) Bloc V_{ready}. *(b) Bloc V_{valid}.*

FIGURE 2.12 – *Implémentation des blocs V_{ready} et V_{valid}.*

Blocs Δ et Λ_{comb}

Ces deux blocs combinatoires sont identiques aux blocs correspondants dans l'architecture synchrone. Le bloc Λ_{comb} implémente la partie de la fonction λ générant les entrées du chemin de données Λ^{i}_{comb}. Le bloc Δ calcule la valeur du prochain état de la FSM et permet d'implémenter avec le bloc V'_{demux} la fonction δ.

Comme ces blocs sont combinatoires, leurs entrées doivent être actives durant toute la durée d'activation d'un état. Dans le cas d'une AFSM, l'activation d'un état correspond à la communication d'un contrôleur appartenant à C. La génération de l'entrée des blocs Δ et Λ_{comb} est donc réalisée par un OU logique, noté *OR*, entre le vecteur C^{req}_{out} et le vecteur C^{ack}_{out}. La figure 2.13 présente le chronogramme d'une communication du contrôleur c^i appartenant à C ainsi que la génération de la sortie de *OR* associée, noté or^{i}_{out}. Les entrées de *OR*, notées $or^{i,req}_{in}$ et $or^{i,ack}_{in}$, correspondent respectivement aux signaux $c^{i,req}_{out}$ et $c^{i,ack}_{out}$.

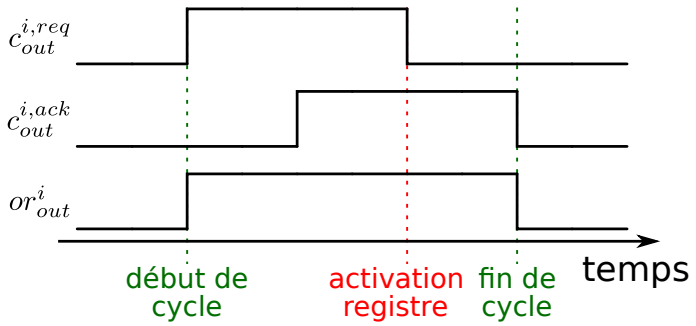


FIGURE 2.13 – *Génération d'un signal or^{i}_{out}.*

Bloc *Cint*

Ce bloc est un contrôleur asynchrone *late-capture* comme les éléments du bloc C et par conséquent leur description s'applique aussi pour *Cint*. Ce contrôleur assure le bon

fonctionnement du protocole de communication dans l'AFSM elle-même. Il est essentiel d'avoir ce contrôleur dans le cas d'un état qui reboucle sur lui-même mais il peut être omis lorsque ce cas n'est pas présent dans la FSM.

Bloc Λ'_{en}

Dans le circuit synchrone, les sorties du bloc Λ_{en} sont connectées aux ports *enable* des registres dans le chemin de données. Les registres sauvegardent de nouvelles données lorsque ces signaux *enable* valent '1'. Dans le circuit asynchrone, nous utilisons ces signaux pour jouer le rôle du signal d'horloge dans les registres du chemin de données.

La figure 2.14 présente la génération d'un signal Λ'^{i}_{en} qui est activé lors du passage dans les états d'indice j et k. Conformément à l'équation 2.5, Λ^{i}_{en} est construit à l'aide d'une porte logique OU. Le signal Λ^{i}_{en} vaut alors '1' lorsqu'un des signaux de requête, $c^{j,req}_{out}$ ou $c^{k,req}_{out}$, est aussi actif. Dans le protocole *late-capture*, les données sont capturées sur le front descendant du signal de requête. Par conséquent, les données en entrée des registres du chemin de données doivent être enregistrées sur ce front. Comme les registres fonctionnent sur le front montant d'une horloge, nous ajoutons un inverseur en sortie de Λ_{en} pour convertir le front descendant de Λ_{en} en un front montant. Le bloc Λ'_{en} est donc formé de Λ_{en} et d'un inverseur. Nous le connectons au port d'horloge des registres.

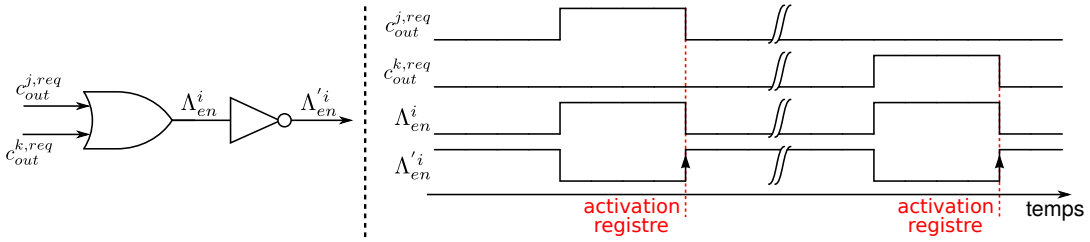
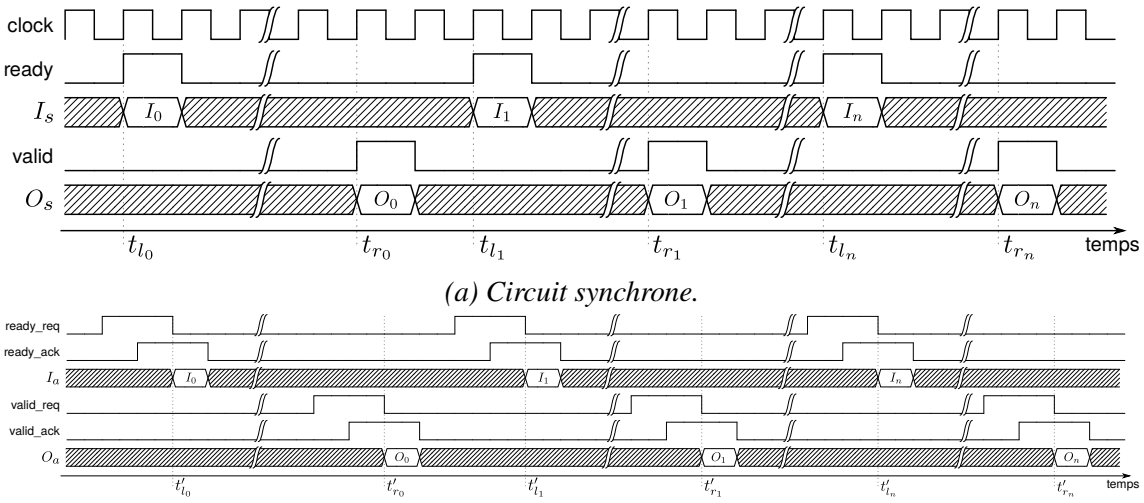


FIGURE 2.14 – *Exemple d'un bloc Λ'_{en} avec le chronogramme associé.*

2.3.2 Preuve de correction

Pour assurer la correction de la méthode, nous devons montrer que pour n'importe quelle séquence d'entrée, la séquence de sortie dans le circuit synchrone équivaut à la séquence de sortie dans le circuit désynchronisé. Dans les circuits synchrones, l'équivalence des sorties est vérifiée à chaque cycle d'horloge. Toutefois, dans un circuit asynchrone, la notion d'horloge n'est pas présente. L'équivalence reposera donc sur les signaux de synchronisation *ready* et *valid*.

Notations Dans le circuit synchrone, nous définissons $\tilde{I}_s$ la séquence des valeurs d'entrées $I_s(t_{l_0}),\ldots, I_s(t_{l_n})$, telle que t_{l_i} est l'instant du i^{ieme} front montant de *ready*. De la même façon, $\tilde{O}_s$ est la séquence des valeurs de sorties $O_s(t_{r_0}),\ldots, O_s(t_{r_n})$ telle que t_{r_i} est l'instant du i^{ieme} front montant sur le signal *valid*. Pour le circuit asynchrone, de manière analogue, nous notons $\tilde{I}_a$ la séquence des valeurs d'entrées $I_a(t'_{l_0}),\ldots, I_a(t'_{l_n})$ telle que t'_{l_i} est l'instant du i^{ieme} front descendant du signal *ready_req* et $\tilde{O}_a$ la séquence des valeurs de sorties $O_a(t'_{r_0}),\ldots, O_a(t'_{r_n})$ où t'_{r_i} est l'instant du i^{ieme} front descendant du signal *valid_req*. Toutes ces notations sont résumées sur les chronogrammes de la figure 2.15.



(a) Circuit synchrone.

(b) Circuit asynchrone.

FIGURE 2.15 – *Chronogrammes des circuits (a) synchrone et (b) asynchrone.*

Théorème 2.1 (Équivalence des circuits)**.** *Le circuit asynchrone est équivalent au circuit synchrone original si et seulement si lorsque les séquences des valeurs d'entrées du circuit synchrone $\tilde{I}_s$ et du circuit asynchrone $\tilde{I}_a$ sont égales, alors leur séquence des valeurs de sorties respectives sont elles aussi égales.*

$$\tilde{I}_s = \tilde{I}_a \implies \tilde{O}_s = \tilde{O}_a$$

Le théorème 2.1 donne la définition générale de l'équivalence entre deux circuits. Néanmoins, ce théorème est trop complexe à prouver en l'état. Nous divisons la preuve en nous concentrons sur un seul calcul du circuit. Nous montrons l'équivalence du comportement des deux circuits sur ce calcul pour en déduire la preuve du théorème d'équivalence générale entre le circuit synchrone et sa version désynchronisée.

Dans le circuit synchrone, nous supposons qu'aucun nouvel événement sur le signal *ready* ne peut être émis tant qu'un événement sur le signal *valid* ne s'est pas produit. Ce comportement s'apparente à un protocole 4 phases entre le circuit et l'environnement. Dans le circuit asynchrone, cette hypothèse est garantie par le protocole de communication des canaux *ready* et *valid*, ainsi que par la structure de la FSM. Le théorème 2.1 peut alors être transformé en :

Théorème 2.2. *Pour tout instant t_{l_i} et t'_{l_i}, si les entrées des circuits synchrone et asynchrone sont égales, alors leurs sorties seront identiques elles aussi.*

$$\forall i \in \mathbb{N}, I_s(t_{l_i}) = I_a(t'_{l_i}) \implies O_s(t_{r_i}) = O_a(t'_{r_i})$$

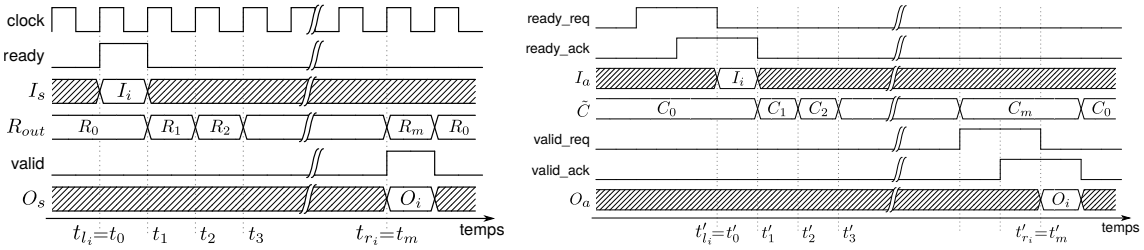
Lemme 2.1. *Si le théorème 2.2 est vrai, alors cela implique que le théorème 2.1 est juste.*

Démonstration. Nous supposons que $\tilde{I}_s = \tilde{I}_a$. Alors pour tout indice i, nous avons $I_s(t_{l_i}) = I_a(t'_{l_i})$. En admettant que le théorème 2.2 est vrai, cette égalité implique $O_s(t_{r_i}) = O_a(t'_{r_i})$. Comme chaque élément des séquences des valeurs d'entrées implique l'équivalence des valeurs de sorties. Par conséquent $\tilde{O}_s = \tilde{O}_a$ et le théorème 2.1 est vrai. ■

Pour démontrer le théorème 2.2, nous divisons la preuve en deux parties. Nous établissons tout d'abord l'équivalence des FSMs, qui correspond à l'équivalence de la séquence des états ainsi qu'à l'équivalence des sorties. Cette équivalence signifie que l'ordonnancement des opérations est le même pour les deux circuits. Puis nous démontrons l'équivalence des comportements des chemins de données en synchrone et asynchrone. Par conséquent, l'équivalence des entrées du chemin de données implique l'équivalence des sorties de celui-ci et le théorème est prouvé.

Équivalence des FSMs

Les deux FSMs sont implémentées suivant le même modèle. Les deux implémentations, synchrone et asynchrone, sont équivalentes si et seulement si leur séquence d'états sont équivalentes. Dans la FSM synchrone, nous notons $\tilde{R}$ la séquence $R_0, R_1, \ldots, R_m$ des valeurs de sortie des registres de R, telle que $R_0 = R_{out}(t_0)$ correspond à l'état initial *wait* où t_0 est l'instant t_{l_i} et $R_m = R_{out}(t_m)$ à l'état *end* avec t_m correspondant à l'instant t_{r_i}. Chaque élément de la séquence coïncide avec un cycle d'horloge comme le montre la figure 2.16a.



(a) Circuit synchrone. *(b) Circuit asynchrone.*

FIGURE 2.16 – *Chronogrammes des séquences d'états des circuits (a) synchrone et (b) asynchrone.*

De manière similaire dans l'implémentation asynchrone, nous définissons $\tilde{C}$ la séquence des valeurs de sortie des contrôleurs $C_0, C_1, \ldots, C_m$, telle que C_0 équivaut à l'état initial *wait* et C_m à l'état *end* (voir figure 2.16b). Chaque élément C_i correspond au vecteur des valeurs C_{out}^{req} où un élément $c_{out}^{j,req}$ du vecteur est actif. Entre deux éléments de la séquence, le contrôleur actif est acquitté et sa requête est remise à 0.

Nous souhaitons démontrer que si au début de chaque séquence les signaux *ready* et *ready_req* sont actifs, alors nous avons le théorème suivant :

Théorème 2.3. *Lors de chaque calcul des circuits, les séquences d'activation des registres $\tilde{R}$ et des contrôleurs $\tilde{C}$ sont égales.*

$$\tilde{R} = \tilde{C}$$

Démonstration. Le bloc combinatoire Δ est identique dans les deux implémentations, synchrone et asynchrone. Étant donné que la valeur du prochain état de la fonction de transition d'états δ est construite par ce bloc, la preuve de correction repose uniquement sur la correction des blocs V'_{demux} et V_{merge}, et de la correction des contrôleurs. Nous supposons pour le moment que les blocs V'_{demux}, V_{merge} sont corrects pour réaliser la preuve de

la méthode. Leur correction peut être réalisée avec des méthodes de *model-checking* [91] ou de "prouveur" de théorème. Pour les contrôleurs, leur correction est assurée par le STG et la communication entre contrôleurs est donc correcte par construction. ■

De la même manière, nous pouvons construire deux couples de séquences de sortie, $(\tilde{\Lambda}_{comb,s}, \tilde{\Lambda}_{en,s})$ pour la FSM synchrone et $(\tilde{\Lambda}_{comb,a}, \tilde{\Lambda}_{en,a})$ pour l'AFSM, et prouver l'équivalence entre ces deux couples :

Théorème 2.4. *Lors de chaque calcul des circuits, les séquences des valeurs des sorties de la FSM synchrone* $(\tilde{\Lambda}_{comb,s}, \tilde{\Lambda}_{en,s})$ *et asynchrone* $(\tilde{\Lambda}_{comb,a}, \tilde{\Lambda}_{en,a})$ *sont équivalentes.*

$$\begin{aligned} \tilde{\Lambda}_{comb,s} &= \tilde{\Lambda}_{comb,a} \\ \tilde{\Lambda}_{en,s} &= \tilde{\Lambda}_{en,a} \end{aligned}$$

Démonstration. Dans les deux FSMs, synchrone et asynchrone, les blocs combinatoires Λ_{en}, Λ_{comb} et Δ sont identiques. Prouver l'équivalence des couples $(\tilde{\Lambda}_{comb,s}, \tilde{\Lambda}_{en,s})$ et $(\tilde{\Lambda}_{comb,a}, \tilde{\Lambda}_{en,a})$ revient à prouver l'équivalence des entrées de ces blocs combinatoires. Dans le circuit synchrone, ces entrées sont connectées au vecteur R_{out} qui représente l'état courant de la FSM. Pour le circuit asynchrone, elles sont reliées au vecteur C^{req}_{out}, qui correspond lui aussi à l'état courant de l'AFSM. Par conséquent, comme $\tilde{R} = \tilde{C}$ d'après le théorème 2.3, nous avons $\tilde{\Lambda}_{comb,s} = \tilde{\Lambda}_{comb,a}$ et $\tilde{\Lambda}_{en,s} = \tilde{\Lambda}_{en,a}$. Les deux couples sont donc équivalents. ■

Il est important de remarquer que pour tout i, $\Lambda^i_{en,a}$ est différent de 0 et qu'entre $\Lambda^i_{en,a}$ et $\Lambda^{i+1}_{en,a}$ il y a un instant t où $\Lambda_{en,a}(t)$ est nul. Ce comportement est dû au protocole *late-capture*, ce qui assure la génération d'un front montant d'horloge pour les registres du chemin de données.

Avec la démonstration des théorèmes 2.3 et 2.4, les deux FSMs possèdent les mêmes séquences d'activation d'état ainsi que des sorties équivalentes, même si leur implémentation est différente. Par conséquent, la FSM du circuit synchrone et l'AFSM du circuit désynchronisé sont équivalentes.

Équivalence du chemin de données

L'équivalence entre les deux chemins de données est plus complexe. Dans le paradigme synchrone, le signal d'horloge contrôle globalement les registres. Dans le paradigme asynchrone, les différents signaux de contrôle synchronisent les registres.

Sans perte de généralité, nous pouvons étudier seulement un registre r^i appartenant à R. En effet, si chaque registre du chemin de données possède le même comportement en synchrone et en asynchrone, alors l'enchaînement du stockage des données dans le chemin de données sera équivalent. Dans le circuit synchrone, son port *enable* r^i_{en} est relié à Λ^i_{en} et son entrée à $r^i_{in,s}$. Dans le circuit asynchrone, il n'y a plus de port *enable* mais le port d'horloge est connecté à Λ'^i_{en}, qui équivaut à $\neg\Lambda^i_{en}$, et l'entrée du registre est notée $r^i_{in,a}$. Nous souhaitons démontrer l'équivalence d'un registre en synchrone et asynchrone définie de la manière suivante :

Théorème 2.5. *Si lorsque le signal Λ^i_{en} est actif les entrées des registres en synchrone et asynchrone sont identiques, alors cela implique les sorties des registres synchrone $r^i_{out,s}$ et asynchrone $r^i_{out,a}$ seront égales.*

$$r^i_{in,s} = r^i_{in,a} \implies r^i_{out,s} = r^i_{out,a}$$

Démonstration. Soit un registre r^i où $r^i_{in,s} = r^i_{in,a}$. Dans le circuit synchrone, si Λ^i_{en} est à '1', alors sa valeur ne peut changer qu'au prochain front montant du signal d'horloge. Suivant l'équation 2.3, l'entrée $r^i_{in,s}$ sera donc copiée sur $r^i_{out,s}$ au prochain front montant d'horloge. Dans le circuit asynchrone, étant donné que l'inverse de Λ^i_{en} correspond à l'entrée du signal d'horloge locale, le port d'horloge du registre est à '0'. Grâce au protocole de communication dans l'AFSM, un front descendant se produira sur le signal Λ^i_{en}. Ce front descendant est converti en front montant ce qui active les registres. Sa sortie équivaut à $r^i_{out,a} = r^i_{in,a}$. Par conséquent, comme $r^i_{in,s} = r^i_{in,a}$, nous obtenons $r^i_{out,s} = r^i_{out,a}$ qui montre donc l'égalité des sorties. ■

Les deux chemins de données sont donc équivalents. Or comme les FSMs sont équivalentes, les registres sont activés dans le même ordre dans le cas synchrone et asynchrone. Le théorème 2.2 est alors prouvé et par voie de conséquence le théorème 2.1 aussi. Les circuits synchrone et asynchrone sont donc équivalents, et la méthode de désynchronisation est correcte.

Dans cette démonstration, nous supposons que les blocs combinatoires sont sans délai et nous ne prenons pas non plus en compte les temps de *setup* et de *hold* des registres. En pratique, il est nécessaire d'inclure ces délais : dans chaque bloc de l'AFSM, nous ajoutons des délais équivalents au chemin critique pour éviter des pertes de données ou une métastabilité dans les registres.

Nous avons esquissé manuellement la preuve de correction de notre méthode de désynchronisation et donné les grandes étapes de celle-ci. Il reste encore à réaliser cette preuve avec un logiciel prouveur de théorème, afin de s'assurer formellement de la justesse de la preuve et de la validité de nos hypothèses. Ce travail a été débuté avec la modélisation de notre méthode avec le langage *Prototype Verification System* (PVS), un langage de spécification associé à un logiciel prouveur de théorème.

2.3.3 Contraintes temporelles

Le fonctionnement correct des circuits à données groupées repose sur des contraintes temporelles locales. Ces contraintes s'établissent entre chaque paire des contrôleurs du circuit de contrôle et sont exprimées à l'aide des RTCs décrite en section 1.3.3. Généralement, un circuit à données groupées possède des contraintes temporelles internes à son circuit de contrôle ainsi que des contraintes entre le circuit de contrôle et le chemin de données pour assurer le bon fonctionnement du circuit. Ce deuxième type de contraintes temporelles s'effectue entre deux bancs de registres pour vérifier le calcul des données par la logique combinatoire.

Dans notre cas, l'AFSM génère en plus des signaux de sélection pour la logique combinatoire (les signaux Λ^i_{comb}). Cette spécificité ajoute des contraintes supplémentaires,

TABLEAU 2.1 – *Contraintes temporelles sur les registres (signaux Λ^i_{en})*

Type de contraintes	RTC associée
Contrainte d'établissement (*setup*)	$c^{i,req}_{out}- \mapsto r^{i+1}_{in} \prec \Lambda^{i+1}_{en}++\delta_S$
Contrainte de maintien (*hold*)	$c^{i,req}_{out}- \mapsto \Lambda^{i}_{en}+ \prec r^{i}_{in}-\delta_H$

mais sans changer celles déjà existantes. Pour les contraintes temporelles entre les bancs de registres, il n'y a aucune différence et les contraintes sont répertoriées table 2.1.

Les valeurs δ_S et δ_H correspondent respectivement aux temps de *setup* et de *hold* des registres du circuit. La contrainte de *setup* est assurée car elle fait intervenir le délai sur le signal de requête. Ce délai doit donc couvrir le délai de la logique combinatoire et le temps de *setup* pour assurer un fonctionnement correct. Comme notre AFSM a uniquement un seul état actif à un instant donné, cela signifie qu'un seul contrôleur est activé en même temps. La contrainte de *hold* est par conséquent assez légère, sauf dans le cas d'un état rebouclé sur lui-même. Les contraintes de *hold* sont résolues au niveau du chemin de données du circuit.

Comme dit précédemment, l'AFSM génère aussi des signaux pour la logique combinatoire, les signaux Λ^i_{comb}. Ces signaux possèdent aussi leurs propres contraintes temporelles (voir table 2.2).

TABLEAU 2.2 – *Contraintes temporelles sur la logique combinatoire (signaux Λ^i_{comb})*

Type de contraintes	RTC associée
Contrainte d'établissement (*setup*)	$c^{i,req}_{out}+ \mapsto r^{i}_{in} \prec \Lambda^{i}_{en}++\delta_S$
Contrainte de maintien (*hold*)	$c^{i,req}_{out}- \mapsto \Lambda^{i}_{en}+ \prec r^{i}_{in}-\delta_H$
Stabilité du signal	$c^{i,ack}_{out}+ \mapsto or^{i,ack}_{in}+ \prec or^{i,req}_{in}-$

Les deux premières contraintes temporelles de la table 2.2, *setup* et *hold*, sont très similaires aux contraintes de registres à registres vues précédemment. La seule différence correspond au chemin de calcul des données qui va cette fois passer par les signaux Λ^i_{comb}. Elles seront donc résolues de la même manière. Pour la contrainte de stabilité du signal Λ^i_{comb}, c'est une contrainte interne à l'AFSM. Elle signifie simplement que l'entrée des blocs Λ_{comb} et Δ doit être active durant toute la période de calcul. Pour cela, la sortie de la porte OR or^i_{out} associée doit être à '1', et le fil connectant $c^{i,ack}_{out}$ à la porte logique doit être . Comme cette contrainte s'applique sur un fil, elle est facile à résoudre.

Toutes ces contraintes temporelles locales doivent être vérifiées lors de chacune des phases de conception du circuit, la synthèse et le PnR. Pour cela, les outils commerciaux synchrones reposent sur l'utilisation de la STA. Pour assurer la vérification avec ces outils de conception, nous utiliserons la méthode LCS, expliquée dans la section 1.3.2, qui utilisent des horloges pour construire les contraintes temporelles locales.

2.4 Application à Catapult HLS

Maintenant que notre méthode a été présentée et formalisée, nous l'appliquons à un outil de HLS classique dans l'industrie, Catapult HLS de Siemens EDA, afin de démontrer la faisabilité de notre méthode. Cet outil permet de générer un code RTL à partir d'un algorithme décrit dans un sous-ensemble de C++. Nous travaillons à partir de la description RTL pour mettre en place notre méthode. Une fois le circuit asynchrone obtenu, celui-ci suit le flot standard de synthèse et de placement/routage pour la suite de sa conception. Pour rendre efficace ces étapes, notre flot génère aussi à haut niveau les contraintes temporelles du circuit grâce à la méthode LCS [70] décrite dans le chapitre précédent.

2.4.1 Architecture des circuits

La figure 2.17 montre les connexions entre le chemin de données et la FSM, ainsi que les interconnexions des sous-blocs du chemin de données. Comme dans notre modèle, la FSM contrôle l'unité de calcul au travers des signaux Λ^i_{comb} et Λ^i_{en}. Le futur état de la FSM est choisi à l'aide des signaux Δ^i_{in}. Dans un circuit généré par Catapult, la synchronisation entre le système et l'environnement est réalisée avec les blocs de synchronisation d'entrée/sortie (*Sync Block* dans la figure 2.17) ainsi qu'avec des signaux de requête/acquittement insérés au niveau du chemin de données. Si l'environnement n'active pas les blocs de synchronisation, le bloc *Stall Block* fige le circuit entier grâce au signal global *stall*.

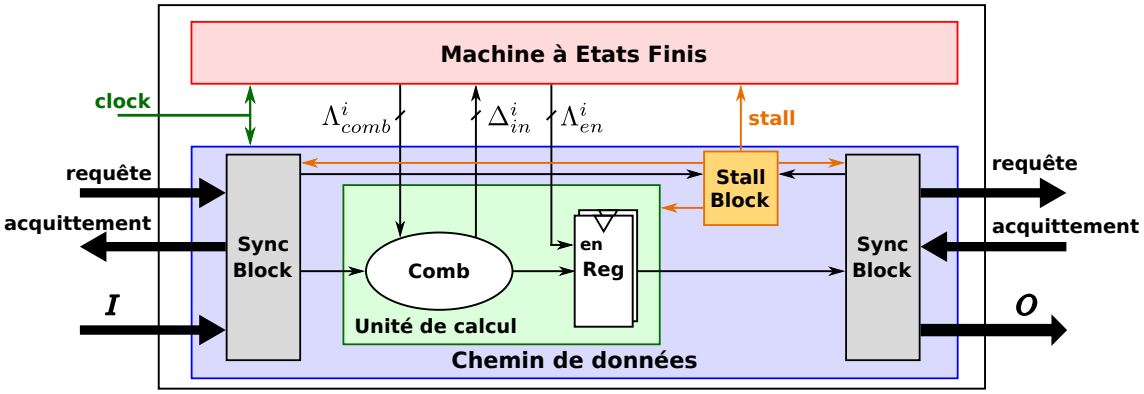


FIGURE 2.17 – *Architecture d'un circuit synchrone produit par Catapult.*

Si l'architecture des circuits réalisés par Catapult HLS est globalement similaire à notre modèle, il semble que l'interface de communication des entrées/sorties avec le circuit soit différente. En effet, elle est présente dans le chemin de données et gérée par les blocs *Sync Block* et *Stall Block*. Néanmoins, le principe reste le même : une communication est attendue avec l'environnement extérieur dans un état précis avant de démarrer les calculs du circuit et une autre communication est nécessaire en fin de calcul afin d'envoyer le résultat et accepter de nouvelles données en entrée. Cela correspond respectivement à nos états *wait* et *end*. Les signaux *ready* et *valid* sont quant à eux, respectivement les signaux de requête et d'acquittement des états correspondants.

Nous pouvons donc nous ramener à notre modèle initial pour désynchroniser ce circuit. Le circuit résultant de la désynchronisation est présenté figure 2.18. Les états de la FSM sont extraits pour modéliser le graphe d'états du circuit et chaque état est converti en un contrôleur asynchrone. Tous les éléments du chemin de données utilisés pour la

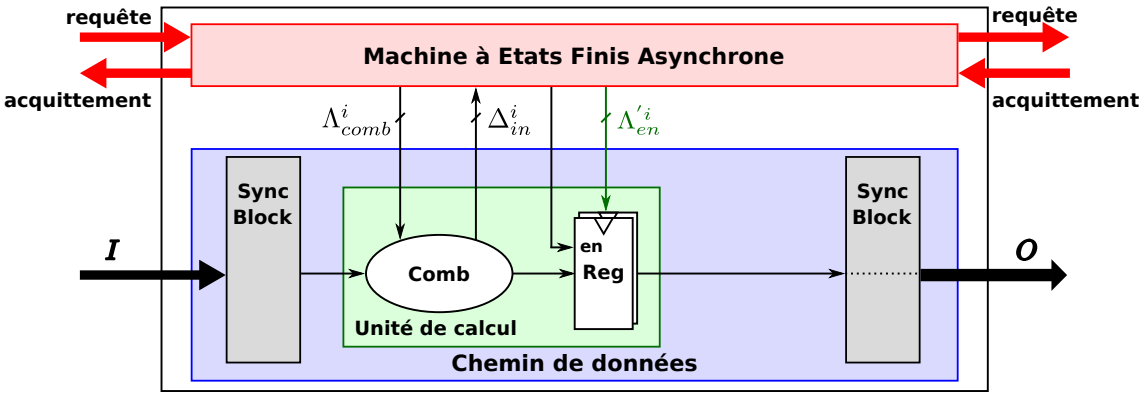


FIGURE 2.18 – *Architecture du circuit désynchronisé.*

synchronisation entre le circuit et son environnement sont supprimés pour être remplacés par les blocs V_{ready} et V_{valid} dans l'AFSM. Pour optimiser l'architecture de notre AFSM et éviter des interconnexions inutiles, nous utilisons une version distribuée des composants *merge* et *demultiplexer*. En effet, lorsqu'il y a peu d'états avec plusieurs prédécesseurs/successeurs, ou que le nombre d'états dans la graphe est important, une architecture distribuée ne gardant que les connexions effectives entre les états est plus efficace (voir section 3.2).

2.4.2 Génération des contraintes

Pour rendre accessible la conception des circuits désynchronisés, nous ciblons l'utilisation des outils de CAO classiques, fait pour des circuits synchrones. Afin de faciliter cela, nous exploitons la méthode LCS et automatisons sa mise en place. Comme le graphe d'état de la FSM est extrait à haut niveau, il est possible de générer les contraintes temporelles au même moment car le graphe donne les connexions à l'intérieur de l'AFSM.

La figure 2.19 présente le détail du générateur des contraintes temporelles du circuit désynchronisé. Tandis qu'une partie du flot génère l'AFSM, ce générateur définit automatiquement les contraintes temporelles. Le générateur de contraintes a besoin de plusieurs types d'informations pour fonctionner correctement :

— La structure de circuit de contrôle, c'est-à-dire de l'AFSM. Cette information est donnée par le graphe d'états de la FSM. Ce dernier est obtenu directement depuis la description RTL du circuit synchrone.

— Les caractéristiques internes du circuit de contrôle. Elles sont de deux types : les règles de propagation des signaux entre les contrôleurs et la structure des contrôleurs utilisés dans l'AFSM. Les règles de propagation sont données par le protocole de communication et sont par conséquent fixes. Dans notre cas, il s'agit du protocole *late-capture* décrit en section 2.3. L'implémentation au niveau portes logiques des contrôleurs est dépendante de la technologie cible et doit donc être définie par l'utilisateur sous forme de librairie. Il est alors facile d'ajouter une nouvelle implémentation en cas de besoin.

Le résultat final est un fichier de contraintes standard *Synopsys Design Constraints* ou SDC, indépendant des conditions PVT et utilisable par tous les outils de CAO industriels. La méthode LCS réalise la vérification des contraintes temporelles sur les chemins locaux

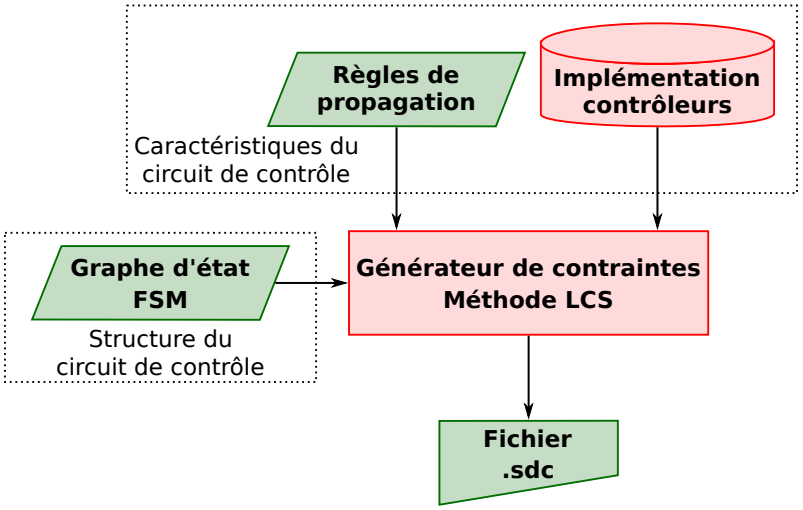


FIGURE 2.19 – *Générateur de contraintes temporelles.*

dans le circuit désynchronisé et rend totalement transparent pour les ingénieurs les spécificités de notre architecture ainsi que de la méthode (comme les horloges intermédiaires).

2.5 Conclusion

Dans ce chapitre, nous avons présenté notre méthode de désynchronisation. Cette méthode est fondée sur la désynchronisation du seul chemin de contrôle, qui n'est rien de plus qu'une FSM contrôlant le chemin de données dans le cas des circuits générés par HLS. L'AFSM, qui possède un encodage *one-hot*, est construite à partir du graphe d'états de la FSM synchrone originale. Nous avons établi une formalisation des circuits et des transformations effectuées par la méthode de désynchronisation. De plus, les premiers pas de la preuve de correction ont été avancés. Cette méthode a été implémentée avec succès sur l'outil de HLS industriel Catapult HLS. Désormais, il nous est possible de concevoir rapidement des circuits asynchrones dans l'écosystème de conception des circuits synchrones.

Maintenant que le cadre formel de la méthode a été posé, la prochaine étape de ces travaux serait naturellement la mise en place de la preuve formelle dans un outil de preuve de théorème. Ce travail a été entamé avec la description sous PVS de notre modèle qui, pour le moment, est incomplète. Il existe aussi d'autres perspectives à ces travaux. Tout d'abord, une étude approfondie de l'AFSM semble nécessaire afin d'identifier la sous-classe des protocoles maintenant l'équivalence entre les circuits synchrones et leurs versions désynchronisées. De plus, un certain nombre de limitations sont présentes dans notre modèle et restreignent les classes de FSM que nous pouvons désynchroniser. Pour atteindre un ensemble étendu de FSMs désynchronisables, il est nécessaire de généraliser notre modèle avec, par exemple, un nombre indéfini d'états communicants avec l'environnement extérieur du circuit.

À présent, notre méthode nécessite d'être évaluée comparativement aux différentes méthodes déjà existantes. Cette analyse est réalisée dans le chapitre suivant et donne une

idée des avantages de notre méthode de désynchronisation.

3
Évaluation de la méthode

Sommaire

3.1 Introduction 52
3.2 Étude comparative avec les autres méthodes 52
3.2.1 Description du circuit 52
3.2.2 Résultats de la désynchronisation 55
3.3 Application à la HLS 57
3.3.1 Architecture du prototype 57
3.3.2 Résultats 59
3.4 Conclusion 61

3.1 Introduction

La méthode de désynchronisation détaillée dans le chapitre précédent a été formalisée et intégrée dans un flot de conception de haut niveau. Il est nécessaire de vérifier la fonctionnalité de notre flot ainsi que d'étudier l'efficacité de la méthode. Notre but est, qu'avec cette méthode, les performances des circuits désynchronisés soient comparables aux circuits synchrones d'origine, tout en apportant le caractère événementiel et la robustesse de l'asynchrone.

Une première partie de ce chapitre comparera la méthode de désynchronisation avec son homologue synchrone, ainsi qu'avec la principale méthode de désynchronisation existante dans l'état de l'art. Cette étude montre les améliorations qu'apporte cette méthode par rapport à la désynchronisation standard en termes de surface, vitesse et consommation. Les circuits désynchronisés sont alors très similaires aux originaux synchrones avec une amélioration notable en consommation d'énergie. La deuxième partie validera le flot de conception HLS développé dans ce travail au travers d'un circuit de chiffrement classique, un AES (pour *Advanced Encryption Standard*). Ce circuit a été évalué en technologie 65 nm de STMicroelectronics et a mené à la fabrication d'un prototype afin de valider l'ensemble du flot, de la HLS au PnR. Ces études permettront de montrer l'efficacité de notre méthode et de notre flot de conception afin d'obtenir rapidement grâce à la HLS des circuits asynchrones à données groupées.

3.2 Étude comparative avec les autres méthodes

Notre méthode de désynchronisation, même si elle cible des circuits générés par HLS, peut être appliquée à n'importe quel circuit possédant une séparation entre les chemins de données et de contrôle. Afin de faire une démonstration de notre méthode ainsi que de la comparer aux travaux de l'état de l'art, nous désynchronisons un circuit de chiffrement AES, conçu de manière standard (c'est-à-dire manuellement). Les performances du circuit ont été évaluées en 28 nm FDSOI de STMicroelectronics.

3.2.1 Description du circuit

L'algorithme AES, présenté en figure 3.1, est très largement utilisé dans la cryptographie. Il permet un chiffrement symétrique d'un bloc de données d'entrée de 128 bits à l'aide d'une clé de taille variable suivant la sécurité souhaitée. Dans notre cas, la clé utilisée est d'une longueur de 128 bits. Celle-ci sert de base pour la génération de sous-clés qui servent à chiffrer les données intermédiaires lors de l'étape *AddKey*. Les étapes *SubBytes*, *ShiftRow*, *MixColumn* et *AddKey* sont effectuées neuf fois puis un dernier tour réalise les opérations *SubBytes*, *ShiftRow* et *AddKey*. La fonction précise de chacune des étapes est détaillée dans la littérature [92].

Nous avons conçu manuellement un AES synchrone, avec une clé de 128 bits. Le circuit est présenté par la figure 3.2. Chaque bloc du chemin de données est composé de la façon suivante : une partie combinatoire calcule les valeurs souhaitées et son résultat est mémorisé dans un banc de registres. Les étapes *SubBytes* et *ShiftRow* sont rassemblées au sein du même bloc *SUBSR*, tandis que la génération des sous-clés de l'algorithme est

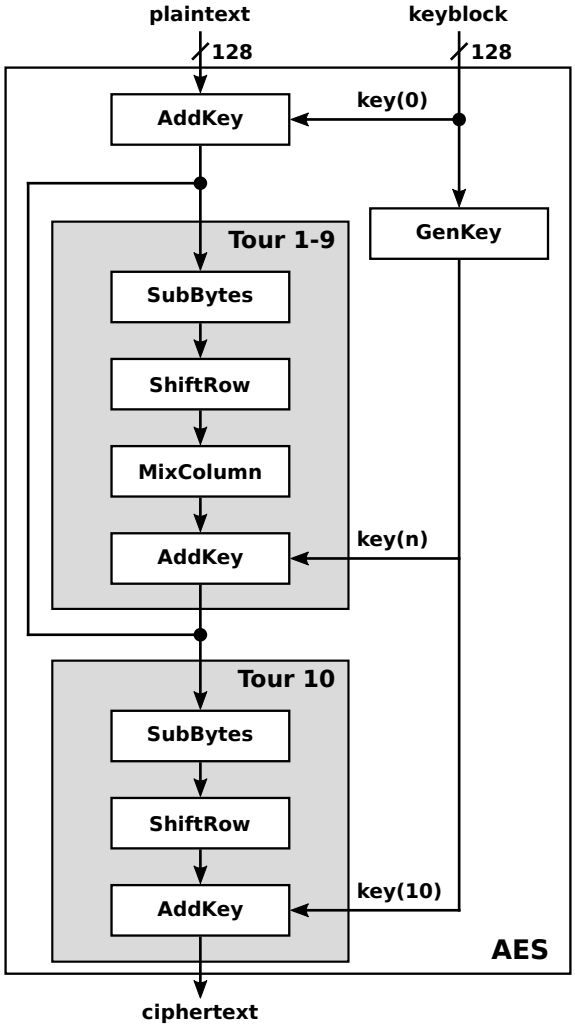


FIGURE 3.1 – *Algorithme de l'AES avec une clé de 128 bits.*

réalisée directement dans les blocs *ADDKEY*.

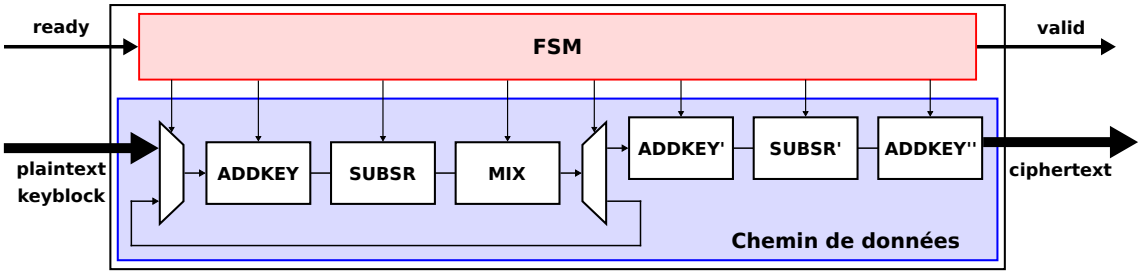


FIGURE 3.2 – *Schéma bloc de l'architecture du circuit.*

La FSM a été conçue pour être compatible avec les restrictions présentées en section 2.2. Par conséquent, un signal *ready* active la FSM au démarrage et un signal *valid* indique la fin du calcul. La FSM, donnée en figure 3.3, contient huit états. Les états s_1, s_2 et s_3 calculent les opérations des blocs *ADDKEY*, *SUBSR* et *MIX* sur la figure 3.2. Une fois que les neuf premiers tours sont terminés, le signal *exit* passe à '1' et le dernier tour est réalisé par les états s_4, s_5 et s_6 qui activent respectivement *ADDKEY'*, *SUBSR'* et *ADDKEY"*. Les états s_0 et s_7 sont, quant à eux, les états *wait* et *end* qui gèrent les signaux *ready* et *valid*.

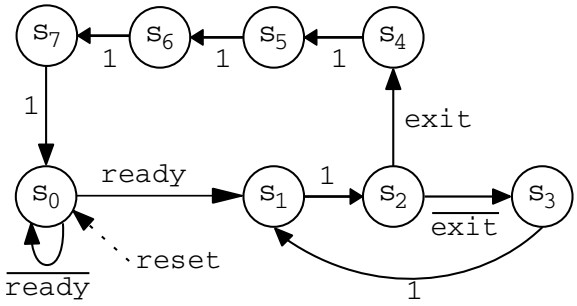


FIGURE 3.3 – *FSM du circuit AES.*

Nous utilisons deux approches pour désynchroniser le circuit : la méthode de désynchronisation présentée dans ce travail de thèse et la méthode proposée par Cortadella *et al.* [48]. Dans la suite, nous appellerons notre méthode DESYNC-AFSM. La FSM est remplacée par l'AFSM de la figure 3.4. Contrairement au modèle présent section 2.3, cette implémentation prend en compte les connexions réelles entre états pour optimiser l'architecture de l'AFSM. Au lieu d'avoir des composants de sélection reliant tous les états entre eux, nous obtenons alors seulement des *merge* et *demultiplexer* avec seulement les deux successeurs ou prédécesseurs ce qui optimise grandement la taille de l'AFSM. Étant donné qu'aucun état ne reboucle sur lui-même, il n'y a pas besoin de contrôleur C^{int} dans l'architecture. Dans la seconde méthode, appelée DESYNC-STD, un circuit de contrôle est ajouté au circuit de la figure 3.2. Les contrôleurs utilisés dans le circuit de contrôle, correspondant au protocole *semi-decoupled*, sont les mêmes que dans le travail original de [48].

Dans ce circuit, le nombre d'états de la FSM est plus grand que le nombre de bancs de registres (six dans le chemin de données et un pour la FSM de l'AES). Ce n'est pas le cas idéal pour notre méthode, mais cette comparaison apporte des éléments sur les limites de celle-ci et une évaluation pertinente avec la méthode de désynchronisation classique

en termes de performance. Pour avoir une comparaison juste avec le circuit synchrone, ce dernier est conçu avec du *clock gating*. Il est noté SYNC dans la suite du texte.

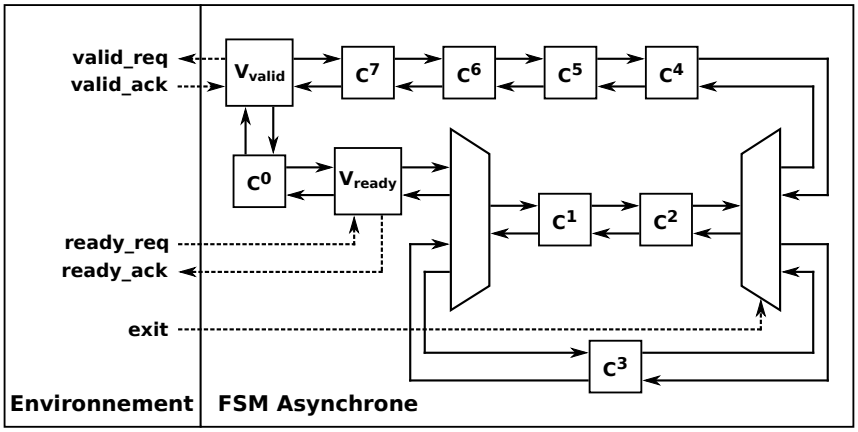


FIGURE 3.4 – *AFSM du circuit désynchronisé.*

3.2.2 Résultats de la désynchronisation

Les circuits sont conçus avec des outils de CAO commerciaux, Design Compiler de Synopsys pour l'étape de synthèse et Cadence Innovus pour le PnR. Toutes les analyses post-conception sont faites avec PrimeTime de Synopsys. Afin d'avoir une conception efficace avec ces outils, nous utilisons la méthode LCS pour les deux circuits désynchronisés. Dans le cas de DESYNC-AFSM, le flot de désynchronisation génère le fichier de contraintes associé. Pour le circuit DESYNC-STD, la méthode LCS est appliquée manuellement. En plus de vérifier les contraintes temporelles locales, cette méthode permet de contraindre le chemin de données à une fréquence précise comme on le pratique avec les circuits synchrones.

TABLEAU 3.1 – *Résultat en surface et impact de la désynchronisation*

	SYNC	DESYNC-AFSM	DESYNC-STD
Surface (μm^2)	18874	19043	19105
Overhead (%)	–	+0,9 %	+1,22 %

Les trois circuits sont synthétisés avec la technologie FDSOI 28 nm de STMicroelectronics à une fréquence de 800 MHz avec une tension d'alimentation de 0,9 V. La table 3.1 présente les résultats en surface après synthèse. Pour que ces résultats soient pertinents, les blocs de délais sur les signaux de requête dans les circuits asynchrones sont insérés durant la synthèse. Les circuits désynchronisés, DESYNC-AFSM et DESYNC-STD, ont respectivement une augmentation de 0,9 % et 1,22 % comparé au circuit original. Dans les deux cas, cette augmentation est faible mais notre méthode la réduit de 30 %. En effet, DESYNC-STD instancie deux contrôleurs par banc de registres au lieu d'un pour chaque état de la FSM dans DESYNC-AFSM. Grâce au protocole *late-capture*, la taille des lignes de délais est aussi divisée par deux. Néanmoins, nos composants de sélection (*demultiplexer* et *merge*) sont plus complexes que ceux utilisés dans DESYNC-STD. Le circuit de contrôle représente respectivement 1,1 % et 2,6 % de la taille du circuit dans

DESYNC-AFSM et DESYNC-STD. DESYNC-AFSM tire parti de la suppression de la FSM synchrone. D'un autre côté, DESYNC-STD possède un chemin de données plus petit grâce à l'utilisation des *latches*. Dans ce dernier cas, il est probable que la taille finale du circuit augmente durant la phase de PnR car le nombre de cellules est plus grand, comme montré dans [48].

Pour la vitesse ainsi que la consommation d'énergie, les circuits sont étudiés après la réalisation du PnR. Ceci permet de prendre en compte l'impact de l'arbre d'horloge, que ce soit pour le circuit synchrone ou pour les circuits désynchronisés qui possèdent des arbres locaux en sortie des contrôleurs. Pour avoir des résultats réalistes en consommation d'énergie, nous définissons l'activité de commutation du circuit suivant une simulation rétro-annotée au niveau portes. Tous les résultats sont présentés dans la table 3.2.

TABLEAU 3.2 – *Caractéristiques des circuits synchrone et désynchronisés*

	SYNC	DESYNC-AFSM	DESYNC-STD
Temps de calcul (ns)	38,75	38,47	48,91
Time Overhead (%)	–	-0,72 %	+26,22 %
Puissance moyenne (mW)	1,31	1,17	2,14
Power Overhead (%)	–	-10,71 %	+63,36 %
Énergie (pJ)	50,8	45	105
Overhead (%)	–	-11,4 %	+106,7 %

En termes de temps de calcul, les résultats ne montrent quasiment aucune amélioration du circuit DESYNC-AFSM par rapport au circuit synchrone. En effet, même si un circuit asynchrone à données groupées calcule des données en temps moyen, le chemin critique est toujours présent dans le circuit (les blocs *ADDKEY* dans ce cas). Le circuit asynchrone réalise quelques gains sur les autres chemins, qui sont plus rapides, mais ceux-ci sont contrebalancés par le temps de cycle du protocole de communication entre chaque paire de contrôleurs. Cette interprétation est aussi applicable pour le circuit DESYNC-STD car les deux circuits asynchrones partagent un circuit de contrôle similaire. Dans DESYNC-STD, comme il y a deux fois plus de contrôleurs que dans DESYNC-AFSM et que la moitié d'entre eux communique, ce phénomène est encore plus présent. Par conséquent, le temps de calcul du résultat est majoré de 26,2 %.

La puissance moyenne de DESYNC-AFSM est réduite de 10,71 % comparée au circuit synchrone d'origine. Étant donné que leur temps de calcul est équivalent, la diminution de la consommation d'énergie entre le circuit asynchrone et synchrone est similaire, où la baisse atteint 11,4 %. Comme le circuit synchrone a été conçu avec des cellules de *clock gating*, il n'active les registres que lors d'un cycle spécifique, de manière analogue à un circuit asynchrone. Toutefois, le circuit synchrone continue d'activer la partie de l'arbre d'horloge en amont des cellules de *clock gating* à chaque cycle tandis que DESYNC-AFSM active seulement l'arbre d'horloge d'un banc de registres précis (avec en plus la ligne de délais et le contrôleur). Comme ces arbres d'horloge locaux sont plus petits, cela amène à un gain de consommation en énergie. Notons aussi que notre analyse inclut uniquement le circuit. Dans un système synchrone, un générateur d'horloge est présent et ajoute sa surface et sa consommation au bilan du circuit synchrone. Par

conséquent, une analyse au niveau système serait bien plus avantageuse pour le circuit asynchrone. Dans le cas de DESYNC-STD, la puissance moyenne consommée est plus grande que son équivalent synchrone avec une augmentation de 63,36 %. Cette augmentation est due au fait que la moitié des contrôleurs sont actifs dans le circuit en permanence. De ce fait, un plus grand nombre d'éléments séquentiels sont activés à chaque cycle dans le circuit DESYNC-STD que dans SYNC. Nous pouvons dire que DESYNC-STD est la désynchronisation d'un circuit sans *clock gating*. La consommation d'énergie est quant à elle doublée, avec une augmentation de 107 %. Si la comparaison est faite avec la version sans *clock gating* du circuit synchrone, DESYNC-STD montre une diminution de 17,1 % de la puissance moyenne. Néanmoins, comme DESYNC-STD est plus lent que le circuit synchrone, il n'y a pas d'amélioration de la consommation d'énergie.

Cette étude montre que la technique de désynchronisation employée dans ces travaux apporte une bonne amélioration par rapport à la désynchronisation classique présente dans l'état de l'art. Le circuit désynchronisé avec une AFSM mène à une moins grande surface, une amélioration de la vitesse ainsi qu'une diminution de la consommation d'énergie comparé aux travaux réalisés en [48]. Nous pouvons noter que la consommation est presque divisée par deux. En comparaison avec le circuit synchrone d'origine, le circuit asynchrone résultant de la désynchronisation est similaire en surface et en vitesse, mais permet de réduire la consommation d'énergie d'environ 10 % (hors circuit de génération du signal de l'horloge). Nous avons donc une méthode permettant d'obtenir rapidement un circuit asynchrone très similaire à son équivalent synchrone, mais qui présente d'autres avantages comme une plus grande robustesse.

3.3 Application à la HLS

Maintenant que la méthode de désynchronisation a été comparée à l'état de l'art, il reste à démontrer que notre flot de conception, appliqué à la HLS, fonctionne correctement. Pour ce faire, nous avons réalisé un AES grâce à l'outil Catapult HLS. Ce circuit a été conçu en 65 nm de STMicroelectronics et a mené à la fabrication d'un prototype ASIC pour valider l'ensemble du flot, de l'architecture asynchrone développée et la génération des contraintes temporelles à l'implantation physique.

3.3.1 Architecture du prototype

Le prototype, visible figure 3.5, est une collaboration entre plusieurs doctorants et intègre plusieurs circuits : une primitive de sécurité et deux AES asynchrones, dont notre AES fait par HLS. Ces deux derniers sont intégrés dans le même système de test.

Le circuit AES a été réalisé afin qu'un tour de l'algorithme en figure 3.1 soit effectué en 3 cycles. Au final, le circuit obtenu possède une FSM comportant 7 états et réalise un calcul complet en 29 cycles. Pour de plus amples explications sur l'architecture du prototype, voir l'annexe A.

La plupart des étages ont été synthétisés avec une fréquence de 110 MHz pour une tension de 1,10 V. En revanche, il est possible avec la méthode LCS de contraindre avec

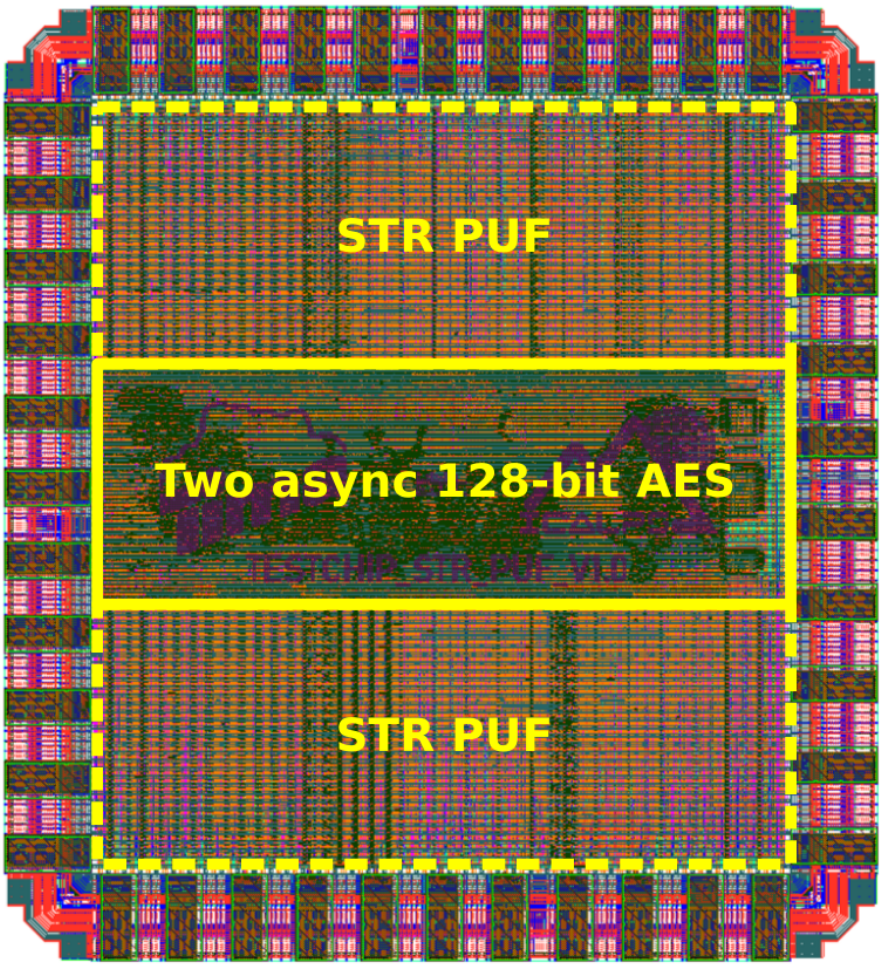


FIGURE 3.5 – *Layout du prototype en 65 nm.*

une fréquence spécifique chaque étage afin d'optimiser le circuit. Dans notre cas, deux étages du circuit peuvent être contraints jusqu'à 180 MHz. Le circuit a été conçu avec les logiciels de la suite Synopsys : Design Compiler pour la synthèse, IC Compiler pour le PnR et PrimeTime pour les différentes analyses des contraintes temporelles.

3.3.2 Résultats

La surface est donnée après la phase de PnR finale du circuit. Les autres mesures sont faites directement sur le circuit après fabrication.

En termes de surface, l'AES asynchrone possède une taille de 49640 μm^2. L'AFSM représente 1,85 % de la surface totale du circuit, pour une augmentation de 1,6 % par rapport au circuit synchrone d'origine. Cette augmentation est un peu plus grande que celle obtenue dans les résultats de la section 3.2.2, mais des marges ont été ajoutées sur les délais des lignes de requête pour augmenter la robustesse du circuit. Cela permet d'implémenter très facilement des méthodes de type DVS.

Le circuit est tout d'abord analysé après PnR à l'aide de simulation logique rétro-annotée. Les résultats en vitesse et en consommation pour une tension nominale de 1,2 V à 25 °C sont présentés dans la table 3.3.

TABLEAU 3.3 – *Simulation du temps de calcul et consommation du prototype*

	AES-AHLS
Temps de calcul (ns)	222
Puissance statique (nW)	311
Puissance dynamique (mW)	6,83
Consommation d'énergie (nJ)	1,49

Le prototype a été testé et validé fonctionnellement pour une plage allant de 1,3 V à 600 mV. Les mesures réalisées sur le circuit sur le temps de calcul et la consommation d'une opération sont présentées sur les graphes en figure 3.6. L'énergie du circuit, présente en figure 3.7, est déduite de ces deux mesures. Malheureusement, il n'a pas été possible d'effectuer des mesures de courant en dessous d'une tension de 700 mV car les équipements de test n'étaient pas assez précis.

Tout d'abord, il est à noter que les mesures à la tension nominale 1,2 V sont cohérentes avec les résultats de simulation en table 3.3, avec moins de 10 % d'écart entre les deux. En passant à une tension d'alimentation de 700 mV, le temps de calcul du circuit est multiplié par 13,5 par rapport à celui à la tension nominale tandis que la puissance moyenne est diminuée de 98 %. La puissance moyenne consommée diminue fortement en étant divisée par 47, mais comme le temps de calcul augmente aussi, la diminution ne sera pas la même pour l'énergie. Dans ce cas, on observe une diminution de 71,4 % entre l'énergie à tension nominale et celle à 700 mV.

Comme nous pouvons l'observer avec la plage étendue de fonctionnement du circuit, les circuits désynchronisés apportent une robustesse qu'il serait difficile d'obtenir avec un circuit synchrone. Notre flot de désynchronisation nous permet donc d'acquérir rapidement les avantages des circuits asynchrones au prix d'une augmentation de surface anecdotique tout en conservant des performances similaires.

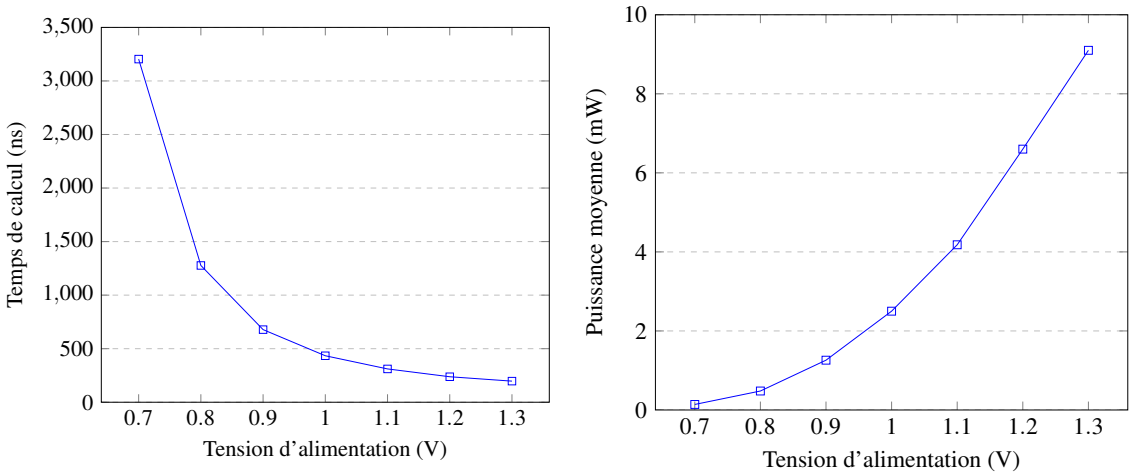


FIGURE 3.6 – *Caractérisation du prototype suivant la tension d'alimentation pour (a) le temps de calcul et (b) la puissance moyenne consommée.*

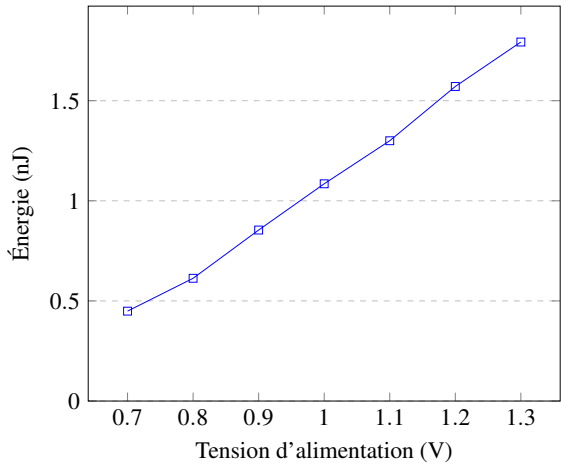


FIGURE 3.7 – *Mesure de la consommation d'énergie suivant la tension d'alimentation.*

3.4 Conclusion

L'étude menée dans ce chapitre a montré une nette amélioration des performances des circuits désynchronisés par notre méthode par rapport à la méthode de désynchronisation classique proposée par J. Cortadella. L'augmentation de surface est limitée par la diminution du nombre de contrôleurs dans l'AFSM et la vitesse et la consommation d'énergie sont grandement améliorées (respectivement de 21 % et 45 %). Il en résulte des circuits asynchrones très semblables aux circuits synchrones d'origine : la surface et la vitesse restent similaires tandis qu'une diminution de la consommation d'énergie est observée.

Cette méthode de désynchronisation, remplaçant la FSM d'origine par une AFSM, a été implémentée sur un outil de HLS. Ce flot a été validé par un prototype, un AES, fabriqué sur silicium. Ceci montre que la désynchronisation est correctement effectuée et que les contraintes temporelles des circuits désynchronisés sont convenablement générées. Un autre prototype a été développé avec un circuit plus complexe, mais n'a pas encore été testé. Il sera décrit dans la suite de ce travail.

Le flot de conception développé dans cette thèse permet de concevoir facilement et rapidement des circuits à données groupées. Il est désormais possible de générer des circuits asynchrones complexes à partir d'une description de haut niveau, tout en automatisant la génération et la vérification des contraintes temporelles locales à l'intérieur du circuit. De plus, ce flot repose sur des outils de CAO commerciaux standards et, par conséquent, ne nécessite aucun coût additionnel en termes d'outils ou de formations.

Les circuits asynchrones obtenus sont similaires à leurs équivalents synchrones mais possèdent deux grands avantages. Tout d'abord, ces circuits sont intrinsèquement événementiels. Il n'y a en effet aucune activité en dehors du calcul des données. De plus, ils sont bien plus robustes. Ces deux éléments rendent les circuits asynchrones intéressants pour implémenter des méthodes de gestion de consommation événementielles comme nous le verrons dans la suite de ce travail, avec l'implémentation d'une stratégie de polarisation dynamique du substrat en technologie FDSOI.

Deuxième partie

Stratégie de polarisation de substrat

4
Technologie FDSOI

Sommaire

4.1 Introduction 66
4.2 Limitations des technologies classiques 66
4.3 Alternatives au Bulk CMOS 68
4.3.1 FinFET 68
4.3.2 FDSOI 68
4.4 Caractéristiques des technologies FDSOI 69
4.4.1 Isolation du substrat 69
4.4.2 Polarisation du substrat 70
4.5 Méthodes de polarisation de substrat 71
4.6 Conclusion 72

4.1 Introduction

La réduction de la dimension des transistors a été une priorité dans l'industrie microélectronique. Cette réduction, représentée par la loi de Moore, a été rapide et a permis de grandes améliorations en taille, en vitesse et en consommation. Néanmoins, ces dernières années il est devenu de plus en plus difficile de suivre cette tendance. En effet, les transistors bulk CMOS atteignent leurs limites. Des phénomènes physiques tel que les effets de canal court empêchent d'obtenir les performances espérées. Si leur taille est bien réduite, leurs performances ne sont pas réellement améliorées dans les noeuds technologiques récents. Il a donc été nécessaire que l'industrie se tourne vers de nouveaux types de transistor [93].

En plus de cette thématique, avec la demande croissante de systèmes économes en énergie ciblant les applications portables ou l'IoT, la consommation d'énergie a acquis une considération accrue au cours des dernières années. Les techniques traditionnelles pour surmonter ce problème sont le DVS ou le *power gating*. La première méthode réduit la consommation d'énergie au prix d'une dégradation des performances. La seconde élimine la majeure partie de la consommation pendant la période d'inactivité du circuit, au prix d'une mise en oeuvre complexe et d'une latence élevée pour son activation et sa désactivation.

Désormais, la technologie FDSOI (*Fully Depleted Silicon on Insulator*) [94] offre de nouvelles possibilités de conception dans les noeuds technologiques récents. En plus de limiter les fuites grâce à son isolation entre les transistors et le substrat, il permet une utilisation intensive de la polarisation de substrat. Cette technique, qui modifie la tension de seuil du transistor V_{th}, était marginale en raison de sa portée limitée dans les technologies standards bulk CMOS. Néanmoins, la technologie FDSOI améliore considérablement la gamme possible des tensions de polarisation de substrat et permet son utilisation en remplacement ou en complément des techniques conventionnelles.

Ce chapitre présente les limitations existantes dans les transistors conventionnels ainsi que les solutions existantes comme le FDSOI ou le FinFET. Nous nous focaliserons sur la technologie FDSOI, ses caractéristiques ainsi que les opportunités de conception possibles. De plus, nous présenterons les méthodes existantes utilisant la polarisation de substrat.

4.2 Limitations des technologies classiques

Les transistors bulk CMOS constituent la technologie "traditionnelle" des circuits intégrés. La figure 4.1 présente l'architecture d'un transistor de type N, ou NMOS. Le drain et la source sont respectivement notés D et S sur le schéma, tandis que la grille qui contrôle le canal est notée G. Un transistor consomme de l'énergie statique, c'est-à-dire en l'absence d'activité, et de l'énergie dynamique lorsque le transistor change d'état. Si la première était négligeable au-dessus d'une taille de canal de 100 nm, cette consommation devient importante dans les noeuds technologiques inférieurs avec notamment l'apparition des effets de canaux courts ou *Short Channel Effects* (SCE).

Tout d'abord, le substrat forme une jonction PN avec le drain et la source. Il y a donc une consommation statique dans le transistor, représentée par les courants de fuite I_{pn}

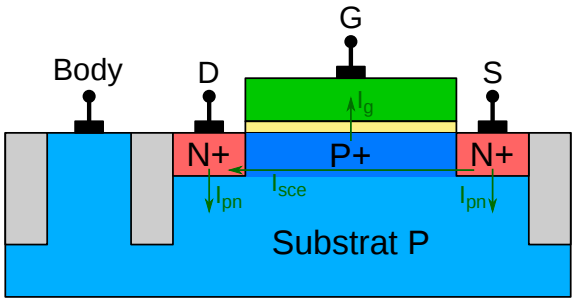


FIGURE 4.1 – *Schéma d'un transistor NMOS en technologie bulk CMOS.*

sur la figure. De plus avec une structure isolante de quelques couches atomiques entre la grille et le canal, des courants par effet tunnel se forment entre grille et canal. Ce courant de fuite est noté I_g.

Le courant I_{sce} apparaît lorsque la taille du canal devient faible. Du fait de la jonction PN présente au niveau du drain et de la source, il existe deux zones de déplétion. Ces zones sont représentées en pointillé rouge figure 4.2. Ces zones de déplétion se rejoignent pour les transistors de petite taille, comme décrit sur le schéma de droite. La barrière de potentiel est alors diminuée et un courant se crée entre le drain et la source. Lorsqu'une tension est appliquée sur le drain, la barrière de potentiel est encore diminuée. Le courant parasite due aux SCE est donc accentué par cet effet appelé *Drain Induced Barrier Lowering* (DIBL).

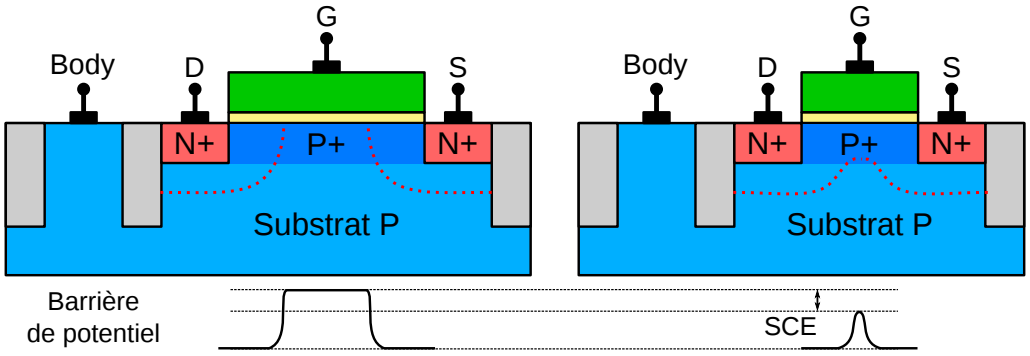


FIGURE 4.2 – *Apparition des SCE avec la diminution de la taille du canal.*

En plus des courants de fuite au sein du transistor, le dopage du canal influe lui aussi sur les caractéristiques du transistor. Lors de la fabrication du circuit, ce dopage varie aléatoirement par un phénomène appelé *Random Dopants Fluctuation* (RDF). Cette variation augmente avec la diminution de la taille du canal et fait différer la tension de seuil V_{th} du transistor par rapport à la valeur ciblée. Par conséquent, pour prendre en compte ces variations de fabrication lors de la conception, des marges additionnelles doivent être ajoutées, ce qui limite les performances du circuit.

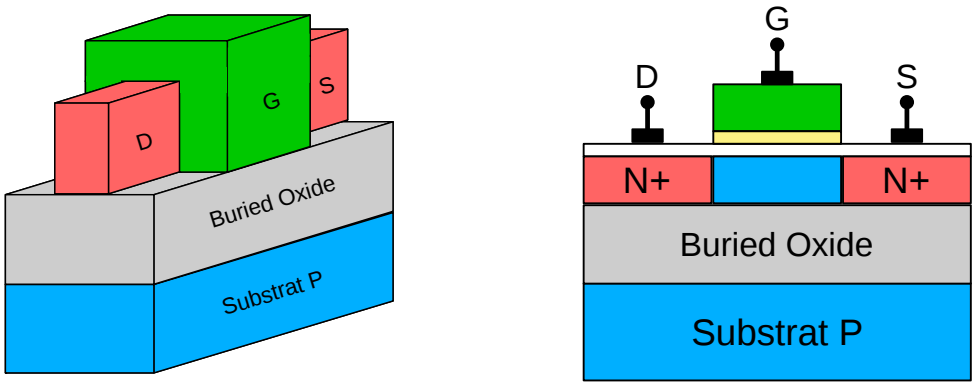
L'augmentation de la consommation due aux SCE et les pertes de performances avec le RDF entravent l'utilisation des transistors bulk CMOS avec les procédés de fabrication les plus avancés.

4.3 Alternatives au Bulk CMOS

Les limitations présentées dans la section précédente empêchent l'amélioration des caractéristiques des circuits malgré la réduction de la dimension des transistors. Il est donc nécessaire de se tourner vers de nouveaux types de transistors pour résoudre ces problèmes. Dans ce cadre, deux technologies se sont répandues cette dernière décennie dans la mise en place des nouveaux noeuds technologiques : les transistors FinFET et la technologie FDSOI.

4.3.1 FinFET

Un transistor FinFET [95] (pour *Fin Field Effect Transistor*) possède une structure 3D, comme présenté en figure 4.3. Le transistor lui-même est réalisé hors du substrat, et ses composants sont isolés par une couche d'oxyde. La grille, désignée par la lettre G sur les schémas, entoure quasiment complètement le canal pour un meilleur contrôle électro-statique [96].



(a) Vue 3D d'un transistor.

(b) Vue en coupe d'un transistor.

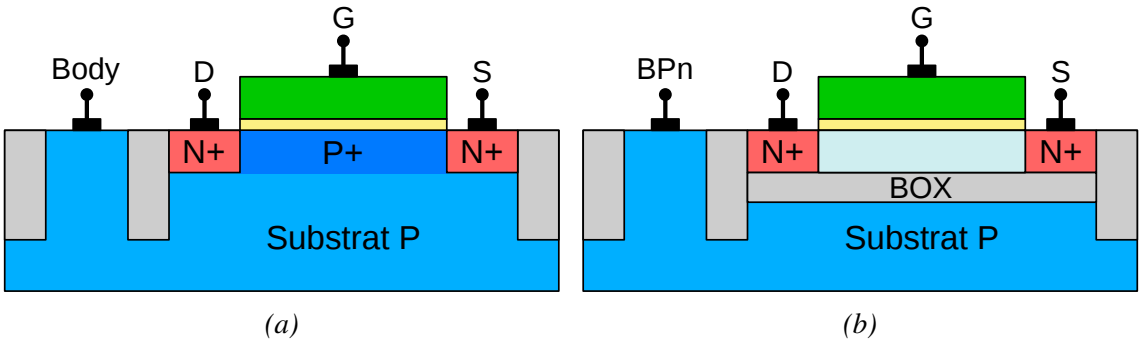
FIGURE 4.3 – *Schéma d'un transistor FinFET.*

Ce type de transistor apporte une grande performance [95] car le canal est contrôlé par la grille sur trois de ces faces. Mais son caractère 3D demande à changer les équipements de production et les circuits ont donc des coûts de fabrication plus élevés.

4.3.2 FDSOI

La technologie FDSOI [94] diffère de la technologie standard bulk CMOS par l'ajout d'une couche d'oxyde (BOX pour *buried oxide*) entre le substrat et les transistors. Cette couche isole les connexions de drain et de source du substrat, ce qui réduit ainsi les fuites du transistor. La figure 4.4 montre les différences entre les transistors bulk et FDSOI. De plus, le canal est désormais non dopé, ce qui élimine la fluctuation aléatoire du dopant.

Cette structure offre de nouveaux degrés de liberté aux concepteurs [97], tout en conservant une technologie planaire. Par exemple, la polarisation du substrat est grandement améliorée et peut être utilisée dans des méthodes de gestion de la consommation. De plus, il est possible de modifier le comportement du transistor en changeant sa tension de seuil. Ces avantages seront détaillés dans la section suivante.



(a) *(b)*

FIGURE 4.4 – *Transistors NMOS en technologie bulk CMOS (a) et FDSOI (b).*

Ces deux technologies sont actuellement utilisées pour remplacer le traditionnel bulk CMOS dans les transistors à très faible dimension. Par exemple, les transistors FinFET sont largement utilisés pour les procédés de fabrication de 7 nm ou 5 nm tandis que la technologie FDSOI est produite pour des noeuds de 28 et 22 nm et est annoncée en 12 nm. La technologie FinFET est largement adoptée dans les domaines où la performance est essentielle mais demande des coûts plus élevés pour fabriquer des transistors 3D [98]. Dans des applications où les coûts et la consommation sont primordiaux, la technologie FDSOI s'avère prometteuse [99]. C'est pourquoi notre choix se porte sur cette dernière pour concevoir des circuits basse consommation.

4.4 Caractéristiques des technologies FDSOI

Nous allons maintenant voir en détail ses caractéristiques principales qui sont l'amélioration des performances globales des transistors, la configuration possible de ces derniers par polarisation du substrat.

4.4.1 Isolation du substrat

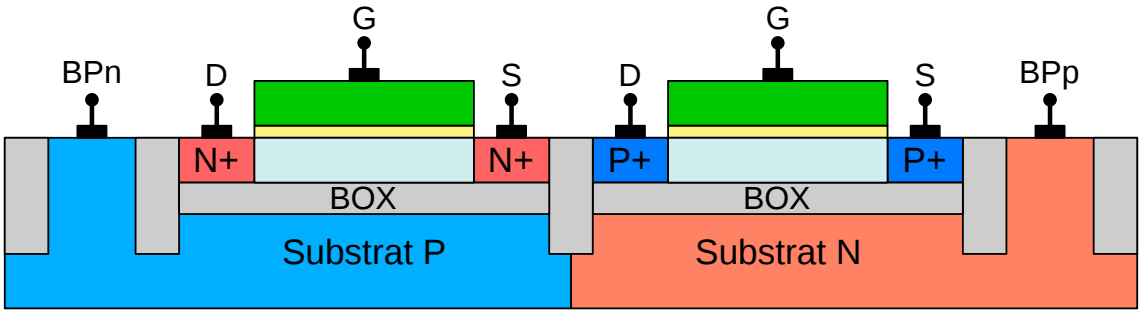


FIGURE 4.5 – *Schéma des transistor NMOS et PMOS en technologie FDSOI.*

Un schéma des transistors NMOS (à gauche) et PMOS (à droite) est présenté en figure 4.5. L'ajout de la BOX permet plusieurs améliorations [100] :

— La connexion entre le substrat et les ports du transistor, le drain et la source, est

éliminée. Par conséquent, les courants de fuite des transistors sont diminués sans la jonction PN formée par ces éléments.

— Les capacités parasites des transistors sont fortement réduites, car la surface de diffusion du drain et de la source est plus faible. La consommation dynamique est elle aussi diminuée.

— Le phénomène de *latch-up* est impossible. En effet, un court-circuit entre la masse et la tension d'alimentation pouvait se former sous certaines conditions avec l'apparition d'un transistor bipolaire parasite. Comme les transistors sont désormais isolés les uns des autres, cet effet n'existe plus.

Le canal est désormais non dopé. Ceci permet de supprimer le RDF, et donc d'avoir moins de variabilité sur la tension de seuil et un meilleur contrôle sur le canal. Les SCE sont aussi fortement réduits [101]. Les performances des transistors en sont alors améliorées.

Comme les jonctions PN entre le substrat et le transistor sont supprimées, la seule jonction PN encore existante correspond à la jonction entre les substrats des transistors NMOS et PMOS. Grâce à cela, la tension pour polariser le substrat peut être bien plus importante que pour les transistors standards, où la tension de polarisation ne peut pas dépasser 0,3 V. La polarisation du substrat sera détaillée dans la section suivante.

4.4.2 Polarisation du substrat

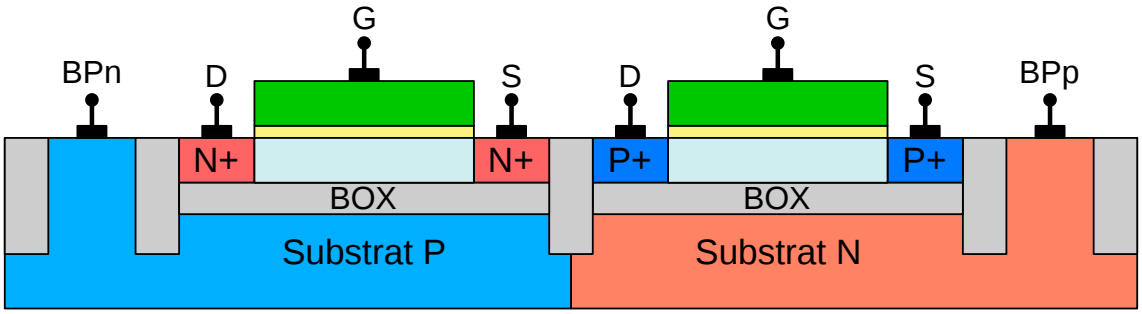
Comme le substrat est isolé du reste des composants, il est possible d'inverser le dopant du substrat comme le montre la figure 4.6. Dans ce cas, la source, le drain et le substrat ont le même type de dopant, appelé configuration *flip-well*. En technologie FDSOI, les configurations conventionnelles sont utilisées pour des transistors à tension de seuil standard (RVT - Regular V_{th}) et les *flip-wells* avec des transistors avec une tension de seuil faible (LVT - Low V_{th}). Les transistors RVT utilisent une polarisation initiale typique : 0 V pour le transistor NMOS et 1 V pour le transistor PMOS. Au contraire, les transistors LVT utilisent une tension nulle pour les deux types de transistors.

Il existe deux types de polarisation de substrat :

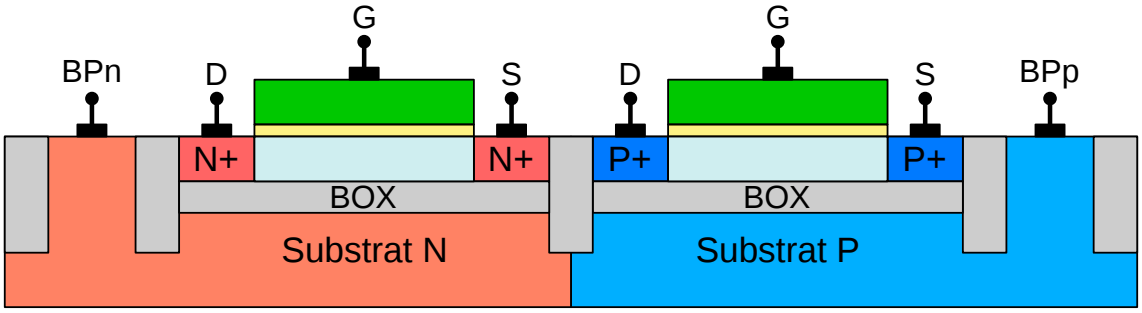
— La polarisation de substrat inverse, ou *Reverse Body Biasing*, (RBB) augmente les tensions de polarisation PMOS et diminue les tensions de polarisation NMOS par rapport à la tension de polarisation de substrat standard. Il en résulte une diminution de la puissance des courants de fuite au prix d'une détérioration des performances du transistor.

— La polarisation de substrat direct, ou *Forward Body Biasing* (FBB), diminue les tensions de polarisation PMOS et augmente les tensions de polarisation NMOS par rapport à la tension de polarisation de substrat standard. Il en résulte une augmentation de la vitesse du circuit au détriment de courants de fuite plus importants.

Si ces deux types de polarisation peuvent être appliqués aux transistors RVT et LVT, la plage autorisée est différente. En effet, la jonction entre les substrats N et P limite la plage de polarisation du substrat. Par conséquent, le FBB (respectivement RBB) s'applique aux transistors LVT (respectivement RVT) car la gamme de polarisation de substrat peut aller jusqu'à 3.3 V.



(a) Configuration conventionnelle.



(b) Configuration flip-well.

FIGURE 4.6 – *Configurations possibles pour des transistors FDSOI.*

4.5 Méthodes de polarisation de substrat

Pour polariser une région donnée d'un circuit, il est nécessaire d'isoler cette région du reste du circuit. Ainsi, il est possible de concevoir un circuit avec plusieurs îlots de polarisation, appelé *Body Bias Domain* (BBD), chacun avec leur propre tension de polarisation du substrat. Plusieurs travaux [102, 103] s'intéressent à la partition optimale des BBDs en fonction de l'utilisation du circuit et des tensions de polarisation possible. Néanmoins, il manque une analyse des résultats prenant en compte les coûts d'implémentation. D'autres [104, 105] cherchent à former des BBDs sans polarisation de substrat, mais pour mélanger les configurations RVT et LVT et améliorer les performances globales du circuit.

Il est aussi possible de changer dynamiquement la polarisation de substrat. Pour améliorer la consommation ou la performance d'un circuit, de nombreuses techniques [106, 107] ont été proposées pour tirer parti des capacités de polarisation de substrat. Le circuit est divisé en grands BBDs distincts de la taille d'une IP et ont leur propre tension de polarisation de substrat. Une unité de contrôle (PMU sur la figure 4.7) est ajoutée pour contrôler la tension de polarisation en fonction du mode de fonctionnement de l'IP. Ce type de stratégie permet d'obtenir de grandes améliorations en performance [108] mais est compliqué à mettre en place. La polarisation de substrat dynamique est aussi utilisée pour compenser les variations PVT. L'unité de contrôle est alors connectée à des capteurs pour définir une tension de polarisation qui change avec l'environnement.

Afin de réaliser ces stratégies, il est nécessaire d'ajouter des cellules pour polariser le substrat de chaque îlot de polarisation. Ce sont des générateurs de tension de substrat, ou *Body Bias Generator* (BBG) [109, 110]. Les BBGs sont généralement de grandes IPs analogiques qui peuvent polariser précisément le substrat d'un grand BBD à une tension donnée. Leur architecture est composée de convertisseurs numérique-analogique ainsi que

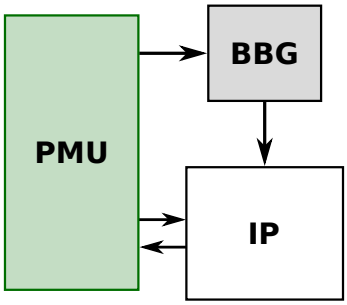


FIGURE 4.7 – *Architecture d'un système utilisant une polarisation de substrat dynamique.*

de *charge pumps* [111]. Comme ces BBGs sont gros et complexes, les BBDs doivent avoir une taille minimale conséquente pour que leur utilisation soit pertinente.

Certaines approches ont couplé la polarisation de substrat avec les circuits asynchrones pour mettre en place des solutions originales. Afin de simplifier les implémentations vues précédemment et optimiser la consommation d'énergie, Hamon *et al.* [112] proposent de polariser des zones à l'échelle d'un étage de pipeline dans des circuits QDI. La figure 4.8 donne un schéma de principe de cette stratégie. Pour simplifier la gestion de la polarisation, celle-ci est réalisée en fonction de l'activité dans le circuit à l'aide de détecteurs d'activité. En revanche, les BBGs classiques ne sont pas bien adaptés à une telle stratégie, car ils sont trop grands et trop lents. Par conséquent, la méthode repose sur de petits BBG polarisant les BBDs dans un mode ON-OFF. Cette idée a été étudiée et implémentée dans [113, 114], mais tous ces travaux sont basés sur des circuits QDI. Bien que les circuits QDI soient très robustes, ils sont également très volumineux par rapport à leurs homologues synchrones et ne sont pas supportés par les outils de CAO conventionnels.

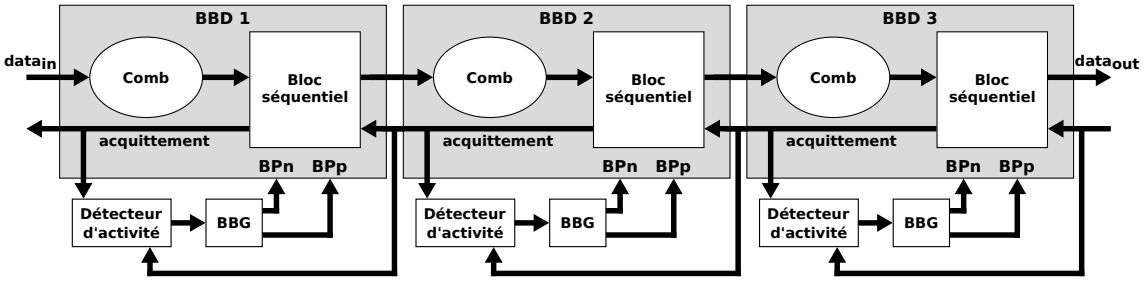


FIGURE 4.8 – *Principe de la méthode de polarisation de substrat pour un circuit linéaire.*

4.6 Conclusion

Afin de répondre aux enjeux actuels des systèmes de l'IoT, la technologie FDSOI semble être une candidate de choix. Avec sa couche d'oxyde entre les transistors et le substrat, elle résout tout d'abord les limitations observées dans les transistors conventionnels, permettant la mise en place de noeuds technologiques avancés. Les courants de fuite, statiques et dynamiques, sont fortement diminués et un meilleur contrôle du canal permet une amélioration des performances. Enfin, l'intervalle possible pour la tension de

la polarisation de substrat est fortement augmenté pour modifier la tension de seuil des transistors. Cette caractéristique a ouvert de nouvelles stratégies de conception.

Un certain nombre de méthodes ont été développées tirant profit de la polarisation du substrat. Ces méthodes permettent de compenser les variations des procédés de fabrication, de l'environnement ou bien de réaliser un compromis entre performance et consommation du circuit. Néanmoins ces méthodes nécessitent la plupart du temps une mise en place complexe, avec des composants externes imposants comme les BBGs ou les unités de contrôle.

Dans ce domaine, les circuits asynchrones et leur comportement événementiel ouvrent de nouvelles perspectives. Ces circuits sont automatiquement en veille lorsqu'il n'y a aucune donnée et sont plus robustes que les circuits synchrones. La méthode présentée par Hamon *et al.* [112] est un bon exemple d'une méthode maximisant les gains en consommation à l'aide de l'architecture particulière des circuits asynchrones. La gestion de la polarisation de substrat en est alors simplifiée.

5

Stratégie de polarisation de substrat

Sommaire

5.1 Introduction 76
5.2 Définition de la stratégie 76
5.2.1 Principe de la stratégie 78
5.2.2 Éstimation de l'efficacité 81
5.3 Implémentation de notre stratégie 83
5.4 Évaluation de la méthode 86
5.4.1 Simulations numériques 86
5.4.2 Conception d'un prototype 88
5.5 Conclusion 90

5.1 Introduction

La technologie FDSOI présenté au chapitre précédent est prometteuse pour remplacer les transistors bulk CMOS dans les applications basse consommation, tout d'abord grâce aux caractéristiques intrinsèques à cette technologie qui réduisent les courants de fuite, mais surtout grâce à la polarisation de son substrat. Ce levier a permis de développer un grand nombre de techniques qui permettent soit de compenser les variations du circuit, soit de trouver un compromis entre les performances et la consommation d'énergie.

Néanmoins la quasi-totalité de ces méthodes polarise une grande surface du circuit et possède un système de contrôle conséquent. En effet, les BBGs ainsi que l'unité de gestion associée sont grands et complexes. Afin de minimiser la consommation d'énergie sans perte de performance, l'approche introduite par Hamon *et al.* [112] est très intéressante. Elle polarise de petits BBDs en fonction de l'activité des données et simplifie grandement la mise en place de la polarisation de substrat. De plus, son caractère événementiel la rend adaptée pour les circuits asynchrones. Originellement, cette approche est basée sur des circuits QDI. Bien que les circuits QDI soient très robustes, ils sont également très volumineux par rapport à leurs homologues synchrones et ne sont pas supportés par les outils de CAO conventionnels.

Nous avons donc décidé d'appliquer cette méthode aux circuits asynchrones à données groupées développés dans cette thèse. Ils nous permettent de profiter de la plus grande robustesse de ces circuits asynchrones avec un surcoût en surface modeste, voire inexistant. Nous présenterons donc les principes de cette méthode et son adaptation à notre architecture asynchrone spécifique. Pour faciliter la mise en place de cette méthode, nous avons défini un flot de conception pour implémenter la polarisation de substrat dynamique et les composants nécessaires. Le flot est présenté en figure 5.1. La partie désynchronisation correspond au flot décrit dans le chapitre 2. Tout comme le flot de désynchronisation, la partie du flot gérant la polarisation de substrat est totalement compatible avec les outils de CAO standards utilisés dans l'industrie.

Pour obtenir une première estimation des gains en consommation de notre travail, des simulations numériques sont présentées et nous comparons notre méthode à une polarisation de substrat basique. De plus, nous avons conçu puis fabriqué un prototype afin de valider le flot de conception dans son ensemble et mesurer les résultats de notre méthode. Ce circuit est un réseau de neurones convolutionnel (CNN).

5.2 Définition de la stratégie

La stratégie événementielle de polarisation dynamique du substrat est particulièrement bien adaptée pour les circuits asynchrones, et permet de simplifier les unités de contrôle qui fonctionne en mode ON-OFF. Nous rappelons le principe de base de cette stratégie et nous présentons son adaptation à l'architecture asynchrone développée dans la première partie de ce travail de thèse. Afin d'estimer les gains de notre stratégie, nous appliquons le modèle présent en [113] sur notre architecture. La mise en place d'une polarisation de substrat nécessite l'ajout d'étapes supplémentaires durant l'implémentation physique du circuit. Nous définissons par conséquent un flot de PnR facilitant cette phase.

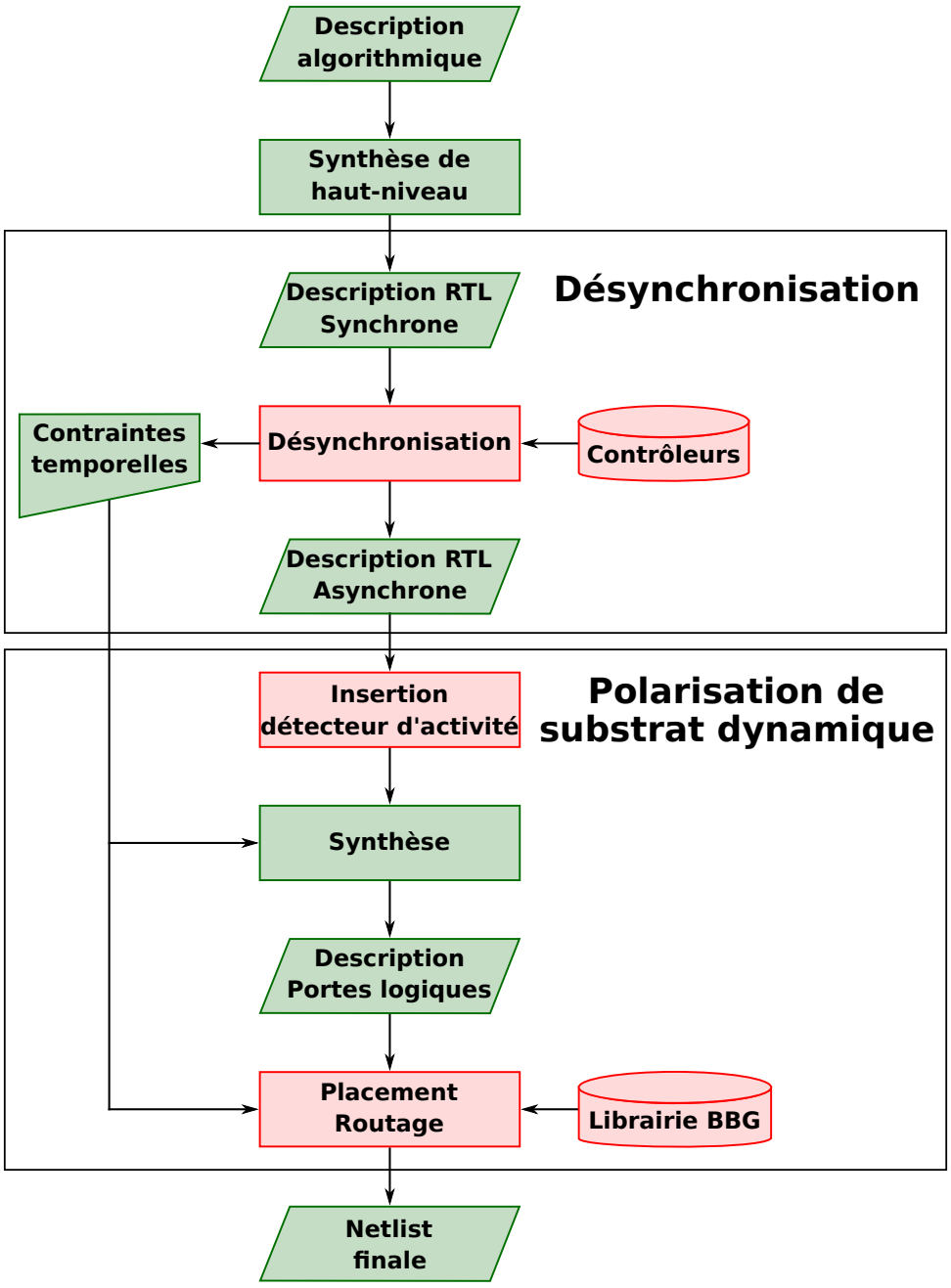


FIGURE 5.1 – *Flot implémentant la polarisation de substrat dynamique.*

5.2.1 Principe de la stratégie

Dans un but d'amélioration de l'efficacité énergétique, la stratégie adoptée consiste à abaisser autant que possible la tension d'alimentation du circuit, ce qui inévitablement nuit à la performance. Afin de ne pas dégrader les performances du circuit, la polarisation de substrat est utilisée en mode FBB (polarisation en direct), l'idée étant de compenser la perte de performance liée à la diminution de la tension d'alimentation. De cette façon, nous obtenons un circuit avec des performances identiques mais bénéficiant d'une efficacité énergétique grandement améliorée. Il est à noter que le gain est d'autant plus conséquent que l'on travaille à basse tension avec une polarisation de substrat maximale.

Afin de minimiser la complexité de l'unité de contrôle de la polarisation, nous utilisons une stratégie assez simple, appelée V_{th}-*hopping*. Lorsqu'il y a de l'activité à l'intérieur d'un BBD, la tension V_{th} des transistors est réduite pour améliorer la vitesse du BBD. D'autre part, les courants de fuite des transistors sont limités lorsqu'il n'y a plus d'activité en augmentant le V_{th}. Ainsi, nous utilisons des transistors LVT avec un schéma FBB : la tension de polarisation du substrat est appliquée lorsque le circuit traite des données et est désactivée dès qu'il est inactif, fixant V_{th} à sa valeur par défaut. Avec cette stratégie, chaque BBD nécessite un unique détecteur d'activité et un BBG avec un comportement ON-OFF.

Les circuits asynchrones présentent deux avantages principaux pour une stratégie de polarisation de substrat dynamique. Premièrement, leur robustesse est améliorée par rapport à un circuit synchrone. Par conséquent, il est possible de changer le paramètre V_{th} des transistors dans le circuit à la volée, sans aucune violation des contraintes temporelles. L'activation et la désactivation de la tension de polarisation sont alors simples. Deuxièmement, les signaux de communication entre les contrôleurs asynchrones (les signaux de requête et d'acquittement) indiquent l'activité des sous-blocs du circuit. Cela apporte une grande liberté pour choisir la granularité la plus appropriée pour les BBDs. Il sera très facile d'adapter les détecteurs d'activité en fonction de la granularité choisie.

La combinaison entre les circuits asynchrones et une stratégie simple de polarisation de substrat en V_{th}-*hopping* nous permet de réduire considérablement la complexité du contrôle de l'alimentation ainsi que la latence de l'activation et de la désactivation de la polarisation de substrat. Il est alors possible d'utiliser cette stratégie dynamique sur des blocs plus petits et plus rapides que ceux utilisés avec les méthodes habituelles de polarisation du substrat.

L'étude de Fesquet *et al.* [113] montre des gains mineurs en consommation lorsque la granularité des BBDs est au niveau du pipeline alors que la surface globale du circuit augmente fortement. Par conséquent, nous décidons de définir la granularité des BBDs au niveau d'un sous-bloc du système pour avoir un compromis entre augmentation de surface et gains en consommation. Cette granularité reste beaucoup plus fine que celle des IPs utilisée habituellement dans les stratégies conventionnelles de gestion de la consommation.

Dans notre cas, un sous-bloc du système correspond à une AFSM et son chemin de données associé. Notre stratégie a donc le fonctionnement suivant : le début d'une opération dans l'AFSM active la tension de polarisation, comme indiqué sur la figure 5.2. Cette tension de polarisation est active durant l'ensemble des calculs du circuit. Une fois les calculs terminés, la tension de polarisation est désactivée et l'AFSM attend une nouvelle donnée d'entrée. Le détecteur d'activité indique si le circuit est en fonctionnement. Pour ce faire, le détecteur est connecté aux signaux de requête de l'AFSM ainsi qu'aux inter-

faces avec l'environnement. Sa sortie est le signal *activity*, qui sert de signal de contrôle pour les BBGs. Les BBGs, connectés à la sortie du détecteur d'activité, génèrent la tension nécessaire à la polarisation des transistors NMOS et PMOS. Comme nous utilisons une stratégie simple, où la polarisation de substrat est soit active, soit inactive, l'architecture du BBG peut être grandement simplifiée.

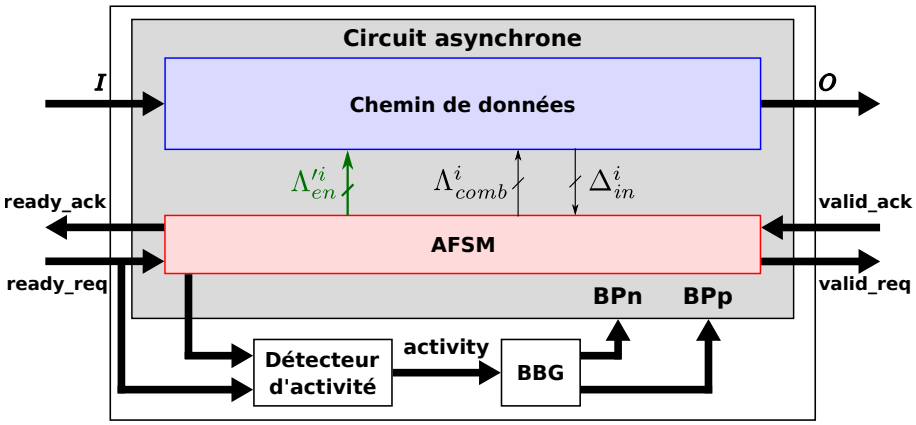


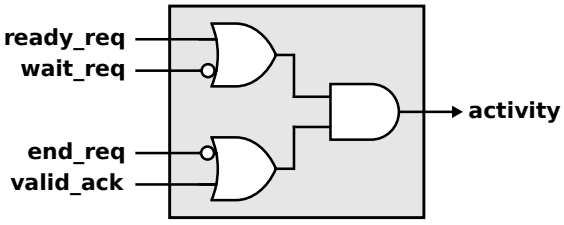
FIGURE 5.2 – *Principe de la stratégie de polarisation dynamique du substrat.*

Détecteur d'activité

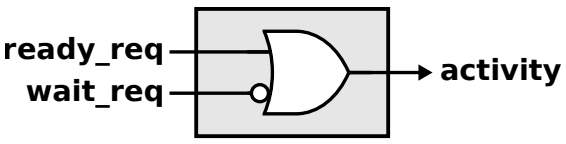
Comme notre stratégie de polarisation de substrat se situe au niveau d'un système asynchrone, la tension de substrat doit être active dès que l'AFSM se trouve dans un état de calcul. Les deux seuls états qui ne sont pas des états de calcul sont les états *wait* et *end*, qui communiquent avec l'environnement extérieur et attendent pour échanger les données respectivement d'entrée et de sortie.

Le détecteur d'activité va donc avoir une sortie active lorsque le circuit ne se trouve pas dans les états d'attente. Ceci est réalisé avec les signaux de requête *wait_req* et *end_req*, qui correspondent respectivement aux états *wait* et *end*. Les connexions avec ces états seraient suffisantes mais la prise en compte des signaux venant de l'environnement, *ready_req* et *ready_ack*, permet de réduire la latence de l'activation de la tension de substrat lors de l'arrivée de nouvelles données.

La figure 5.3a présente l'architecture du détecteur d'activité. Pour une question de simplicité de conception et d'intégration, il est réalisé en portes logiques standards. Son implémentation contient cinq portes logiques, sa taille est donc négligeable comparé au reste du circuit.



(a) Implémentation générique.



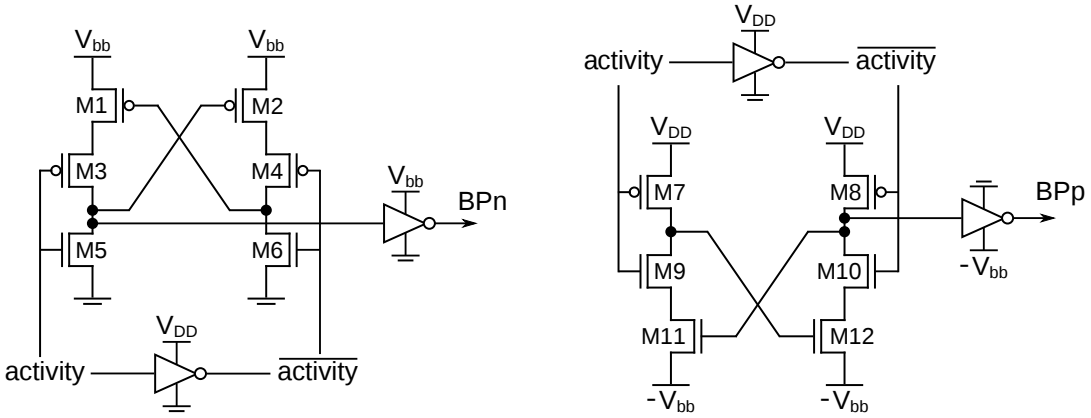
(b) Version optimisée.

FIGURE 5.3 – *Implémentation du détecteur d'activité.*

Lorsque le concepteur sait en amont que les données de sortie sont acceptées rapidement par l'environnement, il est alors possible d'optimiser le détecteur d'activité comme montré en figure 5.3b. Dans ce cas, il nous est possible de supprimer la connexion entre l'état de sortie *end* et le détecteur d'activité. Le détecteur obtenu est alors composé de seulement deux portes logiques, un inverseur et une porte OU.

Générateur de tension de substrat

Comme expliqué précédemment, le BBG n'a pas besoin de générer une gamme de tension de substrat précise mais seulement de fonctionner en mode ON-OFF. Son architecture s'en trouve grandement simplifier par rapport aux BBGs traditionnels. Le BBG est alors un translateur de tension (ou *level-shifter*) comme proposé dans [112]. Il existe un grand nombre d'architectures possibles pour ce composant, mais dans notre cas nous utilisons l'architecture classique de *Contention Mitigated Level Shifter* [115].



(a) Partie positive. *(b) Partie négative.*

FIGURE 5.4 – *Architecture d'un BBG.*

La figure 5.4 présente le schéma transistor du BBG. La partie positive correspond à un *level shifter* classique. Elle génère la tension V_{bb} pour les transistors NMOS lorsque le signal *activity* est actif. La partie négative est le symétrique du *level shifter* précédent et génère la tension négative $-V_{bb}$ pour les transistors PMOS. Il est conçu comme une cellule standard pour faciliter son intégration dans le flot de conception des circuits numériques.

Système global

L'architecture pour mettre en place la méthode au niveau d'un unique caisson de polarisation a été décrite, ainsi que ses différents composants. Il est désormais nécessaire de s'intéresser à la manière de générer un circuit possédant plusieurs BBDs.

Nos circuits sont générés par HLS pour être ensuite désynchronisés. Sous Catapult HLS, il est possible de définir des sous-blocs hiérarchiques au sein d'un même circuit. Ces sous-blocs possèdent chacun leur propre FSM et chemin de données. Un bloc hiérarchique servira alors de base pour un BBD. Cela permet de définir les différents BBDs du circuit dès sa description algorithmique, comme présenté sur la figure 5.5 où les lignes *#pragma design* indiquent les sous-blocs hiérarchiques.

Une fois la description RTL du circuit générée par l'outil de HLS, chaque bloc hiérarchique est désynchronisé par le flot de désynchronisation décrit dans le chapitre 2.

```
#pragma hls_design top
void circuit(...) {
   ...
   sous_bloc_1(...);
   sous_bloc_2(...);
   sous_bloc_3(...);
}

#pragma design
void sous_bloc_1(...) {
   ...
}

#pragma design
void sous_bloc_2(...) {
   ...
}

#pragma design
void sous_bloc_3(...) {
   ...
}
```

FIGURE 5.5 – *Exemple d'un circuit avec trois sous-blocs.*

L'interconnexion entre les différents sous-blocs se fait naturellement grâce aux signaux de requête et acquittement. Ensuite, les composants nécessaires à la polarisation dynamique du substrat (le détecteur d'activité et le BBG) sont ajoutés et connectés à chaque FSM asynchrone du circuit. Le circuit obtenu correspond à l'exemple en figure 5.6.

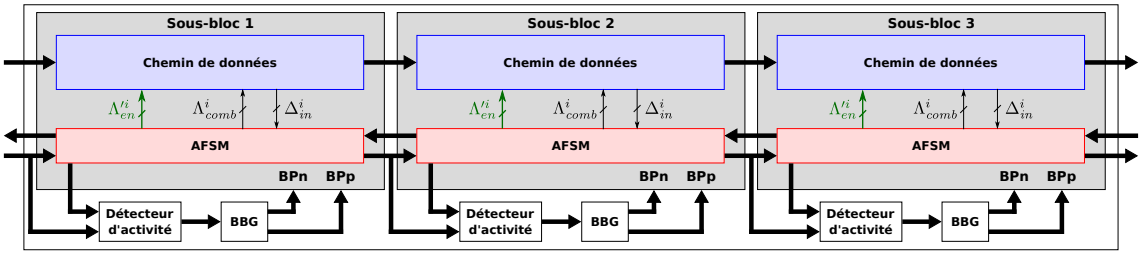


FIGURE 5.6 – *Exemple d'un circuit avec trois sous-blocs.*

Maintenant que la méthode événementielle de polarisation de substrat a été définie, il est nécessaire de connaître ses gains potentiels en consommation. Pour cela, nous modélisons dans la section suivante un système implémenté avec notre méthode.

5.2.2 Éstimation de l'efficacité

La stratégie de polarisation de substrat utilisée dans notre travail n'est efficace que sous certaines conditions. En effet, l'activation successive des BBDs produit un surcoût énergétique. Ainsi, l'énergie économisée pendant la période d'inactivité du circuit doit

être supérieure à l'énergie d'activation des BBDs du circuit. Sinon, il n'y a aucune amélioration par rapport à un système où la tension de substrat est constamment active.

Nous analysons un système linéaire composé de N étages et représenté sur la figure 5.7. Les données d'entrée, représentées en rouge, sont chargées en continu dans le circuit. Lorsque les données atteignent un étage, la tension de polarisation de substrat est activée et incrémente la courbe verte (indiquant le nombre d'étages polarisés). Lorsque toutes les données sont dans le système, les premiers étages commencent à entrer en mode inactif. Il y a donc trois phases dans ce scénario : une phase de chargement des données, une phase de régime permanent et, enfin, une phase de déchargement. Nous définissons Δt_{BB} comme la période où au moins un étage est actif et Δt_{noBB} comme le temps d'inactivité à partir duquel il n'y a plus de données dans le circuit. Nous désignons également par Δt_{BBi} la période active du i^{ieme} BBD.

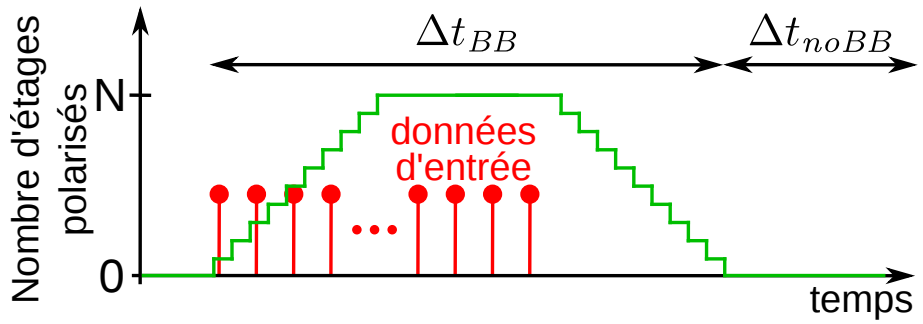


FIGURE 5.7 – *Variation du nombre de BBDs actifs dans le système (adapté de [114]).*

Le substrat d'un BBD est représenté par une capacité C_b et est chargé lorsque le BBD est activé. Pour le i^{ieme} BBD du système, l'énergie $E_{bias,i}$ consommée pendant l'activation ou la désactivation du substrat à une tension V_{bb} est définie par :

$$E_{bias,i} = C_{b,i} V_{bb}^2 \tag{5.1}$$

Ainsi, l'énergie globale E_{total} consommée par le circuit entier est donnée par l'équation suivante :

$$E_{total} = E_{BB} + E_{noBB} + 2 \sum_{i=0}^{N-1} E_{bias,i} \tag{5.2}$$

Durant la période Δt_{noBB}, la seule puissance consommée est la puissance statique sans polarisation du substrat, que nous notons P_l^0. L'énergie E_{noBB} est alors définie par l'équation 5.3. En toute rigueur il serait nécessaire d'inclure la consommation du détecteur d'activité et des BBGs. Néanmoins, la consommation de ces composants, qu'elle soit dynamique ou statique, est négligeable comparée au reste du système.

$$E_{noBB} = \Delta t_{noBB} P_l^0 \tag{5.3}$$

De même, l'énergie E_{BB} correspond à la puissance moyenne $P_{tot}^{V_{bb}}$ avec une tension de substrat V_{bb} durant la période Δt_{BB}. Mais dans ce cas, les phases de chargement et de déchargement du système permettent une réduction de la consommation d'énergie. En effet, durant la période Δt_{BB0} par exemple, seul le premier étage est polarisé avec une puissance moyenne de fuite notée $P_{l,0}^{V_{bb}}$. Par conséquent, la consommation statique des autres étages est réduite à $P_{l,j}^0$ comparé au cas où le circuit aurait été complètement polarisé. Une analyse similaire est faite pour chaque étage.

Nous notons $E_{s,pl}$ l'énergie économisée durant la phase de chargement. Si les phases de chargement et de déchargement sont équilibrées, alors ces deux cases économisent la même quantité d'énergie. Cela mène à la définition suivante :

$$E_{BB} = \Delta t_{BB} P_{tot}^{V_{bb}} - 2E_{s,pl}$$
$$\text{avec } E_{s,pl} = \sum_{i=0}^{N-1} \Delta t_{BBi} (\sum_{j=i+1}^{N-1} (P_{l,j}^{V_{bb}} - P_{l,j}^{0})) \tag{5.4}$$

Grâce aux équations 5.2, 5.3 et 5.4, nous connaissons précisément la consommation d'énergie de notre méthode. Afin de comparer cette méthode à un circuit avec une polarisation de substrat constante, nous définissons l'augmentation d'énergie E_O et l'énergie économisée E_S de notre méthode :

$$E_O = 2 \sum_{i=0}^{N-1} E_{bias,i}$$
$$E_S = \Delta t_{noBB} (P_l^{V_{bb}} - P_l^0) + E_{s,pl} \tag{5.5}$$

Pour améliorer l'efficacité énergétique du circuit avec une polarisation de substrat dynamique, nous cherchons le cas où E_S est supérieur à E_O. Nous déduisons donc de l'équation 5.5 le temps minimum de repos Δt_{noBB} pour une réduction effective de la consommation :

$$\Delta t_{noBB} > \frac{2V_{bb}^2 \sum_{i=0}^{N-1} (C_{b,i} - E_{s,pl})}{P_l^{V_{bb}} - P_l^0} \tag{5.6}$$

L'équation 5.6 présente la limite inférieure de notre méthode. Cette période d'inactivité, également connue sous le nom de *Minimum Idle Time* (MIT), permet de compenser l'augmentation d'énergie induite par cette stratégie de polarisation du substrat. Par conséquent, les gains en énergie de notre méthode dépendent fortement des scénarios de fonctionnement du système. À cette fin, nous définissons le taux d'activité *AR* du circuit :

$$AR = \frac{\Delta t_{BB}}{\Delta t_{BB} + \Delta t_{noBB}} \tag{5.7}$$

Ce taux d'activité *AR* est utilisé pour comparer les différents scénarios présents dans les simulations de notre méthode.

5.3 Implémentation de notre stratégie

Pour mettre en place une stratégie de polarisation de substrat, il est nécessaire de suivre un certain nombre de règles lors de l'implémentation physique du circuit. Les détecteurs d'activité sont ajoutés automatiquement dans la description du circuit avant la synthèse et ne posent aucune difficulté. En revanche, la phase de PnR nécessite des étapes supplémentaires plus complexes. Le flot de conception associé au PnR développé dans

ce travail de thèse est présenté en figure 5.8. En entrée de PnR, la description du circuit correspond à un circuit asynchrone désynchronisé et synthétisé et les contraintes temporelles sont générées par notre flot de désynchronisation. Tout comme pour la synthèse, ces contraintes issues de la méthode LCS permettent de vérifier toutes les contraintes temporelles locales d'un circuit asynchrone à l'aide des outils de CAO classiques.

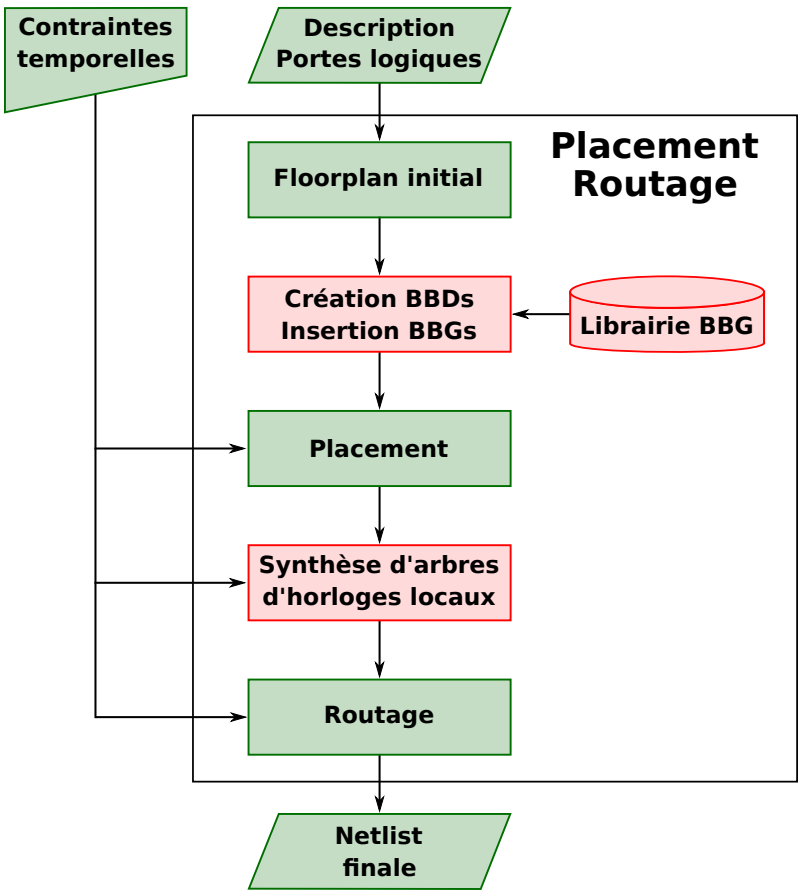


FIGURE 5.8 – *Flot implémentant la polarisation de substrat dynamique.*

Le PnR nécessite deux étapes supplémentaires, indiquées en rouge sur la figure 5.8. La première consiste à partitionner chaque sous-système au sein de son propre BBD et à insérer les BBGs afin de polariser localement les BBDs pour faciliter l'intégration de notre méthode à un flot de conception numérique. La seconde est la synthèse des arbres d'horloge locaux. Cela peut sembler incohérent car il n'y a plus de signal d'horloge dans le circuit, mais notre circuit désynchronisé possède des signaux de contrôle qui activent des groupes de registres. Ces signaux sont considérés comme des signaux d'horloge locaux et doivent être traités comme tout signal d'horloge dans un circuit synchrone. Cette étape n'est pas différente de la synthèse d'un arbre d'horloge standard, mais elle est effectuée localement.

Lorsque plusieurs BBDs sont implémentés en FDSOI, chacun d'entre eux possède sa propre tension de substrat. Pour éviter des courts-circuits dans le substrat de type P, il est nécessaire d'isoler les transistors avec une couche profonde de substrat de type N appelé *deep N-well*. La figure 5.9 représente cette isolation. En plus de cette isolation, les BBDs doivent avoir un éloignement minimal de plusieurs μm, ce qui induit une augmentation de surface conséquente.

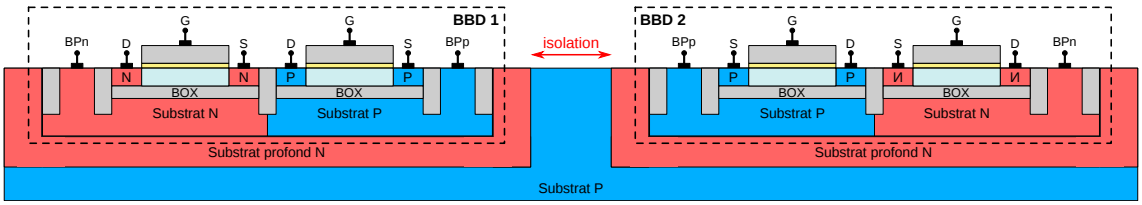


FIGURE 5.9 – *Vue en coupe de l'isolation de deux BBDs.*

Pour réaliser ces contraintes lors de l'implémentation physique du circuit, les cellules appartenant à un même BBD doivent être regroupées. Cela mène au plan de puce de la figure 5.10 où le circuit est partitionné en quatre BBDs représentés en pointillé. Maintenant que le partitionnement est effectué, il reste à mettre en place le système de polarisation du substrat. Une grille d'alimentation locale est réalisée à l'intérieur de chaque BBD pour alimenter les substrats N et P. Les BBGs sont ensuite placés au sein du BBD (en gris sur la figure). Ceux-ci sont connectés aux rails d'alimentation apportant les tensions au substrat ainsi qu'aux détecteurs d'activité. Ses sorties sont quant à elles connectées à l'alimentation du substrat du BBD.

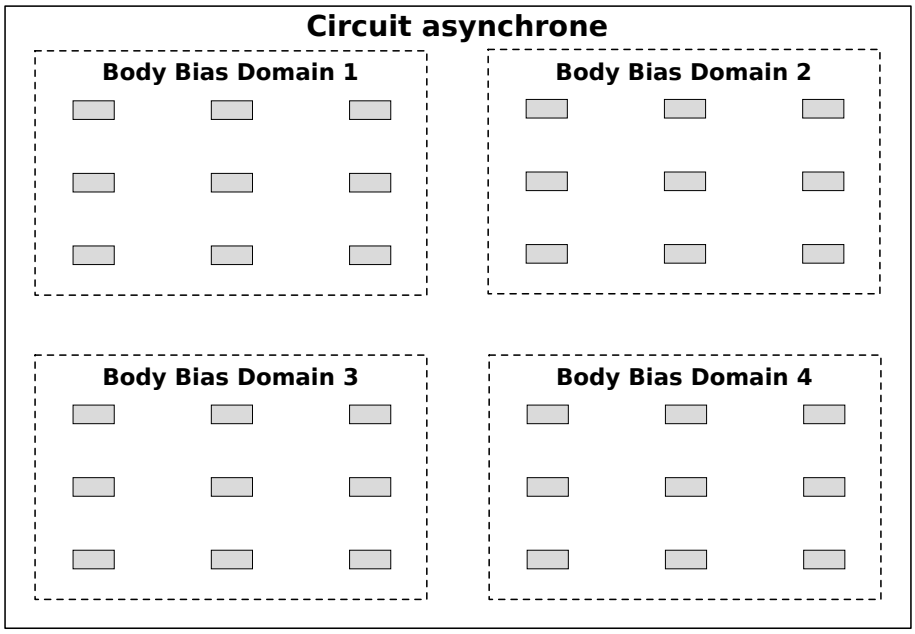


FIGURE 5.10 – *Partitionnement des BBDs lors de l'implémentation physique d'un circuit.*

Une fois la partition des BBDs et la mise en place des BBGs réalisées, l'implémentation de notre méthode de polarisation dynamique du substrat est terminée. Le reste du flot de conception suit les étapes standards de la conception numérique, à une étape près. En effet, nous utilisons des circuits asynchrones qui nécessitent la génération d'arbres locaux d'horloges pour chaque contrôleur asynchrone du circuit. Pour le reste, les contraintes temporelles utilisées rendent transparentes les étapes de conception spécifiques aux circuits asynchrones. Après le routage et les dernières vérifications temporelles et géométriques effectuées, nous obtenons la description finale du circuit.

La méthode développée dans ce travail repose sur un principe simple qui permet une adaptation aisée du flot de conception standard d'un circuit synchrone. Par rapport aux

précédents travaux, l'augmentation de surface est moindre grâce à l'architecture utilisée tout en conservant une robustesse accrue. De plus, l'utilisation de la HLS permet de choisir la division des BBDs au niveau de la description algorithmique du circuit. Le flot de conception décrit ici permet de faciliter au maximum l'implémentation la méthode de polarisation de substrat. Les étapes ajoutées sont automatiques et s'intègrent parfaitement au flot de conception numérique classique.

5.4 Évaluation de la méthode

Pour évaluer les gains de notre méthode de polarisation de substrat dynamique, nous appliquons notre méthode sur une chaîne d'AES conçue par HLS. Ce système est suffisamment complexe pour montrer les avantages de la méthode, mais suffisamment petit pour utiliser les outils de CAO conventionnels avec des licences académiques qui limitent leur efficacité. Les performances du circuit ont été évaluées en technologie FDSOI 28 nm de STMicroelectronics à l'aide de simulations numériques. Afin d'avoir des mesures réelles, nous avons aussi conçu un CNN, lui aussi en FDSOI 28 nm. Ce circuit a été fabriqué mais doit encore être testé.

5.4.1 Simulations numériques

Présentation du système

Le circuit global est un système linéaire d'AES avec une clé de 128 bits, comme le montre la figure 5.11. Il a été synthétisé hiérarchiquement avec Catapult HLS de Siemens EDA. Tous les AES générés possèdent la même architecture.

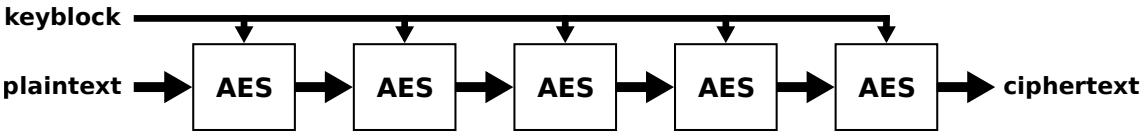


FIGURE 5.11 – *Circuit linéaire avec cinq AES.*

La deuxième étape de notre flot consiste à désynchroniser le circuit. Nous appliquons donc la méthode présentée dans le chapitre 2 sur chaque AES. Comme la granularité des BBDs est définie au niveau du système, nous associons chaque AES à un BBD spécifique. Nous voulons une activation de la polarisation de substrat de l'ordre de la nanoseconde. Nous avons donc attribué 16 BBGs pour chaque BBD.

Pour avoir un aperçu des avantages de notre stratégie, nous comparons notre méthode de polarisation de substrat dynamique à un circuit avec un seul BBD toujours activé. Nous l'appliquons à trois taux d'activité : 0,5, 0,25 et 0,1. Pour chaque scénario, nous envoyons les mêmes données d'entrée sur chaque circuit et nous adaptons la période d'inactivité Δt_{noBB} pour atteindre le taux d'activité souhaité. La tension de polarisation de substrat appliquée est de V_{bb} = 1,1 V.

Le circuit est synthétisé grâce au Design Compiler de Synopsys et la phase de PnR utilise Cadence Innovus. Tous les résultats en vitesse et consommation sont donnés après le

PnR. Les simulations électriques au niveau des transistors permettent d'obtenir une évaluation précise de la consommation électrique. Comme la durée de ce type de simulation est rédhibitoire pour les circuits numériques, nous définissons l'activité de commutation au sein du circuit à partir d'une simulation rétro-annotée au niveau portes logiques. Cette analyse, réalisée avec PrimeTime, n'est qu'une approximation car l'énergie consommée pour l'activation de la polarisation de substrat n'est pas incluse, mais elle donne tout de même une bonne idée de la consommation électrique du circuit.

Étude d'un AES seul

Nous nous intéressons tout d'abord à un seul AES. Nous comparons l'AES asynchrone à son homologue synchrone sans aucune polarisation de substrat. Les circuits sont synthétisés avec une fréquence cible de 1,25 GHz à 0,9 V.

TABLEAU 5.1 – *Impact de la désynchronisation sur l'AES*

	SYNC	DESYNC	Overhead
Surface (μm^2)	10667	10570	-0,9 %
Temps de calcul (ns)	24	24,76	+3,2 %
Puissance moyenne (mW)	21	17	-23 %
Énergie (pJ)	500	420	-20 %

Les résultats sont présentés dans la table 5.1. La dernière colonne affiche le surcoût entre le circuit désynchronisé et le circuit synchrone. Après synthèse, la surface de l'AES désynchronisé diminue de 0,9 % par rapport à la version synchrone. Cette diminution, quoique négligeable, est due à la suppression des blocs de synchronisation dans le chemin de données. L'AES asynchrone a une latence de 24,7 ns, avec une différence d'environ 3 % avec le circuit synchrone. La consommation en puissance montre une amélioration de plus de 20 % après la désynchronisation, ce qui se traduit par une amélioration de 20 % en consommation d'énergie.

Nous remarquons que notre méthode de désynchronisation fournit un circuit asynchrone similaire au circuit original avec une amélioration de la consommation d'énergie, comme montré précédemment dans le chapitre 3.

Résultats du système global

Une comparaison équitable avec les autres stratégies de polarisation n'est pas simple. En effet, les BBGs conventionnels sont plus grands que les nôtres, mais ils sont également utilisés pour polariser des systèmes plus grands avec une tension précise.

Un AES après PnR possède une surface de 12473 μm^2. L'ajout des BBGs augmente la surface de 4,8 %. De plus, chaque BBD doit être isolé du reste du circuit et entraîne une augmentation supplémentaire de 16,4 %. Enfin, un AES avec notre stratégie de polarisation de substrat dynamique atteint une surface de 15215 μm^2. Pour le système entier, cela conduit à une augmentation de la surface globale de 22 %. En ce qui concerne la vitesse du système, il n'y a pas de différence de vitesse par rapport à un circuit toujours polarisé,

seulement des fluctuations négligeables. Un AES avec une tension de polarisation de substrat active calcule les données en 20,2 ns, ce qui représente une amélioration d'environ 18 % par rapport au résultat de l'AES sans polarisation de substrat.

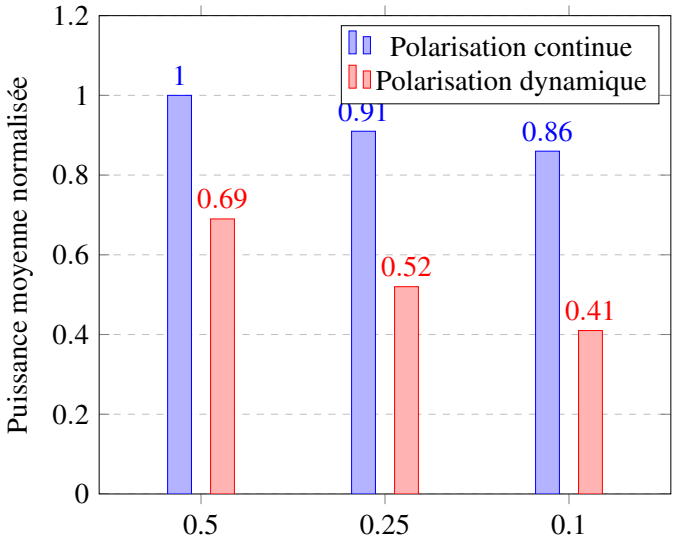


FIGURE 5.12 – *Puissance moyenne normalisée en fonction du taux d'activité.*

La figure 5.12 présente la consommation d'énergie de chaque scénario pour notre système et le circuit de référence polarisé en continu. Nous avons normalisé les résultats en fonction de la consommation d'énergie du circuit de référence dans le scénario où *AR* vaut 0,5.

Nous observons que les deux circuits consomment moins avec la diminution du taux d'activité *AR*, ce qui était attendu puisque la consommation dynamique est liée à l'activité du circuit. Comme prévu, la stratégie de polarisation dynamique réduit la consommation par rapport à la référence. La consommation est divisée de 1,4 à 2,1 en fonction du taux d'activité, qui correspond à des économies d'énergie de 31 % à 52 %. Moins le circuit est actif, plus le gain est important. Ceci est dû à des courants de fuite plus importants dans le circuit polarisé en continu, même pendant la période d'inactivité. Cependant, il faut savoir que ces résultats ne tiennent pas compte de l'énergie consommée pendant l'activation et la désactivation de la polarisation de substrat.

Le gain serait encore plus significatif si la tension de polarisation de substrat était plus élevée. Nous ne pouvons pas montrer ces résultats car les modèles des librairies de cellules standards ne sont caractérisés que pour quelques valeurs de tensions de substrat.

5.4.2 Conception d'un prototype

Maintenant que la fonctionnalité de notre flot a été vérifiée sur un circuit relativement simple, il est souhaitable de concevoir des circuits désynchronisés plus complexes. En effet, la HLS permet de réaliser rapidement des circuits complexes qu'il serait trop long et fastidieux de réaliser manuellement. Une application typique de la HLS est le CNN. Ce circuit, utilisé pour la reconnaissance de chiffres manuscrits, a été évalué en technologie FDSOI 28 nm de STMicroelectronics, puis fabriqué sur silicium. Celui-ci est toujours en cours de test.

Principe d'un CNN

Un CNN permet de résoudre des problèmes de classification. Ce type de réseaux est devenu populaire ces dernières années et est très utilisé dans le domaine de la reconnaissance des images. Ces réseaux sont composés de plusieurs types de couches, dont le nombre et l'agencement diffèrent d'un réseau à l'autre, afin d'obtenir les meilleurs résultats. Les opérations élémentaires sont les suivantes :

— Couche de convolution : lors de cette opération, un filtre balaie sur l'image d'entrée et un calcul de convolution est effectué sur chaque sous-ensemble de l'image. Le nombre de filtres détermine le nombre de canaux d'une couche.

— Couche de *Pooling* : cette couche réalise un sous-échantillonnage des données en entrée pour garder les valeurs les plus significatives. Généralement, les couches les plus utilisées échantillonnent les valeurs maximales ou moyennes, appelés respectivement *max pooling* et *average pooling*. Un exemple de *max pooling* de taille 2x2 est présenté en figure 5.13. Le nombre de données est alors divisé par deux.

— Couche complètement connectée (FC) : dans cette couche, un nombre spécifique de neurones est implémenté où chaque donnée est connectée à chaque neurone de la couche. Un neurone réalise la somme pondérée de ces entrées notées x_j avec des coefficients w, appelés poids. En fin de calcul, un biais θ est ajouté. Pour le i^{ieme} neurone de la couche, sa sortie o_i vaut :

$$o_i = \phi\left(\sum_{j=0}^{n} x_j w_{i,j} + \theta_i\right) \tag{5.8}$$

L'apprentissage du CNN correspond à la définition des valeurs des filtres et des paramètres des couches complètement connectées afin de reconnaître des images ou objets spécifiques. Celui-ci est réalisé de manière supervisée, c'est-à-dire que des images sont données en entrée du réseau et les paramètres sont modifiés en fonction de la réponse attendue.

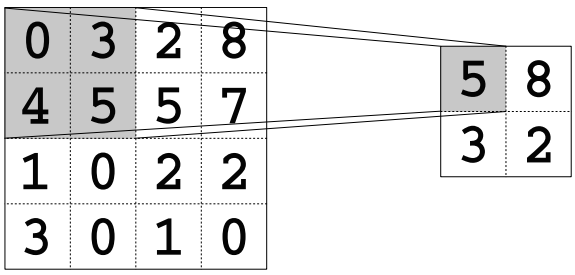


FIGURE 5.13 – *Exemple d'une couche de max pooling.*

Architecture du circuit

Nous travaillerons sur la base de données MNIST [116] afin de faire de la reconnaissance de chiffres manuscrits. Cette base de données contient 70000 images (60000 pour l'apprentissage et le reste pour le test) de taille de 28 par 28 en niveau de gris, avec dix classes correspondant aux chiffres de 0 à 9.

L'architecture CNN choisie, inspirée du Lenet-5 [117], est composée de 5 couches (voir figure 5.14). Nous avons deux couches de convolution C1 et C3, avec des filtres de

taille 5x5 contenant respectivement 6 et 16 canaux, ainsi que deux couches de *Pooling* S2 et S4. La dernière couche du réseau est une couche complètement connectée avec 10 neurones, un pour chaque classe de la base de données.

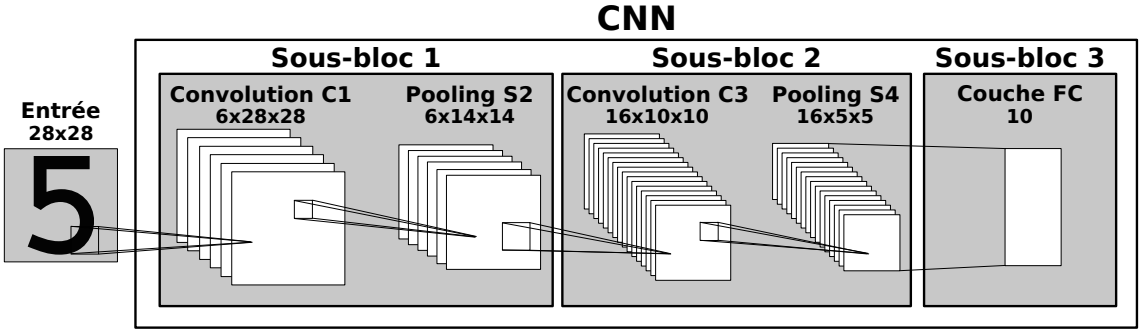


FIGURE 5.14 – *Architecture du CNN.*

Le modèle du CNN et son apprentissage sont réalisés sous Python à l'aide de Keras [118] et ce réseau a ensuite été implémenté en C++ avec Catpult HLS. L'architecture du circuit est présentée figure 5.14. Afin de limiter la taille du circuit, l'encodage des données a été effectué en virgule fixe. De plus, nous utilisons une architecture avec trois blocs hiérarchiques : un premier avec les couches C1 et S2, le deuxième avec les couches C3 et S4, et enfin le dernier avec la couche de sortie. Ces blocs possèdent chacun leur propre FSM et peuvent donc traiter des données indépendamment les uns des autres.

La désynchronisation du CNN est réalisée sur chaque sous-bloc du circuit. Notre flot génère donc un circuit asynchrone avec trois AFSMs ayant chacune leurs propres contraintes temporelles. Ces AFSMs sont bien plus conséquentes que celles des précédents circuits, avec des sous-blocs ayant respectivement un nombre de 16, 16 et 7 états. Les connexions entre les AFSMs ne demandent pas d'attention particulière, car les blocs V_{valid} et V_{ready} s'interfacent naturellement entre eux.

5.5 Conclusion

Dans ce chapitre, nous avons présenté un flot de conception associé à une stratégie simple de polarisation de substrat dynamique. Grâce à un schéma de V_{th}-*hopping* et à la robustesse intrinsèque des circuits asynchrones, nous réalisons une stratégie de polarisation de substrat à grains très fins reposant sur l'activité des données. Selon nos simulations, nous démontrons des économies d'énergie de 31 % à 52 % pour une faible activité de données par rapport aux circuits polarisés de manière constante. Il est à noter qu'une forte activité du circuit impliquerait des gains négligeables pour un surcoût important en surface, car les caissons seraient très souvent activés.

De plus, ce flot de conception repose sur la facilité de conception des circuits asynchrones grâce à des outils HLS commerciaux et le travail de désynchronisation développé dans cette thèse au chapitre 2. Les principales étapes de conception sont les suivantes. Tout d'abord, un circuit à données groupées est obtenu à partir d'une description de haut niveau en C ou C++, qui est compilée par Catapult HLS de Siemens EDA. Cette description permet aussi de définir le partitionnement du circuit pour la polarisation de substrat. Ensuite,

la partie contrôle issue de la HLS est désynchronisée et nous extrayons les contraintes de conception (principalement les contraintes temporelles). Enfin, les BBGs, qui ont été intégrés dans un format de cellule standard, sont insérés dans le plan du circuit durant les opérations PnR.

Les simulations présentées ne sont que des estimations au niveau portes logiques. De futurs travaux viseront des simulations au niveau des transistors pour une meilleure précision. Une puce de test en FDSOI 28 nm de STMicroelectronics a également été conçue en suivant ce flot de conception, néanmoins il reste à la tester pour valider notre flot de conception et nos résultats de simulation. Nous prévoyons également d'étudier la réduction de puissance obtenue avec une tension d'alimentation plus faible, tout en compensant la baisse de performance grâce à la polarisation du substrat. Les simulations nécessitant des modèles dédiés pour chaque tension d'alimentation, le mieux est d'obtenir directement ces résultats à partir des mesures d'un prototype.

Si notre méthode utilise des transistors LVT et polarise les BBDs lors d'une activité pour augmenter les performances du circuit, il est toutefois possible d'utiliser l'approche inverse. Un circuit composé de transistors RVT rend possible l'usage d'une polarisation de substrat de type RBB, qui augmente la tension de seuil et réduit donc les courants de fuite. La polarisation de substrat serait alors appliquée lorsqu'aucune activité n'est détectée dans les BBDs. Cette autre approche serait facilement implémentable au sein de notre flot de conception et pourrait minimiser la consommation statique dans un circuit.

Conclusion

La consommation d'énergie est devenue une des préoccupations majeures de ces dernières années. En parallèle, les contraintes sur les circuits synchrones sont de plus en plus fortes et empêchent des gains significatifs sur les noeuds technologiques avancés. Il est donc nécessaire de rechercher des architectures alternatives afin de maximiser la réduction de consommation d'énergie dans les circuits intégrés. Dans ce cadre, les circuits asynchrones sont prometteurs. Libérés de la contrainte d'un signal d'horloge global, ces circuits suppriment une bonne partie de la consommation superflue liée à ce signal. De plus, la synchronisation locale des composants internes apporte une robustesse supplémentaire aux variations PVT. Toutefois, ces circuits restent encore complexes à concevoir pour la majorité des ingénieurs dans l'industrie. En effet, l'asynchrone repose généralement sur des outils de CAO et des langages non-conventionnels. Cet environnement de conception différent des outils traditionnels de l'industrie demande un investissement supplémentaire en connaissance, en savoir-faire et donc en temps et en argent pour quiconque souhaiterait entrer dans le domaine asynchrone.

Pour amoindrir ces difficultés, nous avons mis en place un flot de conception haut niveau permettant de concevoir des circuits asynchrones, qui sont couplés à une gestion dynamique de la polarisation du substrat. Ce travail facilite grandement la génération des circuits asynchrones et est entièrement compatible avec les outils de CAO commerciaux. Les circuits asynchrones obtenus par ce flot sont similaires à leurs équivalents synchrones tout en étant plus robustes. La polarisation dynamique du substrat permet d'augmenter l'efficacité énergétique des circuits. Les étapes d'implémentation physique, nécessitant des règles spécifiques pour une polarisation distribuée et dynamique du substrat, sont très largement automatisées. Les travaux issus de cette thèse ont été publiés dans plusieurs conférences internationales.

Contributions

Notre flot permet de générer des circuits asynchrones automatiquement à partir d'une description algorithmique. Pour être le plus simple possible, nous utilisons les logiciels de CAO synchrones et désynchronisons les circuits obtenus. Ceci est fait simplement en remplaçant le chemin de contrôle, qui n'est rien de plus qu'une FSM, en une AFSM. La méthode, appliquée à l'outil de HLS industriel Catapult HLS, a été formalisée dans le chapitre 2 afin de démontrer la correction de la méthode. La méthode a été intégrée dans un flot de conception permettant de réaliser toutes les étapes de conception facilement, de la synthèse au PnR, avec des outils de CAO commerciaux. La gestion des contraintes temporelles locales, point central du flot, est définie à haut niveau et repose sur l'utilisation d'horloge. Les circuits asynchrones obtenus sont donc corrects par construction et possèdent des performances similaires aux circuits d'origine. L'étude menée dans le chapitre 3 démontre qu'en plus de posséder une surface et une vitesse analogues, la consommation est légèrement améliorée. Ces résultats sont une grande amélioration par rapport à

l'état de l'art des méthodes de désynchronisation. L'architecture développée dans la méthode a aussi été validée par la conception d'un circuit de test. Notre flot de conception de circuits asynchrones est donc complètement validé, et les caractéristiques du circuit mettent en lumière la robustesse des circuits à données groupées que nous utilisons.

Une fois les circuits asynchrones générés, il est possible d'utiliser des méthodes nouvelles de gestion de la consommation. En technologie FDSOI, la polarisation de substrat ouvre de nouvelles possibilités pour changer dynamiquement la tension de seuil des transistors. Nous avons donc mis en place une méthode polarisant dynamiquement et localement des zones restreintes du circuit en fonction de l'activité dans le circuit. Lorsqu'une donnée est détectée dans un BBD local à l'aide des signaux de communication asynchrone, la polarisation de substrat est activée pour augmenter la vitesse du circuit. À l'inverse, la polarisation de substrat est éteinte à la fin du calcul pour limiter la consommation d'énergie. Ces changements peuvent être réalisés aisément grâce à la robustesse des circuits asynchrones. Cette stratégie, décrit dans le chapitre 5, permet de grandement simplifier les unités de contrôle nécessaire à la polarisation. Seul un détecteur d'activité est présent pour contrôler la génération de la tension de substrat par les BBGs. Ces derniers, conçus sous la forme de translateur de tension, sont distribués localement à l'intérieur des BBDs et sont bien plus simples que dans les BBGs usuellement employés. Afin de faciliter les opérations de PnR, ces composants ont été conçus comme des cellules standards afin d'être intégrés au flot de conception numérique. De plus, la technologie FDSOI nécessite des règles de dessins spécifiques pour la polarisation de substrat. Par conséquent, pour rendre accessible notre méthode, la génération des BBDs et l'insertion des BBGs ont été automatisées. Les résultats en simulation de notre méthode montrent des résultats prometteurs avec une diminution importante de l'énergie consommée comparée à des circuits polarisés en permanence, surtout si l'activité du circuit est faible. Un circuit de test a aussi été conçu en FDSOI 28 nm pour réaliser des mesures réelles. Il reste encore à le tester.

Perspectives

Il est possible de diviser les perspectives en deux grandes catégories, une pour les circuits asynchrones et l'autre pour la méthode de polarisation du substrat en FDSOI. La méthode de désynchronisation présentée dans ce travail a été prouvée manuellement, mais pour être le plus rigoureux possible, il est nécessaire de réaliser la preuve formelle à l'aide d'un prouveur de théorème. Ce travail a déjà débuté avec le logiciel PVS mais n'a pas encore été finalisé. Si notre modèle permet de démontrer notre méthode, une voie intéressante serait d'étudier la généralisation de l'approche de désynchronisation à tout type de FSMs. En effet, nous sommes aujourd'hui limités dans les FSMs que nous pouvons désynchroniser. Si ces restrictions ne posent pas de problèmes pour les circuits ciblés dans cette thèse, les prochaines étapes de ce travail nécessitent de les dépasser. Par exemple, une FSM avec un nombre quelconque d'états communicant avec son environnement pourrait permettre de désynchroniser des circuits bien plus complexes. De plus, notre AFSM fonctionne avec un protocole de communication spécifique. L'idée serait de trouver également l'ensemble des protocoles fonctionnels respectant l'équivalence entre la version synchrone et son homologue asynchrone.

Le flot de conception associé à notre méthode est complètement intégré à l'environnement des outils de CAO industriels fait pour les circuits synchrones. Ceci est largement dû à la méthode LCS, qui permet d'utiliser les analyses temporelles synchrones sur des

chemins locaux. Néanmoins, deux points importants méritent notre attention pour améliorer cette méthodologie de conception. Premièrement, l'extraction des chemins des LCS est réalisée à partir du modèle des contrôleurs asynchrones mais rien ne permet de s'assurer de la bonne correspondance entre les deux. La démonstration de l'équivalence entre les LCS et le modèle est donc nécessaire, pour éviter tout oubli ou faux chemin. Ensuite, le nombre de contraintes temporelles peut être très important dans le cas d'un contrôleur asynchrone complexe, avec notamment des contraintes internes aux contrôleurs. Ce grand nombre de contraintes peut s'avérer fastidieux à mettre en place et lourd à gérer pour les outils d'analyses temporelles. C'est pourquoi une approche hiérarchique du circuit de contrôle peut permettre une vérification automatique avec l'intégration des contraintes internes directement dans les librairies temporelles des cellules. Afin d'avoir un flot de conception totalement industrialisable, il est essentiel d'ajouter des solutions de *Design For Test* (DFT). Dans ce cadre, la plupart des solutions existantes restent non conventionnelles. Un travail récent de Guazzelli *et al.* [119] propose une méthode pour insérer une DFT complète à l'aide des outils de CAO standards. Il resterait à adapter cette solution à notre architecture spécifique.

La méthode de polarisation dynamique et locale du substrat en FDSOI a montré des résultats prometteurs en simulations. Il reste désormais à effectuer des mesures réelles, sur notre circuit de test notamment, afin d'évaluer les gains réels prenant en compte la capacité du substrat. Une fois la méthode validée sur silicium, nous nous concentrerons sur la réduction drastique de la consommation d'énergie : la réduction de la tension d'alimentation du circuit et l'utilisation de la polarisation de substrat pour éviter une dégradation des performances. Les gains en énergie devraient alors être substantiels. L'approche complémentaire à notre proposition pour la polarisation du substrat mériterait d'être explorée également pour optimiser la consommation d'énergie. En effet, nous pouvons très simplement implémenter une stratégie polarisant le substrat en l'absence de données pour diminuer les courants de fuite. Cette méthode, qui cible des applications très différentes de celles étudiées dans cette thèse, permettra néanmoins d'optimiser l'énergie du circuit pour des applications où l'activité est faible et les temps de veille prolongés.

Plus largement, ce flot de conception permet de concevoir simplement des circuits asynchrones à partir d'une description de haut niveau. Par conséquent, de futurs travaux pourraient aussi se concentrer sur le développement de nouveaux d'algorithmes adaptés à des circuits événementiels. De ce point de vue, l'échantillonnage non-uniforme [120] ou les réseaux de neurones à impulsions [121] semblent des applications particulièrement intéressantes étant donné qu'elles traitent des données parcimonieuses. Ces circuits peuvent d'ailleurs servir de blocs élémentaires pour des méthodes existantes plus complexes, comme cela a montré dans cette thèse. Il est alors possible de simplifier grandement les méthodes de conception tout en ayant des gains de consommation importants. Par exemple, les méthodes de DVS gagneraient à utiliser les circuits asynchrones pour simplifier la gestion du changement de tension d'alimentation.

Bibliographie

[1] J. Shalf, "The future of computing beyond Moore's Law," *Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2166, p. 20190061, Mar. 2020.

[2] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth : Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[3] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi : A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[4] E. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.

[5] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, Feb. 2002.

[6] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous CAD tools," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 107–117, Jul. 2002.

[7] A. Yakovlev, P. Vivet, and M. Renaudin, "Advances in Asynchronous Logic : From Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD Tools," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2013, pp. 1715–1724.

[8] S. M. Nowick and M. Singh, "Asynchronous Design—Part 1 : Overview and Recent Advances," *IEEE Design & Test*, vol. 32, no. 3, pp. 5–18, Jun. 2015.

[9] C. Molnar, T.-P. Fang, and F. Rosenberger, "Synthesis of delay-insensitive modules," in *Proc. 1985 Chapel Hill Conf. VLSI*, May 1985, pp. 67–86.

[10] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," in *Beauty Is Our Business*, W. H. J. Feijen, A. J. M. Gasteren, D. Gries, and J. Misra, Eds., New York, NY, 1990, pp. 302–311.

[11] K. Fant and S. Brandt, "NULL Convention Logic/sup TM/ : a complete and consistent logic for asynchronous digital circuit synthesis," in *Proceedings of In-*

ternational Conference on Application Specific Systems, Architectures and Processors : ASAP '96, Chicago, IL, USA, 1996, pp. 261–273.

[12] R. D. Jorgenson, L. Sorensen, D. Leet, M. S. Hagedorn, D. R. Lamb, T. H. Friddell, and W. P. Snapp, "Ultralow-Power Operation in Subthreshold Regimes Applying Clockless Logic," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 299–314, Feb. 2010.

[13] D. E. Muller, "A theory of asynchronous circuits," University of Illinois, Tech. Rep. 75, 1956.

[14] D. Huffman, "The synthesis of sequential switching circuits," *Journal of the Franklin Institute*, vol. 257, no. 3, pp. 161–190, Mar. 1954.

[15] S. Nowick and D. Dill, "Synthesis of asynchronous state machines using a local clock," in *ICCD 1991*, Cambridge, MA, USA, 1991, pp. 192–197.

[16] K. Yun and D. Dill, "Automatic synthesis of extended burst-mode circuits. I. (Specification and hazard-free implementations)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 101–117, Feb. 1999.

[17] ——, "Automatic synthesis of extended burst-mode circuits. II. (Automatic synthesis)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 118–132, Feb. 1999.

[18] R. M. Fuhrer, S. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana, "MINIMALIST : An Environment for the Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines," Department of Computer Science, Columbia University, Technical report CUCS-020-99, Apr. 2011.

[19] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[20] J. Sparsø and S. Furber, Eds., *Principles of Asynchronous Circuit Design*. Boston, MA : Springer US, 2001.

[21] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, "An asynchronous low-power 80C51 microcontroller," in *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, San Deigo, CA, USA, 1998, pp. 96–107.

[22] K.-L. Chang, J. S. Chang, B.-H. Gwee, and K.-S. Chong, "Synchronous-Logic and Asynchronous-Logic 8051 Microcontroller Cores for Realizing the Internet of Things : A Comparative Study on Dynamic Voltage Scaling and Variation Effects," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, no. 1, pp. 23–34, Mar. 2013.

[23] C.-M. Hung and K. Muhammad, "RF/analog and digital faceoff — friends or enemies in an RF SoC," in *Proceedings of 2010 International Symposium on VLSI Technology, System and Application*, Hsin Chu, Taiwan, 2010, pp. 19–20.

[24] G. Bouesse, N. Ninon, G. Sicard, M. Renaudin, A. Boyer, and E. Sicard, "Asynchronous logic VS Synchronouos logic : Concrete results on electromagnetic emissions and conducted susceptibility," in *6th International workshop on electromagnetic compatibility of integrated circuits (EMC Compo'07)*, Torino, Italy, Nov. 2007, pp. 99–102.

[25] S. Germain, S. Engels, and L. Fesquet, “High Level Current Modeling for Shaping Electromagnetic Emissions in Micropipeline Circuits,” *Journal of Low Power Electronics and Applications*, vol. 9, no. 1, p. 6, Jan. 2019.

[26] J. Cortadella, L. Lavagno, P. Lopez, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, “Reactive clocks with variability-tracking jitter,” in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, New York City, NY, USA, Oct. 2015, pp. 511–518.

[27] T. Murata, “Petri nets : Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[28] T.-A. Chu, “On the models for designing VLSI asynchronous digital systems,” *Integration*, vol. 4, no. 2, pp. 99–113, Jun. 1986.

[29] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “Petrify : a tool for manipulating concurrent specifications and synthesis of asynchronous controllers,” *IEICE Transactions on information and Systems*, vol. 80, no. 3, pp. 315–325, 1997.

[30] A. Yakovlev, A. Koelmans, A. Semenov, and D. Kinniment, “Modelling, analysis and synthesis of asynchronous control circuits using Petri nets,” *Integration*, vol. 21, no. 3, pp. 143–170, Dec. 1996.

[31] I. Sutherland and S. Fairbanks, “GasP : a minimal FIFO control,” in *Seventh International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Salt Lake City, UT, USA, 2001, pp. 46–53.

[32] M. Singh and S. Nowick, “MOUSETRAP : ultra-high-speed transition-signaling asynchronous pipelines,” in *ICCD 2001*, Austin, TX, USA, 2001, pp. 9–17.

[33] A. Peeters, F. t. Beest, M. d. Wit, and W. Mallon, “Click Elements : An Implementation Style for Data-Driven Compilation,” in *IEEE Symposium on Asynchronous Circuits and Systems (ASYNC)*, Grenoble, France, 2010, pp. 3–14.

[34] A. Peeters and K. van Berkel, “Single-rail handshake circuits,” in *Proceedings Second Working Conference on Asynchronous Design Methodologies*, London, UK, 1995, pp. 53–62.

[35] S. Furber and P. Day, “Four-phase micropipeline latch control circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 2, pp. 247–253, Jun. 1996.

[36] G. Birtwistle and K. S. Stevens, “The Family of 4-phase Latch Protocols,” in *14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Newcastle upon Tyne, United Kingdom, Apr. 2008, pp. 71–82.

[37] P. B. McGee and S. M. Nowick, “A lattice-based framework for the classification and design of asynchronous pipelines,” in *Proceedings of the 42nd annual conference on Design automation - DAC ’05*, San Diego, California, USA, 2005, p. 491.

[38] K. Yun, P. Beerel, and J. Arceo, “High-performance asynchronous pipeline circuits,” in *Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Fukushima, Japan, 1996, pp. 17–28.

[39] C. Mannakkara and T. Yoneda, “Asynchronous pipeline controller based on early acknowledgement protocol,” in *8th International Conference on Application of Concurrency to System Design*, Xian, China, 2008, pp. 118–127.

[40] J. Simatic, A. Cherkaoui, R. P. Bastos, and L. Fesquet, "New asynchronous protocols for enhancing area and throughput in bundled-data pipelines," in *29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Belo Horizonte, Brazil, Aug. 2016, pp. 1–6.

[41] D. Edwards and A. Bardsley, "Balsa : An Asynchronous Hardware Synthesis Language," *The Computer Journal*, vol. 45, no. 1, pp. 12–18, Jan. 2002.

[42] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits," in *Proceedings of the European Conference on Design Automation.*, Amsterdam, Netherlands, 1991, pp. 384–389.

[43] M. Renaudin, P. Vivet, and F. Robin, "A design framework for asynchronous/synchronous circuits based on CHP to HDL translation," in *Proceedings. Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Barcelona, Spain, 1999, pp. 135–144.

[44] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, pp. 226–234, Dec. 1986.

[45] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev, "Asynchronous design using commercial HDL synthesis tools," in *International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)*, Eilat, Israel, 2000, pp. 114–125.

[46] M. L. L. Sartori, R. N. Wuerdig, M. T. Moreira, and N. L. V. Calazans, "Pulsar : Constraining QDI Circuits Cycle Time Using Traditional EDA Tools," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Hirosaki, Japan, May 2019, pp. 114–123.

[47] M. L. L. Sartori, M. T. Moreira, and N. L. V. Calazans, "A Frontend using Traditional EDA Tools for the Pulsar QDI Design Flow," in *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Salt Lake City, UT, USA, May 2020, pp. 3–10.

[48] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization : Synthesis of Asynchronous Circuits From Synchronous Specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.

[49] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *10th International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Crete, Greece, 2004, pp. 149–158.

[50] J. Paykin, B. Huffman, D. M. Zimmerman, and P. A. Beerel, "Formal Verification of Flow Equivalence in Desynchronized Designs," in *26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Salt Lake City, UT, USA, May 2020, pp. 54–62.

[51] F. Bertrand, A. Cherkaoui, J. Simatic, A. Maure, and L. Fesquet, "CAR : On the highway towards de-synchronization," in *24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Batumi, Dec. 2017, pp. 339–343.

[52] J. Xu and H. Wang, "Desynchronize a legacy floating-point adder with operand-dependant delay elements," in *IEEE International Symposium on Circuits and Systems (ISCAS 2011)*, Rio de Janeiro, Brazil, May 2011, pp. 1427–1430.

[53] H. Wu, W. Chen, Z. Su, S. Wei, A. He, and H. Chen, "A method to transform synchronous pipeline circuits to bundled-data asynchronous circuits using commercial EDA tools," in *IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, Xi'an, China, Jun. 2019, pp. 1–2.

[54] N. Srivastava and R. Manohar, "Operation-Dependent Frequency Scaling Using Desynchronization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 4, pp. 799–809, Apr. 2019.

[55] Y. Zhang, H. Cheng, D. Chen, H. Fu, S. Agarwal, M. Lin, and B. Peter, "Challenges in Building an Open-Source Flow from RTL to Bundled-Data Design," in *24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Vienna, May 2018, pp. 26–27.

[56] L. A. Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE Transactions on Computers*, vol. C-31, no. 12, pp. 1133–1141, Dec. 1982.

[57] C. Sotiriou, "Direct-mapped asynchronous finite-state machines in CMOS technology," in *Proceedings 14th Annual IEEE International ASIC/SOC Conference (IEEE Cat. No.01TH8558)*, Arlington, VA, USA, 2001, pp. 105–109.

[58] D. L. Oliveira, G. C. Duarte, G. C. Batista, and N. N. M. Cardoso, "Converting Synchronous Digital Systems to Asynchronous Systems using Local-Clock," in *2020 IEEE XXVII International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, Lima, Peru, Sep. 2020, pp. 1–4.

[59] M. Iizuka, N. Hamada, H. Saito, R. Yamaguchi, and M. Yoshinaga, "A tool set for the design of asynchronous circuits with bundled-data implementation," in *IEEE 29th International Conference on Computer Design (ICCD 2011)*, Amherst, MA, USA, Oct. 2011, pp. 78–83.

[60] F. Rosenberger, C. Molnar, T. Chaney, and T.-P. Fang, "Q-modules : internally clocked delay-insensitive modules," *IEEE Transactions on Computers*, vol. 37, no. 9, pp. 1005–1018, Sep. 1988.

[61] J. Simatic, R. P. Bastos, and L. Fesquet, "High-level synthesis for event-based systems," in *Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, Krakow, Poland, Jun. 2016, pp. 1–7.

[62] A. Prost-Boucle, O. Muller, and F. Rousseau, "Fast and standalone Design Space Exploration for High-Level Synthesis under resource constraints," *Journal of Systems Architecture*, vol. 60, no. 1, pp. 79–93, Jan. 2014.

[63] K. Stevens, R. Ginosar, and S. Rotem, "Relative timing," in *Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Barcelona, Spain, 1999, pp. 208–218.

[64] W. Hua and R. Manohar, "Exact Timing Analysis for Asynchronous Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 203–216, Jan. 2018.

[65] N. Xiromeritis, S. Simoglou, C. Sotiriou, and N. Sketopoulos, "Graph-Based STA for Asynchronous Controllers," in *29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Rhodes, Greece, Jul. 2019, pp. 9–16.

[66] W. Hua, Y.-S. Lu, K. Pingali, and R. Manohar, "Cyclone : A Static Timing and Power Engine for Asynchronous Circuits," in *26th IEEE International Symposium*

on Asynchronous Circuits and Systems (ASYNC), Salt Lake City, UT, USA, May 2020, pp. 11–19.

[67] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of Asynchronous Templates for Integration into Clocked CAD Flows," in *15th IEEE Symposium on Asynchronous Circuits and Systems (ASYNC)*, Chapel Hill, NC, USA, May 2009, pp. 151–161.

[68] N. Andrikos, L. Lavagno, D. Pandini, and C. P. Sotiriou, "A Fully-Automated Desynchronization Flow for Synchronous Circuits," in *44th ACM/IEEE Design Automation Conference (DAC)*, San Diego, CA, USA, Jun. 2007.

[69] M. Gibiluka, M. T. Moreira, and N. L. Vilar Calazans, "A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework," in *2015 Euromicro Conference on Digital System Design*, Funchal, Aug. 2015, pp. 79–86.

[70] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static Timing Analysis of Asynchronous Bundled-Data Circuits," in *24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Vienna, May 2018, pp. 110–118.

[71] G. Gimenez, J. Simatic, and L. Fesquet, "From Signal Transition Graphs to Timing Closure : Application to Bundled-Data Circuits," in *25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Hirosaki, Japan, May 2019, pp. 86–95.

[72] P. Coussy, D. Gajski, M. Meredith, and A. Takach, "An Introduction to High-Level Synthesis," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 8–17, Jul. 2009.

[73] B. Khailany, E. Krimer, R. Venkatesan, J. Clemons, J. S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y. S. Shao, S. Srinath, C. Torng, S. L. Xi, Y. Zhang, and B. Zimmer, "INVITED : A Modular Digital VLSI Flow for High-Productivity SoC Design," in *55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, San Francisco, CA, Jun. 2018, pp. 1–6.

[74] "Catapult HLS."

[75] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp : high-level synthesis for FPGA-based processor/accelerator systems," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '11*, Monterey, CA, USA, 2011, p. 33.

[76] P. Coussy and A. Morawiec, Eds., *High-level synthesis : from algorithm to digital circuit*. New York : Springer, 2008, oCLC : ocn227032779.

[77] J. Sparso, "Current trends in high-level synthesis of asynchronous circuits," in *16th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2009)*, Yasmine Hammamet, Dec. 2009, pp. 347–350.

[78] C. G. Wong and A. J. Martin, "High-level synthesis of asynchronous systems by data-driven decomposition," in *Proceedings of the 40th conference on Design automation - DAC '03*, Anaheim, CA, USA, 2003, p. 508.

[79] G. Venkataramani, M. Budiu, T. Chelcea, and S. Copen Goldstein, "C to asynchronous dataflow circuits : An end-to-end toolflow," in *IWLS 2004*, 2004.

[80] C. S. Ananian, "The static single information form," PhD Thesis, Massachusetts Institute of Technology, 1999.

[81] R. Li, L. Berkley, Y. Yang, and R. Manohar, “Fluid : An Asynchronous High-level Synthesis Tool for Complex Program Structures,” in *27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Beijing, China, Sep. 2021, pp. 1–8.

[82] S. F. Nielsen, J. Sparso, and J. Madsen, “Behavioral Synthesis of Asynchronous Circuits Using Syntax Directed Translation as Backend,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 248–261, Feb. 2009.

[83] S. F. Nielsen, J. Sparsø, J. B. Jensen, and J. S. R. Nielsen, “A Behavioral Synthesis Frontend to the Haste/TiDE Design Flow,” in *15th IEEE Symposium on Asynchronous Circuits and Systems (ASYNC)*, Chapel Hill, NC, USA, May 2009, pp. 185–194.

[84] J. Hansen and M. Singh, “Concurrency-Enhancing Transformations for Asynchronous Behavioral Specifications : A Data-Driven Approach,” in *14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Newcastle upon Tyne, United Kingdom, Apr. 2008, pp. 15–25.

[85] M. Tranchero, L. M. Reyneri, A. Bink, and M. d. Wit, “An Automatic Approach to Generate Haste Code from Simulink Specifications,” in *15th IEEE Symposium on Asynchronous Circuits and Systems (ASYNC)*, Chapel Hill, NC, USA, May 2009, pp. 175–184.

[86] R. Badia and J. Cortadella, “High-level synthesis of asynchronous systems : Scheduling and process synchronization,” in *EDAC 1993*, Paris, France, 1993, pp. 70–74.

[87] R. van Leuken, T. van Leeuwen, and H. L. Arriens, “High Level Synthesis of Asynchronous Circuits from Data Flow Graphs,” in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, J. L. Ayala, B. García-Cámara, M. Prieto, M. Ruggiero, and G. Sicard, Eds., Berlin, Heidelberg, 2011, vol. 6951, pp. 317–330, series Title : Lecture Notes in Computer Science.

[88] Naohiro Hamada, Yuuki Shiga, Hiroshi Saito, Tomohiro Yoneda, C. Myers, and Takashi Nanya, “A behavioral synthesis method for asynchronous circuits with bundled-data implementation (Tool paper),” in *2008 8th International Conference on Application of Concurrency to System Design*, Xian, China, 2008, pp. 50–55.

[89] J. Hansen and M. Singh, “A Fast Branch-and-Bound Approach to High-Level Synthesis of Asynchronous Systems,” in *IEEE Symposium on Asynchronous Circuits and Systems*, Grenoble, France, 2010, pp. 107–116.

[90] K. Garcia, D. L. Oliveira, R. d’Amore, L. A. Faria, and J. L. V. Oliveira, “FPGA implementation of optimized XBM specifications by transformation for AFSMs,” in *ReConFig 2016*, Cancun, Mexico, Nov. 2016, pp. 1–6.

[91] K. Alsayeg, K. Morin-Allory, and L. Fesquet, “RAT-based formal verification of QDI asynchronous controllers,” in *FDL 2009*, Sophia Antipolis, France, Sep. 2009.

[92] National Institute of Standards and Technology, “Advanced encryption standard (AES),” National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST FIPS 197, Nov. 2001.

[93] C. Hu, “New sub-20nm transistors : why and how,” in *Proceedings of the 48th Design Automation Conference on - DAC ’11*, San Diego, California, 2011, p. 460.

[94] B. Pelloux-Prayer, M. Blagojevic, O. Thomas, A. Amara, A. Vladimirescu, B. Nikolic, G. Cesana, and P. Flatresse, "Planar fully depleted SOI technology : The convergence of high performance and low power towards multimedia mobile applications," in *2012 IEEE Faible Tension Faible Consommation*, Paris, France, Jun. 2012, pp. 1–4.

[95] J.-P. Colinge, Ed., *FinFETs and other multi-gate transistors*, ser. Series on integrated circuits and systems. New York : Springer, 2008, oCLC : ocn171549428.

[96] Bin Yu, Leland Chang, S. Ahmed, Haihong Wang, S. Bell, Chih-Yuh Yang, C. Tabery, Chau Ho, Qi Xiang, Tsu-Jae King, J. Bokor, Chenming Hu, Ming-Ren Lin, and D. Kyser, "FinFET scaling to 10 nm gate length," in *Digest. International Electron Devices Meeting,*, San Francisco, CA, USA, 2002, pp. 251–254.

[97] R. Taco, I. Levi, A. Fish, and M. Lanuzza, "Exploring back biasing opportunities in 28nm UTBB FD-SOI technology for subthreshold digital design," in *IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, Eilat, Israel, Dec. 2014, pp. 1–4.

[98] H. Jones, "Why migration to 20nm bulk CMOS and 16/14nm FinFETs is not best approach for the semiconductor industry," *International Business Strategies, Los Gatos, CA, Tech. Rep*, 2014.

[99] E. Beigne, J.-F. Christmann, A. Valentian, O. Billoint, E. Amat, and D. Morche, "UTBB FDSOI technology flexibility for ultra low power internet-of-things applications," in *2015 45th European Solid State Device Research Conference (ESSDERC)*, Graz, Austria, Sep. 2015, pp. 164–167.

[100] T. Sakurai, A. Matsuzawa, and T. Douseki, *Fully-depleted SOI CMOS circuits and technology for ultralow-power applications*. Dordrecht, the Netherlands : Springer, 2006.

[101] E. Beigne, A. Valentian, B. Giraud, O. Thomas, T. Benoist, Y. Thonnart, S. Bernard, G. Moritz, O. Billoint, Y. Maneglia, P. Flatresse, J. Noel, F. Abouzeid, B. Pelloux-Prayer, A. Grover, S. Clerc, P. Roche, J. Le Coz, S. Engels, and R. Wilson, "Ultra-Wide Voltage Range Designs in Fully-Depleted Silicon-On-Insulator FETs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, Grenoble, France, 2013, pp. 613–618.

[102] J. M. Kühn, D. Peterson, H. Amano, O. Bringmann, and W. Rosenstiel, "Spatial and Temporal Granularity Limits of Body Biasing in UTBB-FDSOI," in *DATE 2015*, Grenoble, France, 2015, pp. 876–879.

[103] J. M. Kühn, H. Amano, O. Bringmann, and W. Rosenstiel, "Leveraging FDSOI through body bias domain partitioning and bias search," in *The 53rd Annual Design Automation Conference (DAC '16)*, Austin Texas, Jun. 2016, pp. 1–6.

[104] B. Pelloux-Prayer, A. Valentian, B. Giraud, Y. Thonnart, J.-P. Noel, P. Flatresse, and E. Beigne, "Fine grain multi-VT co-integration methodology in UTBB FD-SOI technology," in *IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, Istanbul, Turkey, Oct. 2013, pp. 168–173.

[105] H. Fatemi, A. B. Kahng, H. Lee, and J. Pineda de Gyvez, "Heuristic Methods for Fine-Grain Exploitation of FDSOI," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2860–2871, Oct. 2020.

[106] S. Kulkarni, D. Sylvester, and D. Blaauw, "A Statistical Framework for Post-Silicon Tuning through Body Bias Clustering," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Double Tree Hotel, San Jose, CA,USA, Nov. 2006, pp. 39–46, iSSN : 1092-3152.

[107] H. Xu, W.-B. Jone, and R. Vemuri, "Novel Vth Hopping Techniques for Aggressive Runtime Leakage Control," in *23rd International Conference on VLSI Design : concurrently with the 9th International Conference on Embedded Systems Design (VLSID)*, Bangalore, India, Jan. 2010, pp. 51–56.

[108] D. Jacquet, F. Hasbani, P. Flatresse, R. Wilson, F. Arnaud, G. Cesana, T. Di Gilio, C. Lecocq, T. Roy, A. Chhabra, C. Grover, O. Minez, J. Uginet, G. Durieu, C. Adobati, D. Casalotto, F. Nyer, P. Menut, A. Cathelin, I. Vongsavady, and P. Magarshack, "A 3 GHz Dual Core Processor ARM Cortex TM -A9 in 28 nm UTBB FD-SOI CMOS With Ultra-Wide Voltage Range and Energy Efficiency Optimization," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 4, pp. 812–826, Apr. 2014.

[109] N. Kamae, A. K. M. M. Islam, A. Tsuchiya, and H. Onodera, "A body bias generator with wide supply-range down to threshold voltage for within-die variability compensation," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, KaoHsiung, Taiwan, Nov. 2014, pp. 53–56.

[110] M. Blagojevic, M. Cochet, B. Keller, P. Flatresse, A. Vladimirescu, and B. Nikolic, "A fast, flexible, positive and negative adaptive body-bias generator in 28nm FD-SOI," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, Honolulu, HI, USA, Jun. 2016, pp. 1–2.

[111] G. Palumbo and D. Pappalardo, "Charge Pump Circuits : An Overview on Design Strategies and Topologies," *IEEE Circuits and Systems Magazine*, vol. 10, no. 1, pp. 31–45, 2010.

[112] J. Hamon and E. Beigne, "Automatic Leakage Control for Wide Range Performance QDI Asynchronous Circuits in FD-SOI Technology," in *19th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, Santa Monica, CA, USA, May 2013, pp. 142–149.

[113] L. Fesquet, Y. Decoudu, A. R. I. Jadue, T. F. de Paiva Leite, O. Rolloff, M. Diallo, R. P. Bastos, K. Morin-Allory, and S. Engels, "A Distributed Body-Biasing Strategy for Asynchronous Circuits," in *IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, Cuzco, Peru, Oct. 2019, pp. 27–32.

[114] T. Ferreira de Paiva Leite, "FD-SOI technology opportunities for more energy efficient asynchronous circuits," PhD Thesis, Université Grenoble Alpes, Grenoble, Jan. 2019.

[115] C. Tran, H. Kawaguchi, and T. Sakurai, "Low-power high-speed level shifter design for block-level dynamic voltage scaling environment," in *International Conference on Integrated Circuit Design and Technology (ICICDT 2005)*, Austin, TX, USA, 2005, pp. 229–232.

[116] Y. Lecun, C. Cortes, and C. Burges, "The MNIST database of handwritten digits," 1998.

[117] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[118] "Keras."

[119] R. A. Guazzelli and L. Fesquet, "At-speed DfT Architecture for Bundled-data Design," in *2020 IEEE International Test Conference (ITC)*, Washington, DC, USA, Nov. 2020, pp. 1–9.

[120] B. Bidegaray-Fesquet and L. Fesquet, "Levels, peaks, slopes… which sampling for which purpose ?" in *Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, Krakow, Poland, Jun. 2016, pp. 1–6.

[121] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593–616, Dec. 2004.

Publications de l'auteur

Conférences internationales avec actes

1. Y. Decoudu, K. Morin-Allory, and L. Fesquet, "A High-Level Design Flow for Locally Body Biased Asynchronous Circuits," in *28th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Octobre 2021.
2. Y. Decoudu, J. Simatic, K. Morin-Allory, and L. Fesquet, "From High-Level Synthesis to Bundled-Data Circuits," in *Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS 2020)*, Juillet 2020.
3. L. Fesquet, Y. Decoudu, R. Iga Jadue, T. Ferreira de Paiva Leite, O. Rollof, M. Diallo, R. Possamai Bastos, K. Morin-Allory, and S. Engels, "A Distributed Body-Bias Strategy for Asynchronous Circuits," in *27th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Octobre 2019.
4. Y. Decoudu, J. Simatic, P. Alexandre, K. Morin-Allory, and L. Fesquet, "Comparison of Synchronous and Asynchronous FIR Filter Architectures," in *5th International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, Mai 2019.

Communications et workshop internationaux

5. L. Fesquet, Y. Decoudu, R. Iga Jadue, T. Ferreira de Paiva Leite, O. Rolloff, M. Diallo, R. Possamai Bastos, K. Morin-Allory, and S. Engels, "Body-Bias Micro-Generators for Activity-Driven Power Management," in *FDSOI workshop at DATE Conference 2020*, Mars 2020.
6. Y. Decoudu, J. Simatic, K. Morin-Allory, and L. Fesquet, "Desynchronizing Circuits Synthesized with CatapultC," IP-SOC 2019, Grenoble, France.

Liste des acronymes

AES *Advanced Encryption Standard*. vi, vii, XVI, XVII, 52, 54, 57, 59, 61, 86–88

AFSM Machine à états finis asynchrone (*Asynchronous Finite State Machine*). v, 12, 13, 20, 21, 26, 30, 35, 36, 39, 40, 43–45, 47, 48, 54, 57, 59, 61, 78, 79, 90, 93, 94

ASIC Circuit intégré spécialisé (*Application-Specific Integrated Circuit*). 57

BBD *Body Bias Domain*. vi, 71, 72, 76, 78, 80–82, 84–87, 91, 94

BBG *Body Bias Generator*. vi, 71–73, 76, 78–82, 84–87, 91, 94

CAO Conception Assistée par Ordinateur. 2–4, 12, 13, 16, 18, 20, 22, 23, 27, 47, 55, 61, 72, 76, 84, 86, 93–95

CDFG graphe de flot de contrôle et données (*Control and Data Flow Graph*). 25

CMOS *Complementary Metal–Oxide–Semiconductor*. 12, 23, 66–69, 76

CNN Réseau de neurones convolutionnel (*Convolutional Neural Network*). vi, 76, 86, 88–90

DFT *Design For Test*. 95

DI Insensible aux délais (*Delay Insensitive*). 11, 12

DIBL *Drain Induced Barrier Lowering*. 67

DIMS *Delay Insensitive Min-terms Synthesis*. 12

DVS *Dynamic Voltage Scaling*. 1, 59, 66, 95

FBB *Forward Body Biasing*. 70, 78

FDSOI *Fully Depleted Silicon on Insulator*. 1, 3, 4, 52, 55, 61, 66, 68–70, 72, 76, 84, 86, 88, 91, 94, 95

FinFET *Fin Field Effect Transistor*. 66, 68, 69

FPGA *Field-Programmable Gate Array*. 2, 20

FSM Machine à états finis (*Finite State Machine*). v, 20, 25, 26, 30, 31, 33–36, 39–44, 46–48, 54–57, 61, 80, 81, 90, 93, 94

GALS Globalement Asynchrones Localement Synchrones (*Globally Asynchronous Locally Synchronous*). 14–16

HDL Langage de description de matériel (*Hardware Description Language*). 18, 19

HLS Synthèse de haut niveau (*High Level Synthesis*). XVI, 3, 21, 24–26, 30, 48, 52, 57, 61, 80, 86, 88, 90, 91, 93

IoT Internet des Objets (*Internet of Things*). 1, 2, 66, 72

IP Propriété intellectuelle (*Intellectual Property*). 71, 78

LCS *Local Clock Sets*. 23, 24, 30, 45–47, 55, 57, 84, 94, 95

LVT *Low Voltage Threshold*. 70, 71, 78, 91

NCL *Null Convention Logic*. 12, 19

NMOS Transistor MOS de type N. vi, 66, 69, 70, 79, 80

PLL *Phase Locked Loop*. 14

PMOS Transistor MOS de type P. 69, 70, 79, 80

PnR Placement/Routage (*Place and Route*). 30, 45, 52, 55, 56, 59, 76, 83, 84, 86, 87, 91, 93, 94

PVS *Prototype Verification System*. 44, 48, 94

PVT Procédés de fabrication, tension et température (*Process Voltage temperature*). 11, 14, 15, 20, 23, 24, 47, 71, 93

QDI Quasi-insensible aux délais (*Quasi-Delay Insensitive*). 12, 15, 19, 72, 76

RBB *Reverse Body Biasing*. 70, 91

RDF *Random Dopants Fluctuation*. 67, 70

RTC *Relative Timing Constraint*. 21, 23, 44

RTL Niveau transferts de registres (*Register Transfer Level*). 19, 22, 24, 25, 30, 46, 47, 80

RVT *Regular Voltage Threshold*. 70, 71, 91

SCE Effets de canal court *Short Channel Effects*. 66, 67, 70

SDC *Synopsys Design Constraints*. 47

SI Indépendant de la vitesse (*Speed Independent*). 12

STA Analyse temporelle statique (*Static Timing Analysis*). 22–24, 45

STG Graphe de transitions de signaux (*State Transition Graph*). 16, 17, 24, 37, 43

WCHB *Weak Contention Half Buffer*. 12, 16, 37

XBM *Extended Burst-Mode*. 13, 20

A

Prototype ASIC 65 nm

Le prototype présenté en figure 3.5 contient plusieurs circuits, dont deux AES asynchrones. Ceux-ci ont été intégrés ensemble et isolés du reste de la puce. Le système de test et ses connexions avec les AES sont décrits figure A.1. Les circuits AES sont *AHLS AES*, fait par HLS, et *Testable AES*, qui contient une solution de test structurelle des circuits asynchrones. Mais nous nous concentrerons sur le circuit *AHLS AES*, qui est l'AES réalisé dans le cadre de ce travail.

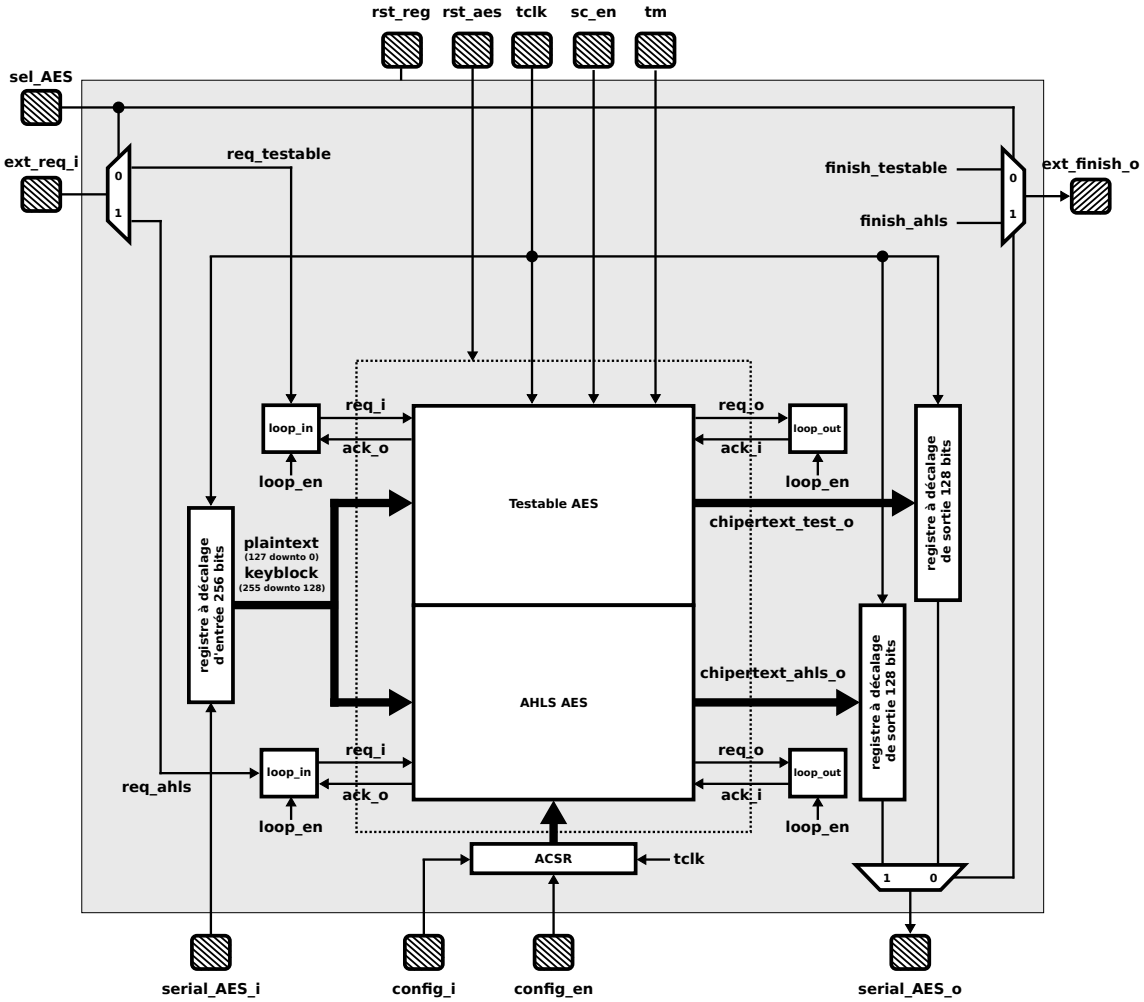


FIGURE A.1 – *Architecture des circuits AES asynchrones au sein du prototype.*

Le système de test entourant les circuits AES correspond à des registres à décalage synchrones contrôlés par l'horloge de test. Les données d'entrée et de sortie des AES sont récupérées par des registres à décalage pour utiliser les ports en série *serial_AES_i* et *serial_AES_o* sur le prototype. Un autre registre à décalage, le registre ACSR, permet de contenir des bits de configuration qui définissent le mode de fonctionnement du système, ainsi que le choix du circuit AES testé. Tous les ports du prototype sont répertoriés table A.1.

Il existe plusieurs modes de fonctionnement pour ce prototype :

— Mode chargement des données d'entrée : Active le registre à décalage d'entrée afin de charger les données (*plaintext* et *keyblock*) dans le circuit.

— Mode calcul une opération : Le circuit AES calcul une valeur de sortie lorsqu'il reçoit une requête d'entrée et s'arrête ensuite. Ce mode démarre lorsque le signal *ext_req_i* devient actif.

— Mode calcul en boucle : Active les blocs *loop_in* et *loop_out* et reboucle les signaux de communication en entrée et sortie des AES. Le circuit va alors calculer de nouvelles valeurs indéfiniment. Ce mode est actif lorsqu'à la fois les signaux *ext_req_i* et *loop_en* sont actifs.

— Mode acquisition des données en sortie : Active le registre à décalage de sortie voulu afin de récupérer les données (*ciphertext*) en sortie du circuit AES.

— Mode déchargement des données de sortie : Active le registre à décalage de sortie voulu afin de récupérer les données (*ciphertext*) à l'extérieur du circuit.

TABLEAU A.1 – *Pins d'entrées et sorties de la partie AES du prototype*

Signal	Direction	Usage
tclk	Entrée	Horloge de test pour les registres à décalage
rst_reg	Entrée	Signal de reset actif des registres à décalage
rst_aes	Entrée	Signal de reset actif au niveau haut des circuits AES
serial_AES_i	Entrée	Port série des données d'entrée des circuits AES
serial_AES_o	Sortie	Port série des données de sortie des circuits AES
config_i	Entrée	Port série des données d'entrée des registres ACSR
config_en	Entrée	Signal *enable* des registres ACSR
ext_req_i	Entrée	Requête d'entrée externe des circuits AES
ext_finish_o	Sortie	Requête de sortie des circuits AES
tm	Entrée	Activation du mode de test (uniquement pour l'AES testable)
sc_en	Entrée	Signal *enable* de la chaîne de scan (uniquement pour l'AES testable)

Flot de conception pour circuits asynchrones : de la HLS à l'implémentation en FDSOI

Résumé

La consommation est désormais une préoccupation majeure de l'industrie microélectronique, notamment avec l'émergence des objets de l'internet et de dispositifs sans batterie. L'immense majorité des circuits étant synchrones, des stratégies de réduction de l'énergie telles que le clock gating, le DVFS (Dynamic Voltage and Frequency Scaling) ou la polarisation du substrat en FDSOI ont été mises au point. Toutefois, ces approches, aussi vertueuses soient-elles, se traduisent par des contraintes très fortes sur l'arbre d'horloge. De plus, avec les nœuds de petites dimensions, la sensibilité aux variations des procédés de fabrication, de tension et de température accroît encore les contraintes de conception des puces. Une approche disruptive, basée sur des circuits asynchrones, permet d'augmenter la robustesse des circuits à ces variations et de réduire leur consommation énergétique. Afin de favoriser l'adoption des circuits asynchrones dans l'industrie, un flot de conception original, s'appuyant sur un outil commercial de synthèse de haut niveau, est proposé dans cette thèse. Par ailleurs, cette approche de haut niveau a été couplée à un système distribué de polarisation dynamique du substrat à grains fins afin d'optimiser l'efficacité énergétique des circuits. Ce flot a été en très grande partie automatisé.

Mots-clés : circuits asynchrones, synthèse de haut niveau, FDSOI, polarisation du substrat, basse consommation

Abstract

Power consumption is today a main concern for the microelectronic industry, especially with the emergence of the internet of things and batteryless devices. Since the vast majority of circuits are synchronous, energy reduction strategies such as clock gating, DVFS (Dynamic Voltage and Frequency Scaling) or FDSOI body biasing have been developed. However, these approaches, as virtuous as they are, impose very strong constraints on the clock tree. Moreover, with the most advanced nodes, their sensitivity to process variations, voltage and temperature further increase the chip design constraints. A disruptive approach, based on asynchronous circuits, is able to increase the circuit robustness to these variations while reducing their power consumption. In order to push the adoption of asynchronous circuits in the industry, an original design flow, based on a commercial high-level synthesis tool, is proposed in this thesis. Moreover, this high-level approach has been coupled to a distributed fine-grained and dynamic body biasing system in order to optimize the circuit energy efficiency. This flow has been mostly automated.

Keywords : asynchronous circuits , high-level synthesis, FDSOI, body biasing, low-power