



HAL
open science

Concevoir des circuits sécurisés à très faible consommation : une alternative basée sur l'asynchrone

Grégoire Gimenez

► **To cite this version:**

Grégoire Gimenez. Concevoir des circuits sécurisés à très faible consommation : une alternative basée sur l'asynchrone. Micro et nanotechnologies/Microélectronique. Université Grenoble Alpes [2020-..], 2021. Français. NNT : 2021GRALT006 . tel-03245307

HAL Id: tel-03245307

<https://theses.hal.science/tel-03245307v1>

Submitted on 1 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Grégoire GIMENEZ

Thèse dirigée par **Laurent FESQUET**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale EEATS**

Concevoir des circuits sécurisés à très faible consommation : une alternative basée sur l'asynchrone

Thèse soutenue publiquement le **12 février 2021**,
devant le jury composé de :

M. Laurent FESQUET

Maître de Conférence, Université Grenoble Alpes, Directeur de thèse

M. Lionel TORRES

Professeur, Université de Montpellier, Président

M. Sylvain GUILLEY

Professeur associé, TELECOM-ParisTech, Rapporteur

M. Olivier SENTIEYS

Professeur, Université de Rennes, Rapporteur

M. Marc RENAUDIN

HDR, Directeur technique, Tiempo Secure, Examineur

M. Antoine CHRISTIN

Ingénieur, Architecte composants de sécurité, Thales, Invité



Remerciements

Il paraîtrait que tout bon discours commence par une bonne anecdote. Je vous laisse juges de la qualité de mon discours. Mais l’anecdote suivante mérite certainement d’être racontée.

Nous sommes le 16 septembre 2015. Je suis alors employé depuis quelques temps chez Dolphin Integration. Assis à mon bureau, je travaille sur un circuit quelconque quand ma cheffe d’équipe, Lucille, débarque en trombe dans mon bureau : – « Il paraît que tu veux faire une thèse?! »

Attendez, non... Cette histoire commence plutôt la veille, le 15 septembre. Je discute avec mon beau-frère, Paul, dans les couloirs du CIME. Paul est entrepreneur. Sa startup est hébergée au sein de Minatec. Au détour de la conversation, je lui mentionne que la sécurité matérielle est un sujet qui m’intéresse particulièrement. Paul me répond : – « Oh, tu sais, toi, il faudrait que tu fasses une thèse ». Et nous nous séparons là-dessus.

Mais Paul a de la suite dans les idées. Il en parle à Skandar, son voisin de bureau. Skandar est alors co-directeur du TIMA. Il s’entend bien avec Laurent dont le bureau est deux étages plus bas. Skandar va donc voir Laurent. – « Dis Laurent, tu connais un certain “Grégoire Gimenez” ? Il paraît qu’il veut faire une thèse... »

Mais il s’avère qu’à ce moment précis, dans le bureau de Laurent, se trouve Sylvain... Sylvain qui, lui, connaît bien Grégoire. Et pour cause, c’est le mari de Lucille, la cheffe de Grégoire. Et vous connaissez la suite !

Merci donc à Paul, Skandar, Laurent, Sylvain, Lucille : vous avez tous été instigateurs de ce crime, et je vous en suis reconnaissant ! Un merci tout particulier à Louis qui a rendu possible ce projet au sein de Dolphin.

Merci à l’équipe CDSI pour l’accueil chaleureux qui m’a été fait. Merci à la bande des Brésiliens : Thiago, Leonel, Otto, Ricardo, Matheus, Rodrigo (oui je sais, tu es chilien, c’est tout à fait différent). J’ajoute aussi Jeanito, qui est presque brésilien ! Merci à vous pour la bonne ambiance, le maté, les horaires décalés et les cocoricos. J’attends toujours le barbecue...

Merci à mes colocataires de bureau successifs, Karim, Sylvain, Sophie, Yoan. Merci pour les discussions autour d’un café, les séances de craquage entrecoupées de débats passionnés et passionnants.

Merci aussi à la nouvelle garde de CDSI, Mohamed, Medhi, Julie, Jérémie, Nils, Liège, Yoan (le même).

Merci à mes compères de papiers. Karim, Jean, Guillaume, Raphaël. Ce fut un plaisir de bosser avec vous.

Merci à Robin qui a toujours été disponible pour m’aider à monter des manip, à Mohamed et Abdelhamid : votre aide a été très précieuse.

Merci à Olivier qui m’a encadré dans un premier temps chez Dolphin.

Merci à IC’Alps. A Jean-Luc et Lucille qui m’ont renouvelé leur confiance. A Guillaume qui m’a appris tout ce que je connais de la micro-électronique. Et à tous les autres qui se demandent encore où j’étais pendant tout ce temps.

Merci aussi à ceux qu'on oublie tout le temps. Aux services IT, à Fred, Ahmed, Jérôme, et à l'administration, à Anne-Laure et Laurence qui travaillent dans l'ombre.

Merci à tous ceux que j'oublie, Assia, Amani, Hugo...

Merci encore aux membres du jury, qui ont accepté de relire ma prose... longue prose. Merci à Antoine, c'était ta première expérience dans ce rôle, et je ne t'ai pas épargné!

Merci à toi Laurent pour tes conseils, tes relectures, et tes encouragements. Merci d'avoir su me convaincre de rédiger ce fameux papier alors que je ne pensais pas être prêt!

Merci à Paul, mon relecteur favori. Mon manuscrit serait illisible sans ton aide précieuse. J'ai sûrement été inconsciemment jaloux de ta propre thèse, ce qui m'a motivé à finir la mienne!

Merci aux gars du mardi, qui m'ont permis de relever les yeux chaque fois que nécessaire.

Merci à ma famille, de droite comme de gauche. A mes amis et proches, qui se reconnaîtront. Merci pour vos encouragements et votre soutien sans faille.

Merci à Louise, Amy et Augustin qui ont bien trop souvent accepté de prêter leur papa.

Et merci à Audrey, ma meilleure alliée. Toi qui a su pallier mes absences, supporter mes nuits blanches, coacher mon anglais, et profiter de tous ces voyages!

Pour finir, je ne voudrais pas oublier celui sans qui rien de tout cela n'aurait été possible. On l'appelle souvent hasard. Il a l'habitude de bien faire les choses. Mais comme Einstien le faisait remarquer, il se promène souvent incognito.

Table des matières

Table des matières	v
Liste des figures	ix
Liste des tableaux	xiii
Préambule	1
Introduction	3
I Flot de conception très basse consommation	13
1 Circuit asynchrones	15
1.1 Introduction	16
1.2 La communication dans les circuits logiques	16
1.2.1 Protocoles de communication	17
1.2.2 Encodage des données et signalisation	18
1.3 Classification des circuits logiques	19
1.3.1 Circuits synchrones	20
1.3.2 Circuits insensibles aux délais	21
1.3.3 Circuits quasi-insensibles aux délais	22
1.3.4 Circuits asynchrones à données groupées	22
1.3.5 Circuits désynchronisés	25
1.4 Atouts des circuits asynchrones	27
1.4.1 Performances accrues	27
1.4.2 Consommation réduite	27
1.4.3 Bruit et émissions électromagnétiques restreintes	28
1.4.4 Modularité optimale	28
1.4.5 Sécurité améliorée	29
1.5 Modélisation des circuits asynchrones	29
1.6 Conception de circuits asynchrones	30
1.7 Implémentation des circuits asynchrones	31
1.7.1 Contrainte de temps relatifs	31
1.7.2 Analyse temporelle des circuits asynchrones	31
1.7.3 Performances et validité fonctionnement	32
1.8 Conclusion	33
2 Asynchrone, où est le problème ?	35
2.1 Introduction	36
2.2 Nœud gordien et cercles vicieux	36
2.3 Couper le nœud	37
2.4 Les limites des solutions existantes	38
2.4.1 Limitations des outils synchrones	39

2.4.2	Limitations des approches min/max	40
2.5	Contraintes de temps relatifs et circuits à données groupées	41
2.5.1	Modèles et RTC	41
2.5.2	Classification des RTC	43
2.5.3	RTC, primitives et hiérarchies	47
2.6	Conclusion	49
3	La méthode LCS	51
3.1	Introduction	52
3.2	Principes de fonctionnement	52
3.2.1	Oscillateurs locaux	53
3.2.2	Horloges pour circuits sans horloge	53
3.2.3	LCS : séries d'horloges locales	55
3.2.4	Le cas des courses de signaux	57
3.3	Du STG aux LCS en passant par les RTC	59
3.3.1	Du STG aux RTC	60
3.3.2	Casser les boucles	60
3.3.3	Identifier les points de divergence	61
3.3.4	Simplifier les horloges	63
3.3.5	Construire les LCS	65
3.3.6	Restreindre les chemins	66
3.3.7	Générateur de contraintes LCS	67
3.3.8	Comparaison de différents protocoles	68
3.4	LCS et flot d'implémentation physique	71
3.4.1	Comparaison avec la méthode min/max	72
3.4.2	Contraindre le chemin de données	74
3.4.3	Synthèse des arbres locaux	75
3.4.4	Insertion des délais <i>matchés</i>	76
3.4.5	Flot d'implémentation LCS	78
3.5	Conclusion et perspectives	79
II	Primitives de sécurité à base d'anneaux auto-séquencés	83
4	Génération d'aléa	85
4.1	Introduction	86
4.2	Oscillateur en anneau auto-séquencé	87
4.2.1	Architectures d'oscillateur en anneau	87
4.2.2	Modes d'oscillation des STR	89
4.2.3	Effet <i>drafting</i> et effet <i>Charlie</i>	90
4.2.4	Répartition homogène des phases et finesse de résolution	91
4.2.5	Abstraction jetons/bulles	93
4.3	Générateur d'aléa dynamique : TRNG	95
4.3.1	Principes de fonctionnement	95
4.3.2	<i>Jitter</i> dans les oscillateurs en anneau	96
4.3.3	Imprédictibilité	97
4.3.4	Non-manipulabilité	102
4.3.5	Exemples de TRNG	103
4.3.6	Comparaison de différentes architectures	107
4.4	Générateur d'aléa statique : PUF	107
4.4.1	Principes de fonctionnement	108
4.4.2	Évaluation	109

4.4.3	Exemple de PUF	110
4.5	Conclusion	113
5	Optimisation et sécurisation du STR-TRNG	115
5.1	Introduction	116
5.2	Modèle de menaces	117
5.3	Attaques	118
5.3.1	Injection et suppression de jetons et de bulles	119
5.3.2	Modification du temps de propagation	120
5.4	Contremesures	120
5.4.1	Choix du nombre d'étages	121
5.4.2	Utilisation du mode interne	121
5.4.3	Moniteur de jetons	122
5.4.4	Fréquence-mètre	124
5.5	Simulations	125
5.5.1	Simulations mixtes	125
5.5.2	Simulations de haut-niveau	128
5.6	Implémentation ASIC	131
5.6.1	Architecture du circuit	131
5.6.2	Résultats	133
5.6.3	Discussion	135
5.7	Nouveau modèle stochastique basé sur le bruit du signal d'échantillonnage	135
5.7.1	Probabilité et entropie	136
5.7.2	Ordre du modèle	137
5.7.3	Limite basse de l'entropie	138
5.7.4	Comparaison des deux modèles	139
5.8	Conclusion et perspectives	140
6	Fonction physique non-clonable à base d'anneau auto-séquéncé	143
6.1	Introduction	144
6.2	Modèles stochastiques	145
6.2.1	PUF basé sur la mesure de fréquence	145
6.2.2	PUF basé sur la mesure de temps	147
6.2.3	Mesurer le temps et non la fréquence	149
6.3	TDC à base de STR	149
6.4	STR-PUF	150
6.4.1	Architecture	151
6.4.2	Principes de fonctionnement	152
6.5	Implémentation sur FPGA	153
6.5.1	Détails d'implémentation	153
6.5.2	Qualité des réponses brutes	154
6.5.3	Évaluation	155
6.5.4	A propos de la latence	157
6.6	Circuit de test	158
6.6.1	Objectifs	158
6.6.2	Architecture	159
6.6.3	Implémentation	160
6.7	Conclusion et perspectives	161
	Conclusion	163
A	Liste des acronymes	I

Liste des figures

1	Décomposition en couches d'un système embarqué.	7
2	Opposition entre consommation et sécurité pour les algorithmes de chiffrement symétriques.	8
1.1	Parallèle entre le langage naturel et la communication logique	16
1.2	Différents protocoles de communication.	17
1.3	Différentes conventions de signalisation.	19
1.4	Classification des circuits logiques	19
1.5	Modèle de circuit synchrone.	20
1.6	Schéma d'un oscillateur en anneau classique.	20
1.7	La porte de Muller aussi appelée C-element.	21
1.8	Modèle de circuit QDI.	22
1.9	Modèle de circuit asynchrone à données groupées.	23
1.10	Implémentations typiques de différents schémas de circuits.	24
1.11	Méthode de désynchronisation de CORTADELLA <i>et al.</i>	26
1.12	Modèle de désynchronisation de la méthode <i>Edge</i>	26
1.13	Exemples de STG	30
2.1	Cercles vicieux et nœud gordien.	37
2.2	Boucles combinatoires dans les circuits à données groupées.	39
2.3	Chevauchement de contraintes dans les circuits à données groupées.	40
2.4	Simplification des contraintes et impact sur l'analyse temporelle.	40
2.5	Boucles combinatoires dans les circuits à données groupées.	42
2.6	Arbre de classification des contraintes de temps relatifs.	43
2.7	Contraintes temporelles relatives du contrôleur Mousetrap.	47
2.8	Contraintes temporelles relatives du contrôleur Mousetrap à base de multiplexeur.	48
2.9	Différents cas d'utilisation d'un modèle hiérarchique.	49
3.1	RTC au niveau protocole d'un circuit à données groupées générique.	52
3.2	Oscillateurs dans les circuits à données groupées.	53
3.3	Différentes utilisation d'horloges dans les circuits de contrôle.	54
3.4	Vérification de <i>setup</i> dans un circuit synchrone.	55
3.5	Transformation d'une horloge en un évènement.	56
3.6	Décomposition sous forme de LCS de 3 RTC d'un circuit à données groupées non linéaire.	57
3.7	Principe de fonctionnement des vérifications de <i>clock-gating</i>	58
3.8	Inférence automatique de <i>clock-gating</i> dans les circuits à données groupées.	59
3.9	Protocole à données groupées Maximus.	60
3.10	Trois boucles combinatoires dans le protocole Maximus.	61
3.11	Contrainte de <i>regroupement</i> du protocole Maximus.	61
3.12	Contrainte de <i>écrasement de données</i> du protocole Maximus.	62

3.13	Contrainte de <i>temps de phase minimum</i> du protocole Maximus.	63
3.14	Contrainte de <i>course de chemins locaux</i> du protocole Maximus.	63
3.15	Décomposition des RTC en LCS.	64
3.16	Décomposition des RTC en LCS (<i>cont.</i>).	66
3.17	Générateur de fichier de contraintes SDC de la méthode LCS.	67
3.18	LCS SDC generator pseudo code.	69
3.19	Flots d'implémentation pour circuits à données groupées.	72
3.20	Circuit de chiffrement AES asynchrone.	73
3.21	Détail des différents délais existants dans un circuit à données groupées générique.	75
3.22	Circuit de chiffrement AES asynchrone.	77
3.23	Flot d'implémentation physique LCS complet.	78
3.24	Flot de conception LCS d'un circuit à données groupées.	80
4.1	Oscillateur en anneau à inverseurs.	88
4.2	Oscillateur en anneau à effet transitoire.	88
4.3	Architecture d'un oscillateur auto-séquence.	89
4.4	Comportement d'un anneau auto-séquence.	89
4.5	Comparaison de schéma de porte logique au niveau transistors.	91
4.6	Chronogramme des phases d'un STR de 5 étages.	92
4.7	Modèles d'abstraction jetons/bulles du STR.	93
4.8	Architecture générique d'un TRNG.	95
4.9	Principe des techniques d'extraction du <i>jitter</i> basé sur la coïncidence de deux signaux.	96
4.10	Principe des techniques d'extraction du <i>jitter</i> basé sur l'accumulation.	97
4.11	Modèle de TRNG garantissant l'imprédictibilité.	98
4.12	Architecture d'un MURO-TRNG.	101
4.13	Architecture de TRNG garantissant la non-manipulabilité.	104
4.14	Principe de fonctionnement du ERO-TRNG.	105
4.15	Principe de fonctionnement du STR-TRNG.	105
4.16	Modèle de l'extraction d'entropie du STR-TRNG basé sur le <i>jitter</i> de chaque phase.	106
4.17	Architecture générique d'un PUF.	108
4.18	Section d'un circuit intégré prise au MEB	111
4.19	Architecture de l'A-PUF.	112
4.20	Architecture du RO-PUF.	112
4.21	Architecture du SRAM-PUF.	113
5.1	Modèle de menaces du STR-TRNG.	117
5.2	Exemple d'ajout ou de suppression de jetons et de bulles dans un STR.	119
5.3	Illustration d'une attaque par modification du temps de propagation.	120
5.4	Mode interne du STR-TRNG.	121
5.5	Architecture du moniteur de jetons.	122
5.6	Principe de fonctionnement d'un compteur de jetons générique pour circuits asynchrones.	123
5.7	Architecture d'un fréquencemètre élémentaire.	124
5.8	Environnement de simulation développé pour évaluer les attaques et les contremesures.	125
5.9	Principe de fonctionnement du bloc de suppression de jetons.	126
5.10	Traces d'une attaque par suppression de jetons.	127

5.11	Principe de fonctionnement de l'attaque par modification de temps de propagation.	128
5.12	Traces d'une attaque par modification de temps de propagation.	129
5.13	Architecture du circuit de test et de l'environnement de caractérisation. . .	132
5.14	Cellule de Muller analogique.	132
5.15	Modèle stochastique du STR-TRNG basé sur le <i>jitter</i> de l'horloge d'échantillonnage.	137
5.16	Probabilité d'extraire un 1 et entropie en sortie du générateur.	138
5.17	Comparaison des modèles basés sur le bruit des phases et le bruit du signal d'échantillonnage.	139
5.18	Évolution de l'architecture du STR-TRNG pour exploiter le bruit d'échantillonnage.	140
6.1	Comparaison de la latence et de l'efficacité des trois principales architectures de PUF.	144
6.2	Principe de fonctionnement d'un PUF basé sur la fréquence de RO.	146
6.3	Entropie d'un bit d'un PUF basé sur la fréquence.	147
6.4	Principe de fonctionnement d'un PUF basé sur la période de RO.	147
6.5	Entropie d'un bit d'un PUF basé sur la période.	148
6.6	Architecture d'un TDC à base de STR.	150
6.7	Architecture du STR-PUF.	151
6.8	Modèle probabiliste de la largeur d'impulsion.	152
6.9	Floorplan du STR-PUF dans la matrice FPGA.	154
6.10	Réponses brutes des 1024 sources d'un PUF	155
6.11	Métriques évaluant la qualité des 8 STR-PUF	156
6.12	Architecture du circuit de test ST CMOS065.	159
6.13	Dessin du circuit de test.	160

Liste des tableaux

1.1	Différentes primitives habituellement utilisées	30
2.1	Comparaison des circuits synchrones, BD et QDI selon différentes métriques.	38
3.1	Nombre d’horloges constituant les LCS de différents protocoles à données groupées	70
3.2	Impact de la méthode LCS sur l’empreinte mémoire et le temps d’exécution de la STA pour différents protocoles.	71
3.3	Comparaison du contenu des fichiers de contraintes pour les versions synchrone, min/max et LCS du bloc AES.	73
3.4	Temps d’exécution et empreinte mémoire de la commande <code>update_timing</code> pour les versions synchrone, min/max et LCS de 10 blocs AES.	74
4.1	Liste des tests de la suite SP800-22.	99
4.2	Comparaison des performances de trois TRNG	107
5.1	Résultats des simulations haut-niveau.	130
5.2	Évaluation des séquences aléatoires de chacun des générateurs	133
6.1	Évaluation de l’imprédictibilité des réponses du STR-PUF	157
6.2	Latence du STR-PUF et périodes moyennes des RO.	158

Préambule

Les travaux présentés dans ce manuscrit ont été financés par les sociétés Dolphin Integration puis IC'Alps. Chacune de ces sociétés, par son domaine d'activité, est particulièrement concernée par les problématiques de faible consommation et de sécurité des circuits intégrés. Dolphin Integration¹ propose des blocs **IP** et des plateformes pour répondre aux besoins des applications **IoT**, mobiles ou automobiles. Elle doit donc faire face à ces défis. De son côté, IC'Alps s'attache à répondre aux besoins de ses clients dans les domaines médical (les dispositifs médicaux implantés en particulier) et industriel. L'utilisation parcimonieuse de l'énergie et la sécurité des circuits intégrés représentent, pour ces clients, une contrainte à caractère vital. Dans ce contexte industriel et technique, les objectifs que nous nous sommes fixés s'orientaient clairement vers des solutions applicables à plus ou moins court terme, et de nombreux choix ont été faits dans cette perspective d'industrialisation prochaine.

C'est également dans ce contexte industriel que le support d'ingénieurs expérimentés m'a permis d'approcher certaines thématiques avec un regard différent de celui habituellement adopté par le monde académique. Cette rencontre entre les mondes industriel et universitaire est certainement à l'origine des idées développées dans la première partie de ce document. En deuxième partie sont présentés les travaux que j'ai réalisés sur deux primitives de sécurité qui tirent parti de propriétés spécifiques aux circuits asynchrones. Mes prédécesseurs au sein du laboratoire TIMA avaient, avant moi, travaillé sur l'utilisation d'anneaux auto-séquenceés pour la réception de signaux ultra-wide band [Ham09], pour concevoir des liaisons séries à faible bruit de phase [Eli11] et pour concevoir des générateurs d'aléa [Che14]. Par leurs idées et leur soutien, ils m'ont permis de me hisser sur leurs épaules pour regarder un peu plus loin.

Au cours de ces quatre années de thèse, j'ai eu l'opportunité de développer deux circuits de test. Dans ce manuscrit, la place laissée à la description de ces réalisations reflète de manière très parcellaire le travail qu'elles ont demandé. Ces circuits n'ont pas encore révélé toutes les contributions qu'ils peuvent apporter aux deux problématiques qui nous intéressent. À mon tour de prêter mes épaules aux suivants pour les laisser voir un peu plus loin.

« Qui voit le plus loin, un nain ou un géant ? Sûrement un géant car ses yeux sont situés plus haut qu'un nain. Mais si le nain se place sur les épaules du géant, qui voit le plus loin ? (...) Ainsi nous aussi sommes des nains sur les épaules de géants. Nous maîtrisons leur sagesse et nous allons au-delà. Grâce à leur sagesse, nous devenons sages et nous devenons capable de dire ce que nous disons, mais pas parce que nous sommes plus grands qu'eux. »

— Isaiah di Trani

1. La société Dolphin Integration est devenue Dolphin Design depuis septembre 2019.

Introduction

Ce manuscrit synthétise le travail de recherche que nous avons effectué sur l'optimisation de la consommation énergétique des circuits intégrés ainsi que sur leur sécurisation. Nous soutenons que le schéma asynchrone à données groupées est un modèle approprié à la conception de tels circuits. Nous identifions des freins à l'adoption d'un tel paradigme et proposons un flot de conception basé sur des solutions logicielles standards. Nous montrons également comment utiliser les propriétés intrinsèques aux circuits asynchrones pour construire des primitives de sécurité particulièrement efficaces et sécuriser de manière globale un système.

Contexte technique et scientifique

Le début du XXI^e siècle est indéniablement marqué par ce que certains appellent la 3^{ème} révolution digitale [Tar16]. Après l'invention de l'écriture puis celle de l'imprimerie, le monde connaît actuellement un changement sans précédent dans la manière d'appréhender l'information : chacun en devient à la fois un créateur et un consommateur. Cette révolution est étayée par une longue série d'innovations technologiques. Elle prend racine au milieu du XX^e siècle avec l'avènement de l'ordinateur, puis de la téléphonie mobile, du web, des smartphones, du cloud, de l'intelligence artificielle, du big-data, etc. Rien ne paraît se soustraire à cette vague. Les cadres sociétal, culturel, éthique, politique et économique sont un-à-un profondément bouleversés. L'essor de ce nouveau monde cyberphysique accentue deux défis majeurs pour la communauté scientifique et l'industrie de l'électronique.

Basse consommation

Les efforts de développement, qui étaient dans un premier temps principalement guidés par le coût et les performances des circuits intégrés, sont maintenant largement réorientés vers l'autonomie des systèmes embarqués. Les avancées dans ce domaine apparaissent vertigineuses à la simple comparaison² des performances de l'ordinateur de suivi de navigation utilisé par la mission Apollo 11 et de celles d'un smartphone, même vieillissant.

Introduite dès 1957 par le développement des premières montres bracelets électriques, comme la Hamilton Electric 500, la problématique de l'efficacité énergétique n'aura de cesse de s'amplifier par la suite. En 1975, le tout premier ordinateur transportable est commercialisé, l'IBM 5100. Il pèse alors près de 23 kilogrammes. Poussés par un attrait général au nomadisme numérique, les premiers téléphones mobiles apparaissent au courant des années 80. Le *Simon Personal Communicator*, dévoilé en 1992, sera le premier

2. Voir par exemple https://en.wikipedia.org/wiki/Apollo_Guidance_Computer et <https://en.wikipedia.org/wiki/IPhone>.

téléphone intelligent³ avec écran tactile. IBM réunit alors dans un produit les fonctionnalités de téléphonie mobile et d'assistant personnel. Son autonomie était d'environ une heure pour un poids de 510 grammes. Quelques générations plus tard, au tout début de l'année 2007, sort le premier iPhone. Ce nouvel objet marque alors sans contexte un tournant dans cette course à la mobilité. Un même dispositif associe pour la première fois la connectivité de la téléphonie et celle de l'informatique (GSM, Wifi, Bluetooth, USB), la puissance de calcul et d'affichage (GPU), un appareil photo ainsi qu'une nouvelle interface homme-machine (écran tactile multipoints, accéléromètre); le tout pour seulement 135 grammes et une autonomie de près de 8h en communication. Dès lors, le téléphone intelligent prend le rôle de catalyseur de la convergence numérique et devient la principale interface avec le monde nouveau des objets connectés. Son essor coïncide ainsi avec la naissance de l'internet des objets⁴.

En 2011, le Pr. Jonathan Koomey, analysant l'évolution de l'efficacité pic⁵ des circuits numériques depuis le milieu des années 40, conclura que le nombre de calculs par unité d'énergie double tous les 18 mois environ [Koo+11]. Cette tendance est selon lui déterminante dans le développement historique de l'informatique ubiquitaire⁶. Cependant, quelques années plus tard, Koomey actualisera ses données⁷ et constatera un changement dans cette tendance amorcée dès les années 2000. L'efficacité énergétique pic continue de croître, mais à un rythme moindre, avec un doublement tous les 2,6 ans environ. Cette inflexion est sans conteste liée aux ralentissements successifs de la loi de Moore [Moo65] – la tendance à l'augmentation de la densité des transistors – et de l'échelle de Dennard [Den+74] – la tendance à conserver la même densité d'énergie malgré la réduction de la taille des transistors. Les explications physiques et techniques de cette perte de vitesse sont multiples. La part toujours plus importante des courants de fuite à la consommation globale des circuits, qui accompagne la réduction des dimensions, est sans doute l'un des principaux facteurs. L'explosion de la complexité et du coût de mise en place de nouveaux nœuds technologiques du fait des limites de la physique en est un autre. La difficulté toujours accrue à distribuer des signaux d'horloge de manière équilibrée à un nombre d'éléments séquentiels toujours croissants est aussi très souvent pointée du doigt⁸. Mais de ces nouvelles données, Koomey discerne aussi une nouvelle dynamique. Il prédit alors la tendance suivante : l'efficacité énergétique typique, celle mesurée dans un cas d'utilisation moyen, va continuer à croître à un rythme proche de celui initial (doublement tous les 18 mois). Il révèle ainsi le nouveau paradigme de la basse consommation du monde des objets. Pour continuer à améliorer leur efficacité, les systèmes ne peuvent plus être seulement optimisés selon leurs performances brutes. Ceux-ci doivent maintenant être dotés de capacités à s'adapter le mieux et le plus rapidement possible à leur contexte et à leur charge de travail.

3. Paul C. Mugge, l'un des principaux acteurs de cette invention aurait commenté par la suite : « *Don't invent one of these things before they invent the Internet or fiber optics with tremendous bandwidth.* »

4. Le CISCO *Internet Business Group* définit la naissance de l'IoT comme l'instant où plus d'objets sont connectés à internet que de personnes. Selon cette définition l'IoT est né entre 2008 et 2009 (voir https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf)

5. Efficacité mesurée en divisant le nombre maximum de calculs effectués par un dispositif sur une période donnée par sa consommation électrique sur cette même période.

6. Aussi appelée « intelligence ambiante ».

7. <https://www.electronicdesign.com/technologies/microprocessors/article/21802037/energy-efficiency-of-computing-whats-next>

8. Un ordre de grandeur souvent utilisé pour la consommation de l'arbre d'horloge se situe entre 25 % et 50 % de la consommation totale d'un circuit (voir [Tiw+98] ou plus récemment [Kan+16; Sit+16])

Sécurité matérielle

Dans un premier temps restreinte aux cartes à puces, à la gestion des droits numériques et aux applications de la défense, la problématique de la sécurité des circuits intégrés a été portée au tout premier plan par la croissance exponentielle de l'internet des objets. L'avènement du « tous, tout, partout, tout le temps connectés » a subversivement redéfini les frontières du monde physique. Chaque objet connecté devient maintenant une interface entre le privé et le public. L'expansion non-contrôlée du nombre de ces objets a multiplié et élargi considérablement les surfaces exposées aux attaques de personnes malveillantes. Il n'y a donc rien d'étonnant à ce que la barre symbolique du milliard d'attaques visant les objets connectés ait pour la première fois été franchie sur le seul premier semestre de l'année 2019⁹. Ce chiffre ne représente pourtant qu'une partie de la menace. Il comptabilise uniquement les attaques d'exploration, où l'attaquant tente d'identifier et de se connecter à distance sur un objet du réseau en utilisant la force brute. Ces attaques visent notamment à enrôler de nouveaux objets pour alimenter de gigantesques botnets, ou réseau de machine zombies, qui permettent par la suite de mener des attaques de grande ampleur (Spam, attaques DDoS, etc.). Ainsi, entre septembre et octobre 2016, les trois attaques les plus massives par déni de service jamais reportées ont été conduites contre le blog du journaliste Brian Krebs, l'hébergeur français OVH puis le service DynDNS. Ces attaques utilisaient alors un réseau de plusieurs millions d'objets connectés infectés par le maliciel Mirai. Si ces attaques ont eu un retentissement important – elles ont impacté pendant plusieurs heures un nombre important de sites internet très fréquentés (Twitter, Netflix, AirBnB...) – elles auraient pourtant pu être facilement évitées avec une politique de mot de passe adaptée¹⁰. Malgré cela, le mot de passe le plus utilisé en 2019 est toujours 123456¹¹.

Mais depuis quelques années, l'actualité est aussi ponctuée par d'autres types d'attaques et de vulnérabilités tout aussi inquiétantes. Des failles touchant le matériel sont régulièrement révélées par les chercheurs : RowHammer (2014) permet de manipuler le contenu des DRAM, KRACK et KR00K (2016 et 2019) exploitent des vulnérabilités des puces Wifi pour déchiffrer les communications, Spectre et Meltdown (2018), qui affectent la plupart des processeurs modernes, permettent d'accéder à des espaces mémoires sans avoir les privilèges requis, Starbleed (2020) permet de déchiffrer le bitstream de nombreux FPGA Xilinx [EMP20]. Aucun type de circuit ne semble être épargné. Certaines de ces attaques sont par ailleurs assez simples à mettre en œuvre et ne demandent que très peu de matériel. Bien qu'il n'y ait en général pas de preuve qu'elles aient pu être exploitées pour mener de réelles attaques, ces découvertes révèlent une vraie difficulté pour le monde de l'électronique. A l'instar des logiciels, le matériel n'est pas fiable. Et si ces vulnérabilités sont inévitables, le problème vient surtout du fait qu'elles sont très difficilement adressables puisqu'elles restent par définition figées dans le matériel. Même si elles peuvent parfois être mitigées de manière logicielle, les correctifs efficaces ne sont disponibles qu'avec une nouvelle génération du circuit et le remplacement du système. Et souvent, ces deux conditions n'interviennent que de nombreuses années après la découverte de la faille. Ce phénomène est encore amplifié par la diversité et la multiplication du nombre d'objets. En tout état de cause, cette difficulté doit être prise en compte au plus tôt dans la conception des circuits intégrés, et le niveau de sécurité doit impérativement être élevé et adapté aux temps de

9. 1,371 milliard d'attaques en considérant que les attaques utilisant les protocoles telnet et UPnP visent majoritairement l'IoT [F-S19].

10. Les cameras connectées du fabricant chinois Hangzhou XiongMai Technology semblent avoir été particulièrement enrôlées par Mirai. Elles utilisent un login (admin) par défaut sans mot de passe qu'il n'est pas nécessaire de changer pour commencer à utiliser le dispositif (voir à ce sujet : <https://krebsonsecurity.com/2018/10/naming-shaming-web-polluters-xiongmait/>)

11. https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

cycle qui régissent le monde du matériel.

Circuits Asynchrones

Le développement des circuits asynchrones est étroitement lié aux problématiques de basse consommation et de sécurité. Comme leurs homologues synchrones, ces circuits trouvent leurs origines à l'aube de l'informatique, dans les années 50, avec les travaux de HUFFMAN (1954) sur les machines à états finies et de MULLER *et al.* (1959) sur les circuits indépendants de la vitesse. Malgré de premiers succès industriels avec, par exemple, les superordinateurs ILLIAC I et II¹² avec leur puissance de calcul inégalée pour l'époque, ce type de circuit tombera petit à petit en désuétude du fait de leur complexité, laissant place à la suprématie de la logique synchrone, avec son apparente simplicité et son déterminisme rassurant. Il faut attendre la fin des années 80, avec les travaux de SUTHERLAND (1989) sur les circuits Micropipelines ou ceux de MARTIN (1990) sur les circuits quasi-insensible aux délais (*Quasi-Delay Insensitive*) (QDI), pour voir renaître l'asynchrone. Apparaissent alors les premiers outils de Conception Assistée par Ordinateur (CAO) et les premières réalisations académiques de microprocesseurs (AMULET, MiniMIPS, Aspro, etc.). Les premières applications commerciales voient aussi le jour quelques années plus tard, motivées par des applications très basse consommation comme les pagers (Philips Semiconductors, Handshake Solution) ou hautes performances avec les switches Ethernet (Myricom, Fulcrum). Pour ces applications, l'absence d'un signal d'horloge permet de fonctionner plus rapidement tout en réduisant la consommation. La robustesse accrue, liée à des hypothèses temporelles moins strictes (voire absentes), permet aussi à ces circuits de s'adapter automatiquement à des conditions d'alimentation fluctuantes. Plus récemment, la société Tiempo (créée en 2007) met à profit les qualités intrinsèques des circuits QDI – résistance aux fautes, limitation des fuites d'information par canaux cachés, réduction de la surface d'attaque du fait de l'absence d'horloge – pour proposer des blocs asynchrones sécurisés à destination principalement des applications de paiement sans contact. Mais en dépit de ces expériences probantes et du consensus de plus en plus établi sur les avantages des circuits asynchrones, leur utilisation reste anecdotique.

Dernièrement, les circuits asynchrones semblent connaître un intérêt renouvelé pour leur efficacité énergétique. Dans le domaine de l'intelligence artificielle, le circuit TrueNorth, développé par IBM (2014), est le plus large réseau de neurones à impulsion (*Spiking Neural Network*) (SNN) jamais conçu [Mer+14]. Il embarque plus d'un million de neurones qui, grâce à un mélange d'asynchrone (QDI) et de synchrone, consomme seulement 70 mW en fonctionnement nominal. En 2018, Intel annonce à son tour la disponibilité d'un SNN, Loihi, un circuit de près de 130 000 neurones [Dav+18]. Plus petit, ce circuit implémente cependant une densité de synapses quatre fois plus importante que TrueNorth avec 1024 synapses par neurone. Cette fois-ci, pour minimiser la consommation tout en s'adaptant au mieux au caractère événementiel des communications entre les neurones, le schéma asynchrone à données groupées (Micropipeline) est choisi. En Europe, une société comme SynSense (ex aiCTX) a aussi choisi l'asynchrone pour développer des réseaux de neurones très peu consommateurs pouvant être déployés sur des applications IoT et *Edge Computing* [Mor+18].

Même si les applications industrielles à large échelle et la démocratisation de l'asynchrone se font toujours attendre, la conjonction actuelle de contraintes de sécurité et de basse consommation particulièrement sévères de l'internet des objets, l'arrivée à maturité de solutions logicielles de conception ainsi que l'abandon progressif du carcan du déterminisme (architecture many-core, calcul stochastique, réseau de neurones, GALS, etc.)

12. <https://fr.wikipedia.org/wiki/ILLIAC>

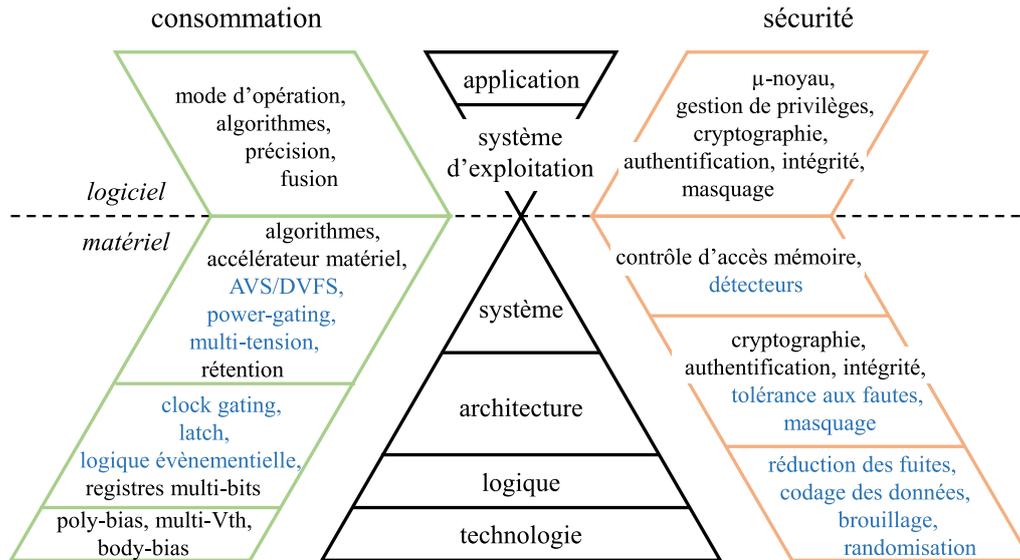


FIGURE 1 – Décomposition en couches d'un système embarqué et liste des différentes techniques d'optimisation (adapté de [Wan+11]). Les solutions impactées positivement par les techniques asynchrones sont accentuées en bleu.

semblent préparer la voie à une intégration plus importante des techniques asynchrones.

Problématique

Au premier abord, la sécurité et la basse consommation peuvent être approchées d'une manière très similaire. Il est possible d'appréhender ces deux défis à l'aide d'un modèle composé de différentes couches superposées : depuis la technologie jusqu'à l'application (figure 1). Dans ce modèle, chaque couche peut être optimisée plus ou moins indépendamment des autres. La performance globale du système est alors garantie par les optimisations apportées à chaque niveau. Celles-ci se cumulent, voire se multiplient. Ainsi, du côté de la consommation, l'utilisation de transistors avec une tension de seuil élevée va permettre de réduire les courants de fuites. En même temps, à plus haut niveau, le choix d'un algorithme approprié va permettre de diminuer le nombre de transistors nécessaires et va encore améliorer la consommation statique du circuit. De la même manière, du côté de la sécurité, l'utilisation d'un schéma robuste de masquage des variables sensibles, au niveau logiciel, permettra de renforcer la résistance d'une fonction cryptographique aux attaques par canaux auxiliaires¹³. Alors qu'au niveau logique, l'utilisation d'un codage double rail (données codées sur deux bits) limitera encore plus les fuites d'information à travers ces mêmes canaux, et la résilience du système à ces attaques s'en trouvera démultipliée. Cette représentation illustre bien l'étendue de ces deux challenges et la diversité des réponses qui peuvent y être apportées. Les aborder de manière exhaustive est difficilement envisageable.

La consommation et la sécurité ont aussi en commun qu'elles sont toutes deux très dépendantes de l'utilisation qui est faite du système. Suivant le contexte et l'application, un système ne doit pas être optimisé de la même manière. Ainsi, pour bien dimensionner les contremesures à développer, la cible de sécurité doit être clairement identifiée et un

13. Attaques qui se basent sur l'analyse des informations émises par l'implémentation d'un algorithme plutôt que l'algorithme lui-même : par exemple l'analyse de la consommation d'un circuit lors d'une opération de chiffrement peut permettre de retrouver la clé utilisée.

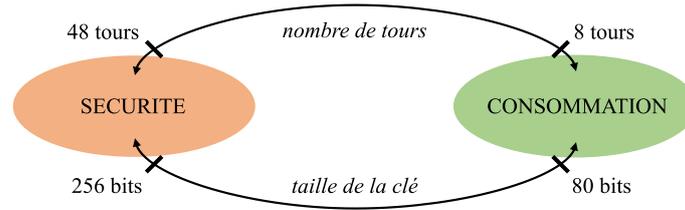


FIGURE 2 – Opposition entre consommation et sécurité pour les algorithmes de chiffrement symétriques (inspirée de¹⁴).

modèle de menaces doit être défini. Un nouveau détecteur ultra-sensible d’attaque laser sera réservé à un dispositif qui traiterait des informations intéressantes d’un adversaire ayant des moyens importants. De la même manière, les scénarios d’utilisation ou profils de mission du système doivent être anticipés pour que les optimisations puissent porter sur les paramètres et les parties du circuit qui contribuent le plus à la consommation. Mettre en place un système complexe d’adaptation dynamique de la tension et de la fréquence (*Dynamic Voltage and Frequency Scaling*) (DVFS) pour un circuit opérant la plupart du temps en mode veille serait certainement contreproductif.

Mais la sécurité et la basse consommation sont également antagonistes, et ce, à plusieurs égards. L’opposition entre ces deux défis se traduit tout d’abord en ce que la sécurité ne s’améliore généralement qu’au détriment de la consommation. L’ajout de détecteurs d’intrusion, de nouveaux protocoles d’authentification, d’algorithmes de chiffrement robustes, etc., nécessite de mobiliser une surface non négligeable de la puce, ou de l’espace mémoire, uniquement pour ces fonctions de sécurité. Au-delà du surcoût silicium, ces transistors supplémentaires augmentent la consommation du circuit; d’autant plus que ces fonctions de sécurité doivent souvent être activées en permanence. La figure 2 illustre bien cette dualité entre la consommation et la sécurité qui se retrouve au sein des fonctions de sécurité elles-mêmes. Dans le cas des algorithmes de chiffrement, différents leviers — le nombre de tours, la taille de la clé — sont couramment utilisés pour trouver un compromis entre efficacité énergétique et robustesse aux attaques. Cette approche, qualifiée de cryptographie légère, peut être généralisée à l’ensemble du système et des primitives de sécurité. Le défi étant alors d’identifier les « bons » curseurs permettant d’équilibrer sécurité et consommation.

Par ailleurs, ces deux problèmes sont de natures bien différentes. Si la basse consommation peut être vue comme une corde — la contribution de chacun des brins venant renforcer la solidité de l’ensemble — la sécurité doit, quant à elle, être comprise comme une chaîne. La cohérence de l’ensemble dépend de la résistance de chacun des maillons et négliger l’un de ces maillons peut compromettre toute la chaîne. De plus, si réduire la consommation d’un circuit est un problème connu — les différents paramètres influant sur celui-ci (tension, fréquence, activité, capacité, courant, température, etc.) sont clairement identifiés et peuvent être reliés par des équations simples¹⁵ — la sécurité est à l’inverse un problème non borné. Le nombre de variables en jeu est considérable et évolue au fur et à mesure que de nouvelles attaques sont imaginées. Des fuites d’information utilisant de nouveaux canaux auxiliaires sont régulièrement trouvées (consommation, rayonnement électromagnétique, acoustique, radio-fréquence, etc.). Les dernières avancées technologiques (intelligence artificielle, calcul quantique, etc.) apportent aussi leur lot de nouveaux moyens de cryptanalyse

14. http://cryptowiki.net/index.php?title=Lightweight_ciphers

15. Par exemple, la part de la consommation dynamique d’un circuit induite par les courants de commutation peut être simplement modélisée par l’équation $P_{SW} \propto CV^2f$.

qui peuvent compromettre la sécurité de nombreux algorithmes de chiffrement pourtant considérés comme sûrs. Ainsi, la sécurité est un domaine mouvant, qui est beaucoup plus difficilement modélisable. En d'autres termes, la sécurité est une appréciation relative et variable de la résistance d'un système à un attaquant. Cette appréciation peut évoluer au cours du temps et un circuit, jugé sécurisé à un moment donné, pourra être compromis quelques temps plus tard. La consommation est quant à elle une mesure objective et stable de la performance d'un système. A périmètre d'application et d'utilisation constant, un système aura une consommation qui ne varie pas dans le temps¹⁶.

Enfin, dans la perspective d'un monde toujours plus connecté et nomade, réduction de consommation et sécurité paraissent difficilement réconciliables. La complexité et le coût des solutions traditionnelles sont importants si bien que l'antagonisme entre ces deux objectifs se résout généralement par un compromis en faveur de la basse consommation – la partie visible du problème – et ce, au détriment de la sécurité dont l'évaluation est bien plus difficile. Hélas, plus que jamais, cette résolution est discutable compte tenu du progrès continu des attaques, de la sensibilité des données en jeu et des temps de vie du matériel.

Proposition et objectifs

De par leurs qualités intrinsèques, les circuits asynchrones semblent de bons candidats pour résoudre le dilemme entre sécurité et basse consommation. L'exemple le plus marquant est certainement l'absence d'horloge qui permet simultanément de réduire consommation et surface d'attaque. Bien souvent les techniques asynchrones jouent aussi un rôle de catalyseur, décuplant l'efficacité des solutions utilisées habituellement. Supprimer l'horloge permet, par exemple, de simplifier grandement les mécanismes de DVFS. Cette technique d'optimisation de la consommation est très efficace, mais elle est complexe à mettre en œuvre et présente des contreparties non négligeables sur les performances. Elle augmente la latence de réaction du circuit, et l'énergie dépensée pendant les transitions de mode¹⁷ vient grever les gains d'efficacité potentiels. Ces techniques sont de ce fait généralement réservées aux circuits de pointe (processeur, accélérateur graphique, etc.). Mais dans les circuits asynchrones, la « fréquence » de fonctionnement s'adapte automatiquement aux conditions d'opération¹⁸. Il suffit donc de contrôler la tension pour accélérer ou ralentir le circuit. De la même manière, ces circuits étant plus robustes aux variations de tension, les méthodes de masquage de la consommation par injection de bruit sur l'alimentation peuvent être utilisées de manière prononcée et seront ainsi plus efficaces pour se protéger des attaques par canaux auxiliaires. La figure 1 présente en bleu les éléments pour lesquels l'asynchrone est particulièrement adapté ou permet d'en décupler l'efficacité. Mais, malgré des atouts indéniables et les promesses d'une démocratisation imminente^{19,20}, les circuits asynchrones restent toujours minoritaires dans le monde de l'électronique. Par ailleurs, la contradiction entre sécurité et basse consommation se retrouve aussi au sein des tech-

16. Si l'on néglige les effets de vieillissement de l'électronique qui peuvent avoir un léger impact sur la consommation.

17. Les modifications de fréquence et de tension ne peuvent pas être faites simultanément. Un séquencement est nécessaire pour garantir un fonctionnement correct du circuit.

18. Si l'on peut parler de fréquence dans le cas d'un circuit asynchrone.

19. « *An asynchronous approach offers many advantages and is unavoidable in the long run.* » — Alain Martin (2007), *Asynchronous Logic : Results and Prospects*, voir <https://www.slideserve.com/johana/asynchronous-logic-results-and-prospects-powerpoint-ppt-presentation>

20. Le rapport 2015 de l'ITRS prévoit une adoption des circuits asynchrones et une pré-production pour 2020/2021 (voir https://www.semiconductors.org/wp-content/uploads/2018/06/1_2015-ITRS-20_System-Integration.pdf)

niques asynchrones. Par exemple, l'utilisation du schéma **QDI**, avec son codage double rail, est particulièrement adaptée à la sécurité. Ce schéma est, par contre, beaucoup moins compact qu'un schéma à données groupées qui est, lui, plus intéressant pour réduire la consommation, en particulier lorsqu'il est utilisé sur des nœuds technologiques avancés pour lesquels la consommation statique devient prépondérante.

Dans cette présentation de nos travaux, nous nous alignons sur la vision de S.B. FURBER (1993) plaidant pour une intégration progressive de la logique asynchrone, en commençant par ces niches composées « d'applications sensibles à la consommation et de fonctions ayant des charges de travail variables » [Fur93]. Nous complétons simplement cette liste avec les systèmes pour lesquels la sécurisation des données est aussi un pré-requis indispensable. Ainsi, nous proposons d'étudier plus en détails en quoi, et comment, les circuits asynchrones peuvent être utilisés pour concevoir des circuits à la fois sécurisés et basse consommation. Sans pouvoir couvrir l'ensemble de ces deux problématiques, nous donnons des éléments étayant la pertinence de cette approche et montrant qu'elle est appropriée à un contexte industriel. Elle apparaît alors comme une base intéressante pour construire une plateforme sécurisée et très basse consommation répondant aux challenges du nouveau monde des objets connectés. Les objectifs de la thèse sont donc les suivants :

- 1) Identifier le schéma asynchrone le plus approprié pour répondre à la problématique de sécurité et de basse consommation.
- 2) Discerner les freins à l'utilisation de la logique asynchrone et proposer un flot de conception permettant d'en accélérer l'adoption.
- 3) Montrer comment les techniques asynchrones peuvent aussi être utilisées pour concevoir des primitives de sécurité performantes et très basse consommation.
- 4) Évaluer la résilience de ces circuits à des attaques classiques et proposer des contre-mesures simples et efficaces.

Contributions et plan

Ce manuscrit est composé de deux parties. Dans la première, nous présentons un flot de conception pour circuits asynchrones qui a l'avantage d'être bâti uniquement à l'aide d'outils commerciaux habituellement utilisés pour les circuits synchrones. Malgré les nombreux flots déjà proposés par la communauté, tous ont jusqu'à présent utilisé les outils standards de manière détournée, sans pouvoir profiter pleinement de leur capacité d'optimisation. Dans les travaux que nous présentons, nous nous attachons à identifier et à surmonter les limitations existantes et nous espérons ainsi contribuer au rapprochement des mondes avec et sans horloge. Le **chapitre 1** présente un état de l'art général sur les circuits auto-séquenceés. Nous mettons en exergue les avantages habituellement attribués à ces circuits mais aussi la disparité du monde asynchrone. Le **chapitre 2** identifie les obstacles pouvant expliquer l'indifférence de l'industrie pour les solutions asynchrones. Nous montrons que les hypothèses temporelles associées à ces circuits sont certainement la source de ce désamour. Nous proposons alors de franchir cet obstacle à l'aide d'une nouvelle méthodologie décrivant le fonctionnement asynchrones à l'aide d'un formalisme purement synchrone : des séries d'horloges locales. Cette méthodologie peut être utilisée pour construire un flot de conception de circuits auto-séquenceés complet et totalement compatible avec les flots synchrones existants (**chapitre 3**).

La deuxième partie de ce manuscrit s'intéresse à deux primitives de sécurité, un générateur de nombres véritablement aléatoires (*True Random Number Generator*) (**TRNG**) et une fonction physique non-clonable (*Physical Unclonable Function*) (**PUF**), pouvant être construites à l'aide d'un circuit asynchrone particulier appelé oscillateur auto-séquenceé

(*Self-Timed Ring oscillator*) (**STR**). Nous montrons que l'utilisation de la logique asynchrone permet d'améliorer l'efficacité énergétique et la sécurité de ces blocs. Le **chapitre 4** présente un état de l'art synthétique sur ces trois éléments (**STR**, **TRNG** et **PUF**). Nous détaillons ensuite les différentes modifications que nous avons pu apporter à l'architecture du **STR-TRNG** pour optimiser sa consommation et le protéger d'éventuelles attaques (**chapitre 5**). Nous abordons ces problématiques de manière théorique, avec l'optimisation d'un modèle stochastique garantissant l'imprévisibilité du générateur, et pratique, avec des mesures et des attaques sur le silicium d'un circuit développé par nos soins validant la pertinence des contremesures que nous avons développées. Enfin, le **chapitre 6** propose une nouvelle architecture de **PUF** qui utilise le **STR** pour mesurer précisément et rapidement les variations de temps de propagation engendrées par les disparités du processus de fabrication des circuits intégrés. Nous présentons un modèle permettant de dimensionner la primitive ainsi que des mesures faites sur une implémentation **FPGA** de ce **PUF** corroborant le modèle proposé.

In fine, au travers de ce document nous plaidons pour un mariage entre synchrone et asynchrone pour tirer profit du meilleur des deux mondes et construire des circuits toujours moins consommateurs et toujours plus sécurisés.

Première partie

**Flot de conception très basse
consommation**

Chapitre 1

Circuit asynchrones : un état de l'art

Sommaire

1.1	Introduction	16
1.2	La communication dans les circuits logiques	16
1.2.1	Protocoles de communication	17
1.2.2	Encodage des données et signalisation	18
1.3	Classification des circuits logiques	19
1.3.1	Circuits synchrones	20
1.3.1.1	Circuits à horloge réactive	20
1.3.1.2	Circuits globalement asynchrones et localement synchrones	21
1.3.2	Circuits insensibles aux délais	21
1.3.3	Circuits quasi-insensibles aux délais	22
1.3.4	Circuits asynchrones à données groupées	22
1.3.4.1	Protocoles à données groupées	23
1.3.5	Circuits désynchronisés	25
1.4	Atouts des circuits asynchrones	27
1.4.1	Performances accrues	27
1.4.2	Consommation réduite	27
1.4.3	Bruit et émissions électromagnétiques restreintes	28
1.4.4	Modularité optimale	28
1.4.5	Sécurité améliorée	29
1.5	Modélisation des circuits asynchrones	29
1.6	Conception de circuits asynchrones	30
1.7	Implémentation des circuits asynchrones	31
1.7.1	Contrainte de temps relatifs	31
1.7.2	Analyse temporelle des circuits asynchrones	31
1.7.3	Performances et validité fonctionnement	32
1.8	Conclusion	33

1.1 Introduction

Le terme asynchrone est doublement trompeur. Tout d’abord, il laisse penser que les circuits asynchrones sont des circuits non-synchronisés alors qu’ils sont, au contraire, particulièrement bien synchronisés. Mais à la différence des circuits synchrones, ces circuits n’utilisent pas un signal global de synchronisation – communément appelé horloge. Mais ils se basent sur un protocole de communication qui permet d’avoir une synchronisation locale, avec un grain plus ou moins fin. Pour lever cette confusion certains ont alors préféré parler de circuits avec horloge et de circuits sans horloge, ou auto-séquenceés. Cependant, la frontière entre ces deux types de circuits est de moins en moins marquée, et le monde synchrone emprunte régulièrement des concepts et des techniques asynchrones pour résoudre certaines problématiques inhérentes à l’utilisation d’une horloge. Une des illustrations marquantes de ce phénomène est l’apparition d’une catégorie de circuits dits Globalement Asynchrones et Localement Synchrones (**GALS**) qui sont un assemblage de plusieurs blocs synchrones inter-connectés de manière asynchrone. De plus, le terme asynchrone peut laisser entendre que les circuits n’utilisant pas d’horloge forment un ensemble homogène, avec des règles de construction et des protocoles de communication bien définis. Cette vision est trompeuse. Il existe en effet de nombreux types de circuits asynchrones. Le monde asynchrone doit plutôt être compris comme un large espace de conception aux dimensions multiples. Avec cette perspective, les circuits synchrones forment un sous-ensemble particulier de cet espace.

Dans ce chapitre nous donnons un aperçu de ce vaste monde de l’asynchrone. Nous partons de la définition des principes de base de la communication dans un circuit logique (section 1.2) pour identifier les caractéristiques permettant de classifier les différents modèles de circuits (section 1.3). La section 1.4 présente ensuite les principaux avantages liés à l’utilisation de circuit sans horloge. Enfin, les sections 1.5 à 1.7 introduisent différents outils et méthodes permettant de modéliser, concevoir et implémenter les circuits asynchrones.

1.2 La communication dans les circuits logiques

Un circuit logique est un système composé de différents opérateurs s’échangeant de manière séquentielle des informations au travers de canaux de communication. La figure 1.1 illustre une telle communication entre l’opérateur A qui émet une information sur son canal de sortie et l’opérateur B qui la reçoit sur son canal d’entrée.

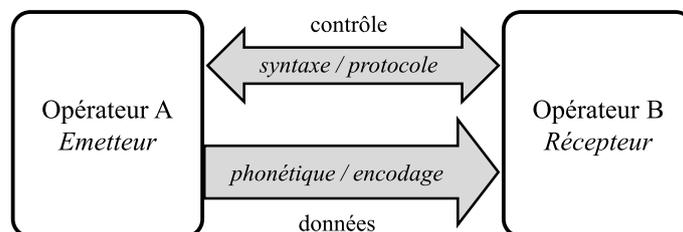


FIGURE 1.1 – Parallèle entre le langage naturel et la communication logique

Pour établir une communication les deux parties doivent s’entendre sur la manière de synchroniser leurs échanges. En ce sens, communiquer revient à s’accorder sur un découpage du temps. A l’instar du langage naturel, une telle convention de communication peut être décomposée en une phonétique et une syntaxe. La syntaxe organise les échanges. Elle définit comment débute et se termine une phrase. Au niveau logique, on parle du protocole de communication qui fixe les périodes de validité des données. La phonétique, elle, fait

référence à la manière dont sont construites les syllabes et comment elles sont agencées pour former des mots¹. D'un point de vue informatique, cela correspond à la façon dont sont encodées les données.

1.2.1 Protocoles de communication

Un protocole de communication permet de synchroniser les échanges d'information. Il spécifie les moyens utilisés par les opérateurs pour indiquer leur capacité à accueillir ou à fournir de nouvelles données et pour signaler l'envoi ou la réception de celles-ci. Un protocole est ainsi découpé en phases combinant les différents états de chaque opérateur : soit actif (émission ou réception), soit passif (en attente). La figure 1.2 présente quelques protocoles généralement utilisés dans les circuits logiques.

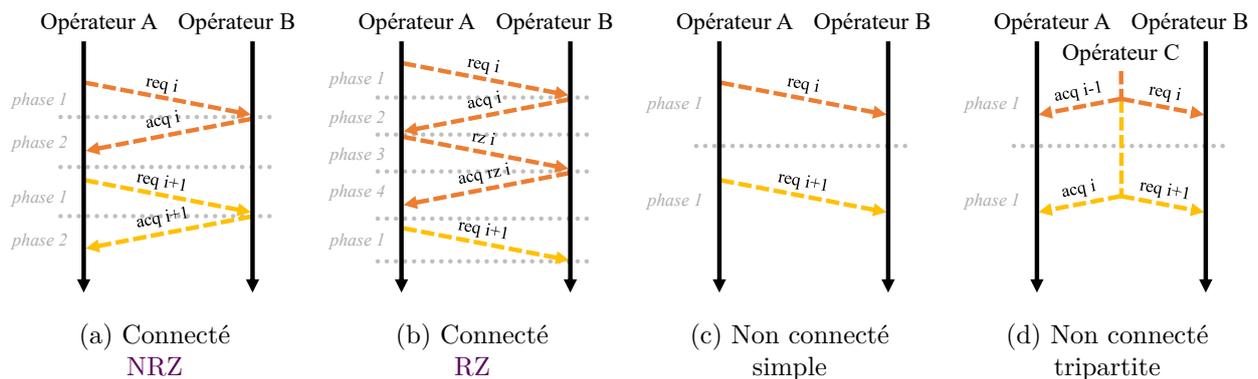


FIGURE 1.2 – Différents protocoles de communication. Seuls les signaux de contrôle sont présentés.

La méthode la plus évidente pour organiser une communication est d'utiliser un protocole connecté. Les deux parties s'échangent des signaux de contrôle explicites sous la forme de requêtes et d'acquittements semblables à une poignée de main (*handshake*). On retient généralement le protocole 2 phases et le protocole 4 phases, même si d'autres protocoles sont imaginables². Le premier (figure 1.2a) correspond à la poignée de main la plus simple : le récepteur renvoie un acquittement à chaque requête de l'émetteur. C'est à priori le plus rapide et le moins consommant des deux puisqu'il nécessite seulement deux événements. Le protocole 4 phases (figure 1.2b) utilise une étape supplémentaire de retour-à-zéro (*Return to Zero*) (RZ) afin de mettre le canal dans son état initial à la fin de la communication. En principe, ce protocole apparaît plus complexe car il met en jeu quatre événements différents pour réaliser un seul cycle de communication. Cependant, son implémentation matérielle est généralement plus simple et les phases supplémentaires peuvent être utilisées à profit pour le découplage des canaux de communication.

De manière à s'affranchir d'un signal d'acquittement, il est aussi possible d'utiliser un protocole non-connecté simple (figure 1.2c). Dans ce cas, il n'y a plus de retour d'information du récepteur vers l'émetteur. Cela permet théoriquement d'accélérer la communication – un seul événement est nécessaire pour décrire un cycle – celle-ci sera donc moins coûteuse. En contrepartie, les deux opérateurs doivent s'accorder sur un créneau temporel qui va borner leur échange. Après une période de temps convenue, le récepteur devra, dans tous

1. Plus précisément, la manière dont sont construits les mots relève de la morphologie.

2. Un protocole 3 phases suivant le séquençement $req \rightarrow acq \rightarrow rz$ est par exemple envisageable. Dans ce cas, les signaux de requête et d'acquittement sont asymétriques, la remise-à-zéro de l'acquittement étant faite de manière implicite.

les cas, avoir fini son traitement et l'émetteur pourra communiquer une nouvelle donnée sans craindre d'écraser la précédente. Ce protocole peut être vu comme un protocole « une phase », même si ce terme n'est pas usuel. Un émetteur-récepteur asynchrone universel (*Universal Asynchronous Receiver-Transmitter*) (**UART**) est l'exemple typique d'un bloc utilisant ce protocole.

Le principal problème des protocoles « une phase » est le déphasage et la déviation entre les bases de temps utilisées par chacun des opérateurs. En pratique, cela limite grandement le débit atteignable et augmente la consommation du fait du sur-échantillonnage nécessaire au niveau du récepteur. Pour s'affranchir de cela, il est possible de faire intervenir une tierce partie. Celle-ci aura la tâche de cadencer les communications. On crée ainsi un protocole tripartite (figure 1.2d). C'est en fait le type de protocole majoritairement utilisé dans les circuits logiques. Il est le support de toute communication dite « synchrone ». Lorsque ce troisième opérateur est mutualisé entre plusieurs, voire tous les canaux de communication, il est alors appelé « horloge ». L'implémentation de ce protocole est extrêmement simple puisqu'un seul signal est suffisant pour synchroniser l'ensemble des communications.

1.2.2 Encodage des données et signalisation

Par opposition avec l'électronique analogique qui manipule des grandeurs physiques continues, un circuit logique utilise des signaux binaires. Cette discrétisation des informations permet d'utiliser l'algèbre de Boole pour concevoir rapidement et efficacement des systèmes très complexes. Mais si le support de l'information reste binaire, chaque bit de donnée peut être encodé de plusieurs manières. On distingue généralement le codage spatial – combien de signaux sont nécessaires pour représenter un bit – et le codage temporel – quelle signalisation est utilisée pour transporter l'information. Au niveau spatial, deux solutions sont privilégiées :

- **Le codage simple rail** : un bit de donnée est encodé à l'aide d'un seul rail. Ce schéma un-pour-un est le plus simple et le plus utilisé. Il est aussi le moins coûteux au niveau matériel.
- **Le codage double rail** : un bit de donnée est encodé à l'aide de deux rails. Il est aussi appelé codage 1-parmi-2. Il est bien sûr plus coûteux en terme de surface. Cependant, cette redondance fonctionnelle laisse la possibilité d'intégrer dans le même code les signaux de contrôle et les signaux de donnée.

Il existe d'autres codages, de type N-parmi-M. Mais ceux-ci sont moins étudiés et rarement utilisés du fait de leur complexité et de leur faible efficacité.

Au niveau temporel, on différencie la signalisation par niveau et la signalisation par transition. Pour la première, c'est le niveau de tension des rails qui représente la donnée. Un '1' logique est généralement représenté par une tension haute, un '0' par une tension basse. Mais pour la signalisation par transition, le niveau de tension sur les rails n'a pas d'importance. C'est uniquement la transition d'une tension vers une autre qui signifie le changement de valeur d'une donnée.

L'implémentation matérielle d'un protocole de communication s'appuie sur une convention de signalisation mariant ces différentes techniques d'encodage temporel et spatial. Par exemple, une convention double rail à trois états (figure 1.3b) permettra d'implémenter un protocole 4 phases. Cette convention utilise un codage 1-parmi-2 avec une signalisation par niveau. Les états « 10 » et « 01 » codent respectivement un '1' et un '0'. L'état « 00 » permet de signaler la phase de retour-à-zéro du protocole et l'état « 11 » est un état interdit. La convention double rail à 4 états (figure 1.3c), implémente également un codage 1-parmi-2, mais utilise par contre une signalisation par transition. C'est la transition du premier rail qui signale la présence d'un bit à '1', alors qu'une transition sur le second signale un '0'. Cette convention est bien adaptée à un protocole 2 phases. Dans les circuits

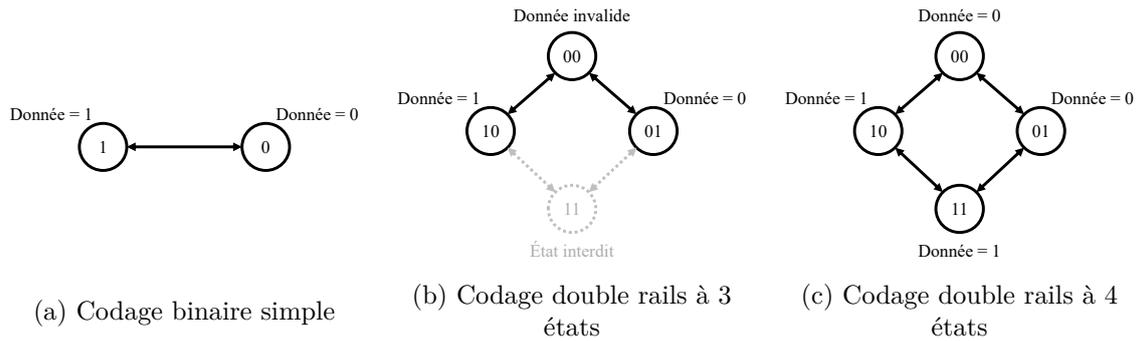


FIGURE 1.3 – Différentes conventions de signalisation.

synchrones, un codage simple rail (figure 1.3a) avec une signalisation par niveau est utilisé pour les données. Une signalisation par transition est généralement utilisée pour le signal d’horloge quand les éléments mémorisants sont des bascules. Une signalisation par niveau est aussi possible si des verrous (*latches*) sont préférés.

1.3 Classification des circuits logiques

La figure 1.4 présente la classification généralement utilisée pour les circuits logiques. Chaque classe est principalement caractérisée par sa convention de signalisation et les exigences sur l’implémentation qui en découlent. Au niveau de la convention, protocole et données peuvent être plus ou moins imbriqués. L’information de validité peut ainsi être intégrée directement dans la manière d’encoder les données, comme c’est le cas dans n’importe quel langage naturel, ou au contraire transportée séparément par un canal externe. Quoiqu’il en soit, pour conserver la cohérence entre les données et le protocole, des exigences plus ou moins importantes sur le séquençement des différents événements sont généralement nécessaires. On parle d’hypothèses temporelles. Celles-ci peuvent être plus ou moins globales suivant qu’elles s’appliquent à tout ou partie du circuit. Selon le niveau d’imbrication et les hypothèses temporelles nécessaires, un circuit logique sera plus ou moins robuste. Il s’adaptera plus ou moins bien aux conditions d’opérations (tension, température).

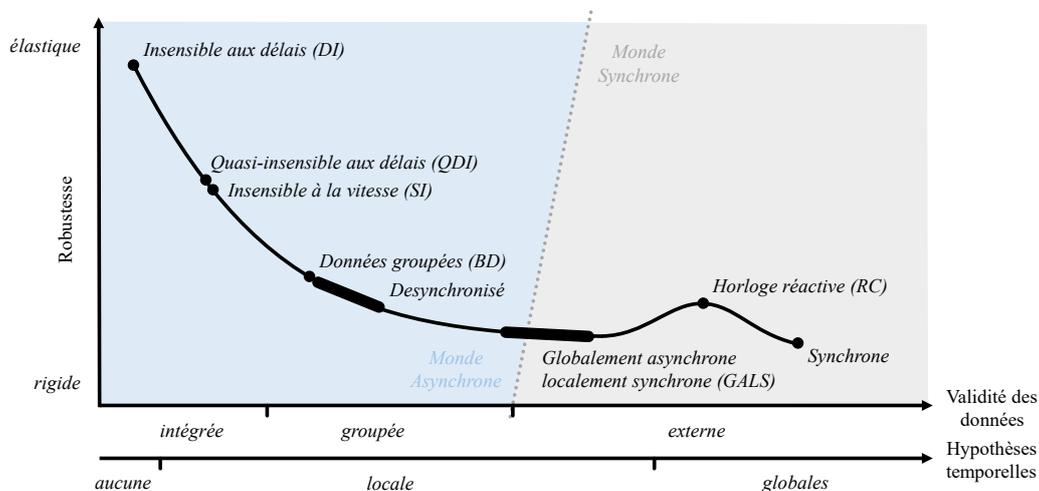


FIGURE 1.4 – Classification des circuits logiques adaptée de [KL02] et de [Sim17].

1.3.1 Circuits synchrones

Le paradigme synchrone, majoritairement adopté, choisit de distinguer physiquement et logiquement les données et le protocole. D'un côté, les données sont généralement codées dans un format binaire simple (simple rail), même si un codage plus complexe n'est pas exclu. De l'autre, un signal externe, l'horloge, sert à valider les données³. Celui-ci ponctue chaque séquence de communication par un seul événement, faisant coïncider en un même instant le début et la fin de différentes phrases : dans un style purement cacophonique⁴, chaque bloc parle en même temps. Pour concevoir ces circuits, on fait l'hypothèse que les délais dans les portes logiques et les interconnexions sont connus et bornés. Ainsi la période de l'horloge pourra être adaptée de manière à ce que les données se soient propagées et que l'ensemble des nœuds du circuit se soient stabilisés avant d'entamer le cycle suivant. Ces hypothèses temporelles fortes permettent de s'affranchir de la considération des aléas dans la logique combinatoire. De ce fait, la conception de fonctions logiques est grandement simplifiée et les circuits synchrones sont très compacts.

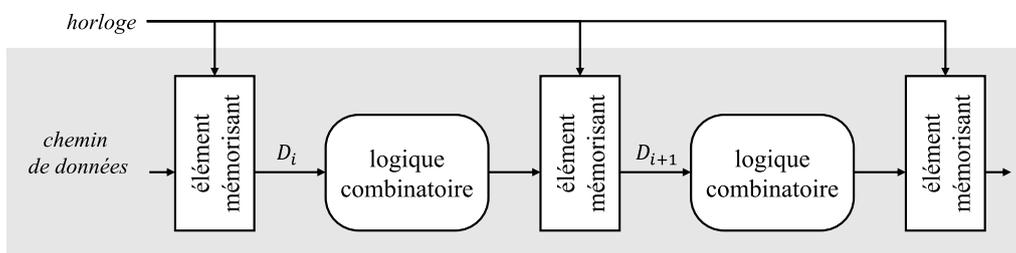


FIGURE 1.5 – Modèle de circuit synchrone.

1.3.1.1 Circuits à horloge réactive

Les circuits à horloge réactive font figure d'exception dans cette classification (figure 1.4). Ils présentent en effet une robustesse améliorée malgré l'utilisation d'une horloge. Ces circuits remplacent le générateur d'horloge fixe (PLL, oscillateur à quartz) par une horloge non-précise et non-stabilisée, dont la fréquence de fonctionnement dépend largement des conditions d'opération. Typiquement, un oscillateur en anneau à base d'inverseur est utilisé (figure 1.6).

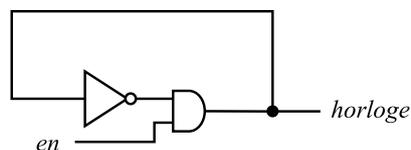


FIGURE 1.6 – Schéma d'un oscillateur en anneau classique.

Les circuits à horloge réactive sont donc bien des circuits synchrones : ils utilisent une hypothèse temporelle globale. Cependant, cette contrainte évolue, tout comme le chemin de données, en fonction des conditions d'opération du circuit : processus de fabrication, tension et température (*Process, Voltage, Temperature*) (PVT). Cela confère à ces circuits une certaine robustesse.

3. Pour l'horloge, l'unique signal de contrôle, une signalisation par transition est généralement utilisée (bascule D). Mais la signalisation par niveau est aussi possible (Verrou D).

4. Nous verrons par la suite que cela peut avoir des avantages !

1.3.1.2 Circuits globalement asynchrones et localement synchrones

Les **GALS** ne forment pas une classe de circuits aux contours bien délimités. Ils sont un assemblage de petits blocs synchrones interconnectés ensemble de manière asynchrone. Cette interconnexion peut avoir comme support une logique synchrone. Dans ce cas, l'asynchronisme est géré au niveau architectural, en général avec des barrières de resynchronisation. Mais cette interconnexion peut aussi utiliser de la logique sans horloge. C'est le cas par exemple lorsqu'elle est assurée par un réseau sur puce asynchrone (*Asynchronous Network on Chip*) (**ANoC**). Ce réseau peut utiliser différents modèles asynchrone (**QDI**, **BD**, etc.). C'est aussi le cas pour le modèle utilisé par chacun des blocs synchrones. Par exemple, FOJTIK *et al.* [Foj+19] proposent un circuit interconnectant de multiples blocs à horloge réactive. Les circuits **GALS** sont ainsi à cheval sur les mondes synchrone et asynchrone. En pratique, ces circuits permettent de limiter les contraintes de distribution du signal d'horloge à de plus petites zones du circuit et d'ainsi limiter l'impact sur les performances. Ils pourront aussi gagner en robustesse suivant le modèle synchrone implémenté. Cependant, la nécessité de resynchronisation entre les différents blocs peut entraîner une augmentation non négligeable de la latence et des communications non sûres (métastabilité).

1.3.2 Circuits insensibles aux délais

A l'autre extrémité de la courbe, les circuits insensibles aux délais (*Delay Insensitive*) (**DI**) forment la classe de circuits asynchrones la plus épurée. Ils utilisent une convention double rail à 4 états sans aucune hypothèse temporelle sur la propagation des signaux (modèle de délais non bornés). Ces circuits sont très robustes [Udd86] car ils fonctionnent quels que soient les délais introduits par les portes logiques et les interconnexions. Cependant, ils sont aussi très complexes à concevoir et sont finalement rarement utilisés. MARTIN [Mar90] prouve en effet que ce type de circuits ne peut être constitué que d'inverseurs et de portes de Muller (figure 1.7) n'existant généralement pas dans les bibliothèques de cellules standard.

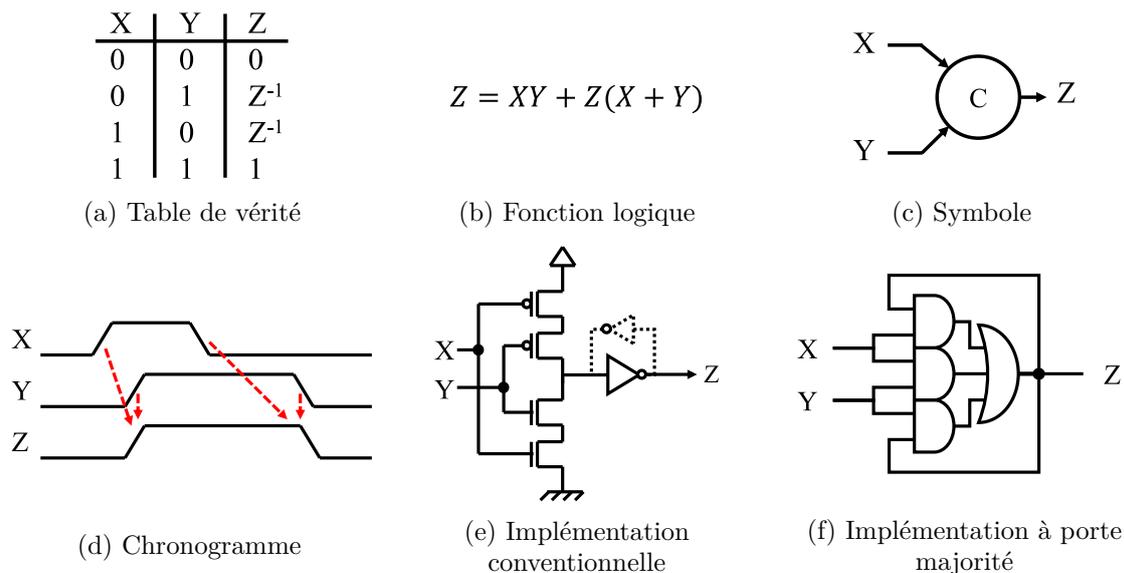


FIGURE 1.7 – La porte de Muller aussi appelée C-element.

1.3.3 Circuits quasi-insensibles aux délais

Les circuits **QDI** ajoutent à la classe **DI** une simple hypothèse temporelle de manière à simplifier leur conception. Afin de garantir le fonctionnement de ces circuits, certaines fourches⁵ doivent alors être isochrones [Mar90] : les temps de propagation sur chacune des branches de la fourche doivent être identiques. En pratique, ces hypothèses sont assez simples à mettre en œuvre et à vérifier. Elles permettent de construire des machines complètes au sens de Turing [MM95] et donc de concevoir n'importe quel système de calcul. La simplicité de cette hypothèse temporelle rend les circuits **QDI** très robustes. Ils restent cependant sensibles aux aléas qui pourraient être injectés (volontairement ou non) dans le circuit et qui peuvent conduire à un blocage ou à une erreur.

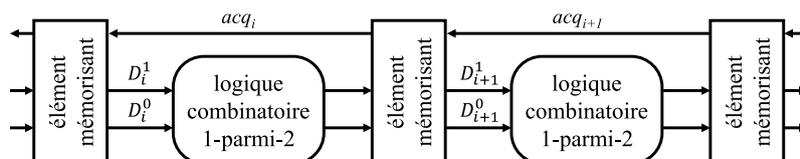


FIGURE 1.8 – Modèle de circuit **QDI**.

Les circuits **QDI** adoptent souvent une convention double rail à 3 états (figure 1.8). Les éléments mémorisant du circuit détectent l'arrivée d'une nouvelle donnée lorsque l'un des rails passe au niveau haut. Ils renvoient alors un acquittement pour signifier la bonne réception. Le cycle de **RZ** permet ensuite de remettre le canal dans son état initial. Malgré l'utilisation d'un état de **RZ**, et donc d'un plus grand nombre de transitions par cycle, la signalisation par niveau est plus simple à implémenter que la signalisation par transition des circuits **DI** et les circuits qui en découlent sont plus petits. Le modèle des indépendants de la vitesse (*Speed Independent*) (**SI**) est très proche de celui des circuits **QDI**. Il considère que les délais dans les interconnexions sont nuls ou identiques et donc, que toutes les fourches sont isochrones. Cette hypothèse est plus difficile à vérifier et n'est pas appropriée aux technologies avancées pour lesquelles le délai induit par les fils devient prépondérant par rapport à celui induit par les cellules logiques. Le modèle **QDI** est ainsi préféré, même si en pratique ces deux modèles de circuit conduisent à des implémentations très similaires.

1.3.4 Circuits asynchrones à données groupées

Les circuits à données groupées (*Bundled-data*) (**BD**) séparent physiquement le contrôle et les données. Ces signaux sont transportés parallèlement et simultanément sur deux supports distincts. Comme illustré par la figure 1.9, on peut donc distinguer deux parties dans ces circuits : la partie chemin de données et la partie chemin de contrôle.

La première est très similaire à un circuit synchrone. Elle est constituée de blocs logiques combinatoires délimités par des éléments séquentiels. Elle utilise le même modèle de délais bornés. On considère souvent que ce chemin de données est un circuit synchrone auquel l'horloge a été enlevée. Il n'est d'ailleurs pas rare que cette partie soit utilisée successivement de manière synchrone et asynchrone⁶.

À l'inverse, la partie contrôle est la plupart du temps un circuit **QDI** voire **DI**. Elle implémente un protocole de poignée de main entre différents contrôleurs de registres qui la constituent. Ceux-ci génèrent les signaux de validité qui contrôlent les registres, permettant ainsi de cadencer la propagation des données à travers le pipeline.

5. Une interconnexion entre la sortie d'un élément du circuit et les entrées d'au moins deux autres éléments.

6. Par exemple, le HT80C51, un micro-contrôleur asynchrone de la société Handshake Solution, propose un mode synchrone (http://www.keil.com/dd/docs/datashts/handshake/ht80c51_um.pdf).

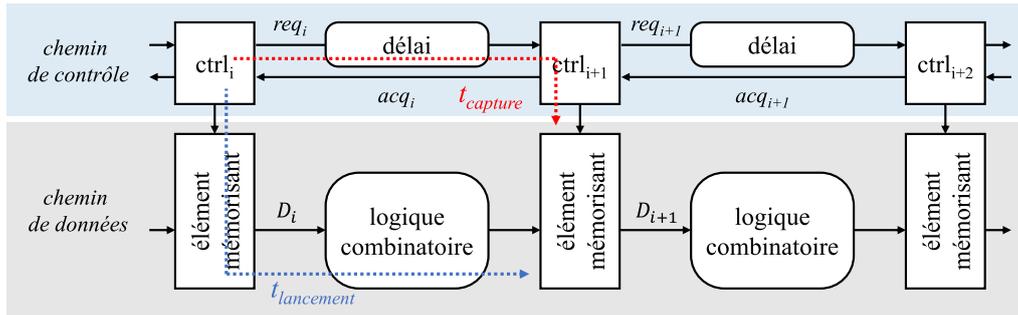


FIGURE 1.9 – Modèle de circuit asynchrone à données groupées.

Pour conserver la cohérence entre les données et les signaux de contrôle, il faut s'assurer qu'ils se propagent à la même vitesse sur leurs chemins respectifs. Ainsi une hypothèse temporelle supplémentaire est indispensable : le temps de propagation des données ($t_{\text{lancement}}$) doit être plus court que le temps de propagation des signaux de contrôle (t_{capture}). De manière à tenir cette contrainte, un élément de retard, aussi appelé délai *matché*, est ajouté entre chaque contrôleur avec un temps de propagation équivalent à la logique combinatoire correspondante.

1.3.4.1 Protocoles à données groupées

Il existe de nombreux protocoles adaptés aux circuits **BDs**. D'un côté, les protocoles 2 phases sont considérés rapides et efficaces grâce à leur poignée de main ne nécessitant que deux transitions : un événement de requête et un événement d'acquiescement. Cependant, ils requièrent un mécanisme de conversion de phase pour transformer les fronts montants et descendants de ces événements en un signal capable d'activer les éléments séquentiels du chemin de données. Ces mécanismes génèrent des chemins temporels et des boucles combinatoires qui rendent l'analyse temporelle statique (*Static Timing Analysis*) (**STA**) plus délicate. D'un autre côté, les protocoles 4 phases utilisent une séquence de **RZ** pour retrouver leur état initial à la fin de la poignée de main. Quatre transitions sont donc nécessaires pour effectuer un cycle de communication complet : deux événements sur la requête et deux événements sur l'acquiescement. Ces protocoles apparaissent donc plus lents et moins efficaces que leurs homologues 2 phases. Ils utilisent cependant des structures de branchements (fourches, démultiplexeurs, multiplexeurs, etc.) plus simples et la phase **RZ** peut être exploitée pour réduire la consommation et la surface de ces protocoles [Sim+16].

La figure 1.10 donne l'implémentation typique d'un contrôleur de registre pour 7 protocoles différents. A titre de comparaison, il donne aussi l'implémentation d'un circuit synchrone à base d'horloge réactive. Dans un souci de généricité et d'adéquation à un contexte industriel, chacune de ces primitives est construite avec des cellules logiques standards. A chaque fois qu'une porte de Muller est requise, nous utilisons donc l'implémentation de type « porte majorité » (figure 1.7f). Trois protocoles 2 phases très connus sont présentés : *Micropipeline*, *Mousetrap* et *Click*.

En 1989, SUTHERLAND [Sut89] proposait le tout premier protocole à données groupées : le *Micropipeline*. Ce protocole est très simple. Il utilise une unique porte de Muller (figure 1.10b) pour resynchroniser les signaux de requête et d'acquiescement de ses canaux d'entrée et de sortie. Comme des registres fonctionnant sur les deux fronts d'horloge (*dual-edge flip-flop*) sont rarement disponibles dans les bibliothèques de cellules standard, nous utilisons un générateur d'impulsions à base d'une porte OU-exclusif (**XOR**) pour faire la

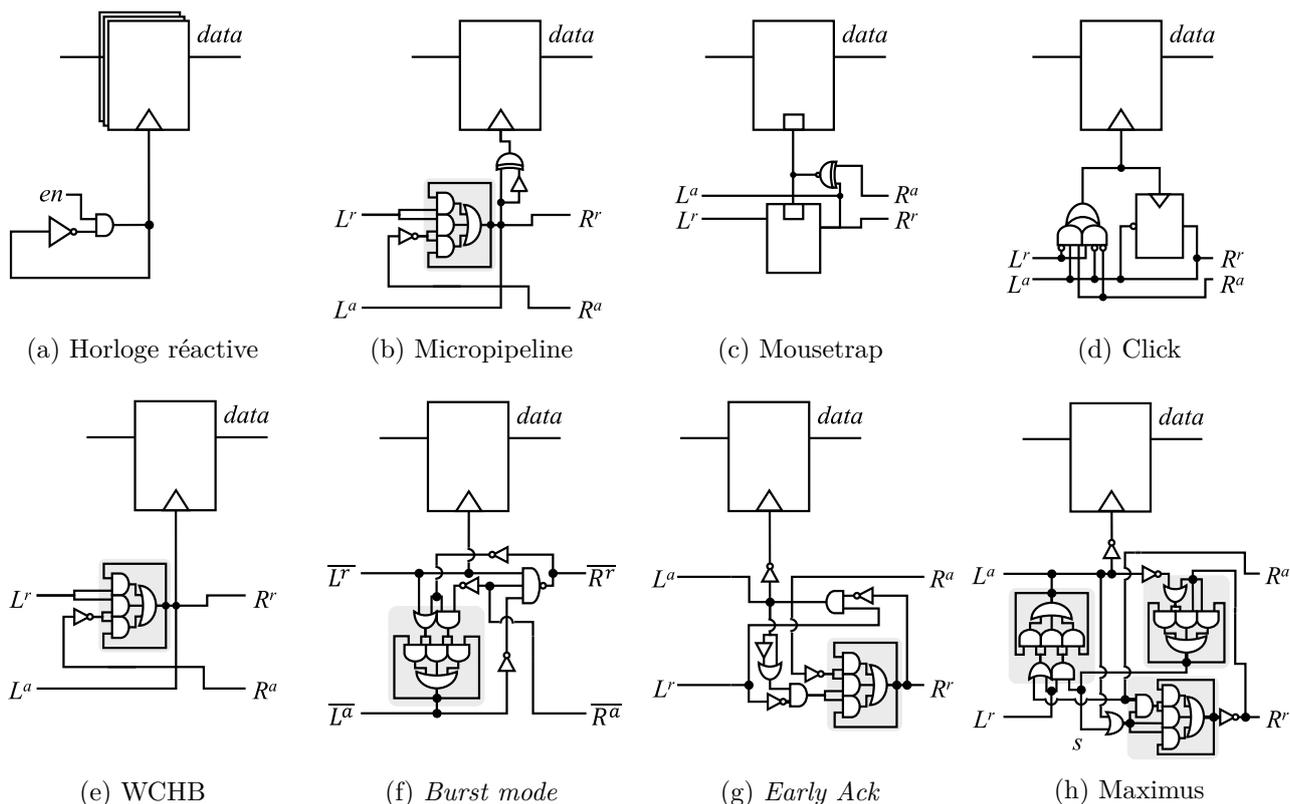


FIGURE 1.10 – Implémentations typiques de différents schémas de circuits.

conversion de phase et ainsi contrôler de simples registres⁷.

Une dizaine d'années plus tard, SINGH et NOWICK [SN01] présentaient le schéma *Mousetrap* (figure 1.10c). Leur objectif était alors de créer des pipelines asynchrones avec un débit plus important tout en conservant une très faible empreinte. Ils remplacent alors la porte de Muller du contrôleur et l'élément mémorisant du chemin de données par des verrous D, initialement ouverts, qui permettent aux données et aux signaux de requête de se propager très rapidement et sans discontinuité dans le circuit. Données et requêtes sont piégées par les contrôleurs – à la manière d'un piège à souris – dès qu'elles ont traversé leur verrou respectif. L'étage reste alors bloqué jusqu'à ce que le contrôleur suivant ait à son tour basculé. La conversion de phase est aussi assurée par une porte XOR qui détecte le statut du canal de sortie⁸ et ferme ou ouvre le verrou en conséquence.

Une autre dizaine d'années plus tard, PEETERS *et al.* [Pee+10] soumettaient à la communauté un nouveau modèle, baptisé *Click* (figure 1.10d), dans le but de simplifier l'analyse temporelle et d'améliorer la testabilité des circuits à données groupées. Cette primitive n'utilise aucune porte logique exotique et l'état interne du protocole est stocké dans un simple registre. Ainsi, ce contrôleur peut être testé avec les méthodes de *Design For Test (DFT)* habituelles. De plus, le registre casse les boucles combinatoires internes du contrôleur qui sinon perturbent les outils de *STA*. La conversion de phase est faite à l'aide de deux portes XOR, chacune d'elle permettant de détecter l'état du canal d'entrée

7. Des verrous sont employés dans l'implémentation proposée par SUTHERLAND [Sut89]. Cependant les concepteurs sont généralement réticents à les utiliser. Nous préférons donc l'utilisation de registres classiques.

8. Première phase du protocole lorsque requête et acquittement ont des valeurs différentes, deuxième phase dans le cas contraire.

ou du canal de sortie. Bien sûr, ces avantages sont apportés au prix d'une taille et d'une consommation plus importante.

La figure 1.10 présente également quatre protocoles 4 phases : *Weak Condition Half-Buffer* (WCHB), *Burst mode*, *Early acknowledgment* et *Maximus*.

Le protocole WCHB [FD96] (figure 1.10e) est identique à un schéma *Micropipeline* pour lequel le bloc de conversion de phase aurait été enlevé. L'implémentation en cellules standard de ce protocole est donc très petite, mais elle présente une limitation majeure : seule la moitié des étages d'un pipeline utilisant ce schéma peuvent simultanément contenir une donnée. En d'autres termes, les données sont toujours espacées par un contrôleur. Le terme *half-buffer* dans le nom de ce protocole retranscrit cette spécificité⁹.

Les schémas *Burst-mode* [YBA96] (figure 1.10f) et *Early acknowledgment* [MY08] (figure 1.10g) implémentent tous deux un protocole à capture tardive. La donnée est mémorisée sur le front descendant de la requête du canal d'entrée. Ce type de protocole permet de réduire la taille du délai *matché*, car celui-ci est parcouru deux fois avant que les données ne deviennent valides. De plus, le protocole *Early acknowledgment* utilise un délai asymétrique de manière à accélérer la phase de retour-à-zéro. Il autorise ainsi un temps de cycle très proche du pire chemin de la partie chemin de données.

Enfin, le protocole *Maximus* [Sim+16] (figure 1.10h) est aussi un protocole tardif. Il a été conçu pour minimiser les phases de RZ sans avoir recours à un délai asymétrique. Il en résulte le schéma le plus complexe de tous ceux présentés (trois portes de Muller sont nécessaires), mais qui est particulièrement efficace lorsque le chemin de données est conséquent (la complexité du contrôleur est alors masquée par le gain associé aux délais *matchés* réduits).

1.3.5 Circuits désynchronisés

Les circuits désynchronisés forment une large classe qui regroupe l'ensemble des circuits asynchrones construits à partir d'une description synchrone. Ce processus de transformation est appelé désynchronisation. De nombreuses méthodes existent. Elles ont toutes en commun la volonté de rapprocher les mondes synchrone et asynchrone et d'ainsi permettre à un concepteur lambda¹⁰ de développer un circuit sans horloge sans connaissances *a priori* sur le monde asynchrone. En pratique, ces méthodes aboutissent toutes à des circuits proches du modèle à données groupées : elles réutilisent le chemin de données synchrone, auquel elles associent un bloc de contrôle asynchrone. La différence entre ces méthodes se situe essentiellement dans la granularité du pipeline obtenu.

La méthode proposée par CORTADELLA *et al.* [Cor+06] (figure 1.11) est certainement la plus connue. Elle consiste à : 1) substituer les registres du chemin de données synchrone par une paire de verrous, maître et esclave, 2) optimiser le chemin de données en appliquant les techniques de *retiming*, 3) construire le circuit de contrôle correspondant au chemin de données, et finalement 4) insérer les délais correspondants entre chaque contrôleur. Cette technique permet d'obtenir très facilement un circuit asynchrone. Cependant la construction du bloc de contrôle s'avère fastidieuse dans la pratique et passe donc difficilement à l'échelle. Des techniques, comme celle présentée par BERTRAND *et al.* [Ber+17], ont été proposées pour automatiser cette tâche. Mais ces techniques de désynchronisation pèchent plus fondamentalement par le niveau de granularité, au niveau registre, qu'elles adoptent. Celui-ci engendre une redondance fonctionnelle. Par exemple, une machine à états finie (*Finite State Machine*) (FSM) à encodage *one-hot* avec dix états sera contrôlée par un bloc

9. Un protocole *full-buffer* peut être facilement construit en assemblant deux contrôleurs WCHB. Mais cela vient au détriment de la taille et des performances. Voir à ce sujet [MBM91].

10. C'est-à-dire, un concepteur synchrone.

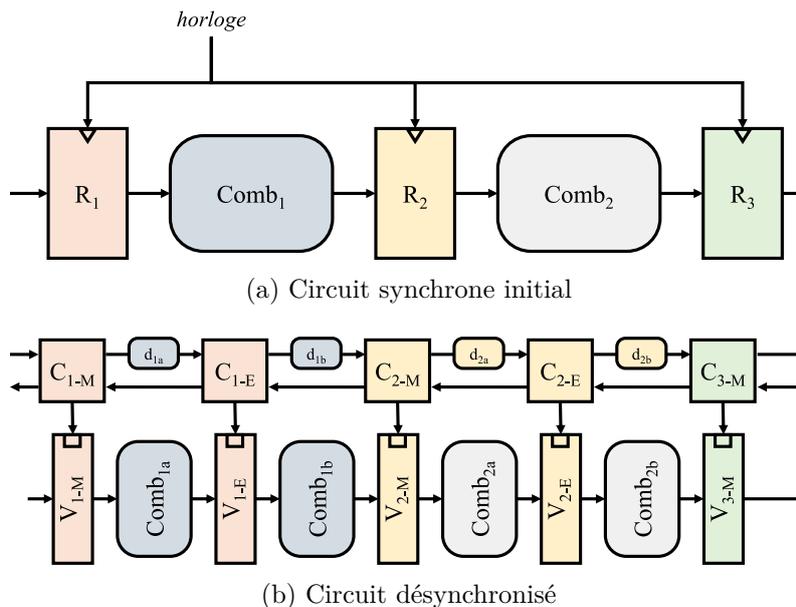


FIGURE 1.11 – Méthode de désynchronisation de CORTADELLA *et al.*

composé de vingt contrôleurs. L'incrément de surface sera donc nécessairement important. Mais surtout, l'état courant de la FSM sera stocké deux fois : une fois dans les registres et une fois dans les contrôleurs.

Pour s'affranchir de cette redondance, des méthodes de désynchronisation à base de machine à états asynchrones (*Asynchronous Finite State Machine*) (AFSM) ont été proposées. Par exemple, SIMATIC [Sim17] propose de remplacer la FSM obtenue en sortie d'un outil de synthèse de haut niveau (*High-Level Synthesis*) (HLS) par une AFSM équivalente. Cette méthode, qualifiée de synthèse de haut niveau asynchrone (*High-Level Synthesis*) (AHL), adopte une granularité plus élevée et supprime les redondances. Elle utilise par contre des contrôleurs complexes et les nombreuses interactions entre chemin de données et bloc de contrôle rendent complexe l'analyse temporelle de ces circuits.

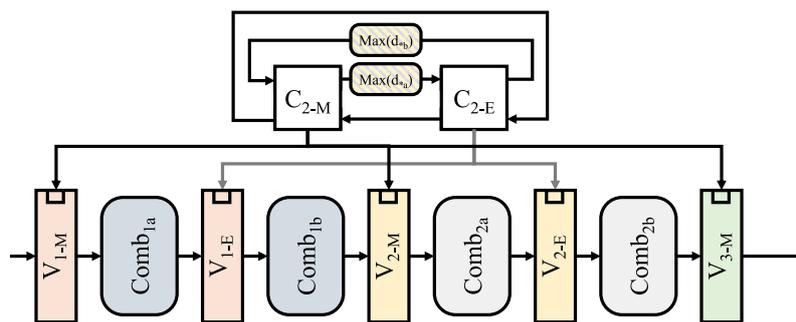


FIGURE 1.12 – Modèle de désynchronisation de la méthode *Edge*

A l'extrême, la méthodologie *Edge* (figure 1.12), présentée dans ZHANG *et al.* [Zha+18], adopte la plus grosse granularité : le circuit synchrone est découpé en deux blocs, chacun dirigé par un contrôleur dédié. Dans un sens, cette approche est en fait très similaire à celle proposée par CORTADELLA *et al.* [Cor+06]. Elle utilise la même technique de transformation des registres en verrous. Les deux blocs du circuit sont respectivement composés des verrous maîtres et des verrous esclaves. Les contrôleurs s'échangent un jeton qui va tour

à tour activer l'un et l'autre de ces groupes, permettant ainsi aux données de se propager dans le circuit. Grâce au protocole de communication utilisé entre les deux contrôleurs, les phases de requête et d'acquiescement sont découplées. Cela donne une liberté supplémentaire pour contrôler le recouvrement entre les signaux de contrôle et ainsi corriger plus facilement des erreurs de violation de temps de maintien. D'un autre point de vue, cette méthodologie génère des circuits très proches du modèle synchrone et plus particulièrement de celui à horloge réactive. Le contrôleur peut être vu comme un oscillateur générant deux horloges réactives alimentant deux blocs synchrones distincts. Ce flot de conception a l'avantage d'être entièrement automatisé¹¹, s'affranchissant du problème de construction du bloc de contrôle. Cela vient cependant au détriment d'une distribution d'horloge légèrement plus complexe : les signaux d'activation des verrous doivent être distribués sur deux arbres d'horloge différents, mais entremêlés.

1.4 Atouts des circuits asynchrones

Malgré leur grande diversité, les circuits asynchrones ont tous un point en commun : ils n'utilisent pas d'horloge. Cette spécificité leur confère de nombreux avantages.

1.4.1 Performances accrues

Dans un circuit synchrone, l'horloge dicte la même vitesse de fonctionnement pour chacun des étages, indépendamment de leur rapidité effective. En effet, pour que chaque étage puisse communiquer simultanément, il est nécessaire de s'adapter au plus lent de tous. Ainsi les étages rapides passent la majorité de leur temps à attendre de nouvelles données. Remplacer l'horloge par des mécanismes de synchronisation locaux permet à chaque étage d'un circuit asynchrone d'adopter une vitesse de fonctionnement différente. Les calculs sont faits en temps minimal et les données se propagent à leur rythme propre. Grâce à ce principe de fonctionnement, les circuits asynchrones ont une latence grandement améliorée. Ils présentent aussi un débit variable, dépendant des données, qui sera en moyenne plus important que leur homologue synchrone.

Même si le protocole synchrone est simple dans son principe, en pratique, la génération et la distribution uniforme sur toute la surface d'un circuit d'un signal d'horloge peut s'avérer extrêmement difficile. Alors que les technologies deviennent de plus en plus fines, les variations à l'intérieur du circuit (*On-Chip Variations*) (OCV) du processus de fabrication sont de plus en plus difficiles à maîtriser et il devient toujours plus compliqué de s'assurer que l'horloge atteigne l'ensemble des registres du circuit simultanément. L'augmentation des fréquences de fonctionnement et la réduction des tensions d'alimentation rendent ce problème de plus en plus criant. L'ajout de marges temporelles pour compenser ces problèmes provoque invariablement une diminution des performances. En s'affranchissant d'un signal d'horloge global, les circuits asynchrones conservent uniquement des hypothèses temporelles locales qui sont moins impactées par l'OCV. Ils peuvent donc prétendre à utiliser moins de marge et à fonctionner plus rapidement.

1.4.2 Consommation réduite

La surconsommation liée au signal d'horloge est un problème connu. En 1997, SAKURAI, KAWAGUCHI et KURODA [SKK97] mentionnaient déjà une empreinte entre 20 % et 45 % du système de distribution de l'horloge. La réduction des dimensions et l'intégration d'un

11. Le code source est disponible sur Github : <https://github.com/nobodybutyou1/Edge/tree/master>

nombre croissant de fonctionnalités dans un même circuit ont depuis accentué ce phénomène. En s'affranchissant du générateur d'horloge (*Phase-Locked Loop* (PLL), oscillateurs...) et d'un arbre d'horloge global, les circuits asynchrones peuvent espérer des gains substantiels en terme de consommation.

La suppression de l'horloge se traduit aussi par une simplification de la gestion des ressources communes. Dans les circuits synchrones les changements de mode de fonctionnement (sommeil profond, veille, mode dégradé, plein régime, etc.) demandent de respecter un séquençement rigoureux entre les modifications des consignes d'alimentation et celles de la fréquence d'horloge. Sans cela, les hypothèses temporelles peuvent être invalidées, entraînant un mauvais fonctionnement du circuit. Dans les circuits asynchrones, en l'absence d'horloge, seule l'alimentation est partagée par tous les éléments du circuit. Les changements de mode d'opération sont donc simplement dictés par la modification de la tension d'alimentation, la vitesse du circuit s'adaptant automatiquement. En plus de simplifier la logique associée à la gestion de ces modes, ce principe permet des changements beaucoup plus rapides, à la volée, et sans attendre la stabilisation de la tension ou de l'horloge. Pour des systèmes au fonctionnement très sporadique, les gains en consommation peuvent être très conséquents.

Par ailleurs, dans l'approche synchrone, données et signaux de contrôle sont dissociés. Ainsi les communications entre les différents éléments du circuit se font indépendamment de la nécessité de transmettre de nouvelles données. En conservant un lien logique entre données et protocole, les circuits asynchrones évitent nativement tous les échanges superflus et sont donc naturellement moins consommateurs d'énergie. Même si des techniques de *clock-gating* ont vu le jour dans les circuits synchrones pour recréer ce lien, leur implémentation n'est pas parfaite. Elle ne permet pas d'atteindre la même granularité et les mêmes gains en consommation.

1.4.3 Bruit et émissions électromagnétiques restreintes

Le fonctionnement synchrone tend à regrouper l'ensemble de la consommation autour des fronts d'horloges. Les appels de courant lors de ces événements sont ainsi très importants. Ils génèrent du bruit sur les lignes d'alimentation et le substrat, et provoquent des rayonnements électromagnétiques conséquents pouvant être très perturbants pour les circuits avoisinants. Les circuits analogiques et radio-fréquences sont particulièrement sensibles à ces perturbations. Au contraire, dans les circuits asynchrones, les appels de courant sont répartis dans le temps. Le bruit et le rayonnement générés sont donc moindres, rendant ces circuits particulièrement adaptés à la manipulation de signaux de très faible puissance.

1.4.4 Modularité optimale

Les circuits asynchrones sont très modulaires. Alors que les problèmes de synchronisation et de convergence temporelle¹² sont récurrents lors de l'assemblage de blocs synchrones, les circuits asynchrones ne nécessitent pas de précaution particulière. La resynchronisation est faite naturellement à partir du moment où le protocole de communication est respecté et les contraintes temporelles peuvent être résolues localement sans impacter le contenu des différents blocs. Ainsi un bloc asynchrone pourra facilement être réutilisé d'un circuit à un autre sans remettre en question son implémentation physique. Il est aussi

12. La convergence temporelle fait référence au processus itératif de budgétisation des contraintes temporelles aux interfaces entre les différents blocs d'un circuit et de résolution des violations qui en découlent.

plus facile de découper un gros circuit en plusieurs petits blocs hiérarchiques, qui pourront être implémentés séparément, réduisant ainsi le temps de développement.

1.4.5 Sécurité améliorée

L'horloge est un canal privilégié par les personnes mal intentionnées pour attaquer un circuit synchrone. Ce signal donne accès direct à l'ensemble des éléments de mémoires du circuit. Il permet d'injecter facilement et à moindre coût des fautes pour corrompre ou détourner le fonctionnement normal du circuit et ainsi révéler les secrets qu'il manipule. En supprimant ce signal global, les circuits asynchrones réduisent leurs expositions aux attaques.

Plus généralement, la grande majorité des attaques physiques sur un circuit nécessitent d'être synchronisées avec les données qu'elles cherchent à récupérer. En l'absence d'un signal de synchronisation global, ces attaques sont bien plus difficiles à mettre en œuvre. Par exemple, une attaque par analyse de consommation superpose de multiples traces pour alimenter un algorithme recherchant des corrélations. Si les traces ne sont pas synchronisées, l'attaque sera bien plus longue et aura une issue incertaine. En s'appuyant sur la robustesse accrue des circuits asynchrones, il est aussi plus facile de construire des contre-mesures contre ce genre d'attaque. Il est notamment possible d'ajouter du bruit sur l'alimentation pour rendre aléatoire toute périodicité résiduelle qui permettrait de se synchroniser.

Les circuits asynchrones utilisant un encodage des données double rails sont aussi intrinsèquement plus résistants aux attaques par canaux auxiliaires. Ces codes permettent de garder un poids de *Hamming* constant, limitant ainsi la dépendance entre la consommation et les données. Les protocoles 4 phases sont particulièrement intéressants car la phase de **RZ** permet aussi de dissocier deux calculs successifs et réduit toujours plus le lien entre données et consommation.

De par leur sensibilité aux aléas, les circuits asynchrones peuvent paraître plus sensibles aux attaques par injections de fautes par laser ou sonde électromagnétique. Mais cette sensibilité peut aussi être utilisée pour construire des détecteurs efficaces [Mon07]. Dans un circuit utilisant un protocole 4 phases, il est par exemple très probable qu'une faute mette à défaut un contrôleur, le plaçant dans l'état '11'. Cet état étant invalide, une alarme peut être générée lorsqu'il est détecté et les mesures nécessaires pourront être prises au niveau système.

1.5 Modélisation des circuits asynchrones

Les graphes de transitions de signaux (*Signal Transition Graphs*) (**STG**) ont été introduits pour modéliser et analyser les circuits asynchrones [Chu87]. Les **STGs** sont une classe de réseaux de Petri [Pet66] pour lesquels les transitions labélisées $X+$ et $X-$ représentent respectivement l'occurrence d'un front montant et d'un front descendant du signal X . Deux transitions peuvent être liées par une flèche représentant une contrainte d'ordonnancement relative sur l'occurrence de ces transitions. Des arcs pointillés sont aussi utilisés pour représenter les conditions imposées par l'environnement du circuit. L'utilisation de **STG** n'est pas appropriée à la description d'opérations arithmétiques. De plus, ces graphes tendent à avoir de grandes tailles ne facilitant pas leur utilisation. Ils sont cependant bien appropriés à la description des protocoles et des primitives et permettent d'identifier visuellement les chemins et boucles combinatoires.

Les langages de description au niveau transactionnel (*Transaction Level Model*) (**TLM**) sont plus à même de décrire des opérateurs de calcul. Ils permettent aussi d'abstraire, sous la forme d'un canal, les considérations liées à l'implémentation bas niveau du protocole.

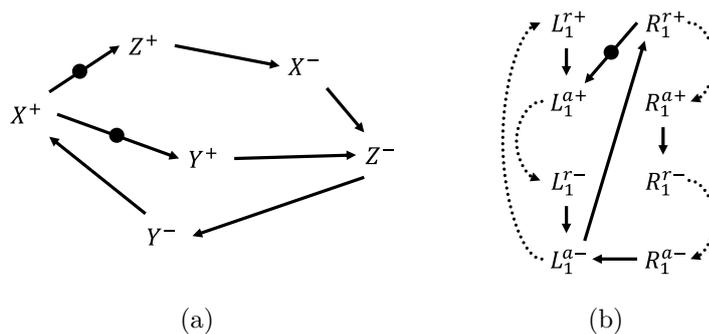
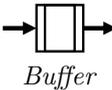


FIGURE 1.13 – Exemples de STG décrivant (a) un circuit quelconque et (b) le protocole Maximus.

TABLE 1.1 – Différentes primitives habituellement utilisées, adapté de [Sim17]

Type de synchronisation	Mémorisation	Non-conditionnelle	Conditionnelle	Non déterministe
1 vers 1	 Buffer			
1 vers N		 Fourche	 Démultiplexeur	
N vers 1		 Union	 Multiplexeur	 Arbitre

Ainsi, des langages comme le *Communicating Sequential Processes (CSP)*, le *Communicating Hardware Processes (CHP)* ou bien les langages *SystemVerilog* et *SystemC* sont bien adaptés à la description de circuits asynchrones complexes.

1.6 Conception de circuits asynchrones

Les circuits asynchrones bénéficient aujourd’hui de nombreuses méthodes de conception. Celles-ci peuvent être classifiées en deux groupes. Le premier regroupe les méthodes de conception directe. Un modèle formel, généralement à base de réseau de Petri ou de STG, est directement construit selon l’architecture du chemin de données. Ce modèle permet de vérifier les propriétés nécessaires au bon fonctionnement du circuit. Par la suite, le circuit est déduit de ce modèle, soit par association directe [Hol82 ; Kis+94], soit par synthèse [Cor+19]. Ces techniques ne conviennent pas pour décrire des systèmes complets car la taille des graphes d’état à manipuler croît de manière exponentielle.

A contrario, le deuxième groupe est constitué des approches de construction des circuits par assemblage de blocs de base. Ces blocs, aussi appelés primitives, réalisent des fonctions élémentaires de synchronisation [FB06 ; Han+15 ; Sim+17] (tableau 1.1). Un modèle formel peut être dérivé de cet assemblage en substituant chaque primitive par son modèle équivalent. L’ensemble ainsi formé permet de réaliser les vérifications garantissant le bon fonctionnement du circuit. Ces approches par primitives choisissent de ne pas synthétiser le modèle formel. Ce faisant, elles perdent en capacité d’optimisation, mais peuvent par

là même traiter des circuits de grandes tailles.

Toutes ces méthodes permettent de spécifier et de vérifier la fonctionnalité de circuits asynchrones, aussi bien au niveau architectural qu’au niveau des protocoles de communication. Cependant, elles ne permettent pas de garantir que cette fonctionnalité soit préservée par les différentes étapes d’implémentation physique. Il faut pour cela s’assurer de la bonne implémentation des hypothèses temporelles.

1.7 Implémentation des circuits asynchrones

1.7.1 Contrainte de temps relatifs

Une contrainte de temps relatifs (*Relative Timing Constraint*) (**RTC**) est un formalisme permettant d’exprimer une exigence temporelle s’appliquant sur un circuit. Elle définit trois points de contrôles [SGR99] : 1) un point de divergence *pod*, 2) un point de convergence précoce *poc₀*, et 3) un point de convergence tardif *poc₁*.

$$pod \mapsto poc_0 \prec poc_1 - \delta \tag{1.1}$$

$$\tag{1.2}$$

L’équation (1.1) décrit une **RTC** générique. Elle doit être lue : « Lorsqu’un événement (front montant ou descendant) se produit au point de divergence *pod*, ses conséquences doivent atteindre le point de convergence précoce *poc₀* avant d’atteindre le point de convergence tardif *poc₁* avec une marge δ ».

1.7.2 Analyse temporelle des circuits asynchrones

A l’instar de leurs homologues synchrones, les circuits asynchrones reposent sur des hypothèses temporelles pour assurer le bon fonctionnement de leur protocole de communication. L’analyse temporelle est le processus de définition et de vérification de ces exigences. On différencie généralement l’analyse temporelle statique (*Static Timing Analysis*) (**STA**) des analyses dynamiques. La première analyse un circuit indépendamment de sa fonctionnalité ou de données appliquées à ses entrées. Elle décompose le circuit en différents chemins et vérifie de manière exhaustive que chacun de ces chemins respecte les exigences qui lui sont appliquées. Elle permet de détecter les chemins critiques et de définir ainsi les performances d’un circuit. Les analyses dynamiques sont quant à elles dépendantes de stimuli. Elles viennent simuler le fonctionnement du circuit lorsqu’un vecteur de test est appliqué. Elles sont donc par nature plus lentes et ne peuvent généralement pas prétendre à l’exhaustivité de la **STA** : la stimulation de tous les chemins d’un circuit complet demanderait un nombre considérable de vecteurs et un temps de simulation prohibitif.

Aujourd’hui, les flots de conception logique s’appuient sur la **STA** pour concevoir et optimiser les performances de circuits synchrones toujours plus complexes. Ainsi, un moteur de **STA** est intégré à tous les outils d’implémentation physique (synthèse, placement-routage) de manière à prendre en compte les exigences temporelles à toutes les étapes de la conception. Un langage spécifique, le format *Synopsys Design Constraints* (**SDC**), est majoritairement utilisé par ces outils pour décrire les exigences temporelles. Il permet de définir des horloges (`create_clock`, `create_generated_clock`) et donc la fréquence de fonctionnement d’un circuit, des points d’arrivées des chemins temporels (`set_data_check`, `set_clock_gating_check`), des contraintes de temps maximal et minimal (`set_min_delay`, `set_max_delay`) ainsi que toutes sortes d’exceptions permettant de décrire précisément les relations entre les différents chemins (`set_false_path`, `set_multicycle_path`, `set_case_analysis`, etc.). Pour simplifier leur analyse, les outils de **STA** s’appuient

aussi sur des modèles d'abstraction décrivant le comportement temporel des différentes cellules constituant un circuit. Ils décrivent par exemple les temps de propagation au travers des différentes cellules en fonction du temps de transition des signaux d'entrée et de la charge vue par les sorties. Ces modèles définissent aussi des arcs séquentiels, spécifiant les temps de *setup* et de *hold*, les temps pour lesquels une donnée doit être stable en entrée d'un élément mémorisant, respectivement avant et après l'arrivée d'un front actif de l'horloge. Le modèle *Liberty* de Synopsys est le modèle d'abstraction temporelle généralement utilisé.

Cependant, l'analyse temporelle des circuits asynchrones n'est pas triviale et les **RTCs** qui soutiennent la cohérence des données de ces circuits sont difficilement définissables pour les outils de **CAO** classiques. Différentes approches ont alors été proposées pour réaliser cette analyse. Le cadre proposé dans [MS16] évalue les **RTCs** selon un critère de robustesse et les classe de manière à résoudre les conflits au sein d'un même ensemble de contraintes. Dans [MHR02; CYD98], des modèles de haut-niveau sont proposés pour vérifier les contraintes temporelles des circuits asynchrones. La plupart des autres travaux s'appuient sur des outils de **CAO** classiques [SXV09; GMC15]. Ils utilisent à chaque fois des contraintes temporelles partielles, `set_min_delay` et `set_max_delay`, pour contraindre à la fois le chemin de données et le chemin de contrôle des circuits à données groupées. Dans [And+07], des horloges virtuelles sont aussi utilisées pour contraindre le chemin de données, mais les commandes de `set_min_delay` sont conservées sur la partie contrôleur. Enfin, PRAKASH et BEEREL [PB15] propose une autre méthodologie utilisant des contraintes de `set_data_check` pour contraindre les fourches isochrones des circuits **QDI**, mais cette approche n'est pas vraiment appropriée aux circuits à données groupées.

1.7.3 Performances et validité fonctionnement

Dans les circuits synchrones, la performance et la validité de fonctionnement dépendent toutes deux d'une seule et même contrainte, la période de l'horloge. Cette contrainte définit un temps de cycle fixe pour chaque étage du circuit et détermine ainsi les performances du circuit. Ainsi, en ne modifiant qu'une seule contrainte, il est aisé de trouver un point d'équilibre entre la performance optimale et la validité de fonctionnement. Ce n'est pas le cas pour les circuits asynchrones. Avec le modèle **QDI**, par exemple, le fait de respecter l'hypothèse de fourches isochrones garantit seulement le respect du protocole. Les performances découlent, elles, de la somme des délais des portes et des interconnexions. Or, ces délais sont largement impactés par les étapes de synthèse, de placement ou de routage. Lors de l'implémentation physique, il est donc généralement nécessaire d'ajouter des contraintes de manière à garantir une performance minimale. Ainsi deux classes d'exigences coexistent :

- 1) Les contraintes de fonctionnement qui garantissent la validité des hypothèses temporelles faites par le protocole. Ce sont des contraintes de temps de propagation minimal et/ou maximal qui s'appliquent plus particulièrement sur les chemins de contrôle (délais *matchés* pour les circuits **BD**, chemin d'acquiescement pour les circuits **QDI**).
- 2) Les contraintes de performance qui garantissent un certain débit et une certaine latence pour le circuit. Ce sont des contraintes de temps de propagation maximal qui s'appliquent sur le chemin de données mais aussi sur le chemin de contrôle.

La coïncidence de ces deux types d'exigences peut engendrer des conflits difficiles à résoudre [MS16]. Cela est particulièrement vrai dans les circuits **QDI** pour lesquels chemins de données et de contrôle ne sont pas clairement séparés.

Une fois les exigences temporelles décrites dans un formalisme compréhensible par les outils d'implémentation, elles peuvent être prises en compte lors des optimisations et des

raffinements successifs apportés par les différentes étapes du flot d'implémentation. De nombreuses approches sont possibles pour optimiser le circuit afin qu'il respecte toutes ces contraintes temporelles. Elles oscillent entre deux extrêmes. Soit chercher en premier lieu à garantir les performances, puis dans un second temps corriger les violations du protocole. Soit, à l'inverse, s'assurer du fonctionnement correct du circuit, puis de rechercher les optimisations qui permettent d'atteindre la performance souhaitée. Dans le cas des circuits à données groupées, ce dilemme revient à choisir si les délais *matchés* doivent être adaptés au chemin de données, ou à l'inverse si le chemin de données doit être optimisé pour correspondre à la contrainte imposée par les délais insérés dans le chemin de contrôle. La plupart du temps une solution intermédiaire et très itérative est utilisée.

1.8 Conclusion

Dans ce chapitre, nous avons présenté un bref état de l'art des circuits asynchrones. Nous avons vu que le terme asynchrone englobe une multitude de circuits aux caractéristiques et aux architectures très variées, mais qui partagent tous un point commun : les communications et échanges qui se jouent en leur sein ne sont pas rythmés par un signal de synchronisation externe.

Nous avons ensuite montré la proximité qui existe entre synchrone et asynchrone. Les premiers peuvent finalement être vus comme une sous-classe des seconds pour lesquels une hypothèse temporelle globale est définie. Nous avons ainsi présenté une classification des ces différents circuits et présenté rapidement les classes les plus étudiées.

Nous avons ensuite présenté les avantages communément reconnus des circuits sans horloge. Nous avons vu qu'ils possèdent des atouts très intéressants dans la perspective de construire des systèmes à la fois basse consommation et sécurisés. Nous avons alors donné quelques éléments sur les méthodes de modélisation et de conception de ces circuits.

Enfin, nous nous sommes attardé sur les techniques d'implémentation de ces circuits. Nous avons présenté la notion de contraintes de temps relatifs. Celles-ci permettent de décrire les hypothèses temporelles régissant le fonctionnement de n'importe quel circuit. Puis nous avons donné quelques références sur les méthodes d'analyse et d'implémentation de ces contraintes.

Dans ce chapitre, nous avons finalement présenté les circuits asynchrones comme une solution intéressante pour répondre aux problématiques de sécurité et de basse consommation. Mais nous avons vu qu'il existe une grande variété d'asynchrone, avec des différences plus ou moins subtiles. Il n'y a alors rien d'étonnant à ce que de nombreux concepteurs trouvent hostile le monde asynchrone. Les circuits à données groupées n'échappent pas à la règle mais paraissent tout de même plus abordables du fait de leur proximité avec les circuits synchrones. Aussi, la disponibilité d'outils et de flots de conception de plus en plus matures semble amener une adoption progressive de ces circuits par les milieux académiques et industriels.

Chapitre 2

Asynchrone, où est le problème ?

Sommaire

2.1	Introduction	36
2.2	Nœud gordien et cercles vicieux	36
2.3	Couper le nœud	37
2.4	Les limites des solutions existantes	38
2.4.1	Limitations des outils synchrones	39
2.4.2	Limitations des approches min/max	40
2.5	Contraintes de temps relatifs et circuits à données groupées	41
2.5.1	Modèles et RTC	41
2.5.2	Classification des RTC	43
2.5.2.1	Paramètres de l'arbre	43
2.5.2.2	Classes de contraintes de temps relatifs	44
2.5.2.3	Exemple du contrôleur Mousetrap	46
2.5.3	RTC, primitives et hiérarchies	47
2.6	Conclusion	49

2.1 Introduction

Parler de logique asynchrone dans le cadre d'une réalisation industrielle rencontrera presque systématiquement l'une des deux réactions suivantes : des yeux écarquillés ou des sourires entendus. Les uns seront choqués de l'hérésie d'une telle proposition, les autres préféreront parler d'arlésienne. Force est de constater qu'en dépit de leurs nombreuses qualités revendiquées, les circuits asynchrones sont loin de faire l'unanimité. Les débats sur la pertinence des techniques asynchrones sont passionnés. Les aficionados [Tri01 ; Pee08 ; Sut12] arguent une évolution inéluctable vers un fonctionnement en adéquation avec les limites de la physique¹ et inspiré des communications naturelles. Ils prennent en exemple le domaine du logiciel qui ne s'assujettit à aucune horloge et proposent, au final, de reléguer les pendules à leur utilité primaire : mesurer le temps. Les détracteurs [Nor01 ; Bor03] en appellent au réalisme et estiment que les problèmes qui ont vu le monde choisir le synchrone n'ont pas disparu. Ils expliquent que les concepteurs ont toujours su trouver des solutions pour résoudre les problèmes prétendument insolubles découlant de l'utilisation d'une horloge. Et malgré quelques percées notables, mais souvent éphémères, de l'asynchrone dans l'industrie (Thesus logic, Handshake Solutions, Fulcrum, Achronix, Tiempo), l'histoire semble pour le moment donner raison à ces derniers.

Dans ce chapitre nous synthétisons quelques clés pour comprendre la contradiction entre les atouts prometteurs des circuits asynchrones et leur si faible démocratisation (section 2.2). Nous proposons une résolution s'appuyant sur une transition progressive vers le monde asynchrone (section 2.3) et identifions l'un des freins qui entrave ce mouvement : les limites des outils de CAO vis-à-vis de l'analyse temporelle des circuits asynchrones (section 2.4). Finalement, nous proposons une étude détaillée et une taxonomie des différentes exigences temporelles s'appliquant aux circuits asynchrones (section 2.5).

2.2 Nœud gordien et cercles vicieux

Le paradoxe asynchrone se cristallise autour de trois grands blocages : deux cercles vicieux et un nœud gordien (figure 2.1). Le premier blocage est économique. L'écosystème asynchrone existant n'est pas suffisamment large pour entraîner avec lui de nouveaux adeptes². L'asynchrone est donc très peu utilisé. Étant donné la faible demande, les efforts nécessaires à son déploiement – outils de conception, cellules standards dédiées, blocs propriété intellectuelle (*Intellectual Property*) (IP) – ne sont pas investis. Au final, cet écosystème ne grandit pas et peine même à s'entretenir. Ce cercle vicieux a une source historique. Le monde asynchrone a pris du retard, une quinzaine d'années peut-être, pendant lequel le synchrone a pu se développer et affirmer sa suprématie. Il y a plus de vingt ans, ce retard était déjà perçu comme trop coûteux par certains acteurs. Lorsqu'en 1997, Intel obtenait une efficacité 6 fois supérieure sur un prototype de *Central Processing Unit* (CPU) asynchrone (trois fois plus rapide pour la moitié de la consommation d'un équivalent synchrone), la décision fut prise de ne pas continuer dans cette voie jugée trop coûteuse [Pan15] : le gain apporté ne compensait pas l'augmentation du temps de développement. Par ailleurs, le monde de la microélectronique s'est peu à peu hiérarchisé et spécialisé de façon à rationaliser et réduire au maximum ses coûts. Cellules standards, mémoires, blocs IP, logiciels de conception sont tous développés par des acteurs distincts. Le tout est utilisé par d'autres sociétés pour concevoir des circuits toujours plus gros. Ces circuits sont enfin fabriqués, testés puis mis en boîtier par des acteurs encore différents. Mais aujourd'hui,

1. Le coût énergétique des communications intra-circuit étant de plus en plus élevé, il faut les réduire au strict nécessaire.

2. Voir à ce sujet : [DL08].

une société souhaitant produire des circuits QDI doit elle-même assumer la conception des bibliothèques de cellules standards spécifiques à l'asynchrone, le développement et la maintenance des outils de CAO dédiés et la construction de leur propres blocs IP.

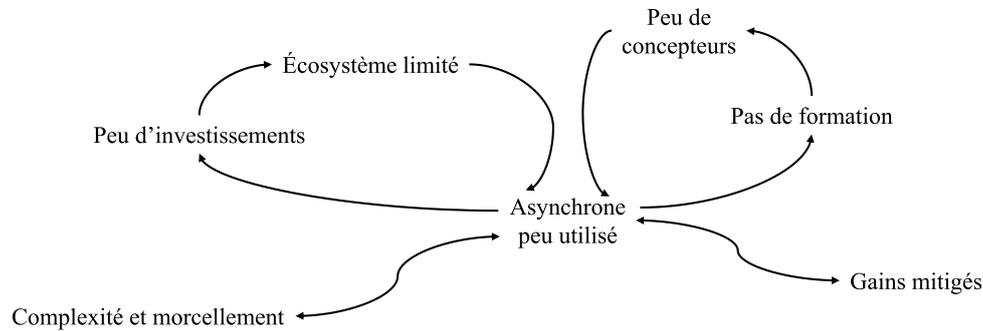


FIGURE 2.1 – Cercles vicieux et nœud gordien.

Le deuxième blocage est sociologique. L'asynchrone n'est pas enseigné dans les universités et écoles d'ingénieurs. Ce manque de formation entraîne une ignorance frappante de la part des ingénieurs des possibilités que cet autre paradigme apporte. Très peu sont ceux qui connaissent ne serait-ce que l'existence de la porte de Muller. Seuls quelques chercheurs ou intrépides s'aventurent dans cette voie, mais cela n'est pas suffisant pour que les circuits sans horloge dépassent le stade du projet de recherche. L'attrait des industriels ne décolle donc pas et la demande de formation reste au plus bas. Ce deuxième cercle vicieux est grandement lié à la réticence des industriels à changer une méthode synchrone qui a fait ses preuves. Ils préfèrent naturellement traiter des problèmes qu'ils connaissent plutôt que d'autres dont ils ignorent l'existence.

Le dernier blocage est technique. Il se présente plutôt comme un nœud gordien. Quelques soient les efforts faits pour le dénouer, celui-ci se resserre d'autant plus. À la complexité apparente du monde asynchrone s'entremêle des gains souvent mitigés. Le [tableau 2.1](#) illustre bien ce problème. Chaque classe de circuits possède ses avantages et ses inconvénients. Se résoudre à une seule de ces classes simplifie le problème, et permet de développer un flot de conception robuste. Mais ce choix entraîne nécessairement une perte de gain dans les situations où une autre classe aurait été plus appropriée. À l'inverse, posséder un flot de conception pour chacune de ces classes permettrait d'associer à chaque projet le schéma de circuit le plus adapté. Mais développer et entretenir de tels flots ne semble pas raisonnable. De son côté, le monde synchrone a, avec le temps, inventé de nouvelles techniques pour combler ses failles et atténuer les différences avec le monde asynchrone. À titre d'exemple, au fonctionnement purement événementiel des circuits asynchrones, répond la technique de *clock-gating* qui permet d'arrêter l'horloge lorsqu'elle n'est pas utilisée. Cette technique est même maintenant totalement intégrée aux outils de CAO. Les techniques de *retiming* ou d'insertion volontaire de *skew* sur l'arbre d'horloge sont d'autres évolutions imaginées pour optimiser les circuits synchrones. Elles permettent d'équilibrer les différents étages combinatoires et d'augmenter les fréquences de fonctionnement tout en diminuant la perte d'efficacité liée à l'approche du « pire chemin ». Finalement, alors que le monde des horloges avance au pas de course, celui des circuits auto-séquenceés forme un front très désuni. Il reste divisé, morcelé, incapable de s'accorder sur le meilleur moyen de s'affranchir de la tyrannie de l'horloge [Sut12].

2.3 Couper le nœud

S'il est peu probable que la tendance actuelle s'inverse dans les années à venir, des opportunités existent toutefois pour exploiter les méthodes de conception asynchrone.

TABLE 2.1 – Comparaison des circuits synchrones, BD et QDI selon différentes métriques.

Métrique	Synchrone	BD	QDI
Modularité	Mauvais	Bon	Meilleur
Vitesse	Bon	Bon	Meilleur
Surface	Bon	Bon	Très mauvais
Énergie	Mauvais	Bon	Mauvais
Multi fréquence	Très mauvais	Bon	Bon
Adaptation de tension	Mauvais	Bon	Meilleur
Sécurité	Mauvais	Bon	Meilleur
Courses temporelles	Bon	Mauvais	Meilleur
Disponibilité bibliothèques	Meilleur	Bon	Très mauvais
CAO	Meilleur	Mauvais	Très mauvais

Avec leurs contraintes extrêmement sévères d'autonomie et de sécurité, les objets connectés forment un champ d'application particulièrement propice à leur utilisation. Par ailleurs, la situation actuelle plaide aussi en faveur d'une intégration progressive et d'une cohabitation avec les circuits à horloges.

Ainsi, les circuits à données groupées semblent les mieux armés pour couper ce nœud. Grâce à leur proximité avec le monde synchrone, il semble aisé de mélanger ces deux classes de circuits. Les outils de CAO commerciaux sont plus facilement ré-exploitable et permettent de rapidement mettre en place des flots de conceptions adaptés [Cor+06 ; Sim17 ; ZZC18]. De plus, ces circuits n'utilisent pas nécessairement de cellules spécifiques et peuvent donc être construits avec n'importe quelle bibliothèque de cellules standards. L'impact économique est contenu. Cette parenté avec le synchrone permet aussi à des concepteurs non-initiés d'appréhender sans peine le fonctionnement de ces circuits. La conception et l'analyse sont facilitées, et les freins psychologiques en sont sûrement atténués. Finalement, les circuits à données groupées semblent être une solution économiquement et sociologiquement pertinente et donc industriellement viable.

Le [tableau 2.1](#) révèle cependant une difficulté qui subsiste dans la conception des circuits à données groupées. Ceux-ci sont particulièrement sensibles aux courses de chemins. Du côté synchrone, on s'accommode assez facilement d'erreurs dans l'analyse temporelle ou de déviations du processus de fabrication. En diminuant légèrement la fréquence d'horloge, des puces non-fonctionnelles peuvent être utilisées dans un mode dégradé. Mais pour les circuits à données groupées, le nombre important de délais *matchés* rend cette approche bien plus difficile. La disponibilité d'outils d'analyse temporelle fiables et performants est donc un prérequis indispensable à l'adoption de ce modèle de circuits.

2.4 Les limites des solutions existantes

Quelques travaux ont envisagé le développement d'outils d'analyse temporelle dédiés aux circuits asynchrones [CYD98 ; MHR02 ; Par+16 ; Xir+19]. Mais la plupart ont préféré tenter de réutiliser les outils synchrones [GMC15 ; And+07 ; PB15] pour faciliter la transition et profiter des performances accrues de ces outils commerciaux. L'une des approches les plus abouties est probablement celle présentée par STEVENS, XU et VIJ [SXV09]. Elle fournit une méthode formelle pour extraire les contraintes temporelles au format SDC à

partir d'une description haut-niveau du circuit (le modèle CCS [Mil89] est utilisé). Mais comme toutes les autres tentatives, cette méthodologie utilise les outils standard de manière non conventionnelle.

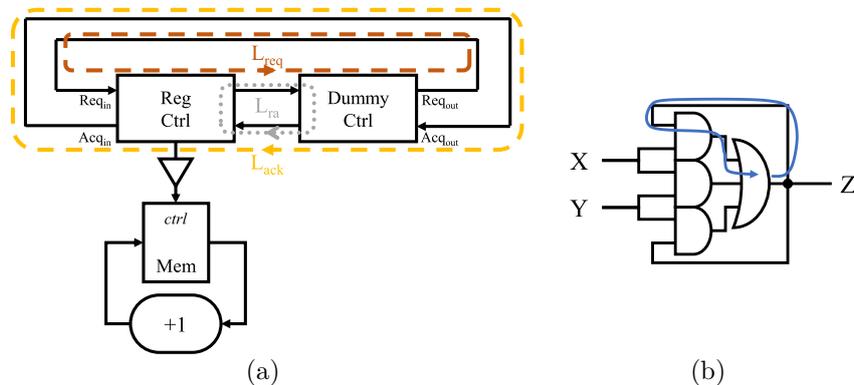


FIGURE 2.2 – Boucles combinatoires dans les circuits à données groupées. (a) Boucles locale et architecturales dans un compteur et (b) boucle locale dans une cellule de Muller à base de porte majorité.

2.4.1 Limitations des outils synchrones

Dans les outils de CAO standards, plusieurs difficultés compliquent la définition des RTC des circuits à données groupées. Ces difficultés sont principalement liées au paradigme synchrone dans lequel ces outils ont été développés et sont utilisés :

- 1) **Horloge ou évènement** – Les vérifications faites par les outils de STA sont par défaut relatives à un temps fixe défini par la période de l'horloge : le temps de propagation dans le chemin de données est comparé au temps séparant deux fronts actifs du signal d'horloge. Ce comportement ne correspond pas au principe de fonctionnement des circuits à données groupées où le temps de propagation d'un même évènement dans le bloc de contrôle doit être comparé au temps de propagation dans le chemin de données. La propagation irrégulière des évènements dans la partie contrôle des circuits à données groupées ne peut pas être capturée par la définition d'une horloge avec une période fixe.
- 2) **Boucles combinatoires locales** – Dans les circuits à données groupées, chaque canal de requête/acquittement forme une boucle combinatoire entre les différents étages du bloc de contrôle. Ces boucles ne sont pas supportées par les moteurs d'analyse temporelle car elles ne peuvent pas être analysées statiquement. Sans précaution particulière, les outils coupent automatiquement et arbitrairement ces boucles, et peuvent désactiver des chemins essentiels. Dans la figure 2.2a, la boucle L_{ra} (en pointillés gris) est un exemple de boucle locale. De telles boucles peuvent aussi exister à l'intérieur des contrôleurs. C'est le cas par exemple des boucles de rétroaction de l'implémentation de type « porte majorité » de la cellule de Muller (figure 2.2b).
- 3) **Boucles combinatoires architecturales** – Les boucles architecturales dans le bloc de contrôle sont la conséquence de boucles dans le chemin de données. Ces boucles ne peuvent être cassées sans désactiver des chemins qui doivent pourtant être vérifiés. Un exemple simple est donné dans la figure 2.2a. Ce compteur présente deux boucles architecturales, L_{req} et L_{acq} (flèches discontinues).
- 4) **Localisation des points de divergences** – les *pod*, qui servent de point de référence aux RTC, ne peuvent pas être devinés par les outils de CAO synchrones. Ils

doivent être décrits manuellement et de manière compréhensible pour le moteur de STA sans casser de chemin temporel significatif.

- 5) **Contrainte du chemin de données** – de nombreuses RTC sont à cheval entre le chemin de données et le bloc de contrôle. Les relations entre ces deux parties du circuit doivent donc être préservées. Celles-ci sont essentiellement décrites dans les modèles d'abstraction temporelle au travers des vérifications usuelles de *setup*, de *hold*, de largeur d'impulsion, etc. Mais dans les outils synchrones, seule la présence d'une horloge sur les entrées de contrôle des éléments séquentiels permet d'activer ces vérifications. Or, c'est uniquement en maintenant cette relation que les outils seront capables de contraindre et d'optimiser simultanément le bloc de contrôle et le chemin de données.

2.4.2 Limitations des approches min/max

Les méthodes de STA des circuits à données groupées envisagées jusqu'à présent peinent à s'affranchir de ces difficultés. Elles adoptent majoritairement une approche consistant à découper les RTC en deux contraintes temporelles disjointes : une contrainte de temps minimum et une autre de temps maximum. Elles utilisent pour cela les commandes `set_min_delay` et `set_max_delay` du format SDC. Mais elles ont de nombreuses limitations.

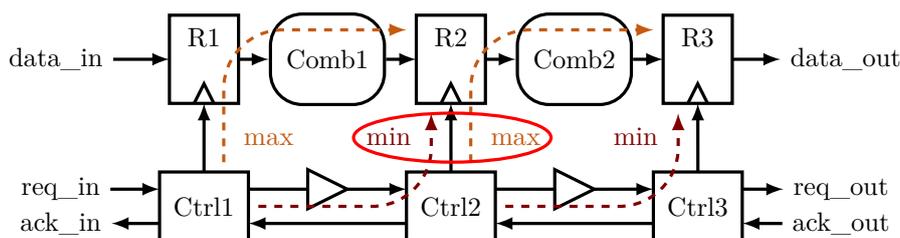


FIGURE 2.3 – Chevauchement de contraintes dans les circuits à données groupées.

La première est liée au chevauchement des RTC. Comme illustré par la figure 2.3, certains chemins du circuit appartiennent à la fois à des contraintes minimales et maximales. Ce conflit de contraintes ne peut pas être résolu par les outils. Pour s'en affranchir, ces méthodes choisissent de réduire la portée des contraintes de manière à éviter les chevauchements : les contraintes de temps minimum et maximum sont appliquées sur une plus petite portion des chemins (figure 2.4). Mais ce faisant, les *poc* et les *pod* sont séparés en points disjointes (losanges rouges) et la relation logique entre les chemins est supprimée (doubles flèches en pointillés). Il en résulte que certaines parties du circuit ne sont plus prises en compte dans l'analyse temporelle. Sur la figure 2.4, les chemins entre chaque contrôleur et son registre associé échappent ainsi à toutes contraintes (ellipse rouge).

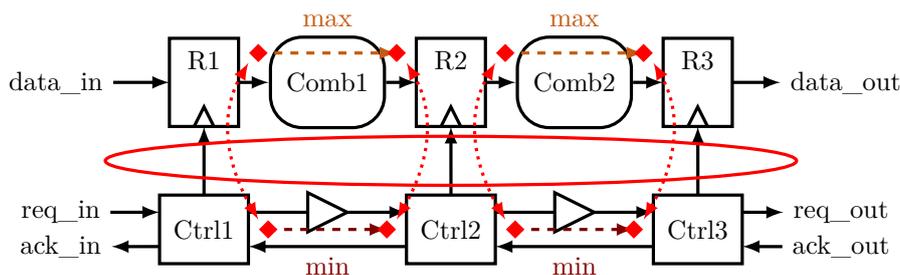


FIGURE 2.4 – Simplification des contraintes et impact sur l'analyse temporelle.

Ensuite, les commandes `set_min_delay` et `set_max_delay` définissent des bornes fixes auxquelles sont comparés les différents temps de propagation. Elles ne permettent donc pas de décrire des temps relatifs entre deux chemins. En d'autres termes, ces méthodes ne peuvent pas exprimer directement une **RTC** sous la forme : le chemin A doit toujours être plus rapide que le chemin B. Elles utilisent donc une variable intermédiaire, X , et se contentent de dire : le chemin A doit prendre moins de X unités de temps et le chemin B doit prendre plus de X unités de temps.

Cependant, les valeurs de ces variables intermédiaires ne sont pas directement disponibles dans les outils de **STA**. Les méthodes min/max s'appuient donc sur un processus itératif d'extraction des temps de propagation, de calcul des contraintes minimales et maximales par post-traitement des rapports, de réinjection de ces contraintes dans les outils puis de ré-analyse des chemins. Ces opérations sont longues et peuvent être nécessaires à de multiples reprises au cours de la conception pour prendre en compte les optimisations apportées par chacun des outils. Dans certains cas, des marges peuvent être ajoutées pour limiter ces itérations, mais cela vient généralement au détriment des performances du circuit.

Par ailleurs, comme ces variables intermédiaires sont fixes, elles ne peuvent décrire le fonctionnement du circuit que pour une condition d'opération donnée. Ces valeurs doivent donc être recalculées pour chacune des conditions **PVT**. Les méthodes min/max nécessitent par conséquent un grand nombre de fichiers de contraintes (au moins un par **PVT**), avec tous les risques que cela comporte, et elles ont tendance à allonger les temps d'analyse.

Les techniques habituelles n'apportent pas non plus de solution satisfaisante pour gérer les boucles combinatoires. Elles adoptent une approche où les boucles sont coupées manuellement (`set_disable_timing`). Et dans le cas où des chemins significatifs doivent être coupés, plusieurs jeux de contraintes sont créés de manière à définir des coupes différentes. En multipliant ainsi les modes de contraintes, tous les chemins peuvent être vérifiés. Mais cela rallonge encore les temps d'analyse.

Finalement, les méthodes min/max ne sont pas automatisées et sont difficilement utilisables sur de grands circuits. N'étant pas en mesure de retranscrire complètement les **RTC** dans un formalisme compréhensible par les moteurs de **STA**, elles laissent les outils ignorants des réelles hypothèses temporelles qu'ils doivent implémenter. Elles ne permettent donc pas d'exploiter pleinement les capacités d'exploration et d'optimisation apportées par les outils de **CAO** commerciaux.

2.5 Contraintes de temps relatifs et circuits à données groupées

L'analyse temporelle des circuits à données groupées est souvent réduite à la vérification de leur principale hypothèse : le regroupement des signaux de contrôle et de données. Celle-ci garantit la propagation cohérente des signaux de requête et d'acquiescement relativement aux données, et ainsi un transfert fiable des informations d'un étage à un autre du circuit. Mais le fonctionnement de cette classe de circuits repose sur de nombreuses autres exigences. Afin d'exploiter efficacement les outils de **CAO** traditionnels, il est donc important de mieux appréhender leur fonctionnement. Il est aussi nécessaire de bien identifier les contraintes qu'ils doivent pouvoir manipuler.

2.5.1 Modèles et **RTC**

Pour simplifier leurs analyses, les outils de **STA** synchrones manipulent des modèles d'abstraction temporelle. Afin de mieux comprendre les interactions entre ces modèles et

les contraintes de temps relatifs, nous proposons de comparer les deux circuits présentés dans la figure 2.5.

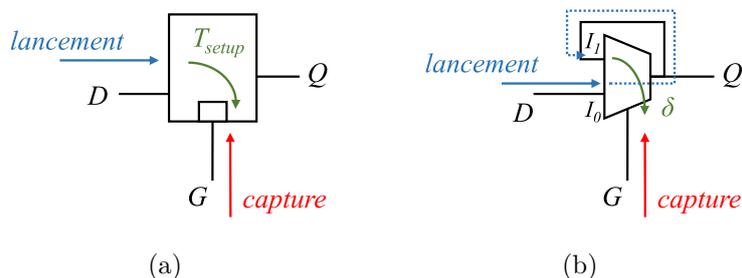


FIGURE 2.5 – Contrainte de temps relatifs de setup dans un verrou. (a) verrou D classique et (b) verrou D à base de multiplexeur..

Le premier (figure 2.5a) est un verrou D, disponible dans toutes les bibliothèques de cellules standards. Son implémentation au niveau transistor n'est pas donnée. Il vient cependant avec un modèle temporel définissant les différentes contraintes à respecter pour assurer un fonctionnement correct. Par exemple, la contrainte de temps de *setup* est représentée sur la figure 2.5a. Cette contrainte peut être exprimée à l'aide de l'équation de RTC (2.1) : un évènement partant d'un point de divergence *pod* doit atteindre l'entrée D du verrou au travers d'un chemin de lancement, avant que ce même évènement, se propageant sur le chemin de capture, n'atteigne l'entrée de contrôle G avec une marge de temps T_{setup} .

$$pod \mapsto D + T_{setup} \prec G \quad (2.1)$$

La figure 2.5b présente ce même verrou, mais cette fois-ci le détail d'une implémentation possible est donné. Elle se base sur une porte de multiplexage rebouclée sur elle-même. Les modèles temporels ne définissent aucune contrainte pour ce circuit car cette cellule est combinatoire. Cependant cette implémentation révèle la vérité habituellement cachée par le modèle : pour assurer une bonne mémorisation d'une donnée dans ce verrou, celle-ci doit traverser le multiplexeur et se propager le long de la boucle combinatoire pour atteindre l'autre entrée du multiplexeur avant que le signal de capture n'arrive. Cette réalité peut être formalisée par l'équation de RTC (2.2).

$$pod \mapsto I_1 + \delta \prec G \quad (2.2)$$

En comparant les équations (2.1) et (2.2), nous comprenons que le temps de *setup* du modèle temporel est équivalent au délai de propagation le long de la boucle combinatoire de la version à base de multiplexeur, comme exprimé par l'équation (2.3). La marge δ permet de modéliser la portion du chemin de propagation au niveau transistor, qui est abstraite par le modèle du multiplexeur.

$$T_{setup} \equiv d_{int} + \delta \quad \text{avec} \quad d_{int} = D \mapsto I_1 \quad (2.3)$$

Une analyse équivalente peut être faite pour les autres contraintes temporelles définies dans les fichiers d'abstraction. Ainsi, les modèles temporels utilisés par les outils standard définissent les contraintes de temps relatifs telles que vues à leurs frontières. Plus précisément, ils spécifient les termes à droite de ces RTC : une paire de points de convergence, un sens d'inégalité ainsi qu'une marge associée. Il est cependant intéressant de noter que ces RTC sont « protocole-agnostique ». Elles sont définies indépendamment d'une classe de circuits. Elles permettent simplement de garantir le fonctionnement électrique et préservent ainsi le circuit des phénomènes de métastabilité dans les éléments séquentiels.

2.5.2 Classification des RTC

De manière à mieux appréhender les contraintes temporelles, nous proposons de les classer sous la forme d'un arbre taxinomique donné en figure 2.6.

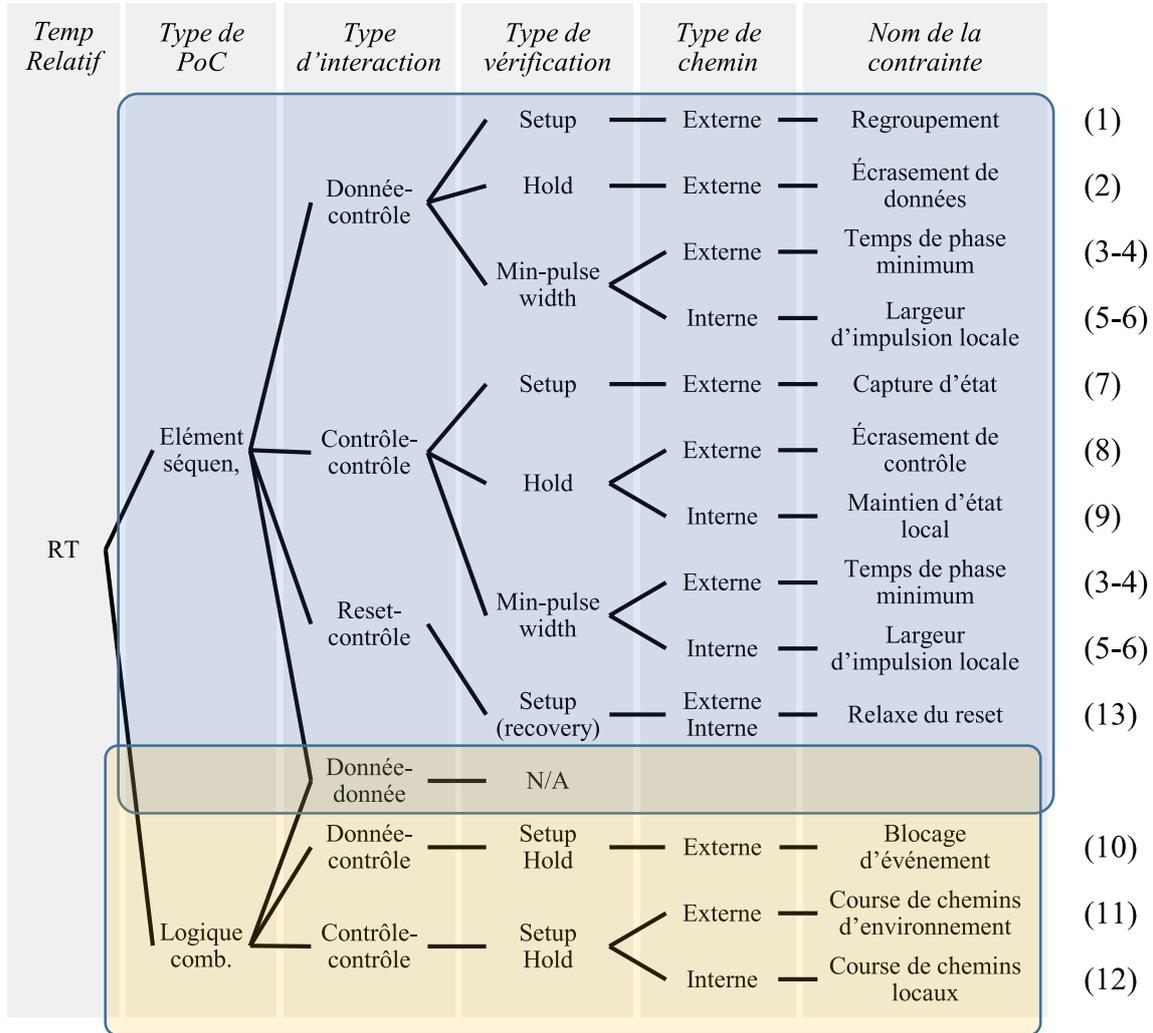


FIGURE 2.6 – Arbre de classification des contraintes de temps relatifs.

2.5.2.1 Paramètres de l'arbre

Nous l'avons vu, les outils, les modèles et les méthodes d'analyse existants, bien que développés dans un paradigme synchrone, manipulent déjà des contraintes de temps relatifs. Les contraintes classiques de *setup*, de *hold* ou de largeur d'impulsion, qui sont associées à tout élément séquentiel, révèlent les RTC masquées par le niveau d'abstraction apporté par le modèle. Pour les circuits synchrones, la plupart des RTC sont agrégées au niveau des éléments séquentiels à cause du modèle de délais bornés utilisé. Mais des exceptions ont été apportées et de nouvelles sémantiques ont été créées pour capturer des RTC qui ne s'intègrent pas aux éléments séquentiels. Par exemple, la commande SDC `set_data_check` permet de définir les point de convergence (POC) et le sens de RTC s'appliquant sur la logique combinatoire. Nous proposons donc de classer les RTC en fonction du type de cellule de POC qu'elles utilisent : soit un élément séquentiel, soit une cellule combinatoire.

Les signaux mis en concurrence par une RTC ne sont pas tous de même nature. Dans

les circuits à données groupées, un signal peut appartenir soit au chemin de données soit au contrôleur. Nous pouvons aussi distinguer le reset qui est un troisième type de signal particulier. Ainsi, les **RTC** peuvent être classées selon le type d'interaction qu'elles décrivent. Alors que les interactions données-données ne sont pas valides (les circuits **BD** utilisent le même modèle à délais bornés dans la partie chemin de données), les interactions donnée-contrôle regroupent l'ensemble des **RTC** s'appliquant à la frontière entre les deux parties du circuit. Les interactions contrôle-contrôle sont celles requises à l'intérieur du contrôleur pour assurer le bon fonctionnement du protocole. La quatrième famille comprend les interactions reset-contrôle. Celles-ci sont souvent oubliées mais peuvent, lorsqu'elles sont mal implémentées, générer de mauvais fonctionnements comme une désynchronisation entre les données et les signaux de contrôle.

Les **RTC** peuvent ensuite être triées suivant le type de vérification qu'elles décrivent. Les vérifications de *setup* et de *hold* s'intéressent à la relation entre deux signaux distincts. Elles se différencient selon l'orientation de l'inégalité (quel signal doit arriver en premier). Elles mettent respectivement en jeu le même cycle de « poignée de mains » ou bien deux cycles consécutifs³. Le troisième type de vérification s'applique au temps séparant deux transitions opposées⁴ du même signal. Il est nommé « largeur minimum d'impulsion haute » quand il contraint l'espacement entre un front montant et un front descendant et « largeur minimum d'impulsion basse » dans l'autre sens.

Enfin, les **RTC** peuvent être séparées en deux catégories suivant le type de chemin qu'elles activent : (i) Certaines sont contenues dans un seul étage du bloc de contrôle. Elles sont internes et peuvent être facilement masquées par un modèle d'abstraction temporelle comme le format *Liberty*. (ii) Les autres impliquent au moins deux contrôleurs. Les chemins temporels s'étalent sur plusieurs étages du bloc de contrôle pour relier leurs points de divergence et de convergences.

Certains cas ont volontairement été ignorés. Par exemple, une contrainte de « groupement interne » n'existe pas par définition. Une contrainte de « capture d'état interne » semble peu probable. Elle impliquerait soit deux éléments séquentiels⁵, dans le même contrôleur, soit un protocole générant de multiples impulsions au cours du même cycle de communication. Nous avons donné un nom à chacune des feuilles de l'arbre de classification. Certains sont inspirés de la littérature. Pour les autres, un nom explicite a été choisi.

2.5.2.2 Classes de contraintes de temps relatifs

Une équation de temps relatifs générique peut être associée à chaque contrainte de la figure 2.6. Le formalisme suivant est adopté : *d-data* et *d-ctrl* représentent respectivement une transition sur l'entrée de données ou d'horloge d'un élément séquentiel dans le chemin de données, *c-data* et *c-ctrl* sont leurs équivalents pour un élément séquentiel présent dans le bloc de contrôle. Les transitions *cl* et *cc* sont deux **POC** dans la logique combinatoire, respectivement de lancement et de capture. La transition *pod* est un point de divergence (**POD**) du circuit. D'autre part, *rst* est la source de *reset* du circuit et *d-rst* est l'entrée de *reset* d'un élément séquentiel. L'indice *i* est utilisé pour indiquer l'appartenance d'une transition au *i*-ème étage du contrôleur. Nous utilisons aussi la notation $\xrightarrow{[i]}$ pour préciser

3. En cela les circuits **BD** se différencient des circuits synchrones pour lesquels les vérifications de *setup* sont faites entre deux cycles d'horloge successifs et celles de *hold* sur le même cycle.

4. Des vérifications de période minimum sont possibles (temps séparant deux transitions identiques du même signal), mais celles-ci sont en général réservées aux blocs hiérarchiques de grosse taille intégrant eux-mêmes un chemin de données (mémoire, sous-circuit). Dans ce cas, le respect des contraintes de largeur d'impulsion n'est pas suffisant à garantir le bon fonctionnement du bloc.

5. Pour lesquels des temps de *setup* et de *hold* existent

l'étage du contrôleur auquel les chemins sont restreints. Le sens des transitions est défini à l'aide de $+$ ou $-$. Dans un souci de simplicité, ceux-ci sont omis lorsque les deux sens sont valables. Finalement, chaque équation utilise un paramètre générique, δ , qui représente l'ensemble des marges prises par les concepteurs, les outils ou les modèles.

- (1) *Regroupement* : Elle est la principale contrainte entre deux contrôleurs d'un circuit à données groupées. Elle garantit que les données et les signaux de contrôle se propagent de manière synchronisée. Elle fait intervenir le délai *matché* sur les fils de poignée-de-main ainsi que le pire chemin de propagation des données. Cette contrainte vérifie que les marges de *setup* de l'élément séquentiel ne sont pas mises en défaut par les temps d'arrivée des données et des signaux de contrôle.

$$pod \mapsto d-data_i \prec d-ctrl_i - \delta \quad (2.4)$$

- (2) *Écrasement de données* : Cette contrainte permet de s'assurer que l'arrivée rapide d'une nouvelle donnée au niveau d'un élément séquentiel ne vienne pas perturber la mémorisation de la donnée précédente. Elle garantit que les données soient maintenues suffisamment longtemps pour respecter les temps de *hold* des éléments séquentiels.

$$pod \mapsto d-ctrl_i \prec d-data_i - \delta \quad (2.5)$$

- (3-4) *Temps de phase minimum* : Elle s'applique à tout élément séquentiel et garantit une capture cohérente des données. En fonction du type d'élément séquentiel utilisé, elle vérifie la durée de l'état haut ou de l'état bas du signal de capture. Cette contrainte s'applique à une impulsion générée par le protocole et dont la largeur dépend de la réponse de l'environnement. Elle implique au moins une phase du cycle de communication.

$$\text{(haut)} \quad pod \mapsto d-ctrl_i^+ \prec d-ctrl_i^- - \delta \quad (2.6)$$

$$\text{(bas)} \quad pod \mapsto d-ctrl_i^- \prec d-ctrl_i^+ - \delta \quad (2.7)$$

- (5-6) *Largeur d'impulsion locale* : Comme la contrainte de *temps de phase minimum*, elle s'applique à tout élément séquentiel. Mais dans ce cas, la largeur d'impulsion dépend de chemins internes au contrôleur. Cette contrainte est typiquement présente dans les protocoles utilisant de la conversion de phase (protocole 2 phases).

$$\text{(haut)} \quad pod_i \xrightarrow{[i]} d-ctrl_i^+ \prec d-ctrl_i^- - \delta \quad (2.8)$$

$$\text{(bas)} \quad pod_i \xrightarrow{[i]} d-ctrl_i^- \prec d-ctrl_i^+ - \delta \quad (2.9)$$

- (7) *Capture d'état* : Cette contrainte s'applique aux protocoles utilisant un élément séquentiel dans leur contrôleur. Ces éléments mémorisent un état interne du contrôleur. La nouvelle valeur de cet état doit être stable pour pouvoir être mémorisée lorsque la phase suivante du protocole commencera.

$$pod \mapsto c-data_i \prec c-ctrl_i - \delta \quad (2.10)$$

- (8) *Écrasement de contrôle* : Cette contrainte s'applique aussi aux contrôleurs incluant un élément séquentiel. La variable d'état ne doit pas être écrasée tant qu'elle n'a été correctement mémorisée.

$$pod \mapsto c-ctrl_i \prec c-data_i - \delta \quad (2.11)$$

- (9) *Maintien d'état local* : Tout comme la contrainte d'*écrasement de contrôle*, celle-ci s'attend à ce que la variable d'état local soit stable lors de sa mémorisation. Chaque fois que cette variable est modifiée localement (c'est-à-dire, sa prochaine valeur dépend directement de sa valeur actuelle), une contrainte de maintien d'état local s'applique.

$$pod_i \xrightarrow{[i]} c-ctrl_i \prec c-data_i - \delta \quad (2.12)$$

- (10) *Blocage d'évènement* : Cette contrainte est nécessaire à chaque fois qu'il existe une interaction directe dans la logique combinatoire (et non au niveau d'un élément séquentiel) entre un signal de contrôle et un signal de données. Cette relation temporelle est habituellement appelée *clock-gating* dans le contexte synchrone. Suivant le sens de l'inégalité, une contrainte de setup ou de hold de blocage d'évènement doit être vérifiée.

$$pod \mapsto cl_i \prec cc_i - \delta \quad (2.13)$$

- (11) *Course de chemins d'environnement* : Une contrainte interne peut être nécessaire pour s'assurer du bon séquençement des évènements dans un contrôleur. Si le chemin relatif s'étend à plusieurs contrôleurs, il décrit un course d'environnement entre deux signaux de contrôle.

$$pod \mapsto cl_i \prec cc_i - \delta \quad (2.14)$$

- (12) *Course de chemins locaux* : Cette contrainte est similaire à une course d'environnement, mais le séquençement des évènements dépend uniquement de transitions internes à un seul contrôleur.

$$pod_i \xrightarrow{[i]} cl_i \prec cc_i - \delta \quad (2.15)$$

- (13) *Relaxe du reset* : Le relâchement du signal de réinitialisation est une phase sensible pour un circuit à données groupées. Un léger décalage entre le redémarrage du contrôleur et celui du chemin de données peut générer un désalignement entre les évènements de contrôle et les données. Cela peut conduire à un comportement incorrect ou à un blocage du système. Habituellement, un délai est inséré pour séparer les signaux de réinitialisation du chemin de données et du contrôleur. Il permet de retarder la réinitialisation du contrôleur jusqu'à ce que le chemin de données soit prêt.

$$rst \mapsto d-rst_i \prec d-ctrl_i - \delta \quad (2.16)$$

2.5.2.3 Exemple du contrôleur Mousetrap

Nous proposons d'utiliser cette classification pour identifier les contraintes temporelles associées au contrôleur de protocole Mousetrap. La [figure 2.7](#) donne l'implémentation typique de cette primitive. En examinant cette implémentation et en identifiant les éléments séquentiels, nous pouvons facilement déduire un premier ensemble de *RTC* s'appliquant à ce contrôleur. Deux verrous doivent être pris en compte : un dans le contrôleur et l'autre dans le chemin de données. Les [équations \(2.4\) à \(2.12\)](#) et [\(2.16\)](#) s'appliquent donc nécessairement. Mais, en considérant que les verrous sont transparents à l'état haut, seule la contrainte de largeur d'impulsion minimum haute doit être vérifiée (les impulsions basses ne génèrent pas de métastabilité dans ce type de verrous). Les [équations \(2.7\)](#) et [\(2.9\)](#) peuvent être ignorées.

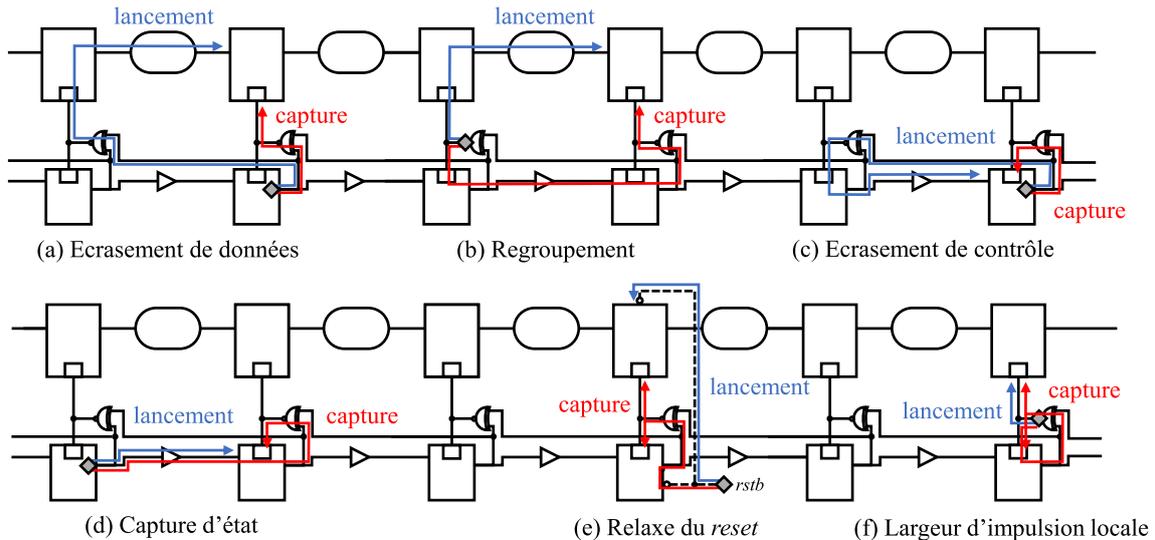


FIGURE 2.7 – Contraintes temporelles relatives du contrôleur Mousetrap.

En regardant le fonctionnement de ce contrôleur, cette liste peut être encore légèrement simplifiée. Le front descendant du signal de contrôle des verrous est généré dans le contrôleur, alors que le front montant est déclenché par le retour d’acquiescement de l’étage suivant. Nous pouvons en déduire que la largeur de l’impulsion haute dépend uniquement de chemins temporels internes. La contrainte de *temps de phase minimum* ne s’applique donc pas et l’équation (2.6) peut être ignorée. L’analyse inverse peut être faite pour la contrainte de *maintien d’état local*. La variable d’état dépend seulement du signal de requête du canal d’entrée. L’équation (2.12) n’a pas de sens dans ce cas.

En parcourant les branches de l’arbre taxinomique, nous avons pu construire une première série de 8 RTC génériques (les contraintes *relaxe du reset* et *largeur d’impulsion locale* doivent être comptées pour chaque verrou). Les autres RTC, s’appliquant sur la logique combinatoire, ne peuvent pas être extraites de la même manière mais peuvent être retrouvées en utilisant des approches formelles. Des outils de synthèse de modèle formel comme Petriify permettent de générer une implémentation d’un protocole donné et de fournir une liste des RTC associées. De telles contraintes n’existent pas dans l’implémentation originale du protocole Mousetrap. Seuls les POC ont été localisés jusqu’à présent. Pour compléter les RTC, chaque POD doit être identifié. Cela peut être fait de manière très intuitive : en parcourant le schéma du bloc de contrôle un point commun connectant les deux POC peut être trouvé. La figure 2.7 trace toutes les RTC identifiées et place leur POD respectif (losange grisé). Il est intéressant de remarquer que trois points sont suffisants pour décrire les POD des 8 RTC.

2.5.3 RTC, primitives et hiérarchies

Pour mieux comprendre le découpage hiérarchique des RTC, nous étudions maintenant une implémentation légèrement différente du protocole Mousetrap. Nous remplaçons les verrous de la partie contrôle par la version utilisant le multiplexeur rebouclé sur lui-même (voir figure 2.5).

Cette nouvelle implémentation révèle une nouvelle RTC initialement cachée par le modèle d’abstraction : la *course de chemins locaux* illustrée par la figure 2.8d. Pour garantir la bonne mémorisation de la variable d’état, le temps de propagation dans le chemin de rétroaction du multiplexeur doit être plus rapide que celui au travers du bloc de

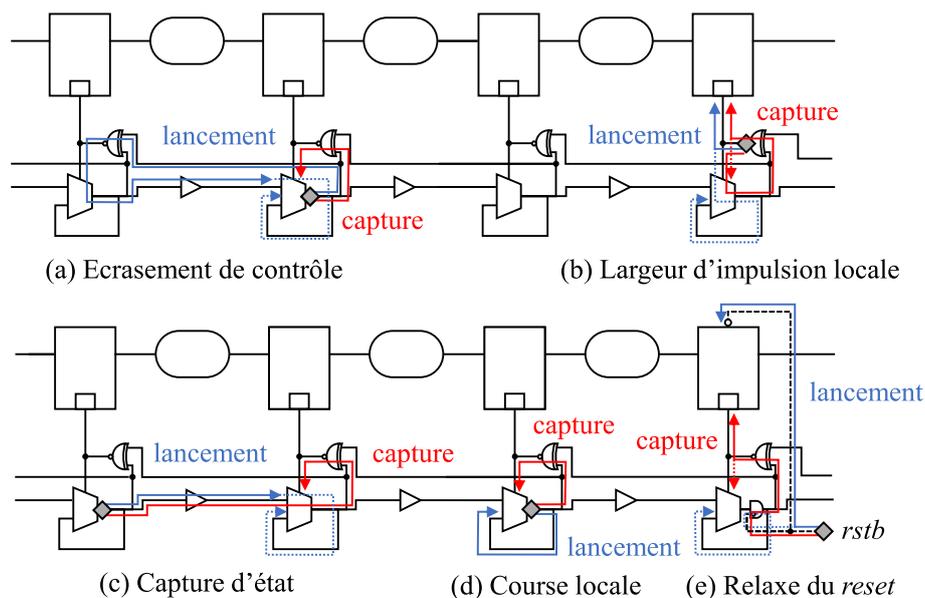


FIGURE 2.8 – Contraintes temporelles relatives du contrôleur Mousetrap à base de multiplexeur.

conversion de phase (la porte XOR). Par ailleurs, les flèches pointillées dans la figure 2.8 montrent l'évolution des RTC existantes qui ont été modifiées du fait de cette nouvelle implémentation. Initialement liées à un élément séquentiel, ces RTC sont maintenant des *courses de chemins d'environnement* ou des *course de chemins locaux*. Cela illustre précisément l'effet d'abstraction du modèle *Liberty*. De plus, certaines RTC apparaissent redondantes. Si la contrainte de course locale (figure 2.8d) est respectée alors celle de *capture d'état* (figure 2.8c) le sera aussi. Le raisonnement identique peut être fait pour les contraintes de *largeur d'impulsion locale* (figure 2.8b) et de *relaxe du reset* (figure 2.8e) du verrou de contrôle. Finalement, 6 RTC doivent être vérifiées pour cette nouvelle implémentation⁶.

De manière assez inattendue, cet exemple montre que l'utilisation d'un modèle d'abstraction temporelle peut générer un plus grand nombre de contraintes. Bien sûr, ces modèles ont d'autres avantages. Ils permettent de décrire des paires de POC de manière standardisée dans les outils de STA synchrones, ou alors de fixer et masquer certaines contraintes et d'ainsi réduire la complexité pour les outils. Mais les bénéfices de leur utilisation dépendent du protocole et de son implémentation, et ne sont donc pas automatiques.

Les modèles *Liberty* permettent d'abstraire les détails de l'implémentation de cellules de base. Mais il est possible de les utiliser pour encapsuler des blocs plus ou moins gros. Cette approche hiérarchique est particulièrement adaptée aux méthodes se basant sur des primitives pour construire un circuit à données groupées. Les contrôleurs et autres primitives (fourche, union, etc.) peuvent ainsi être implémentés séparément puis assemblés pour former le circuit de contrôle. Dans l'idéal, toutes les RTC internes à ces primitives seront masquées. Il ne restera, au niveau de l'assemblage, qu'à vérifier les RTC externes s'étalant sur plusieurs contrôleurs. Cependant, cette approche hiérarchique peut aussi mener à un découpage des RTC. Dans ce cas, il faut arbitrer entre la complexité de l'analyse de STA et le surdimensionnement du circuit.

Revenons à l'exemple du contrôleur Mousetrap basé sur une cellule multiplexeur. Nous

6. Une de plus et trois de moins que dans la version avec un verrou classique

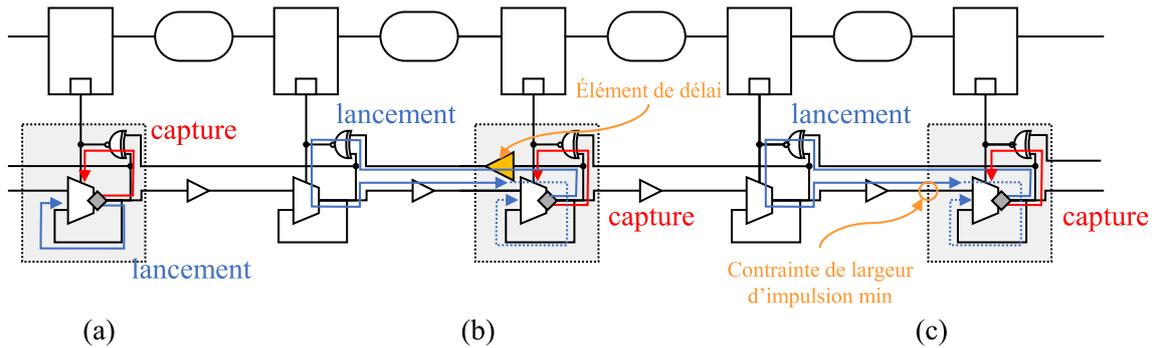


FIGURE 2.9 – Différents cas d'utilisation d'un modèle hiérarchique. a) Masquage de contrainte interne, b) gestion de contrainte dans le bloc hiérarchique et c) définition de nouvelles contraintes au niveau modèle.

pouvons encapsuler le contrôleur dans un modèle hiérarchique tel que schématisé par les zones grises dans la figure 2.9. Ce modèle va complètement masquer la contrainte de *course de chemins locaux* (figure 2.9a). Par contre, la contrainte d'*écrasement de contrôle* est coupée en deux. Une partie de la *RTC* est comprise dans le modèle hiérarchique, l'autre s'étale sur le contrôleur précédent. Deux possibilités s'offrent alors à nous pour gérer cette contrainte :

1. La contrainte peut être réglée dans le bloc hiérarchique, par exemple, à l'aide d'un élément de délai inséré sur la ligne d'acquiescement (en orange dans la figure 2.9b). Dans ce cas, la contrainte n'a pas besoin d'être exposée au niveau supérieur, et la *STA* du circuit est simplifiée. On parle alors de fonctionnement en mode fondamental du bloc hiérarchique. La contrainte temporelle est respectée quoiqu'il arrive au niveau de l'environnement. Cependant, il est fort probable que le temps de propagation dans l'environnement - sur la ligne d'acquiescement, puis à travers la porte *XOR*, le multiplexeur, le délai *matché*, un autre multiplexeur puis la boucle de rétroaction - soit largement suffisant pour garantir la tenue de cette *RTC*. L'ajout d'un délai systématique dans les contrôleurs va donc ralentir inutilement le temps de cycle et diminuer l'efficacité du protocole.
2. La contrainte peut être exposée au niveau du modèle hiérarchique pour être prise en compte lors de la *STA* faite au niveau assemblage. Dans ce cas, une contrainte de largeur d'impulsion minimale doit être ajoutée sur l'entrée de requête du contrôleur (figure 2.9c). Cette solution permet de n'insérer de délai sur la ligne d'acquiescement (ou ailleurs si plus judicieux) qu'en cas de nécessité. Elle conserve ainsi une performance et une efficacité optimale. Par contre, l'analyse de *STA* au niveau du circuit est plus complexe.

2.6 Conclusion

Malgré leurs nombreuses qualités, les circuits asynchrones peinent à convaincre l'industrie de la microélectronique. Ils n'existent qu'au travers de projets de recherche ou de quelques startups en quête de différenciateurs qui profitent de la frilosité des industriels face à la complexité supposée de l'asynchrone. S'il est probable que cette situation ne change pas prochainement, l'asynchrone continuera à être utilisé dans quelques niches pour lesquelles il aura de véritables avantages. Il est aussi probable que synchrone et asynchrone se mêlent de plus en plus pour tirer parti des atouts de chacun. Dans cette optique, les circuits à données groupées semblent les plus appropriés. Ils restent très similaires aux

circuits à horloge tout en ayant une robustesse accrue. Mais nous avons identifié que les outils traditionnels étaient souvent mal utilisés, ne facilitant pas cette cohabitation. Nous avons vu que l'analyse temporelle, qui est au cœur de la conception de tout circuit, est particulièrement délicate à mettre en œuvre sur les circuits à données groupées. Pour répondre à cette problématique, nous avons montré avec quelques exemples concrets que les moteurs de STA synchrones manipulent déjà toutes les notions nécessaires à l'analyse des circuits asynchrones. Nous avons aussi proposé une classification des différentes contraintes temporelles s'appliquant à ces circuits, pour aider n'importe quel concepteur à mieux se les approprier.

Les travaux présentés dans ce chapitre ont été partiellement publiés dans une conférence internationale :

- G. GIMENEZ, J. SIMATIC et L. FESQUET. « From Signal Transition Graphs to Timing Closure : Application to Bundled-Data Circuits ». In : *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2019, p. 86-95

Chapitre 3

La méthode LCS : une vraie analyse temporelle statique des circuits à données groupées

Sommaire

3.1	Introduction	52
3.2	Principes de fonctionnement	52
3.2.1	Oscillateurs locaux	53
3.2.2	Horloges pour circuits sans horloge	53
3.2.2.1	Propriétés des horloges	53
3.2.2.2	De l'horloge à l'évènement	54
3.2.3	LCS : séries d'horloges locales	55
3.2.4	Le cas des courses de signaux	57
3.3	Du STG aux LCS en passant par les RTC	59
3.3.1	Du STG aux RTC	60
3.3.2	Casser les boucles	60
3.3.3	Identifier les points de divergence	61
3.3.4	Simplifier les horloges	63
3.3.5	Construire les LCS	65
3.3.6	Restreindre les chemins	66
3.3.7	Générateur de contraintes LCS	67
3.3.8	Comparaison de différents protocoles	68
3.4	LCS et flot d'implémentation physique	71
3.4.1	Comparaison avec la méthode min/max	72
3.4.2	Contraindre le chemin de données	74
3.4.3	Synthèse des arbres locaux	75
3.4.4	Insertion des délais <i>matchés</i>	76
3.4.5	Flot d'implémentation LCS	78
3.5	Conclusion et perspectives	79

3.1 Introduction

Les méthodes d'analyse temporelle statique des circuits asynchrones utilisant des outils commerciaux ont de nombreuses limitations. Elles se basent toutes sur une interprétation et une simplification des exigences temporelles qui dépendent des conditions d'opération. Elles ne permettent donc pas aux outils de CAO d'appréhender complètement les contraintes de temps relatifs (*Relative Timing Constraints*) (RTC) qu'ils doivent implémenter et vérifier. Dans ce chapitre, nous proposons une nouvelle approche nommée méthode des séries d'horloges locales (*Local Clock Sets*) (LCS). Celle-ci permet de décrire précisément les contraintes de temps relatifs des circuits à données groupées avec un formalisme purement synchrone, en définissant un ensemble d'horloges. Cette méthode générique peut s'adapter à n'importe quel protocole et s'appuie sur un fichier de règles simples permettant de générer automatiquement les contraintes temporelles d'un circuit complexe.

La section 3.2 introduit les principes de base utilisés par la méthode LCS. La section 3.3 s'intéresse plus particulièrement à la manière dont peuvent être construits les fichiers de règles des différents protocoles et fait le lien avec un modèle formel tel que les STG. Finalement, la section 3.4 présente les différentes manières d'utiliser ces LCS lors de l'implémentation physique d'un circuit ainsi que le flot complet que nous avons pu développer grâce à ces LCS.

3.2 Principes de fonctionnement

Dans un souci pédagogique, nous nous proposons d'étudier un circuit à données groupées générique. Nous ignorons ainsi les spécificités de l'implémentation des contrôleurs¹ et nous nous concentrons sur les deux principales exigences temporelles de son protocole : les contraintes de *regroupement* et d'*écrasement des données*. Les RTC correspondantes sont définies par les équations (3.1) et (3.2) et sont illustrées respectivement en rouge et en bleu sur la figure 3.1.

$$ctrl_i \mapsto D_{i+1} \prec C_{i+1} \quad (3.1)$$

$$ctrl_{i+1} \mapsto C_{i+1} \prec D_{i+1} \quad (3.2)$$

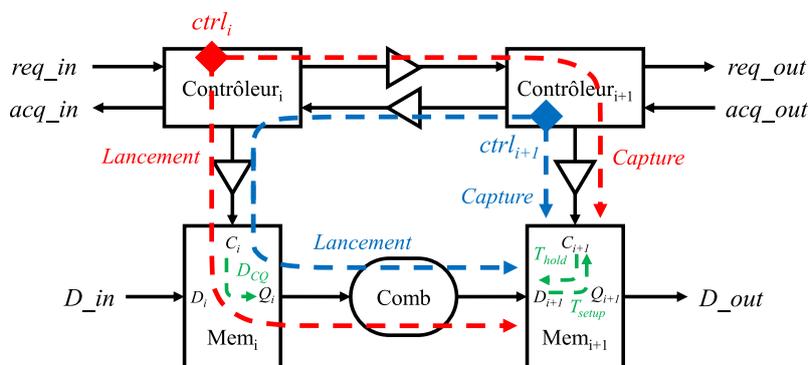


FIGURE 3.1 – RTC au niveau protocole d'un circuit à données groupées générique.

1. Un protocole 2 phases est implicitement utilisé.

3.2.1 Oscillateurs locaux

Pour nous extraire des limitations des méthodes d'analyse min/max, nous proposons de changer de perspective. Plutôt que d'approcher les circuits à données groupées d'un point de vue « canaux de communication », nous proposons de les appréhender comme un « simple » assemblage d'oscillateurs locaux, comme illustré par la figure 3.2.

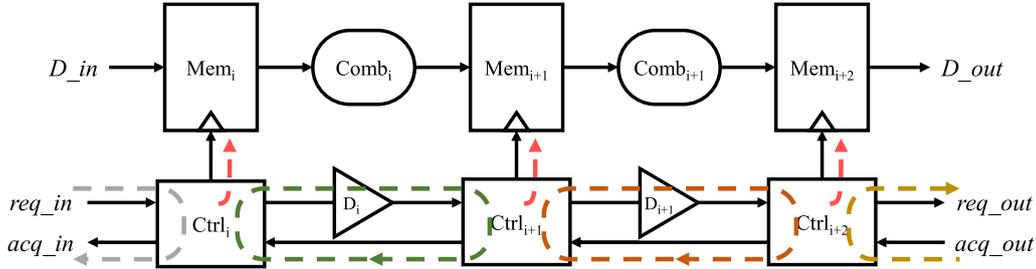


FIGURE 3.2 – Oscillateurs dans les circuits à données groupées.

Ces différents oscillateurs, formés par le rebouclage des lignes de requête et d'acquittement, sont synchronisés deux à deux au niveau des contrôleurs. Chacun d'eux génère une horloge locale avec une fréquence variable, mais qui peut être bornée par une limite haute dépendant principalement du temps de propagation dans la ligne de requête (3.3).

$$F_{osc}^i \leq \frac{1}{2 \times D_i} \quad (3.3)$$

Cette observation érode un peu plus la frontière entre les circuits synchrones et les circuits à données groupées. Ainsi l'idée de base de la méthode LCS est très simple : utiliser une sémantique purement synchrone – la définition d'horloges – pour définir ces différents oscillateurs locaux et ainsi décrire le fonctionnement des circuits à données groupées.

3.2.2 Horloges pour circuits sans horloge

3.2.2.1 Propriétés des horloges

Les horloges sont des éléments singuliers dans les outils de STA. Elles sont utilisées pour décrire le protocole synchrone et, plus particulièrement, la source du signal de synchronisation et sa fréquence d'oscillation. Cela leur confère des propriétés très intéressantes :

- 1) Les horloges définissent les points de départ de tout chemin temporel, qu'il soit de capture ou de lancement. Une horloge peut donc être utilisée pour spécifier le POD d'une RTC. Par exemple, deux horloges pourront être utilisées pour définir les points $ctrl_i$ et $ctrl_{i+1}$ de la figure 3.1. Nous parlons alors d'horloge racine.
- 2) Les vérifications décrites dans les fichiers d'abstraction temporelle relient les deux parties d'un POC. Par exemple, les temps de propagation T_{setup} et T_{hold} , comme illustrés à la figure 3.1, relient les entrées D et C d'un élément séquentiel. Ces entrées forment le POC des contraintes de regroupement et d'écrasement des données. Mais ces vérifications temporelles ne sont activées que lorsqu'une horloge atteint l'entrée de contrôle de cet élément séquentiel. Il en est de même pour les arcs de propagation comme le temps T_{CQ} . En utilisant des horloges, nous pouvons conserver la relation entre le chemin de données et la partie contrôle.
- 3) Comme une horloge définit un point départ pour le moteur de STA, tout chemin temporel traversant ce point est automatiquement bloqué. Les horloges peuvent donc être utilisées pour casser les boucles combinatoires qui perturbent les outils de STA. En sélectionnant soigneusement la source des différentes horloges, les boucles locales

et architecturales peuvent être cassées. Dans le cas où une horloge est utilisée uniquement pour bloquer une boucle combinatoire, nous l'appelons horloge de blocage. La [figure 3.3a](#) illustre la coupure générée par une telle horloge dans des chemins de propagation.

- 4) Contrairement aux commandes de `set_disable_timing`, la définition d'une horloge racine ou de blocage ne désactive pas d'arcs temporels. Il est donc toujours possible d'analyser un chemin coupé par sa source. Un autre type d'horloge peut être utilisé à cet escient : les horloges générées (`create_generated_clock`). Elles permettent de propager un évènement au travers des points de blocages et donc de décrire les différents chemins temporels des `RTC` à implémenter. Nous appelons ces dernières les horloges de propagation d'évènements (*Event Propagation Clocks*) (`EPC`). La [figure 3.3b](#) montre l'utilisation d'une `EPC` (flèche en pointillés) permettant de compléter le chemin de propagation (en vert) au-delà d'une horloge de blocage (matérialisée par le losange gris).

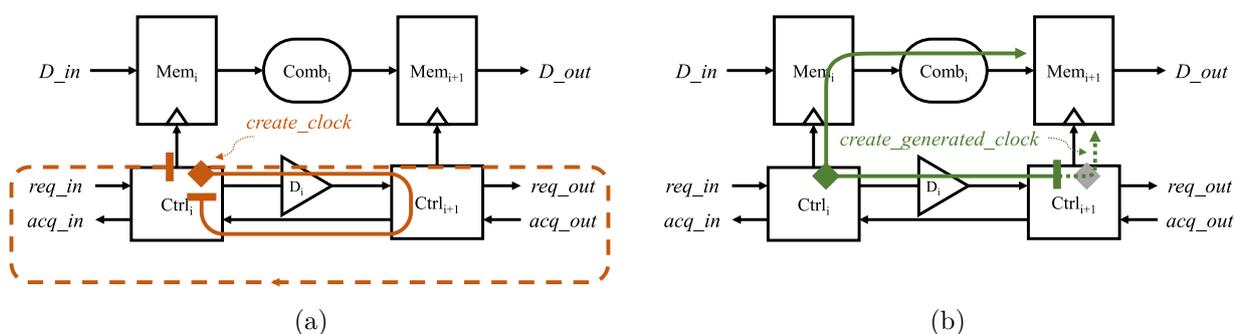


FIGURE 3.3 – Différentes utilisations d'horloges dans les circuits de contrôle. (a) Horloge de blocage des boucles locales (continue) et architecturales (discontinue) et (b) horloge racine (continue) et de propagation (discontinue).

En plus de résoudre le problème des boucles combinatoires, l'utilisation d'horloges permet de décrire tous les éléments constitutifs d'une `RTC`. Le point de divergence correspond à la source d'une horloge racine. Les chemins relatifs peuvent être spécifiés à l'aide d'horloges générées et les points de convergence sont activés lorsque l'une ou l'autre de ces horloges atteint un élément séquentiel.

Par ailleurs, en précisant la période des horloges racines, nous pouvons définir la fréquence de chaque oscillateur formé entre deux étages d'un circuit. Nous pouvons ainsi contraindre indépendamment chacune des parties du chemin de données. Dans cette configuration, l'outil sera par contre incapable de vérifier que les délais *matchés* couvrent bien cette contrainte temporelle. Pour faire cette vérification, l'horloge doit être transformée en évènement.

3.2.2.2 De l'horloge à l'évènement

La [figure 3.4a](#) illustre le comportement standard des outils de `STA` synchrones : ils vérifient les temps de *setup* entre deux fronts actifs successifs de l'horloge. Cette figure présente un étage de pipeline synchrone ainsi que les chemins de lancement (rouge) et de capture (bleu) de cette contrainte. La [figure 3.4b](#) présente les chronogrammes du signal d'horloge vu par chacun des éléments séquentiels et de la donnée en entrée du deuxième élément mémorisant. Alors que le chemin de lancement est initié par le premier front de l'horloge, le chemin de capture démarre une période T_{clk} plus tard. Sur le chronogramme, nous constatons en vert le *slack* positif qui résulte de cette vérification de temps de *setup*.

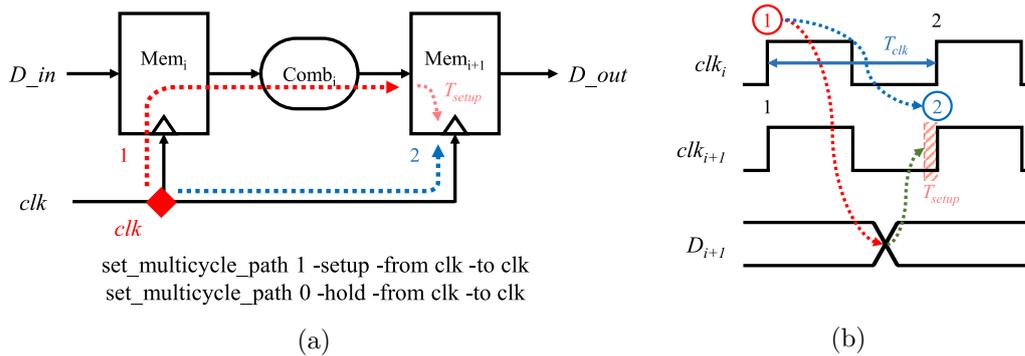


FIGURE 3.4 – Vérification de *setup* dans un circuit synchrone. (a) Illustration des chemins et (b) chronogrammes des signaux correspondants.

Pour décrire le principe d'un circuit à données groupées il faut, au contraire, contraindre l'outil à utiliser le même front de l'horloge — ou évènement — pour le chemin de capture et le chemin de lancement. Les flèches bleues de la figure 3.5 décrivent ce fonctionnement. La commande `set_multicycle_path` du format SDC permet de contraindre les outils de STA à adopter l'un ou l'autre de ces fonctionnements. Elle définit le nombre de cycles d'horloge à prendre en compte lors de la vérification.

Dans le protocole synchrone, une période d'horloge est accordée pour le transfert des données d'un étage à un autre du circuit. Cela correspond à un chemin « multi-cycles » de 1 pour les temps de *setup* et de 0 pour les temps de *hold*². Mais dans les circuits à données groupées, la période d'horloge ne doit pas entrer en compte. Ainsi, l'utilisation de multi-cycles 0 sur les vérifications de *setup* et de multi-cycles -1 pour les vérifications de *hold*³ permet de se conformer au fonctionnement de ces circuits.

Le chronogramme de la figure 3.5b montre cependant que la définition d'une contrainte de multi-cycle n'est pas suffisante. Le délai *matché* n'est pas pris en compte et le *slack* résultant est toujours négatif. Comme illustré sur le chronogramme figure 3.5c, il faut utiliser la commande `set_propagated_clock` pour forcer l'outil à prendre réellement en compte le temps de propagation sur la ligne de requête et ainsi vérifier l'adéquation entre les délais *matchés* et le chemin combinatoire.

Finalement, en combinant les commandes `set_multicycle_path` et `set_propagated_clock`, une horloge initialement caractérisée par sa fréquence d'oscillation peut être transformée en un simple évènement non-périodique. Elle peut alors être utilisée pour décrire n'importe quel chemin de propagation dans les blocs de contrôle des circuits à données groupées.

3.2.3 LCS : séries d'horloges locales

En utilisant uniquement des horloges, nous pouvons traduire les RTC d'un circuit dans un langage compréhensible par les outils de STA. Il faut, pour cela, décrire les chemins reliant chaque POD à ses points de convergence précoce et tardif. Nous appelons une série d'horloges locales (*Local Clock Set*) (LCS) l'ensemble des horloges utilisées pour décrire une

2. Les vérifications de *hold* ne mettent pas en jeu la fréquence de l'horloge!

3. Comme pour le *setup*, les vérifications de *hold* doivent être faites en prenant en compte la propagation du même front d'horloge. Intuitivement un multi-cycle 0 semble approprié. Cependant les vérifications de *hold* sont faites de manière conservatrice relativement à la contrainte de *setup*. Par défaut, lorsqu'un multi-cycle 0 est défini pour le *setup*, les vérifications de *hold* sont faites sur le front précédent et impliquent une période d'horloge. Définir un multi-cycle à -1 permet de s'affranchir de ce phénomène.

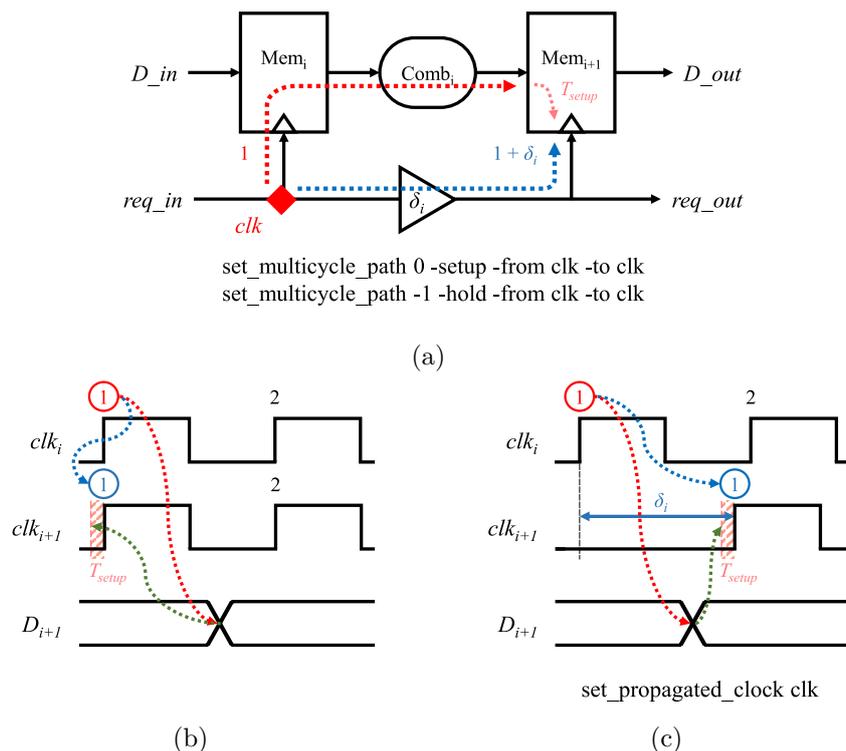


FIGURE 3.5 – Transformation d'une horloge en un évènement. (a) Illustration des chemins de *setup*, (b) chronogrammes des signaux correspondants et (c) chronogrammes lorsque les horloges sont propagées.

RTC. Pour illustrer comment chaque **RTC** peut être décomposée en **LCS**, nous prendrons l'exemple du circuit non linéaire présenté dans la figure 3.6.

Ce circuit à données groupées est composé d'un pipeline de trois étages dont le deuxième est rebouclé sur lui-même. Un contrôleur factice ($Ctrl_{dmy}$) est inséré au niveau du bloc de contrôle pour garantir la vivacité du circuit⁴. En plus des boucles combinatoires locales entre chacun des contrôleurs, il contient donc des boucles architecturales dues à ce chemin de rétroaction. Toutes ces boucles peuvent être cassées à l'aide des différentes horloges racines et de blocage schématisées par les losanges de couleurs. Si nous nous concentrons sur l'horloge racine associée au deuxième contrôleur (clk_2), nous voyons qu'elle atteint l'élément séquentiel associé (Mem_2) et active le chemin de lancement à travers la logique combinatoire jusqu'à l'entrée D_3 de l'élément séquentiel suivant (ligne discontinue rouge). Cette horloge se propage aussi le long de la ligne de requête, traverse le délai *matché* D_2 , puis se retrouve bloquée par la source de l'horloge clk_3 (ligne continue rouge). Une **EPC**, $capture_2$, peut être définie pour continuer à propager le chemin au-delà de ce blocage et atteindre le point de convergence tardif C_3 (ligne rouge en pointillés). Ainsi, la **RTC** de *regroupement* entre les deux derniers étages du circuit est activée. Elle est décrite par une première **LCS** formée des horloges clk_2 et $capture_2$. Un découpage similaire peut être appliqué à la **RTC** d'*écrasement de données* entre les deux premiers étages du circuit (en bleu sur la figure 3.6). L'horloge racine clk_2 se propage le long de la ligne d'acquiescement. Elle est stoppée par l'horloge clk_1 , mais est propagée par l'**EPC** $lancement_2$. La troisième **RTC** est plus complexe (en violet). Elle correspond à la contrainte de *regroupement* du

4. C'est-à-dire, le circuit de contrôle ne présente pas de blocage. Suivant le protocole utilisé, une boucle architecturale doit, pour être vivace, contenir au minimum deux ou trois contrôleurs.

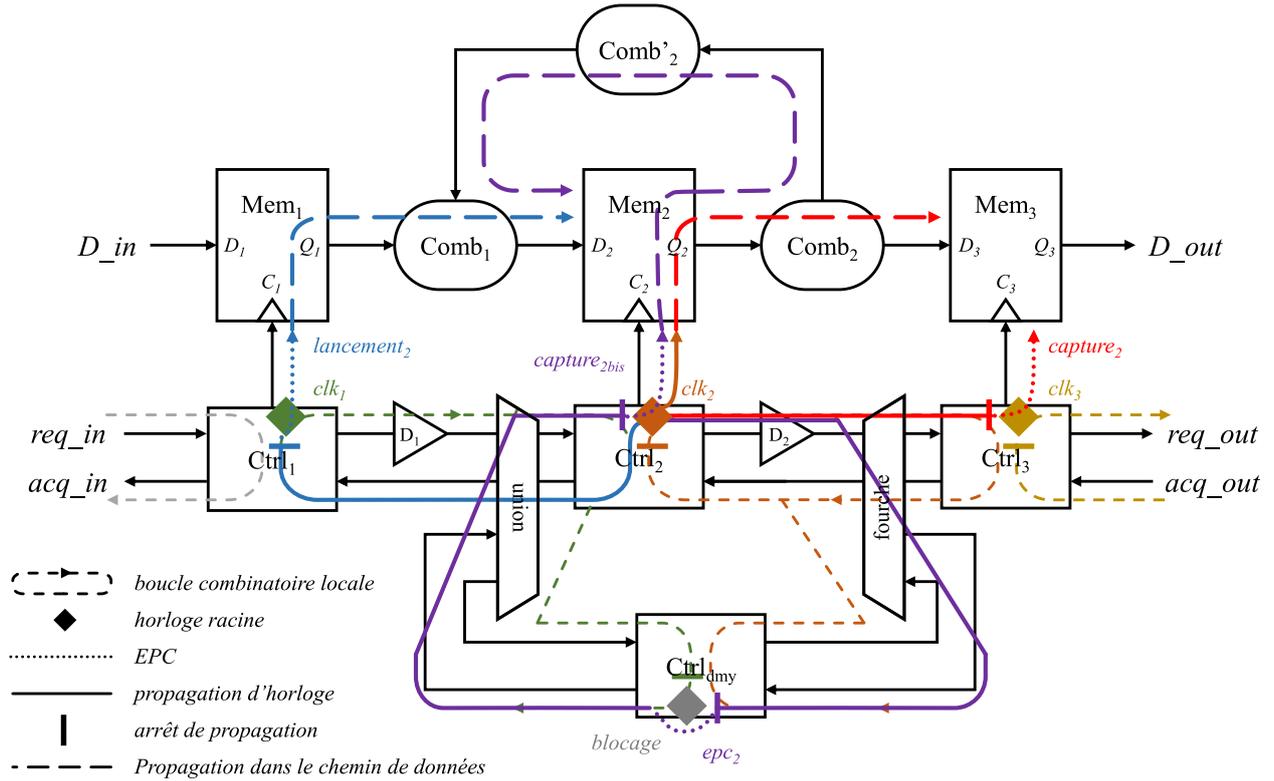


FIGURE 3.6 – Décomposition sous forme de LCS de 3 RTC d'un circuit à données groupées non linéaire.

deuxième l'étage sur lui-même et met en jeu la boucle de rétroaction. La LCS correspondante implique une horloge générée supplémentaire (epc_2) qui permet de propager l'évènement au-delà de l'horloge de blocage ($blocage$). Finalement, la concordance entre chaque RTC et son LCS est résumée par les équations (3.4) à (3.6).

$$LCS_{rouge} = \{clk_2, capture_2\} \Leftrightarrow clk_2 \mapsto D_3 \prec C_3 \quad (3.4)$$

$$LCS_{bleu} = \{clk_2, lancement_2\} \Leftrightarrow clk_2 \mapsto C_2 \prec D_2 \quad (3.5)$$

$$LCS_{violet} = \{clk_2, epc_2, capture_{2bis}\} \Leftrightarrow clk_2 \mapsto D_2 \prec C_2 \quad (3.6)$$

L'équation (3.7) donne finalement la définition générique d'une LCS. Elle est composée d'une horloge racine, identifiée par un soulignement \underline{clk} , suivie et/ou précédée d'une ou plusieurs horloges de propagation d'évènement. Les dernières de ces EPC sont qualifiées d'horloge de lancement ou de capture et sont respectivement identifiées par les symboles $\underline{\underline{clk}}$ et $\underline{\underline{clk}}$. Suivant la topologie de la LCS, l'horloge racine peut aussi faire office d'horloge de lancement ou de capture et est alors respectivement identifiée par les symboles $\underline{\underline{clk}}$ et $\underline{\underline{clk}}$.

$$LCS = \{\underline{\underline{Lancement}}, EPC_{Li}, \dots, EPC_{L1}, \underline{\underline{Racine}}, EPC_{C1}, \dots, EPC_{Ci}, \underline{\underline{Capture}}\} \quad (3.7)$$

3.2.4 Le cas des courses de signaux

À la différence des RTC impliquant un élément séquentiel, les RTC liées aux cellules combinatoires ne s'appuient pas sur les modèles d'abstraction pour décrire leurs POC. Il est nécessaire d'utiliser d'autres commandes SDC (`set_data_check`) pour définir ces points. Cependant, à l'instar de vérifications de *clock-gating* faites par les outils synchrones,

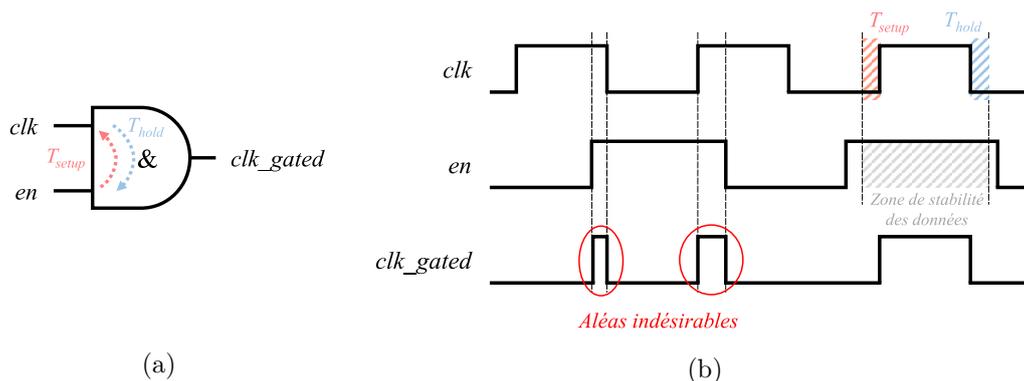


FIGURE 3.7 – Principe de fonctionnement des vérifications de *clock-gating* ; exemple d'une porte ET. (a) schéma de l'interaction horloge/donnée et (b) chronogramme correspondant.

la définition d'horloges dans un circuit à données groupées permet de retrouver automatiquement certains de ces points de convergence.

Si le chemin de données des circuits synchrones s'accommode des aléas liés aux commutations dans les portes logiques, ceux-ci doivent impérativement être évités sur l'horloge. Pour garantir l'intégrité de ce signal gérant le protocole, les outils de STA infèrent automatiquement une vérification de *clock-gating* dès lors qu'ils détectent une interaction avec un signal de donnée. Par exemple, la figure 3.7a présente une interaction entre une horloge (*clk*) et une donnée (*en*) au niveau d'une porte ET. Nous voyons sur la figure 3.7b que cette interaction peut générer des aléas en sortie de la porte logique. Pour s'en prémunir, le signal de donnée doit respecter une zone de stabilité (en gris) qui dépend de la fonctionnalité de la cellule logique⁵. À noter que l'interaction entre deux horloges est aussi interprétée par les outils de STA comme du *clock-gating*. Dans ce cas, deux vérifications sont faites, chacune des horloges étant à son tour considérée comme une donnée. Plusieurs conditions existent pour que le moteur de STA déduise automatiquement une vérification de *clock-gating* :

- La fonction implémentée par la cellule de convergence doit être *unate* (négatif ou positif) afin de pouvoir identifier une zone de stabilité. C'est le cas des portes ET et OU. Par contre, les portes OU-exclusif, multiplexeur et généralement toutes les portes plus complexes sont dites non-*unate*.
- Un chemin temporel doit exister depuis la sortie de la cellule de convergence vers un point « consommateur d'horloge ». Autrement dit, vers l'entrée de contrôle d'un élément séquentiel ou vers la source d'une horloge générée.
- Un point de divergence doit exister entre les chemins arrivant aux deux entrées de la cellule de convergence et ces chemins doivent être actifs.

En déclarant des horloges dans le bloc de contrôle d'un circuit à données groupées, nous héritons de cette spécificité des outils de STA qui permet de retrouver automatiquement certaines RTC combinatoires. C'est par exemple le cas des RTC faisant interagir un signal issu du chemin de données et un signal de contrôle. Celles-ci se retrouvent souvent dans les primitives de branchements conditionnels comme celle présentée en figure 3.8a. La déclaration d'une horloge *clk* sur le contrôleur gérant le canal de sélection va permettre à l'outil de déduire la vérification de *clock-gating* illustrée par la double flèche. Celle-ci peut

5. Pour une porte OU, cette zone correspond à l'impulsion basse de l'horloge

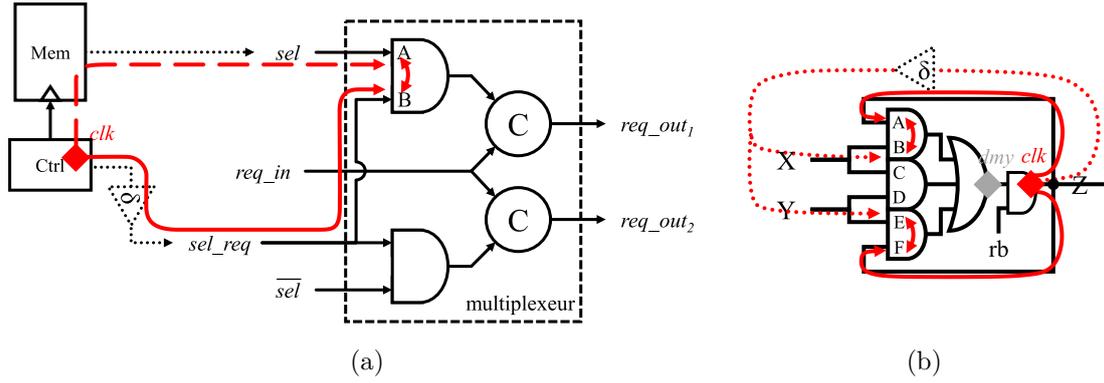


FIGURE 3.8 – Inférence automatique de *clock-gating* dans les circuits à données groupées. (a) Dans un multiplexeur WCHB (*event gating*) et (b) dans une cellule de Muller (*course d'environnement*).

être exprimée par l'équation de RTC (3.8). Pour garantir la tenue de cette contrainte, un élément de délai δ pourra être ajouté dans sur le chemin de requête.

$$clk \mapsto A \prec B + \delta \quad (3.8)$$

Un exemple de course de signaux dans l'implémentation à base d'une porte majorité de la cellule de Muller est donné en figure 3.8b. En déclarant l'horloge clk , et à condition qu'il existe un chemin temporel dans l'environnement, tel que schématisé par les flèches en pointillés, les deux RTC des équations (3.9) et (3.10) sont déduites par les outils de STA. Le délai δ permet de garantir la tenue de ces contraintes. À noter qu'il est nécessaire d'ajouter une horloge générée intermédiaire dmy (en gris), appelée horloge d'activation⁶. Elle définit un point consommateur d'horloge en aval de la cellule de convergence et active l'inférence automatique de ces RTC.

$$clk \mapsto A \prec B + \delta \quad (3.9)$$

$$clk \mapsto F \prec E + \delta \quad (3.10)$$

3.3 Du STG aux LCS en passant par les RTC

Trouver la liste des RTC associées à un protocole et en déduire les LCS puis les contraintes correspondantes au format SDC s'avère être une tâche très fastidieuse. Les RTCs présentées au chapitre 2 donnent parfois la position explicite des points de convergence (les éléments séquentiels) mais identifient, au mieux, un sous-ensemble de transitions pouvant être support de leur POD. Un concepteur expérimenté pourrait retrouver manuellement les informations manquantes. Toutefois, le risque d'erreur est important. Celui-ci devient considérable lorsqu'un contrôleur complexe ou des structures de branchements conditionnels sont utilisés. Pour simplifier le processus, nous proposons une méthodologie partant d'un modèle formel des protocoles, le STG, pour retrouver facilement les RTC, puis les LCS associées. Nous utilisons le protocole Maximus [Sim+16] pour illustrer cette méthodologie.

6. Sans cette horloge, aucun point consommateur n'est détecté par l'outil car les chemins temporels sont bloqués par la source de l'horloge clk .

3.3.1 Du STG aux RTC

Tout commence avec un modèle formel du protocole. Le **STG**, présenté en [figure 3.9a](#), décrit les successions de transitions des entrées et sorties d'un contrôleur Maximus. Le canal d'entrée L (respectivement de sortie R) est constitué d'une ligne de requête L^r (resp. R^r) et d'une ligne d'acquiescement L^a (resp. R^a). Ce protocole nécessite un état interne s permettant d'assurer le découplage entre les deux canaux. Ce STG peut être synthétisé avec l'outil Petrifly pour obtenir l'implémentation niveau porte de la [figure 3.9b](#).

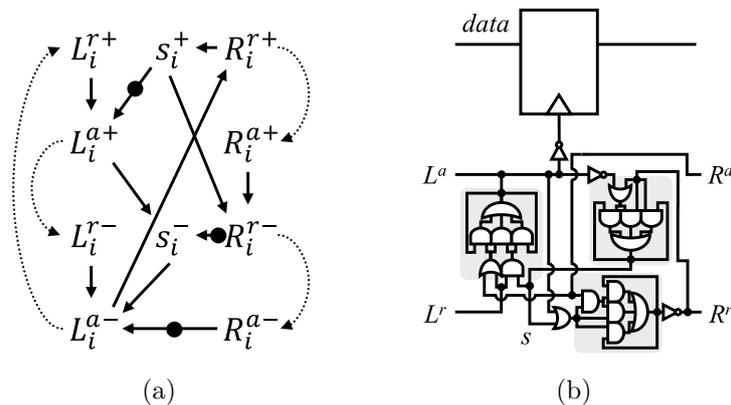


FIGURE 3.9 – Protocole à données groupées Maximus. (a) STG et (b) implémentation au niveau portes logiques.

En plus de l'implémentation au niveau porte, Petrifly donne les différentes **RTC** internes s'appliquant sur le contrôleur. Celles-ci viennent compléter la liste des **RTC** liées à l'élément séquentiel. Au final, douze **RTC** s'appliquent sur cette primitive. La moitié d'entre elles est liée à l'implémentation de la cellule de Muller. Nous nous intéresserons par la suite à 4 de ces **RTC** : *regroupement*, *écrasement des données*, *temps de phase minimum* et une *course de chemins locaux*⁷. Elles sont respectivement données par les [équations \(3.11\)](#) à [\(3.14\)](#).

$$pod \mapsto data_i \prec ctrl_i^+ - \delta \quad (3.11)$$

$$pod \mapsto ctrl_i^+ \prec data_i - \delta \quad (3.12)$$

$$pod \mapsto ctrl_i^- \prec ctrl_i^+ - \delta \quad (3.13)$$

$$L_i^{a+} \mapsto or/A^+ \prec or/B^- - \delta \quad (3.14)$$

3.3.2 Casser les boucles

Pour pouvoir faire l'analyse temporelle statique du contrôleur, il est nécessaire d'identifier et de casser les boucles combinatoires. Le **STG** est particulièrement adapté à cette tâche. Ce graphe peut être parcouru algorithmiquement pour trouver très rapidement ces boucles. Mais il a surtout l'avantage, comparativement à d'autres modélisations formelles, d'être très visuel. Il facilite une identification manuelle des boucles combinatoires. Nous comptons 9 boucles dans un unique contrôleur, sans compter celles liées aux rétroactions internes des cellules de Muller, ni les boucles architecturales. La [figure 3.10a](#) présente 3 de ces boucles sur le **STG** du protocole. Les flèches relient deux transitions du même signal

7. Dans les différents **STG**, les transitions $ctrl_i^-$, or_i/A^+ et or_i/B^- ne sont pas représentées. Elles sont respectivement équivalentes aux transitions L_i^{a+} , L_i^{a-} et s_i^- .

mais de sens opposés, formant ainsi des boucles combinatoires. Les mêmes boucles sont aussi reportées sur la *netlist* de la figure 3.10b. La suite de transitions $R_i^{r+} \rightarrow R_i^{a+} \rightarrow R_i^{r-}$ implique l'environnement. Elle peut être qualifiée de boucle de protocole. Les deux autres, $s_i^+ \rightarrow L_i^{a+} \rightarrow s_i^-$ et $L_i^{a-} \rightarrow R_i^{r+} \rightarrow s_i^+ \rightarrow L_i^{a+}$, sont des boucles internes du contrôleur.

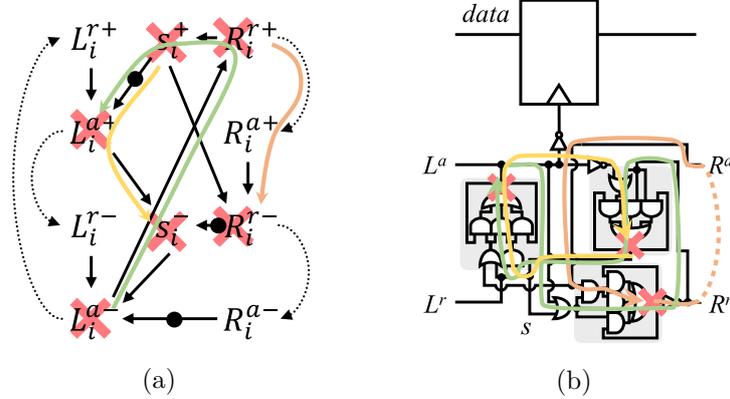


FIGURE 3.10 – Trois boucles combinatoires dans le protocole Maximus, reportées sur (a) le STG et (b) la *netlist* au niveau portes.

Le STG permet de retrouver rapidement toutes les boucles. C'est un modèle plus simple à manipuler que la *netlist* au niveau portes. Il permet aussi de trouver aisément les points communs à chacune de ces boucles qui permettront de les casser. En considérant toutes les boucles, trois points sont candidats : L^a , R^r et s . Deux sont normalement suffisants pour casser les 9 boucles identifiées. Cependant, les trois sont conservés car ils permettent également de casser les boucles internes aux cellules de Muller. Ces points de blocages sont représentés par les croix rouges sur la figure 3.10.

3.3.3 Identifier les points de divergence

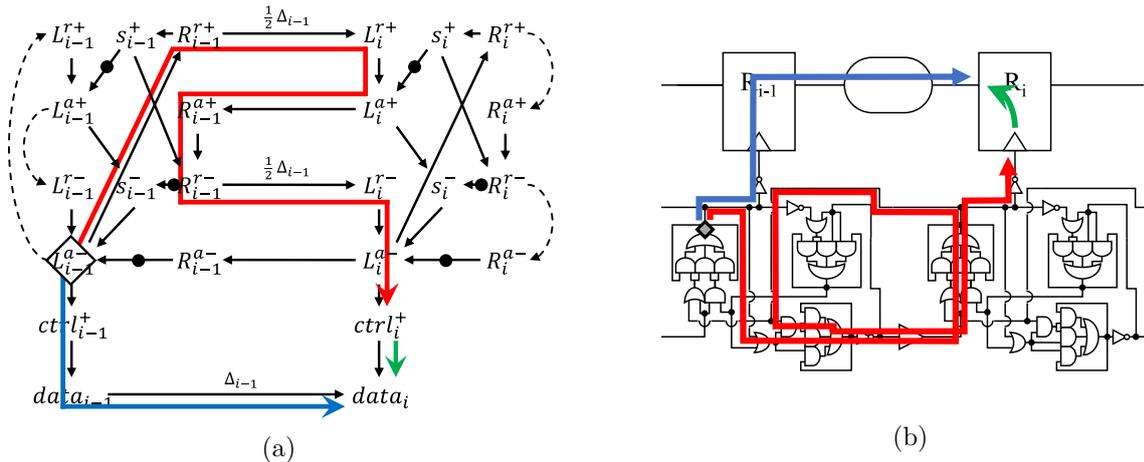


FIGURE 3.11 – Contrainte de *regroupement* du protocole Maximus, reportée sur (a) le STG et (b) la *netlist* au niveau porte.

En considérant les équations (3.11) à (3.14) nous constatons que trois d'entre elles utilisent un point de divergence générique. Les positions de ces POD sont inconnues car ces RTC impliquent des chemins d'environnement. Il est nécessaire de s'intéresser à un

assemblage de contrôleurs pour pouvoir les retrouver. Pour bien expliciter les **RTC**, les **STG** des figures 3.11 à 3.14 sont étendus pour inclure les éléments séquentiels et le chemin de données entre deux étages d'un circuit utilisant le protocole Maximus. Chacune de ces figures correspond à une des quatre **RTC** présentées précédemment. Le **POC** de ces contraintes est à chaque fois représenté par la flèche verte reliant les points précoce et tardif. En parcourant le **STG**, un point de départ commun peut être trouvé pour les chemins de capture et de lancement. Par exemple, pour la **RTC** de *regroupement* illustrée par la figure 3.11, nous identifions le **POD** comme étant à la transition L_{i-1}^{a-} de l'étage précédent ($i - 1$). Les chemins reliant le **POD** aux **POC** peuvent alors être projetés sur la *netlist* en suivant la liste des transitions parcourues :

- *regroupement*

- lancement : $L_{i-1}^{a-} \rightarrow ctrl_{i-1}^+ \rightarrow data_{i-1} \rightarrow data_i$
- capture : $L_{i-1}^{a-} \rightarrow R_{i-1}^+ \rightarrow L_i^{r+} \rightarrow L_i^{a+} \rightarrow R_{i-1}^{a+} \rightarrow R_{i-1}^{r-} \rightarrow L_i^{r-} \rightarrow L_i^{a-} \rightarrow ctrl_i^+$

Le même procédé peut être appliqué aux trois autres **RTC**. Le **POD** de la contrainte d'*écrasement des données* est aussi la transition L_i^{a-} mais, cette fois-ci, de l'étage courant. Les deux dernières **RTC** ont le même **POD** : L_i^{a+} . Finalement, les chemins de lancement et de capture sont les suivants :

- *écrasement de données*

- lancement : $L_i^{a-} \rightarrow R_{i-1}^{a-} \rightarrow L_{i-1}^{a-} \rightarrow ctrl_{i-1}^+ \rightarrow data_{i-1} \rightarrow data_i$
- capture : $L_i^{a-} \rightarrow ctrl_i^+$

- *temps de phase minimum*

- lancement : $L_i^{a+} \rightarrow ctrl_i^-$
- capture : $L_i^{a+} \rightarrow R_{i-1}^{a+} \rightarrow R_{i-1}^{r-} \rightarrow L_i^{r-} \rightarrow L_i^{a-} \rightarrow ctrl_i^+$

- *course de chemins locaux*

- lancement : $L_i^{a+} \rightarrow or_i/A^+$
- capture : $L_i^{a+} \rightarrow s_i^- \rightarrow or_2/B^-$

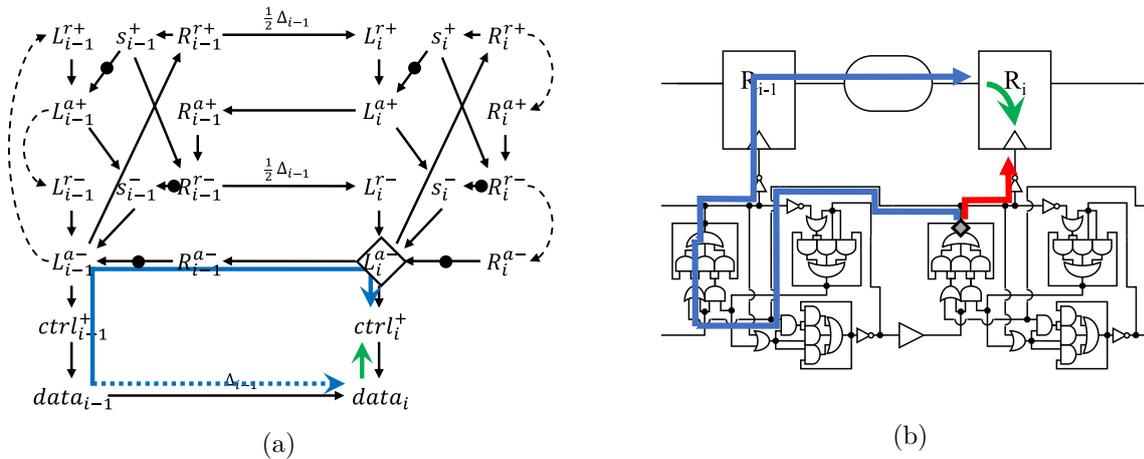


FIGURE 3.12 – Contrainte de *écrasement de données* du protocole Maximus, reportée sur (a) le STG et (b) la netlist au niveau porte.

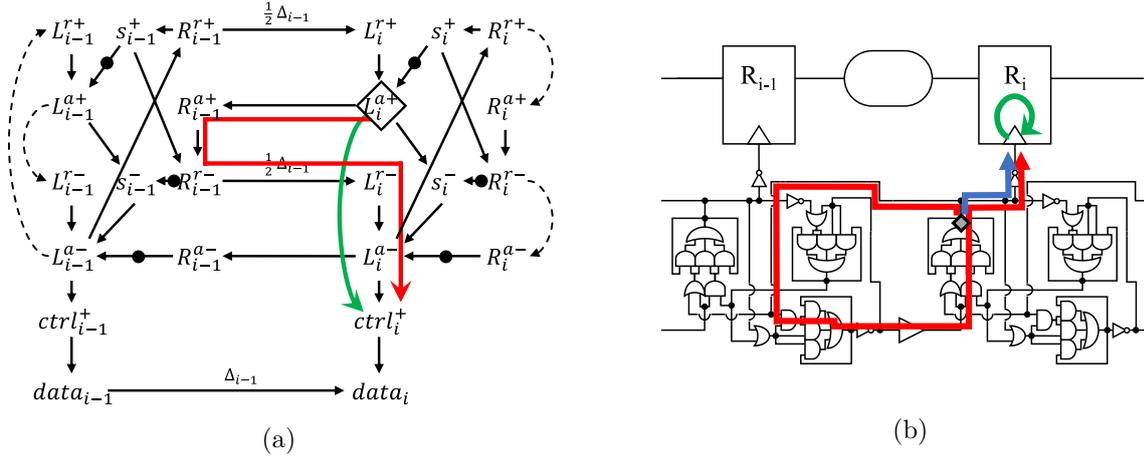


FIGURE 3.13 – Contrainte de *temps de phase minimum* du protocole Maximus, reportée sur (a) le STG et (b) la netlist au niveau porte.

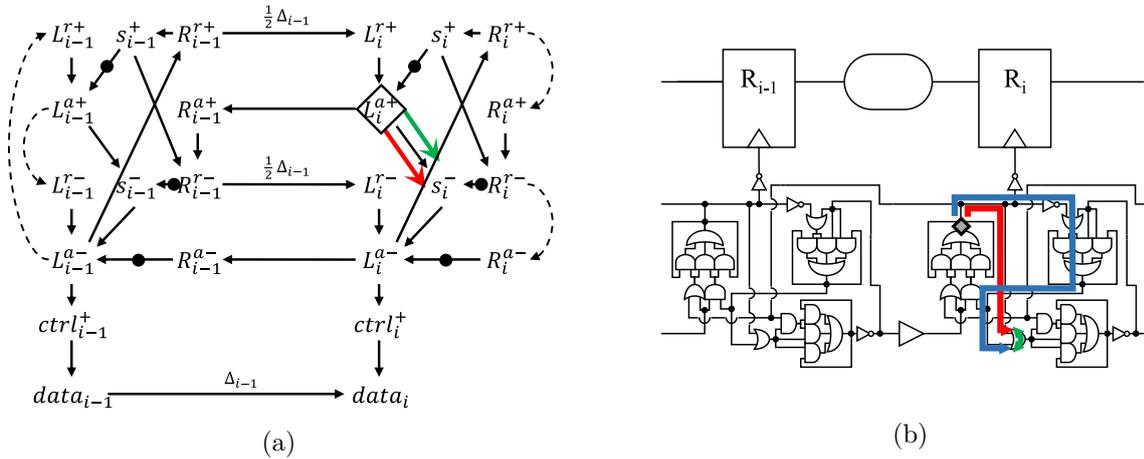


FIGURE 3.14 – Contrainte de *course de chemins locaux* du protocole Maximus, reportée sur (a) le STG et (b) la netlist au niveau porte.

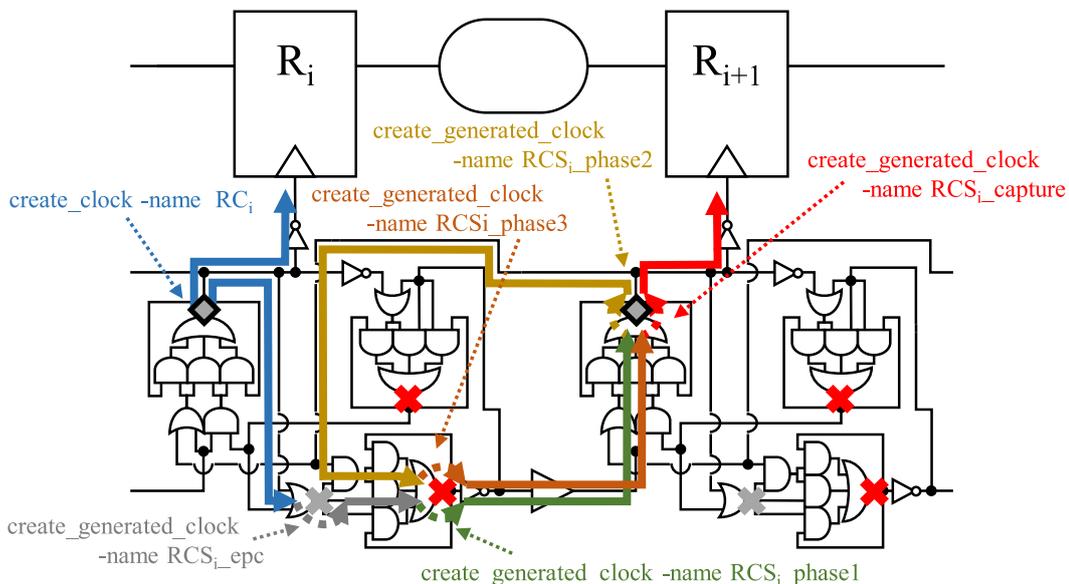
3.3.4 Simplifier les horloges

Nous utilisons des horloges racines pour définir le **POD** de chacune des **RTC**. Pour les 4 contraintes étudiées, les **POD** sont tous positionnés sur le même point physique : L^a , la sortie d’acquiescement du canal d’entrée du contrôleur. Il correspond également à l’un des points de blocage identifié précédemment pour casser les boucles. La simplification est donc triviale. L’une des horloges de blocage et toutes les horloges racines peuvent être fusionnées.

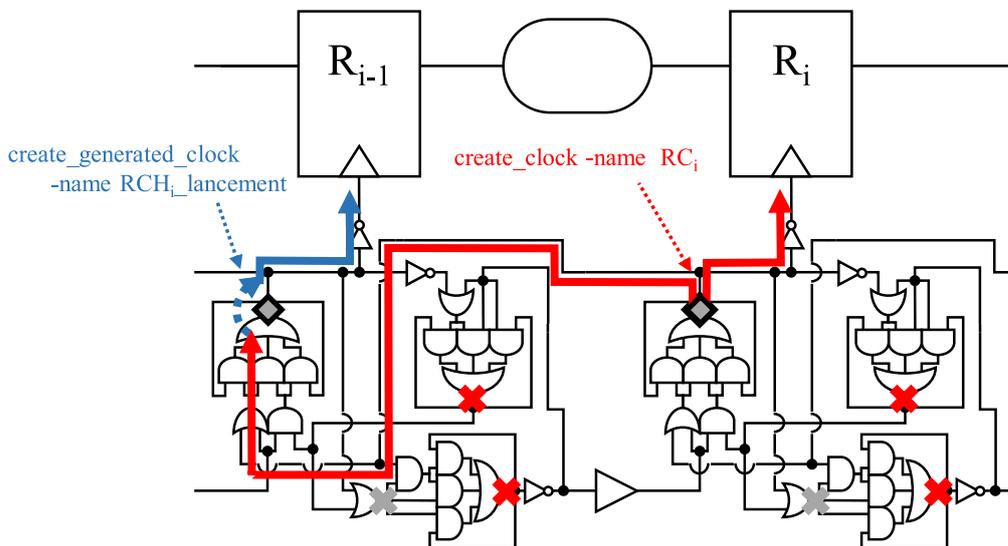
Dans des situations plus complexes, il est possible de déplacer l’horloge source d’une **LCS** en amont de son **POD** pour pouvoir réduire le nombre d’horloge. Dans ce cas, les outils de **STA** reconnaissent automatiquement le **POD** d’origine comme le dernier point commun entre les chemins de lancement et de capture. Les techniques de réduction du pessimisme lié aux reconvergences d’horloges (*Clock Reconvergence Pessimism Removal*) (**CRPR**) embarquées dans les outils de **STA** permettent à l’analyse temporelle de ne pas être impactée.

Dans notre cas, trois horloges sont donc suffisantes. Les 4 **POD** et les 3 points de

blocage peuvent être implémentés avec une horloge racine sur L^a et deux horloges de blocage sur R^r et s . Il est cependant nécessaire de déclarer une quatrième horloge d'activation afin d'ajouter un point de consommation activant l'inférence automatique de la *course de chemins locaux*. Cette horloge peut idéalement être positionnée en sortie de la porte OU, support de cette RTC.



(a)



(b)

FIGURE 3.15 – Décomposition des RTC en LCS. (a) *regroupement* et (b) *écrasement de données*.

3.3.5 Construire les LCS

Une fois les sources des horloges racines et de blocage identifiées, l'équivalent **LCS** de chaque **RTC** peut être construit. En partant des horloges racines, nous parcourons les chemins relatifs de lancement et de capture : dès qu'une autre source d'horloge est rencontrée, une **EPC** doit être créée de manière à propager l'évènement, et ce, jusqu'à ce que le POC soit atteint.

Les figures 3.15 et 3.16 présentent la décomposition **LCS** de chacune des 4 **RTC** analysées précédemment. Cette décomposition est illustrée sur la *netlist* au niveau portes, mais peut également être faite au niveau du **STG**. La contrainte de *regroupement* est, par exemple, décomposée sur la figure 3.15a. L'horloge racine RC_i (en bleu) est déclarée au niveau du **POD** L^a . Elle se propage jusqu'à l'entrée de contrôle de l'élément séquentiel R_i , le **POC** précoce, activant ainsi le chemin de données et complétant le chemin de lancement. Simultanément, cet évènement se propage sur le chemin de capture et rencontre la source de l'horloge d'activation à la sortie de la porte OU. Une première **EPC**, RCS_i_epc est créée (en gris). Celle-ci est très vite arrêtée par une horloge de blocage. Une nouvelle horloge générée est créée (en vert). Nous l'appelons RCS_i_phase1 car elle correspond à la propagation du front montant de la requête, première phase du protocole. Cette horloge se propage jusqu'à l'horloge racine de l'étage suivant. Une nouvelle **EPC** est créée (en bronze). Nous l'appelons RCS_i_phase2 puisqu'elle correspond à l'envoi du front montant de l'acquiescement. Ensuite, l'horloge modélisant la phase 3 (en marron) est générée puis l'horloge de capture (en rouge) vient enfin atteindre le **POC** tardif avec la bonne polarité (front descendant) et complète le chemin de capture. La **LCS** correspondant à la **RTC** de regroupement est ainsi composée de 6 horloges. Elle peut s'écrire sous la forme :

$$LCS_{regroupement} = \{ \underleftarrow{RC_i}, RCS_i_epc, RCS_i_phase1, RCS_i_phase2, RCS_i_phase3, \underline{RCS_i_capture} \} \quad (3.15)$$

La *course de chemins locaux* est décomposée sur figure 3.16a. Deux solutions sont possibles pour implémenter cette **RTC**. Elle peut être explicitement définie en utilisant la commande `set_data_check`, ou être automatiquement inférée par l'outil de **STA** si une horloge d'activation est utilisée. Dans un cas comme dans l'autre, la même décomposition en horloge peut être faite :

$$LCS_{course} = \{ \underline{RC_i}, \underline{RCI_i_capture} \} \quad (3.16)$$

Enfin, les deux autres **RTC**, d'*écrasement de données* (figure 3.15b) et de *temps de phase minimum* (figure 3.16b), se décomposent respectivement selon les équations (3.17) et (3.18).

$$LCS_{écrasement} = \{ \underline{RCH_i_lancement}, \underline{RC_i} \} \quad (3.17)$$

$$LCS_{phase} = \{ \underline{RC_i}, \underline{RCPL_i_phase3}, \underline{RCPL_i_capture} \} \quad (3.18)$$

Finalement, pour décrire les 4 **RTC** étudiées, 13 horloges par étage du circuit sont nécessaires⁸. Une seule horloge racine est définie par étage, celle-ci étant partagée entre les différentes **LCS**. Par soucis de clarification il est aussi possible de découpler les **LCS** en déclarant une horloge racine à chaque fois. En respectant la nomenclature déjà utilisée, celles-ci pourraient être appelées RCS_i , RCH_i , RCI_i et $RCPL_i$ ⁹, et seraient toutes déclarées sur le même point physique.

8. 1 horloge racine, 9 horloges de propagation, 2 horloges de blocage et une horloge d'activation

9. Pour respectivement *Root Clock* de *Setup*, *Hold*, *Interne* et de *Phase Low*.

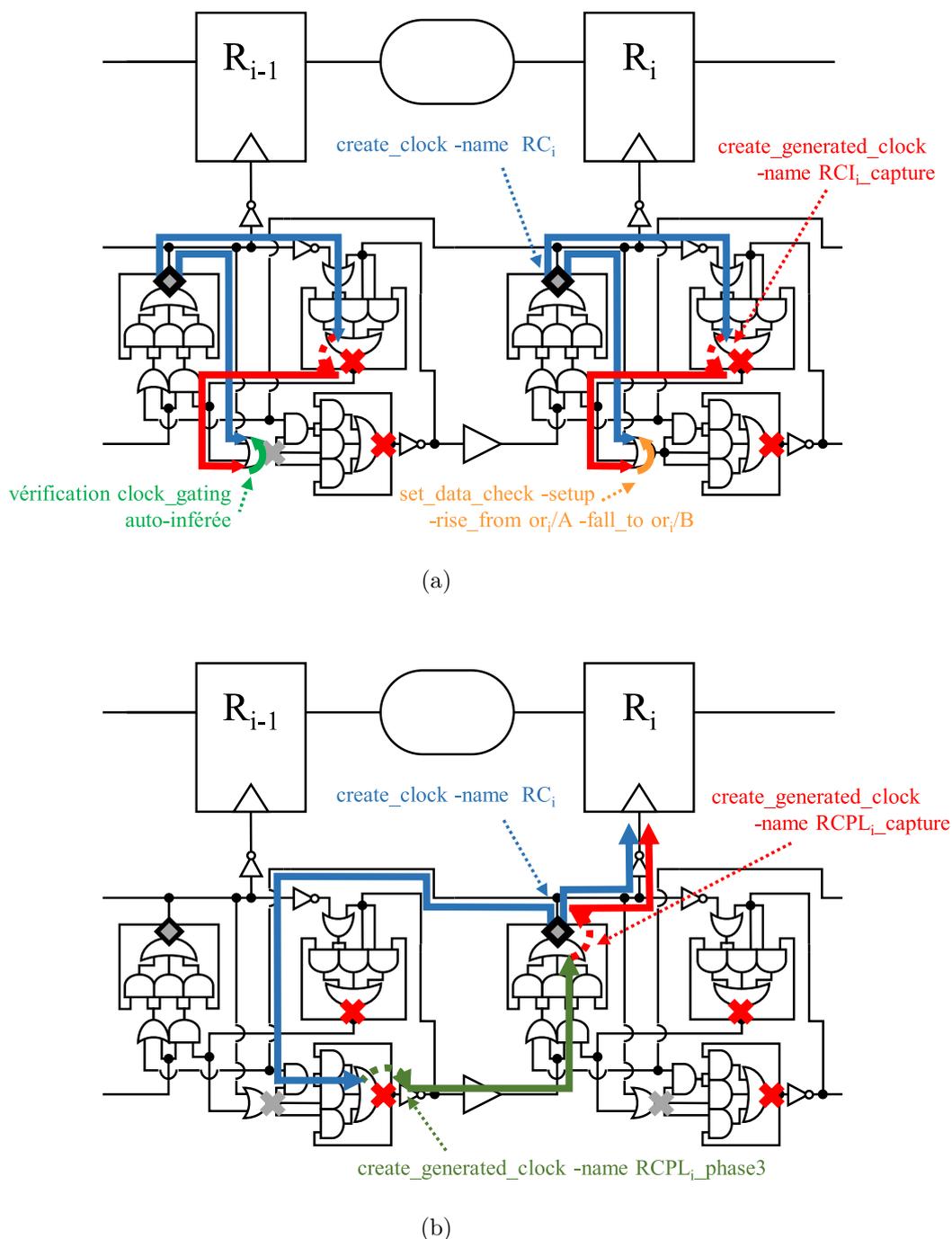


FIGURE 3.16 – Décomposition des RTC en LCS (*cont.*). (a) *course de chemins locaux* et (b) *temps de phase minimum*.

3.3.6 Restreindre les chemins

Pour mener à bien leurs analyses, les moteurs de STA se basent sur les définitions d'horloge pour construire le graphe de tous les chemins temporels possibles. Pour cela, ils considèrent une origine temporelle commune à toutes les horloges : l'instant « 0 ». Même si elles n'ont ni la même fréquence ni la même origine, ces horloges sont donc

considérées comme « synchrones ». Ainsi, toute portion du chemin de données, activée par une horloge de lancement et capturée par une autre, est un chemin relatif à analyser. Dans la méthodologie **LCS**, en multipliant les déclarations d'horloge, nous augmentons la taille de ce graphe mais aussi la possibilité d'activer des **RTC** non valides. Par exemple, un chemin lancé par une horloge racine RC_i peut être capturé par l'horloge racine de l'étage suivant RC_{i+1} . Cette relation ne correspond à aucune **RTC** au niveau du protocole. Elle doit donc être masquée. Une dernière étape de restriction des chemins est donc nécessaire. En particulier les relations suivantes doivent être masquées :

- Les **RTC** lancées par une horloge de capture.
- Les **RTC** capturées par une horloge de lancement.
- Les **RTC** lancées ou capturées par une **EPC**.
- Les **RTC** lancées ou capturées par une horloge de blocage ou d'activation.
- Les **RTC** impliquant des horloges de capture et de lancement de différentes **LCS**.

La commande **SDC set_false_path** et ses options *-from*, *-to*, *-setup* et *-hold* permettent de facilement réduire la liste des **RTC** à analyser. La commande **set_clock_group** est aussi particulièrement intéressante pour isoler chaque **LCS** car elle permet de définir l'asynchronisme directement entre des groupes d'horloges. Lors de la définition des horloges, l'utilisation d'une nomenclature bien adaptée permet de faciliter cette étape de restriction des **RTC**. Idéalement, elle peut alors être indépendante du protocole.

3.3.7 Générateur de contraintes LCS

En systématisant la création des **LCS**, toutes les **RTC** d'un circuit complexe peuvent être définies. Dans cette optique, nous avons développé un générateur de **LCS** permettant de construire automatiquement un fichier de contraintes au format **SDC**, indépendant des conditions **PVT**, à partir d'un nombre restreint de données.

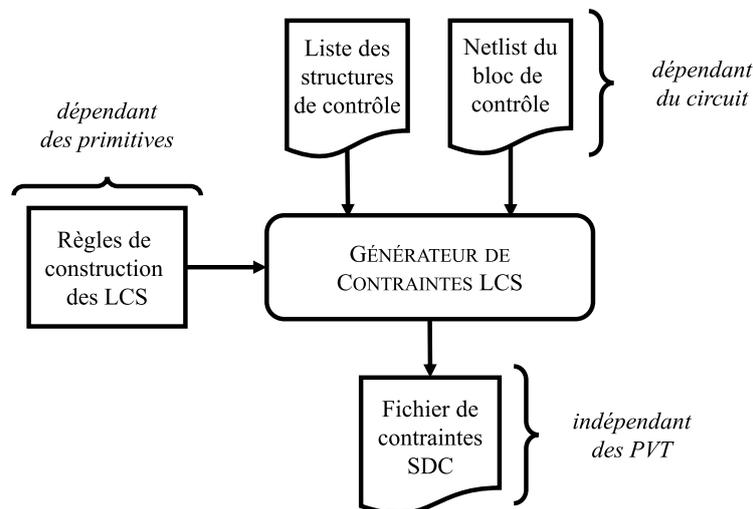


FIGURE 3.17 – Générateur de fichier de contraintes SDC de la méthode LCS.

La figure 3.17 présente cette méthode de génération des **LCS**. Ce générateur doit être alimenté par les fichiers suivants :

- Une *netlist* du bloc de contrôle asynchrone au niveau portes logiques. Cette *netlist* doit être obtenue par une approche d'assemblage de primitives (les contrôleurs ne sont pas synthétisés) de manière à pouvoir définir un jeu de règles unique.

- La liste des structures de contrôle associées. Elle donne les noms hiérarchiques de chacune des primitives de base qui composent le bloc de contrôle (contrôleur de registre, union, fourche, multiplexeur, etc.). Cette liste ne fait que résumer des informations contenues dans la *netlist* du bloc de contrôle. Elle pourrait être retrouvée automatiquement. Elle est cependant très facilement déductible des modèles formels ou de haut-niveaux qui servent à concevoir et à vérifier la vivacité du circuit et, à ce titre, peut être générée très facilement en amont du générateur de contraintes.
- Les règles de construction des **LCS**. Ces règles décrivent comment doivent être déclarées les différentes horloges pour chaque primitive. Elles dépendent du protocole utilisé et non de l'architecture du circuit.

Les règles de construction des **LCS** sont dépendantes de l'implémentation qui est faite de chaque primitive. Idéalement, elles sont utilisables quelle que soit la technologie et les bibliothèques de cellules standards utilisées¹⁰, et peuvent être générées lors de la synthèse de chaque primitive à partir du modèle **STG**. Ces règles décrivent les éléments suivants :

- La position et le nom des horloges racines permettant de définir les **POD** du circuit : `create_clock`.
- La position et le nom des horloges factices permettant de bloquer les boucles combinatoires restantes et d'activer l'inférence des *courses de chemins* : `create_clock`.
- Les contraintes permettant de décrire les courses de chemins additionnelles (si nécessaire) : `set_data_check`.
- Les règles de création des horloges de propagation : `create_generated_clock`. Ces règles sont exécutées de manière itérative. Elles définissent le nom des horloges attendues sur les entrées de chaque primitive (les autres horloges ne sont pas propagées dans la primitive) ainsi que le nom, la source et la polarité des horloges qu'elles doivent générer. Un jeu de règles de propagation existe pour chaque phase du protocole.
- Les commandes supplémentaires permettant de préciser la propagation d'horloge (si nécessaire) : `set_sense`. Ces commandes permettent, par exemple, d'arrêter la propagation d'une horloge en un point du circuit (option *-stop*) ou de spécifier la polarité devant être propagée (options *-rise* ou *-fall*).
- Les exceptions permettant de désactiver les **RTC** non valides : `set_false_path` et `set_clock_group`.

Ces règles permettent d'alimenter le générateur de fichier de contraintes, dont le pseudocode est donné en [figure 3.18](#). Nous avons développé un générateur **LCS** en *Tcl* dans l'outil PrimeTime de Synopsys. Cela nous permet d'accéder directement au moteur d'analyse temporelle pour valider l'adéquation des contraintes avec le circuit. Ce générateur est cependant compatible avec, et a été validé par, toute la suite d'outils Synopsys (Design-Compiler, TetraMax, ICCompiler) et peut facilement être porté dans les outils d'un autre fournisseur de **CAO**.

3.3.8 Comparaison de différents protocoles

Afin de valider la polyvalence de la méthode **LCS**, nous proposons de comparer 7 protocoles à données groupées couramment utilisés. Nous étudions également le schéma de circuit à horloge réactive qui se prête bien à la description **LCS** et, pour avoir un point de référence, nous incluons à ce comparatif un circuit purement synchrone. Un pipeline

10. Du fait des degrés de liberté existants sur la manière de caractériser les cellules standards, de légères adaptations peuvent être nécessaires suivant le fournisseur de bibliothèque.

```

1 # I) root clocks definition
2 FOR each register controller
3   DO create clock on point of divergence
4   DO add it to local clock set list
5 END
6 # II) dummy clocks definition
7 FOR each dummy controller
8   DO create dummy clock on loop-breaker point
9 END
10 # events propagation
11 DO propagate clocks (update_timing)
12 # III) event propagation clocks definition
13 FOR each protocol phase
14   FOR each req/ack input of FCS & dummy controllers
15     FOR each clock propagating to current input
16       IF current clock is not dummy
17         DO create generated clock on controller output \
18                               from req or ack input
19         DO add it to the appropriate local clock set
20 # events propagation
21   DO propagate clocks (update_timing)
22   ENDIF
23   END
24   END
25 END
26 # IV) capture/launch clocks definition
27 FOR each req/ack input of register controllers
28   FOR each clock propagating to current input
29     IF current clock is not dummy
30       DO create generated clock on controller output \
31                               from req or ack input
32       DO add it to the appropriate local clock set
33     ENDIF
34   END
35 END
36 # V) timing exceptions definition
37 DO define hold false path to/from capture clocks
38 DO define setup false path to/from launch clocks
39 DO define asynchronous group foreach local clock set
40 # Transform clock into event
41 DO define 0-cycle path from all clocks to all clocks
42 DO define all clocks as propagated

```

FIGURE 3.18 – LCS SDC generator pseudo code.

TABLE 3.1 – Nombre d’horloges constituant les LCS des contraintes de *regroupement* et d’*écrasement des données* pour différents protocoles à données groupées. i correspond au nombre d’étages du circuit.

Schéma	Nombre d’horloges			
	Total	Racine	Factice	Générée
Synchrone	1	1	0	0
Horloge réactive	6	1	0	5
WCHB	$5 i$	$1 i$	0	$4 i$
<i>Burst mode</i>	$10 i$	$2 i$	0	$8 i$
<i>Early ack.</i>	$9 i$	$2 i$	0	$7 i$
<i>Maximus</i>	$8 i$	$1 i$	$2 i$	$5 i$
<i>Micropipeline</i>	$3 i$	$1 i$	0	$2 i$
<i>Mousetrap</i>	$10 i$	$2 i$	0	$8 i$
<i>Click</i>	$6 i$	$1 i$	0	$5 i$

linéaire de 320 étages et large de 256 bits est construit pour chacun de ces modèles de circuits. Les RTC de *regroupement* et d’*écrasement de données* sont converties en règles LCS qui permettent de générer les contraintes SDC pour tous ces circuits. Ces protocoles sont alors comparés en termes de nombre d’horloges nécessaires, de temps d’exécution et d’empreinte mémoire de la STA.

Le tableau 3.1 montre que le nombre d’horloges nécessaires pour décrire ces deux RTC pour chacun des modèles est très disparate. Pour les circuits à données groupées, nous constatons un facteur supérieur à trois entre les nombres maximum et minimum d’horloges. Cela dépend bien sûr de l’implémentation de chaque contrôleur. Le protocole *Mousetrap* présente les règles les plus complexes avec 10 horloges par étage de pipeline. Il intègre en effet un mécanisme de conversion de phase qui requière la définition d’EPC décrivant la propagation de chacune des polarités des signaux de contrôle. Le protocole *Click* utilise aussi un bloc de conversion de phase. Il y associe cependant un élément séquentiel qui permet d’éviter les boucles combinatoires et de simplifier les règles LCS. A l’opposé, le modèle *Micropipeline* présente les règles les plus simples avec 3 horloges par étage. Le protocole WCHB utilise deux horloges supplémentaires pour décrire les phases de retour-à-zéro. Les autres contrôleurs sont aussi des protocoles 4-phases et nécessitent également de modéliser les phases de RZ. Ils demandent en moyenne 9 horloges par étage pour décrire les 2 RTC.

Étonnamment, si l’empreinte mémoire est relativement corrélée avec le nombre d’horloges, ce n’est pas le cas du temps d’exécution (voir tableau 3.2). Alors que les autres protocoles présentent des durées d’analyse sensiblement identiques, le modèle *Mousetrap* montre un incrément de facteur 4 sur le temps d’exécution par rapport au circuit synchrone. Le nombre d’horloges, plus important, ne semble pas suffisant pour justifier ce surcoût : le protocole *Burst mode* utilise le même nombre d’horloges mais dispose d’un temps d’analyse deux fois moindre. Nous soupçonnons que l’utilisation de verrous soit la cause principale de cette différence. Les outils synchrones n’ont pas été initialement conçus pour analyser les circuits à base de verrous et semblent avoir des performances dégradées dans ces cas-là. Le temps d’exécution du schéma *Click* est très proche de celui du synchrone. Cela semble lié à l’absence de boucle dans le circuit de contrôle, ce qui facilite le travail du moteur d’analyse temporelle. Enfin, le circuit à horloge réactive présente les résultats les plus imprévus. Alors que nous pourrions nous attendre à des performances similaires à la référence synchrone, le temps d’analyse est presque doublé. Nous en concluons que le temps d’exécution ne dépend pas seulement du nombre d’horloges, mais aussi du nombre d’éléments séquentiels « vus » par chacune de ces horloges.

TABLE 3.2 – Impact de la méthode LCS sur l’empreinte mémoire et le temps d’exécution de la STA pour différents protocoles d’un pipeline 320 étages et de 256 bits de large. Le ratio par horloge est donné.

Type	Schéma	Temps d'exécution (s)		Empreinte mémoire (Mo)	
		Total	Ratio	Total	Ratio
Synchrone	Synchrone	26,59	<i>ref</i>	1962.9	<i>ref</i>
	Horloge réactive	47,15	+77 %	2049.3	+4 %
4-phases	WCHB	36,16	+36 %	2501.3	+27 %
	<i>Burst mode</i>	45,39	+71 %	3266.5	+66 %
	<i>Early ack.</i>	50,74	+91 %	3221.9	+64 %
	<i>Maximus</i>	35,93	+35 %	2612.1	+33 %
2-phases	<i>Micropipeline</i>	32,49	+22 %	2305.5	+17 %
	<i>Mousetrap</i>	106,54	+306 %	3046.3	+55 %
	<i>Click</i>	32,27	+21 %	2521.6	+28 %

Bien sûr, cette comparaison ne prend en compte que deux **RTC**. D’autres analyses seraient nécessaires pour précisément comparer ces différents modèles de circuits du point de vue de leur impact sur les performances des outils de **CAO**. Lors de la sélection d’un protocole à données groupées de nombreux autres paramètres doivent aussi être pris en compte : le débit, l’efficacité énergétique ou la testabilité. Ceux-ci ne sont pas traités dans notre travail. Mais en analysant avec succès des circuits de grandes tailles¹¹, cette comparaison valide le principal objectif de la méthode **LCS** : la **STA** des circuits à données groupées est possible, quel que soit le protocole utilisé et la taille du circuit analysé. Ainsi, réduire le nombre d’horloges utilisées pour chaque **LCS** ne semble pas être une priorité. Afin de simplifier la manipulation des **LCS**, il semble possible de dupliquer les horloges racines (une pour chaque LCS) sans trop impacter les temps d’exécution.

Il apparaît tout de même que l’analyse des circuits à base de verrous est moins aisée. Pour faciliter la compréhension des concepteurs habitués aux circuits à base de bascules et pour exploiter au mieux les capacités de *time borrowing* des verrous, les outils de **STA** prennent systématiquement en compte la période d’horloge lors des vérifications temporelles. Les chemins temporels sont vérifiés entre deux fronts successifs ouvrant les verrous. Ce comportement permet d’analyser plus facilement des circuits mélangeant bascules et verrous, mais il empêche de transformer les horloges en événement au sens de la méthode **LCS**. Un mode avancé de l’outil PrimeTime permet de s’affranchir de ce phénomène et d’ainsi retrouver une analyse plus réaliste¹². Nous supposons que ce mode d’analyse explique en partie le rallongement des temps d’exécution.

3.4 LCS et flot d’implémentation physique

Une fois traduites au format **SDC**, les **LCS** peuvent être utilisées tout au long du flot d’implémentation physique. Elles permettent aux moteurs de **STA** de connaître l’ensemble des contraintes temporelles à respecter. L’approche **LCS** ne présume à aucun moment de l’emplacement des délais *matchés*. Ainsi, elle est une véritable traduction, et non une simple interprétation, des exigences temporelles. Elle donne aux outils toute la latitude

11. A titre de comparaison, un processeur peut être implémenté en utilisant 5 étages de pipeline (voir [Dum+09; SYK18])

12. En activant la variable `timing_enable_through_path`, les chemins de *setup* et de *hold* sont lancés par un front ouvrant les verrous et capturés par un front les fermant.

nécessaire pour effectuer leurs optimisations sur le circuit.

3.4.1 Comparaison avec la méthode min/max

La figure 3.19 met en contraste les flots d'implémentation physique basés sur les méthodes d'analyse temporelle min/max et LCS. Le premier adopte une approche itérative. Les fichiers de contraintes doivent être mis à jour à chaque fois que de nouvelles optimisations sont faites sur le circuit. De plus, ces fichiers sont dépendants des conditions d'opération (PVT). Le nombre de fichiers à manipuler est donc important et le temps nécessaire pour les régénérer devient vite prohibitif. Pour limiter ces itérations, il est possible de prendre des marges au niveau des contraintes, mais cela revient systématiquement à sur-dimensionner le circuit. Au contraire, la méthode LCS se base sur un unique fichier SDC définissant l'ensemble des RTC quel que soit les conditions PVT. Ce fichier est généré une seule fois et est ensuite indistinctement utilisé par les différents outils d'implémentation physique pour guider leurs optimisations et leur permettre de vérifier la bonne tenue des exigences temporelles.

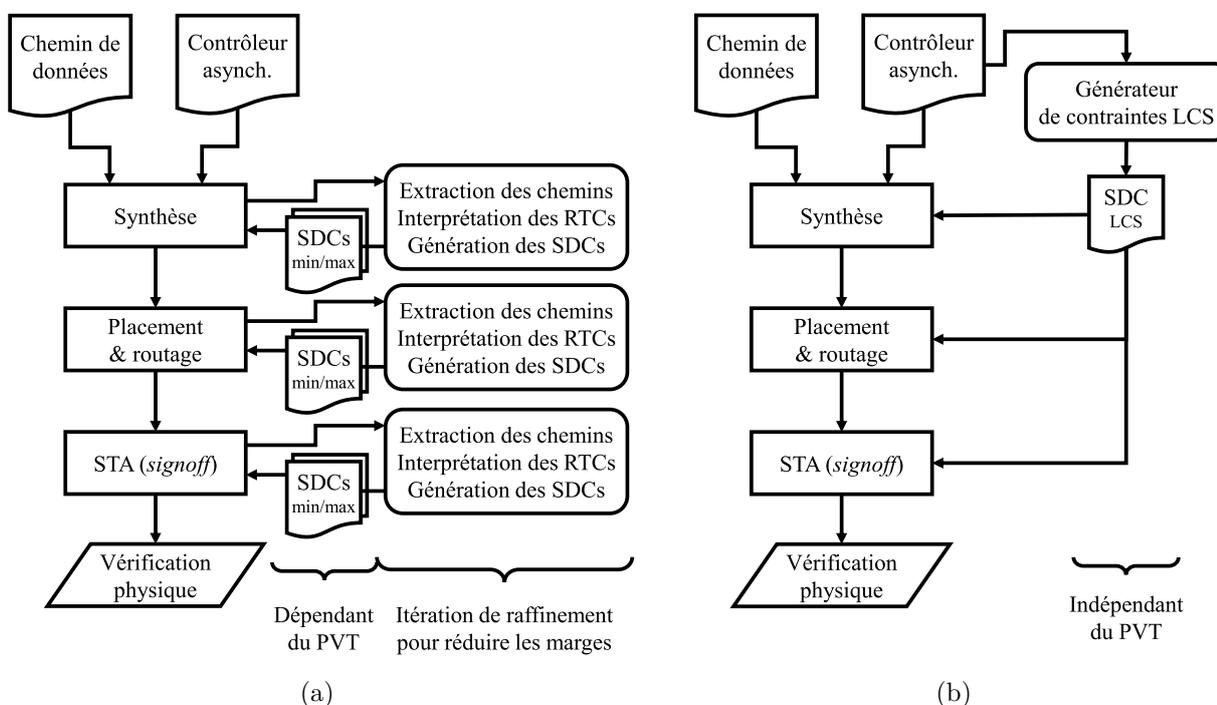


FIGURE 3.19 – Flots d'implémentation pour circuits à données groupées. (a) méthode min/max et (b) méthode LCS.

Afin de comparer plus en détails ces deux méthodes, nous avons utilisé comme cas d'étude un bloc de chiffrement *Advanced Encryption Standard (AES)* de 128 bits. Un circuit synchrone formé à partir du même chemin de données est utilisé comme référence. Ce chemin de données est présenté en figure 3.20a. Pour les circuits à données groupées, l'horloge est remplacée par le contrôleur schématisé en figure 3.20b. Celui-ci utilise un protocole *WCHB 4-phases*¹³. Nous avons donc implémenté le circuit *AES* de trois manières

13. Nous avons opté pour une simplification des LCS pour ce circuit : la phase de RZ n'est pas modélisée dans les règles LCS que nous avons utilisées (qui sont donc équivalentes aux règles du *Micropipeline 2-phases*). Cela permet de réduire le nombre d'horloges mais sur-contraint les chemins d'écrasement des données.

différentes. Pour chacune d'elles, nous avons créé les contraintes au format SDC et exécuté la STA.

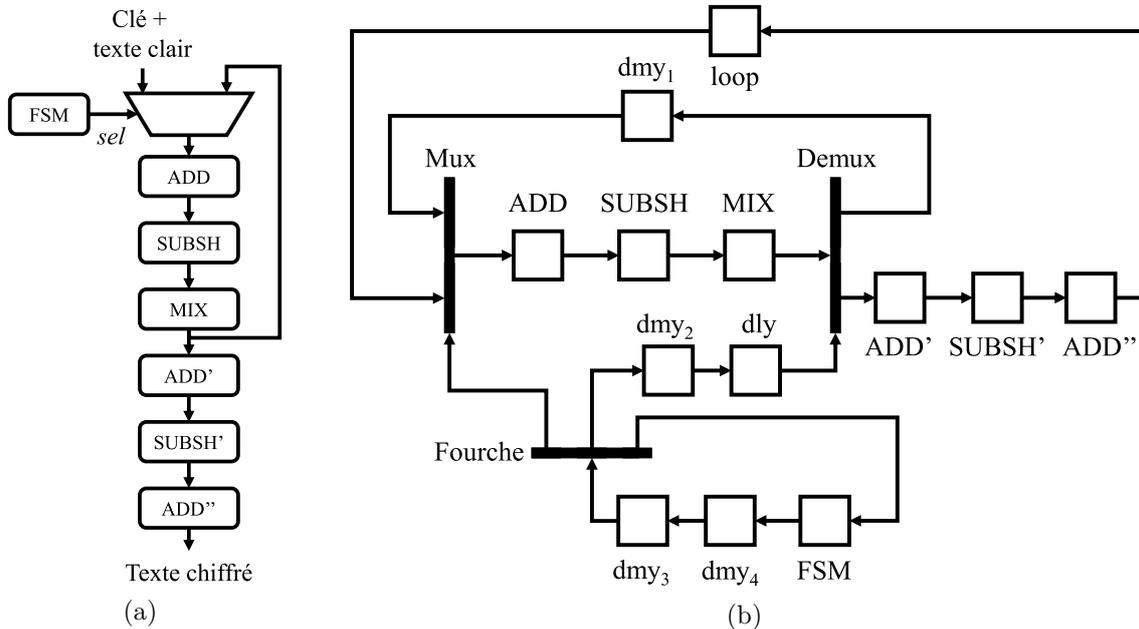


FIGURE 3.20 – Circuit de chiffrement AES asynchrone. (a) chemin de données et (b) bloc de contrôle.

TABLE 3.3 – Comparaison du contenu des fichiers de contraintes pour les versions synchrone, min/max et LCS du bloc AES.

commandes	Synchrone	Asynchrone	
		min/max	LCS
<i>create_clocks</i>	1	0	13
<i>create_generated_clocks</i>	0	0	54
<i>set_disable_timing</i>	0	15	2
<i>set_min/max_delay</i>	0	27	0
autres	3	0	83
TOTAL	4	42	152
Ratio	1	10	38

Le [tableau 3.3](#) permet de comparer le contenu des trois fichiers de contraintes. La méthode [LCS](#) nécessite 9 horloges racines, 4 horloges de blocage et 54 [EPC](#) pour construire les 26 [RTC](#) garantissant le fonctionnement du circuit [AES](#). Son fichier de contraintes est donc bien plus complexe que celui utilisé par les deux autres méthodes. Ce fichier a cependant été généré automatiquement, contrairement au fichier [SDC](#) de la méthode min/max que nous avons dû construire manuellement.

L'analyse des temps d'exécution reportés dans le [tableau 3.4](#) apporte un éclairage différent. Pour la méthode min/max, nous voyons que deux durées d'exécution sont données. Elles correspondent aux deux appels successifs de la commande `update_timing` requis par cette méthode : un pour extraire les temps de propagation et le second pour faire l'analyse temporelle du circuit une fois les contraintes min/max ajustées. Ainsi, l'analyse du circuit avec la méthode [LCS](#) prend 1,5 fois plus de temps que pour le cas synchrone. La méthode min/max, quant à elle, double presque le temps d'exécution.

Pour la méthode [LCS](#), l'augmentation du nombre d'horloges se répercute aussi au

TABLE 3.4 – Temps d’exécution et empreinte mémoire de la commande `update_timing` pour les versions synchrone, min/max et LCS de 10 blocs AES.

	Synchrone	Asynchrone	
		min/max	LCS
Temps d’exécution (s)	9,84	8,96+9,65	14,95
Nb <code>update_timing</code> min	1	2	1
Variation	<i>ref</i>	+89 %	+52 %
Memoire (Mo)	850	840	1 033
Variation	<i>ref</i>	-1,2 %	+21 %

niveau de l’empreinte mémoire. Elle conduit à un agrandissement du graphe temporel manipulé par l’outil de **STA**. Cependant, le fichier de contraintes **SDC** intègre les commandes permettant de désactiver les **RTC** invalides et le nombre de chemins à analyser reste équivalent à celui des deux autres méthodes. Cela explique l’incrément raisonnable du temps d’exécution que nous observons.

Finalement, en comparant les deux approches asynchrones, la méthode **LCS** permet une réduction du temps d’exécution de plus de 20 % par rapport à la méthode min/max. De plus, ce chiffre ne prend pas en compte le coût de génération des fichiers de contraintes pour chaque *corner* **PVT**, ni le fait que de nombreuses itérations sont nécessaires pour appliquer la méthode min/max.

3.4.2 Contraindre le chemin de données

Les performances d’un circuit synchrone sont définies par la période de son horloge. En respectant cette contrainte, les outils garantissent à la fois les performances et le bon fonctionnement du circuit. Mais pour les circuits à données groupées, le respect des exigences temporelles permet seulement d’assurer leur bon fonctionnement. Ces exigences ne garantissent pas les performances. Un circuit peut être fonctionnel – chaque délai *matché* couvrant le temps de propagation du chemin de données correspondant – sans pour autant délivrer les performances attendues. Ce sera le cas si le chemin de données n’a pas été correctement contraint et qu’il n’a pas été suffisamment optimisé par les outils. La méthode **LCS** nous laisse cependant plusieurs alternatives pour contraindre le chemin de données.

Une première technique consiste à insérer préalablement des délais dans le contrôleur. Cela peut être fait au niveau de la description niveau transferts de registres (*Register-Transfer Level*) (**RTL**) en instanciant un nombre d’éléments de retard proportionnel au délai attendu. Ces éléments pourront être conservés tout au long de l’implémentation physique¹⁴ et serviront de référence pour le chemin de données. En utilisant les **LCS** transformées en évènement, chaque chemin sera contraint et les outils essayeront de les optimiser afin qu’ils soient au moins aussi rapides que leur délai préalablement inséré. Cette première approche a l’avantage d’être intuitive et simple d’utilisation. Elle permet également de contraindre indépendamment chaque étage du circuit. Elle n’est cependant pas très précise et reste très sensible aux variations des délais dues à l’implémentation. Il est en effet difficile d’anticiper précisément quel sera le temps de propagation final dans la ligne de retard. Cela dépend grandement du placement et du routage de chacun des éléments la constituant. Tout particulièrement, les effets de diaphonie (*crosstalk*) peuvent avoir un impact non négligeable sur les temps de propagation. Ceux-ci peuvent difficilement être anticipés et n’apparaissent qu’une fois le routage du circuit réalisé. À moins de prendre des marges supplémentaires, cette technique sera donc très itérative et rendra la convergence

14. Par exemple en spécifiant des contraintes de `set_dont_touch`.

ligne d’acquiescement pour compenser les *skew* virtuels qui peuvent exister au niveau des entrées de contrôle des éléments séquentiels. L’inégalité (3.20) se simplifie alors en :

$$D_{A_i} > S_i + T_{hold} - D_{CQ} - D_a \quad (3.21)$$

avec $S_i = D_{L_{i+1}} - D_{L_i}$

Pour éviter de dégrader les performances du circuit (surface, consommation, débit), le délai D_{A_i} doit être le plus petit possible. On peut donc en déduire la contrainte suivante sur le *skew* entre deux étages :

$$S_i < D_{CQ} + D_a - T_{hold} \quad (3.22)$$

Cette contrainte peut être approchée par $S_i < D_a$ si l’on considère que les temps de *hold* et de capture des éléments séquentiels sont équivalents. Par ailleurs, il n’y a pas de limite basse sur le *skew* virtuel. Cependant, toute valeur négative de S_i doit être compensée par le délai D_{R_i} sur la ligne de requête, ce qui entraîne une réduction du débit du circuit¹⁶. De plus, la plupart des circuits contiennent des boucles architecturales. Par propagation de la contrainte, ces boucles impliquent que $-S_i$ soit aussi borné par D_a . Finalement, l’équation (3.23) donne un objectif concret pour le *skew* virtuel pendant l’implémentation physique. En respectant cet intervalle, nous pouvons garantir un délai minimal sur le chemin de requête et éviter l’utilisation d’un délai sur l’acquiescement.

$$|S_i| < D_a \quad (3.23)$$

Contrairement à ce que l’on pourrait croire, les circuits à données groupées conservent une contrainte de création d’arbres d’horloges équilibrés. Mais à la différence des circuits synchrones, cette contrainte reste locale et s’applique souvent dans la pratique à des arbres de très petite taille. Elle est donc plus facile à respecter. Cette contrainte est aussi moins forte, car tout déséquilibre peut toujours être compensé lors de l’insertion des délais *matchés*. Les algorithmes de construction d’arbres d’horloges peuvent facilement être utilisés pour construire ces arbres locaux. En particulier, les options d’ajustement de latences inter-arbres d’horloges permettent d’équilibrer ces multiples arbres locaux.

3.4.4 Insertion des délais *matchés*

L’objectif initial de la méthode *LCS* n’est pas d’insérer les délais *matchés*. Cependant, elle donne aux outils la connaissance des *RTC*. Leurs rapports d’analyse temporelle donnent alors directement la valeur des délais à insérer (ou à retirer dans le cas d’une marge trop importante). En partant de cette connaissance, plusieurs méthodes sont envisageables pour insérer les délais. Une technique naïve consiste à insérer manuellement un élément de retard sur chaque ligne de requête puis de ré-analyser le circuit. De nouveaux délais sont insérés itérativement jusqu’à ce que toutes les contraintes soient tenues. Cette technique est loin d’être efficace, mais permet d’avoir un contrôle assez fin sur l’insertion des délais dans la mesure où l’on peut ajuster la granularité des délais insérés itérativement.

Une technique plus rapide se base sur l’utilisation de contraintes de `set_min_delay`. De telles contraintes peuvent être appliquées sur les lignes de requête entre chaque étage, et adaptées selon les marges temporelles rapportées par l’analyse *LCS* (figure 3.22a). L’outil d’implémentation tentera, lors d’une passe d’optimisation, d’insérer les délais suffisants

16. En compensant de la sorte le *skew* existant entre deux étages du circuit, on vient transférer des éléments de retard d’un arbre de contrôle local vers un chemin de requête. Ce faisant, on déplace des délais qui n’impactent initialement pas le temps de cycle pour les ajouter dans l’un des oscillateurs locaux, ce qui a pour effet de ralentir le circuit.

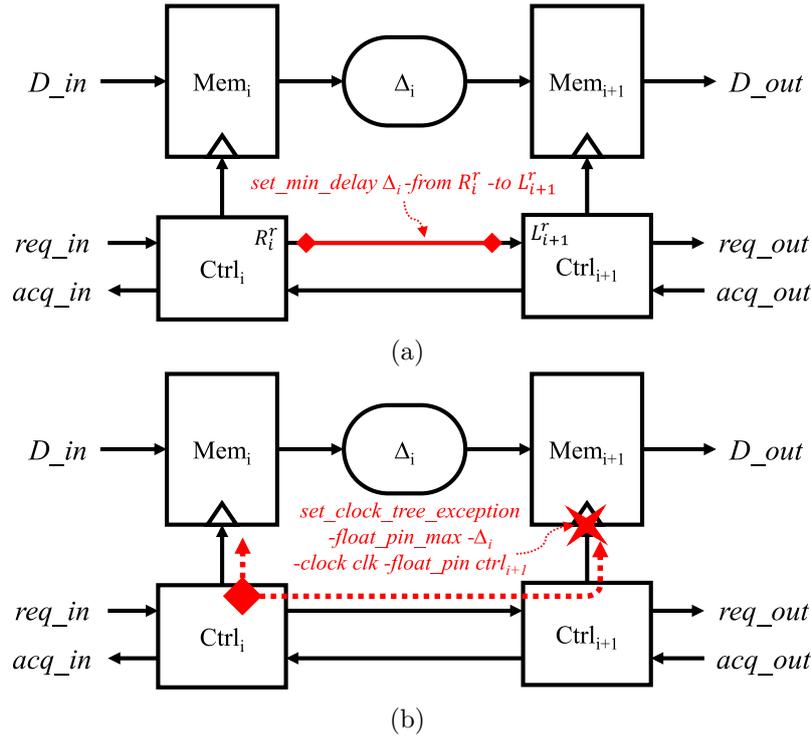


FIGURE 3.22 – Deux méthodes d'insertion des délais *matchés*. (a) utilisation des commandes `set_min_delay` et (b) en utilisant un déséquilibre volontaire lors de la synthèse d'un arbre d'horloge.

pour couvrir ces contraintes de temps minimum. Mais en laissant l'outil ajouter lui-même les délais, nous perdons en précision sur le résultat final. Il est très probable que l'outil ajoute trop de délais, dégradant ainsi les performances du circuit. Ce phénomène est d'autant plus amplifié que la contrainte de temps minimum s'applique indépendamment du contexte électrique : l'ajout de délai peut venir perturber les cellules proches de la ligne de requête (diaphonie, modification du routage, etc.) modifiant les temps de propagation dans les contrôleurs adjacents ou dans le chemin de données et invalidant la contrainte initialement utilisée. Une étape de raffinement itératif, semblable à la technique « naïve », est donc généralement nécessaire pour corriger ces petites variations en ajoutant ou en retirant quelques éléments de retard.

Nous avons développé une troisième approche, plus précise, qui utilise les capacités intrinsèques de construction d'arbres d'horloges des outils de placement-routage. Au lieu de définir une contrainte de temps minimum, nous pouvons forcer l'outil à insérer le délai *matché* au travers d'un arbre d'horloge volontairement déséquilibré. Nous proposons donc de construire un arbre d'horloges pour chaque LCS du circuit. La figure 3.22b illustre cette approche pour un simple étage de pipeline. L'horloge racine (losange rouge) doit être distribuée au travers d'un arbre d'horloge aux différents éléments séquentiels, Mem_i et Mem_{i+1} , atteints par la LCS correspondante. Par défaut, les outils vont chercher à équilibrer cet arbre en adaptant les délais insérés pour que l'horloge racine atteigne chacun des éléments séquentiels au même moment. Nous pouvons cependant les forcer à construire un arbre déséquilibré en leur demandant de ralentir le signal d'horloge destiné aux éléments séquentiels de l'étage de destination (Mem_{i+1}). Dans ICCompiler, la commande `set_clock_tree_exception` permet de définir de telles contraintes. Cette technique a l'avantage de prendre en compte le chemin complet depuis la source de l'horloge racine jusqu'aux entrées de contrôle des différents éléments séquentiels. De plus, contrairement

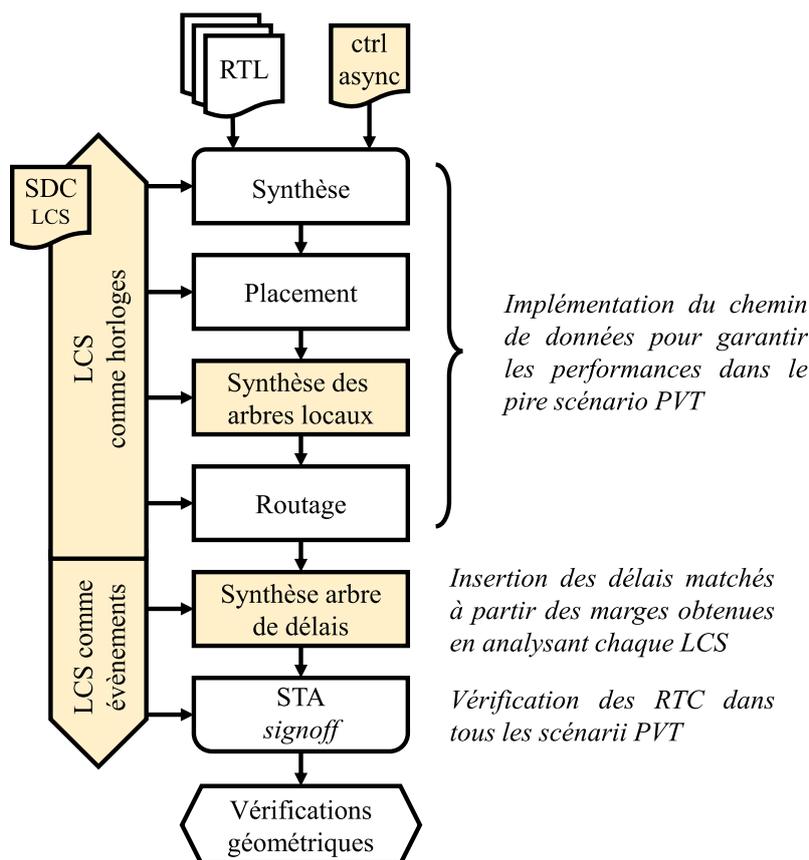


FIGURE 3.23 – Flot d’implémentation physique LCS complet. Les éléments spécifiques aux circuits à données groupées sont indiqués en orange.

à la commande `set_min_delay`, la contrainte d’arbre d’horloge n’est pas une contrainte minimale. L’outil va chercher à s’en approcher le plus possible pour atteindre un *skew* proche de 0 à Δ_i près.

3.4.5 Flot d’implémentation LCS

Nous avons bâti un flot d’implémentation physique complet, basé sur la description temporelle *LCS* et qui utilise les outils de CAO traditionnels. La figure 3.23 présente les différentes composantes ce flot. Nous retrouvons les mêmes grandes étapes que celles utilisées pour les circuits synchrones, à savoir : synthèse, placement, routage, STA. Celles-ci ne sont pas, ou très peu, modifiées pour être adaptées aux circuits à données groupées. Grâce aux *LCS*, les outils ont la connaissance des exigences temporelles et peuvent, au cours de ces étapes, optimiser et vérifier le chemin de données de manière à garantir le fonctionnement et les performances souhaités. Quelques précautions doivent tout de même être prises pour éviter que les outils ne modifient la partie contrôle du circuit (utilisation de commande `set_dont_touch` sur les éléments sensibles). Au niveau du placement, il est aussi recommandé de forcer l’outil à regrouper les cellules formant chaque contrôleur de manière à limiter l’insertion de délais parasites et la sensibilité au routage de cette partie du circuit.

Les vraies différences de ce flot par rapport à un flot d’implémentation synchrone se situent au niveau de l’étape normalement dédiée à la synthèse de l’arbre d’horloge. Celle-ci est remplacée par une étape de synthèse et de routage des arbres de contrôles locaux. Comme vu précédemment, cette étape permet de limiter le surdimensionnement du circuit.

A l’instar de la construction d’un arbre d’horloge, cette étape ne dépend d’aucune **RTC**. Son objectif est uniquement de construire des arbres locaux, équilibrés — pour ne pas impacter les performances — et les plus courts possible — pour ne pas trop impacter la consommation.

L’étape d’insertion¹⁷ des délais *matchés* vient compléter ce flot. Pour celle-ci, nous avons préféré adopter une approche consistant à ajouter les délais le plus tardivement possible. En effet, lorsque les délais sont insérés plus précocement dans le flot, les différentes étapes d’optimisation ont tendance à modifier leur temps de propagation, rendant nécessaires des raffinements itératifs. Comme présenté précédemment, nous utilisons les algorithmes de construction d’arbre d’horloge pour insérer ces délais. Ils permettent, avec des options spécifiques, d’ajouter les éléments de retard en modifiant le moins possible le placement des cellules du chemin de données¹⁸. Ainsi, les performances du circuit ne sont pas altérées par l’adjonction des délais. Cette approche peut cependant présenter des limites. En particulier, si la densité de cellules standard est élevée, il est possible que les éléments de retard soient très espacés les uns des autres à cause du manque de place. Cela peut rendre difficile la convergence des algorithmes. Dans le pire des cas, ce phénomène peut également remettre en cause l’hypothèse de localité — la proximité des chemins de données avec leurs délais *matchés* respectifs — et il ne serait alors plus possible de se contenter d’utiliser des marges **OCV** modélisant uniquement les variations locales. Mais dans les différents essais que nous avons pu mener, c’est cette approche tardive qui a montré la plus grande précision.

3.5 Conclusion et perspectives

Les différents éléments présentés dans ce chapitre peuvent être assemblés pour construire le flot de conception global présenté dans la figure 3.24. Ce flot peut être décomposé en quatre parties. Les deux premières sont des étapes de construction et de caractérisation de primitives. Leurs résultats peuvent être livrés comme complément aux bibliothèques de cellules standards. A l’inverse, les étapes 3 et 4 sont spécifiques au circuit développé.

- 1) En partant du modèle formel — le **STG** — décrivant le comportement des différentes primitives d’un protocole à données groupées, nous synthétisons une *netlist* au niveau portes et la liste des **RTC** associées à chacune de ces implémentations. Nous complétons cette liste avec les **RTC** provenant des modèles d’abstraction temporelle des éléments séquentiels.
- 2) Nous transformons ensuite ces **RTC** en une série de règles génériques permettant de décrire la manière de construire les **LCS** équivalentes.
- 3) À partir de ces règles et de la *netlist* du bloc de contrôle créée par assemblage des différentes primitives, nous générons automatiquement le fichier de contraintes temporelles traduisant ces **LCS** au format **SDC**.
- 4) Enfin, nous utilisons ce fichier de contraintes **SDC** lors de l’implémentation physique pour optimiser le circuit, insérer les délais *matchés* et vérifier les exigences temporelles.

Avec ces différentes étapes, nous proposons un flot compréhensif depuis le **STG** jusqu’à la clôture temporelle, garantissant les performances et le fonctionnement d’un circuit asynchrone. Contrairement aux autres approches de la littérature, notre flot décrit

17. L’insertion correspond à l’ajout, le placement et le routage des délais.

18. Par défaut, la priorité est donnée aux cellules de l’arbre d’horloge, et le placement optimal du chemin de données peut être altéré pour obtenir un arbre mieux équilibré.

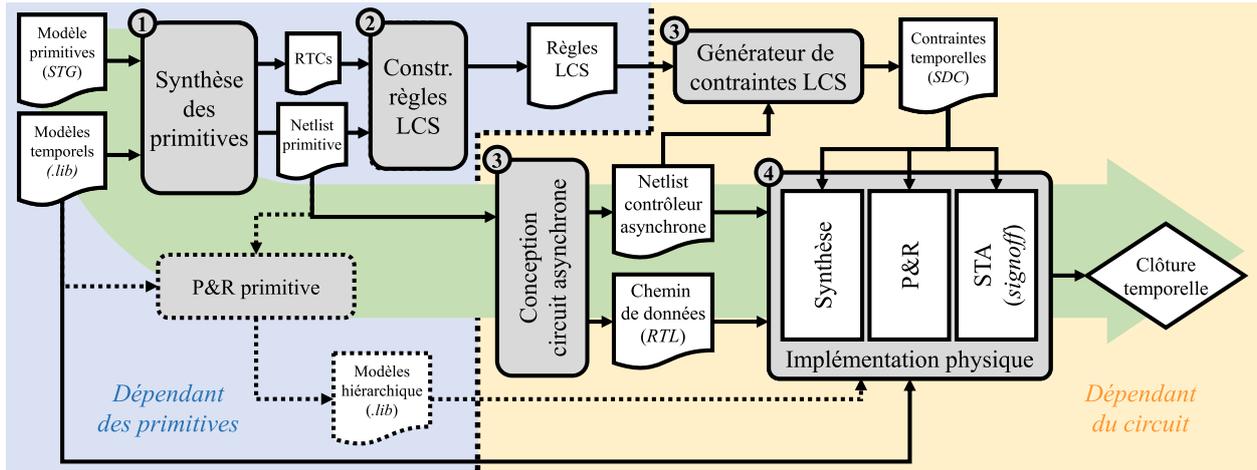


FIGURE 3.24 – Flot de conception LCS d'un circuit à données groupées.

les spécificités des circuits à données groupées avec un formalisme synchrone et permet donc d'utiliser les outils commerciaux de manière optimale. En traduisant réellement les **RTC**, la méthode **LCS** s'affranchit aussi de la dépendance aux conditions d'opération des contraintes temporelles, simplifiant l'analyse et le flot d'implémentation. Il en résulte un impact contenu sur les performances des outils, et notre méthode peut ainsi être utilisée pour des circuits de grandes tailles. Nous avons également montré que cette méthode était utilisable pour toutes sortes de protocoles à données groupées. Finalement, nous avons démontré que l'analyse temporelle des circuits à données groupées est faisable à l'aide des outils synchrones existants. Elle n'est donc plus un véritable frein à l'adoption de ces circuits asynchrones. Nous espérons ainsi contribuer à l'abaissement des barrières entre les mondes synchrone et asynchrone et à l'intégration de ces techniques de conception dans un nombre croissant de circuits du commerce. Dans cette optique, les règles **LCS**, développées dans le cadre de la comparaison présentée dans la [section 3.3.8](#), sont disponibles en libre accès en ligne [[Gim19](#)].

Ce travail ouvre aussi de nombreuses perspectives. En s'appuyant sur les principes de base exposés dans ce chapitre, il est possible de créer les règles **LCS** pour de nouveaux protocoles. Chacun des protocoles à données groupées ayant ses avantages et ses inconvénients, il semble intéressant de travailler sur leur mélange au sein d'un même circuit. Cela permettrait de sélectionner le protocole le plus approprié compte tenu des contraintes s'appliquant sur les différentes parties d'un circuit complexe : taille du chemin de données, équilibre entre performance et consommation, etc. L'impact d'une telle approche sur les **LCS** serait alors à étudier. Un travail sur la standardisation des règles **LCS** apparaît nécessaire pour faciliter cette démarche. Nous avons montré dans le [chapitre 2](#) qu'il est toujours possible de choisir entre une approche hiérarchique, qui cache la complexité des contrôleurs dans des modèles temporels et permet d'améliorer la productivité (étape en pointillés sur la [figure 3.24](#)), ou une approche « à plat » qui autorise des optimisations supplémentaires. Une analyse plus poussée de la frontière entre ces deux approches serait pertinente. Elle couvrirait la sélection des **RTC** significatives, leur caractérisation et leur modélisation dans le format Liberty.

Enfin, les capacités d'optimisation avancées des outils de placement-routage pourraient être utilisées dans la perspective d'automatiser encore plus l'insertion des délais *matched* et ainsi de limiter les marges et les raffinements nécessaires. Depuis quelques années, ces outils proposent des options d'optimisation concurrente de l'arbre d'horloge et du chemin de données (*Concurrent Clock and Data Optimization*) (**CCDO**). Celles-ci permettent de

compenser des violations temporelles dans le chemin de données par du *skew* volontaire sur l'arbre d'horloge. Ainsi, ces techniques insèrent automatiquement des éléments de retard sur certaines branches de l'arbre suivant les résultats de l'analyse temporelle. Ce principe correspond exactement au processus d'insertion des délais *matchés*, et semble donc exploitable à cette fin. Cela est bien sûr envisageable grâce à l'utilisation des descriptions *LCS* qui donne aux outils la connaissance des violations résultant de l'absence de délais *matchés*.

Les travaux présentés dans ce chapitre ont fait l'objet de deux publications dans des conférences internationales :

- G. GIMENEZ *et al.* « Static Timing Analysis of Asynchronous Bundled-Data Circuits ». In : *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2018, p. 110-118. (*Best Paper Award*)
- G. GIMENEZ, J. SIMATIC et L. FESQUET. « From Signal Transition Graphs to Timing Closure : Application to Bundled-Data Circuits ». In : *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2019, p. 86-95

Deuxième partie

**Primitives de sécurité à base
d'anneaux auto-séquencés**

Chapitre 4

Génération d'aléa : un état de l'art

Sommaire

4.1	Introduction	86
4.2	Oscillateur en anneau auto-séquenté	87
4.2.1	Architectures d'oscillateur en anneau	87
4.2.2	Modes d'oscillation des STR	89
4.2.3	Effet <i>drafting</i> et effet <i>Charlie</i>	90
4.2.4	Répartition homogène des phases et finesse de résolution	91
4.2.5	Abstraction jetons/bulles	93
4.3	Générateur d'aléa dynamique : TRNG	95
4.3.1	Principes de fonctionnement	95
4.3.2	<i>Jitter</i> dans les oscillateurs en anneau	96
4.3.3	Imprédictibilité	97
4.3.3.1	Évaluation statistique	98
4.3.3.2	Modélisation	100
4.3.3.3	Le contre-exemple du MURO-TRNG	101
4.3.4	Non-manipulabilité	102
4.3.4.1	Retour sur le MURO-TRNG	102
4.3.4.2	Tests en ligne et monitoring de la source	102
4.3.5	Exemples de TRNG	103
4.3.5.1	Le ERO-TRNG	104
4.3.5.2	Le STR-TRNG	105
4.3.6	Comparaison de différentes architectures	107
4.4	Générateur d'aléa statique : PUF	107
4.4.1	Principes de fonctionnement	108
4.4.2	Évaluation	109
4.4.2.1	Évaluer l'unicité	109
4.4.2.2	Évaluer la stabilité	109
4.4.2.3	Évaluer l'imprédictibilité	110
4.4.2.4	Autres paramètres	110
4.4.3	Exemple de PUF	110
4.4.3.1	Temps de propagation : le A-PUF	111
4.4.3.2	Fréquence : le RO-PUF	112
4.4.3.3	Initialisation d'éléments bistables : le SRAM-PUF	113
4.5	Conclusion	113

4.1 Introduction

Tout système de traitement de l'information stocke, manipule et communique des données. Selon leur degré de criticité, des besoins de sécurité plus ou moins importants s'appliquent. On distingue généralement cinq objectifs. Les données doivent avant tout rester confidentielles. Elles ne doivent être accessibles qu'aux personnes autorisées. Elles doivent ensuite être préservées de toute altération, intentionnelle ou non. Toute modification de leur contenu remet en cause leur intégrité et doit être au minimum détecté. Les données doivent aussi être authentifiées de manière à garantir leur provenance d'un correspondant donné. Ce correspondant doit à son tour être identifié, pour attester qu'il est bien celui qu'il prétend être. Et finalement, un système de non-répudiation est nécessaire pour éviter que l'émetteur des données puisse nier en être à l'origine.

Divers moyens sont employés pour atteindre ces objectifs. Ils reposent généralement sur l'utilisation de blocs, aussi appelés primitives, implémentant des fonctions de sécurité élémentaires. Par exemple, les algorithmes de chiffrement¹ et déchiffrement² transforment de manière réversible des données initialement compréhensibles en une suite de bits intelligibles. Ils s'appuient, pour cela, sur un système de clés, symétriques³ ou asymétriques⁴. Les fonctions de hachage et de correction d'erreurs extraient, quant à elles, une signature à partir des données sans faire intervenir de clé. Cette signature peut alors être utilisée pour détecter et/ou corriger de potentielles altérations. Ces deux types de primitive peuvent être implémentés de manière matérielle ou logicielle. La première approche est plus efficace du point de vue de la surface, des performances et de la consommation, alors que la seconde apporte une flexibilité intéressante dans l'optique de mise à jour et d'évolution des algorithmes. Il existe pourtant d'autres primitives qui ne peuvent s'affranchir d'une implémentation matérielle. C'est le cas des primitives permettant de générer de l'aléa, qu'il soit statique ou dynamique.

L'aléa dynamique est un élément primordial de la sécurisation. La grande majorité des protocoles cryptographiques requièrent des nombres aléatoires de grande qualité. Ils sont utilisés comme clé, masque, nonce ou comme vecteur d'initialisation (*Initialization Vector*) (IV)⁵. Bien souvent, ces nombres aléatoires doivent aussi être parfaitement imprédictibles. C'est par exemple le cas du standard américain de signature algorithmique de signature numérique (*Digital Signature Algorithm*) (DSA) pour lequel la connaissance de quelques bits de l'IV est suffisant pour retrouver la clé privée [Ble00 ; HS01 ; NS02 ; Ara+20]. Bien que des suites de nombres aléatoires uniformément distribués peuvent facilement être obtenues à l'aide de constructions mathématiques appelées générateur de nombres pseudo-aléatoires (*Pseudo Random Number Generator*) (PRNG), ce type de dispositif doit être régulièrement réamorcé avec une graine aléatoire afin d'atteindre un niveau de sécurité optimal [SK02]. Cela repousse donc la problématique de l'imprédictibilité au niveau des graines. Celles-ci sont idéalement générées par un **générateur de nombres véritablement aléatoires** (*True Random Number Generator*) (TRNG) qui exploite des phénomènes physiques jugés totalement aléatoires.

1. A noter que les termes « cryptage » ou « encryptage » sont des anglicismes qui ne sont pas corrects en français.

2. Le terme « décryptage » existe. Mais celui-ci désigne uniquement une opération de déchiffrement illégitime : le fait de retrouver le message original sans connaître la clé de chiffrement.

3. On parle souvent de cryptographie conventionnelle : deux entités s'échangent des informations à l'aide d'un secret qu'elles ont en commun.

4. On parle alors de système de clé publique-clé privée : seul le destinataire à la connaissance de la clé secrète (privée) permettant de décoder le message précédemment chiffré à l'aide de la clé publique associée.

5. Un nonce et un IV sont tous deux des données utilisées une seule fois et qui doivent être uniques. Un IV doit de plus rester secret et être imprédictible.

L'aléa statique est aussi essentiel à tout système sécurisé. Une primitive générant un tel aléa doit être capable de fournir un nombre aléatoire unique et imprévisible, mais cette fois-ci, de manière reproductible. Interrogé à de multiples reprises, le même nombre aléatoire devra être donné. Ce nombre est donc attaché au circuit et peut être utilisé comme identifiant ou comme clé secrète. L'utilisation de mémoires non volatiles (*Non-Volatile Memories*) (NVM) ou de mémoires programmables une seule fois (*One Time Programmable*) (OTP) est très largement répandue. Cependant, celles-ci permettent uniquement de stocker une valeur aléatoire ayant été générée à l'extérieur du circuit. Ces solutions sont aussi sensibles aux techniques d'ingénierie inverse et aux attaques intrusives visant à relire le contenu de ces mémoires. Elles nécessitent alors l'ajout de détecteurs d'intrusion qui viennent grever l'efficacité énergétique du circuit. Au cours des vingt dernière années, l'idée d'une nouvelle primitive de sécurité, appelée **fonction physique non-clonable** (*Physical Unclonable Function*) (PUF), a été étudiée pour palier à ces limitations. Un PUF répond conjointement aux besoins d'unicité, d'imprévisibilité et de résistance à l'altération des secrets manipulés par les systèmes d'authentification.

De nombreux types de TRNG et de PUF existent. Dans ce chapitre, nous présentons un état de l'art synthétique couvrant l'une et l'autre de ces primitives (respectivement sections 4.3 et 4.4). Nous verrons ainsi qu'une implémentation particulièrement efficace de TRNG a déjà été proposée par CHERKAoui [Che14]. Celle-ci repose sur l'utilisation d'un oscillateur auto-séquéncé (*Self-Timed Ring oscillator*) (STR), qui est un circuit asynchrone de la classe DI. Nous commencerons donc par introduire le STR et son principe de fonctionnement section 4.2.

4.2 Oscillateur en anneau auto-séquéncé

Les oscillateur en anneau (*Ring Oscillator*) (RO) sont des constructions très utilisées dans les circuits numériques. Ils peuvent servir de source d'horloge pour les circuits à horloge réactive présentés dans la section 1.3.4.1. Plus généralement, les oscillateur commandé en tension (*Voltage Controlled Oscillator*) (VCO) embarqués dans les PLL sont généralement construits à l'aide de RO [MHO05]. La fréquence d'oscillation d'un RO étant dépendante des variations PVT d'un circuit, ce type de circuit est aussi très souvent utilisé pour construire des moniteurs de température ou de variabilité du processus de fabrication. Enfin, le processus de caractérisation d'une bibliothèque de cellules standards se base aussi généralement sur la fabrication d'un grand nombre de RO incluant les différentes cellules à caractériser. Si la structure d'oscillateur en anneau à inverseurs (*Inverter Ring Oscillator*) (IRO) est la plus simple et la plus connue, il existe de nombreuses autres architectures de RO.

4.2.1 Architectures d'oscillateur en anneau

Un IRO est constitué d'un nombre impair d'inverseurs connectés ensemble de manière à former une boucle combinatoire (figure 4.1a). A l'état initial, lorsque la commande *en* est à l'état bas, cette structure est stable. La sortie *osc* est à 0, et les différents étages de l'anneau inversent successivement cette polarité jusqu'au nœud P_L qui est à l'état haut. La porte ET voit donc à ses bornes deux polarité opposées. Dès que la commande d'activation remonte, cette porte combinatoire devient transparente, laissant passer cette transition qui va pouvoir se propager librement dans le circuit. Un régime oscillatoire s'établit, dont la fréquence F_{RO} dépend du temps de propagation dans chacune des portes constituant le

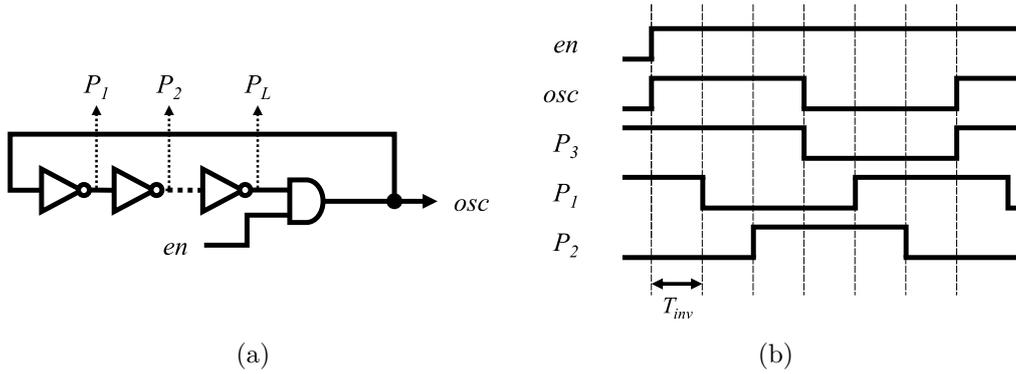


FIGURE 4.1 – Oscillateur en anneau à inverseurs, (a) architecture et (b) chronogramme.

circuit (4.1)⁶.

$$F_{RO} = \frac{1}{2 \times L \times T_{inv}} \quad (4.1)$$

Comme illustré par le chronogramme présenté en figure 4.1b, chacun des nœuds P_i de ce circuit oscille à cette même fréquence mais avec un léger déphasage équivalent au temps de propagation T_{inv} . Ce phénomène est dû au fait qu'une seule transition – ou évènement – se propage dans ce circuit.

L'oscillateur en anneau à effet transitoire (*Transient Effect Ring Oscillator*) (**TERO**) est une adaptation de l'architecture du **RO** permettant de propager plusieurs évènements simultanément [VD10; Bos+14]. Elle décompose l'anneau en différentes branches. A chacune d'elles est associée une porte ET permettant d'injecter un évènement. La figure 4.2a donne l'architecture d'un oscillateur de ce type constitué de deux branches. Lorsque la commande d'activation bascule de 0 à 1, deux transitions vont commencer à se propager dans le circuit. Dans le cas idéal, ce système entre alors dans un régime oscillatoire permanent. Cependant, du fait des imperfections de chacune des portes et des variations de fabrication, les temps de propagation des transitions montantes et descendantes ne sont pas les mêmes dans les deux branches. Comme illustré par le chronogramme 4.2b, les deux évènements vont donc petit à petit se rapprocher, et le nœud *osc* présente une série d'impulsions dont la largeur diminue ou augmente à chaque nouveau cycle. Cet oscillateur a donc en réalité un fonctionnement transitoire et sa sortie se stabilisera au bout d'un certain temps à 0 ou à 1 suite à la collision des deux transitions.

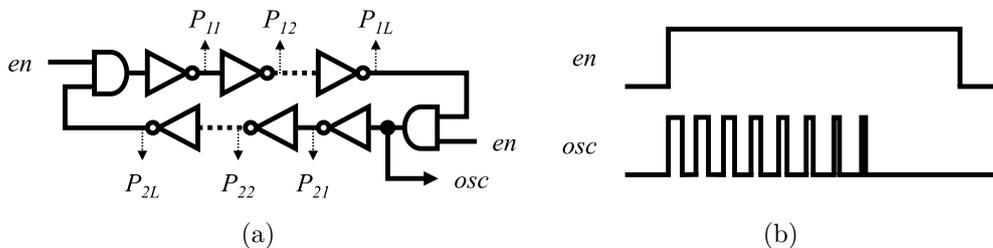


FIGURE 4.2 – Oscillateur en anneau à effet transitoire, (a) architecture et (b) chronogramme.

6. Pour simplifier, les temps de propagation dans chacun des inverseurs sont considérés équivalents (T_{inv}), celui de la porte ET est négligé.

Grâce à l'utilisation d'un protocole de poignée de main, le **STR** permet de propager de multiple évènements sans que ceux-ci ne se collisionnent. L'architecture de ce troisième type d'oscillateur est donnée dans la [figure 4.3](#). Elle correspond au bloc de contrôle d'un circuit *Micropipeline* rebouclé sur lui-même pour former un anneau. Chaque étage de cet oscillateur n'est plus simplement constitué d'un inverseur, mais une cellule de Muller est ajoutée afin d'implémenter le mécanisme de resynchronisation des évènements. On note respectivement D_{ff} et D_{rr} les temps de propagation statiques direct et inverse de chaque étage (représenté en rouge sur la [figure 4.3](#)). Ces temps correspondent aux délais de propagation de la porte du Muller suite à un évènement sur l'une de ses entrées alors que l'autre reste stable.

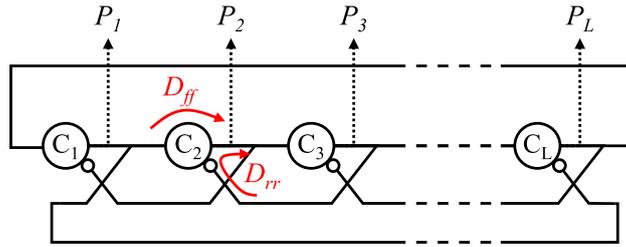


FIGURE 4.3 – Architecture d'un oscillateur auto-séquéncé.

4.2.2 Modes d'oscillation des STR

En initialisant chacun des étages à '0' ou à '1', un certain nombre d'évènements peuvent être injectés dans l'anneau. Ceux-ci vont alors se propager pendant une phase transitoire jusqu'à atteindre un régime permanent pour lequel ils se seront répartis dans l'anneau selon deux modes possibles : soit dans un mode rafale (*Burst mode*) dans lequel les évènements se propagent en groupe, soit dans un mode de répartition uniforme (*Evenly-spaced mode*) lorsque ceux-ci se répartissent tout autour de l'anneau et se propagent avec un espacement temporel constant. Ces modes d'oscillation sont facilement identifiables en observant l'une des phases P_i de l'anneau. La [figure 4.4a](#) donne les chronogrammes caractéristiques pour le mode *evenly-spaced* (en haut) et le mode *burst* (en bas) [WG01]. A noter que dans ces deux modes, chacune des phases de l'anneau présente des caractéristiques d'oscillation équivalente (fréquence, rapport cyclique).

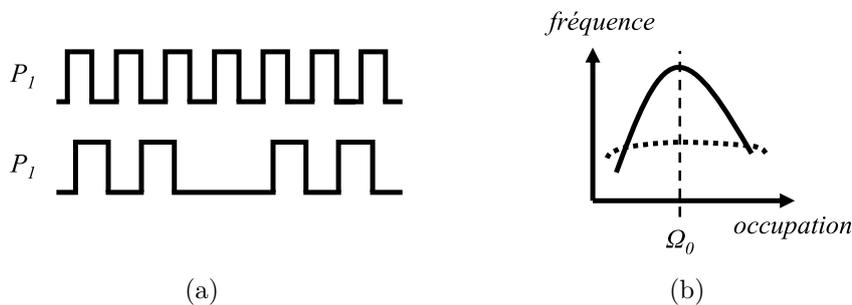


FIGURE 4.4 – Comportement d'un anneau auto-séquéncé. (a) modes d'oscillation possibles et (b) fréquence d'oscillation en fonction du taux d'occupation de l'anneau.

Ces deux modes d'oscillation sont stables et dépendent de paramètres du **STR**. Pour une implémentation donnée, le passage de l'un à l'autre de ces modes dépend uniquement du taux d'occupation de l'anneau $\Omega = N/L$, avec L le nombre d'étage et N le nombre

d'évènements injectés dans l'anneau. En pratique, le mode *evenly-spaced* est obtenu pour un intervalle d'occupation autour de Ω_0 défini par l'équation (4.2) [Ham+08]. Le mode *burst* est observé dès lors que le taux d'occupation dépasse ou est inférieur à cet intervalle.

$$\Omega_0 = \frac{D_{ff}}{D_{ff} + D_{rr}} \quad (4.2)$$

De plus, la fréquence d'oscillation de chaque phase du STR ne dépend pas directement du nombre d'étages comme c'est le cas pour les RO et TERO. Celle-ci dépend plutôt du taux d'occupation Ω . Comme illustré par la figure 4.4b, cette fréquence augmente avec le nombre d'évènements, jusqu'à atteindre un maximum en Ω_0 . Cette fréquence baisse ensuite lorsque le taux d'occupation est plus élevé.

La courbe de fréquence en fonction de l'occupation de la figure 4.4b est particulièrement caractéristique du mode *evenly-spaced* du STR. Elle n'est pas forcément centrée sur la valeur Ω_0 et, selon les caractéristiques intrinsèques de l'anneau et des cellules de Muller le composant, son étendue peut être plus ou moins importante. Dans certains cas, une courbe présentant un méplat peut être observée (en pointillés sur la figure 4.4b). Ce phénomène est caractéristique d'un mode dégradé d'oscillation, à goulot d'étranglement (*Bottleneck mode*). Il apparaît lorsque l'un des étages de l'anneau présente un défaut – un temps de propagation plus important que les autres – qui ne peut être compensé par les effets de rétroaction de l'anneau. L'observation simultanée de plusieurs phases de l'anneau peut confirmer ce phénomène. Chacune d'elles aura une fréquence identique, dépendant du temps de propagation de l'étage en question et du taux d'occupation Ω_0 , mais leur rapport cyclique évolue en fonction de la distance entre la phase observée et l'étage faisant goulot d'étranglement.

4.2.3 Effet *drafting* et effet *Charlie*

Les différents modes d'oscillation du STR sont dus à la présence de deux phénomènes analogiques dans les portes de Muller : l'effet *drafting* et l'effet *Charlie*. Nous renvoyons le lecteur intéressé aux études détaillées et aux propositions de modélisations de ces deux effets présentés dans [WG01 ; WGG02 ; Ham+08 ; Fai09]. L'effet d'aspiration, ou de *drafting*⁷, apparaît dans toute cellule combinatoire. Il résulte des phénomènes de charge et de décharge des capacités de sortie d'une porte logique. Deux commutations successives et rapprochées entraînent un temps de propagation plus court dans la porte : la sortie n'ayant pas eu le temps d'atteindre la tension haute (respectivement basse), il faudra moins de temps pour l'évènement suivant pour décharger ces capacités (respectivement charger). Intuitivement, l'effet *drafting* est à l'origine du mode *burst* des STR. Un étage venant juste de commuter aura un temps de propagation plus court pour la prochaine transition susceptible de le traverser. Les évènements dans un anneau auront donc tendance à se rapprocher les uns des autres pour finalement se propager en groupe resserré.

L'effet *Charlie* décrit l'impact du temps séparant deux évènements, arrivant sur chacune des entrées d'une cellule de Muller, sur le délai de propagation de cette porte : plus le temps de séparation est faible, plus le délai de propagation est long. Cet effet n'est en réalité pas spécifique aux cellules de Muller. CHANDRAMOULI et SAKALLAH [CS96] ont mis en évidence le même phénomène dans les portes combinatoires usuelles. Il est lié à la présence de transistors en série et à la charge plus ou moins anticipée des nœuds internes. Par exemple, dans le schéma classique d'une porte NON-OU donné en figure 4.5a, l'arrivée précoce d'un front descendant sur l'entrée x va pré-charger la capacité C_P . Lors

7. Cet adjectif fait référence aux coureurs cyclistes qui profitent de l'aspiration du coureur les précédant pour économiser leurs efforts [WGG02].

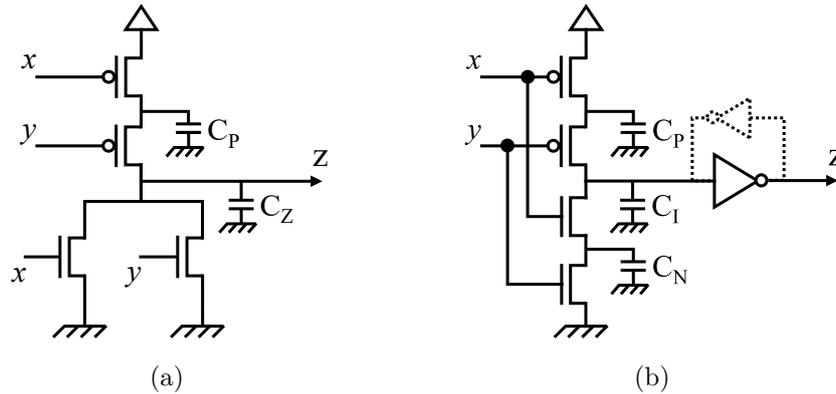


FIGURE 4.5 – Comparaison de schéma de porte logique au niveau transistors. (a) porte NON-OU et (b) porte de Muller conventionnelle.

de l'arrivée plus tardive d'un front descendant sur l'entrée y , l'effort à fournir pour faire basculer la sortie est réduit. Il suffit d'attendre que la capacité C_Z se charge pour que la nouvelle valeur s'établisse en sortie de la porte. À l'inverse, le temps de propagation lors de l'arrivée simultanée de deux transitions montantes sera accéléré dans ce type de porte. Dans ce cas, les deux transistors-n seront simultanément saturés, permettant une décharge plus rapide de la capacité C_Z ⁸. Une telle dissymétrie ne se retrouve pas dans les portes de Muller, pour lesquelles les réseaux de transistors p et n, sont tous deux en séries (figure 4.5b). Ces cellules présentent donc un effet *Charlie* quel que soit le sens du basculement des deux entrées. Dans un anneau, cet effet aura alors tendance à repousser les évènements les uns des autres. Et lorsque le taux d'occupation augmente le premier évènement pourra se retrouver dans la zone de répulsion du dernier évènement : un phénomène de rétroaction se met en place et les évènements se répartissent uniformément dans l'anneau. Lorsque le taux d'occupation augmente encore, les évènements vont retrouver un mode de propagation en rafale. Dans ce cas, la propagation des acquittements devient limitant. L'effet *Charlie* est de moins en moins actif et le phénomène de *drafting* reprend une place prépondérante.

4.2.4 Répartition homogène des phases et finesse de résolution

La principale particularité des STR réside dans leur mode de propagation *evenly-spaced* : les évènements sont répartis uniformément dans l'anneau avec une résolution temporelle plus fine que le délai minimal atteignable par n'importe quelle porte combinatoire. Dans les ROs classiques, cette résolution est limitée par le temps de propagation d'un étage de l'anneau. Comme illustré par la figure 4.1b, les différentes phases d'un tel oscillateur sont espacées de T_{inv} : un seul évènement se propage dans l'anneau. Ainsi la résolution minimale atteignable est équivalente à la demi-période d'oscillation d'un IRO à un seul étage. En revanche, les STR autorisent des différences de phase qui sont des fractions du délai de propagation d'un étage puisque plusieurs évènements évoluent simultanément dans l'anneau.

8. Bien que ces phénomènes liés aux basculements simultanés d'entrées soient connus depuis plus de vingt ans, ceux-ci commencent tout juste à être pris en compte par les outils de caractérisation des cellules standards et les outils d'implémentation physique. Ces effets étant de très faibles amplitudes, l'approximation faite en les négligeant ne devient problématique que dans les technologies très avancées (16 nm et moins).

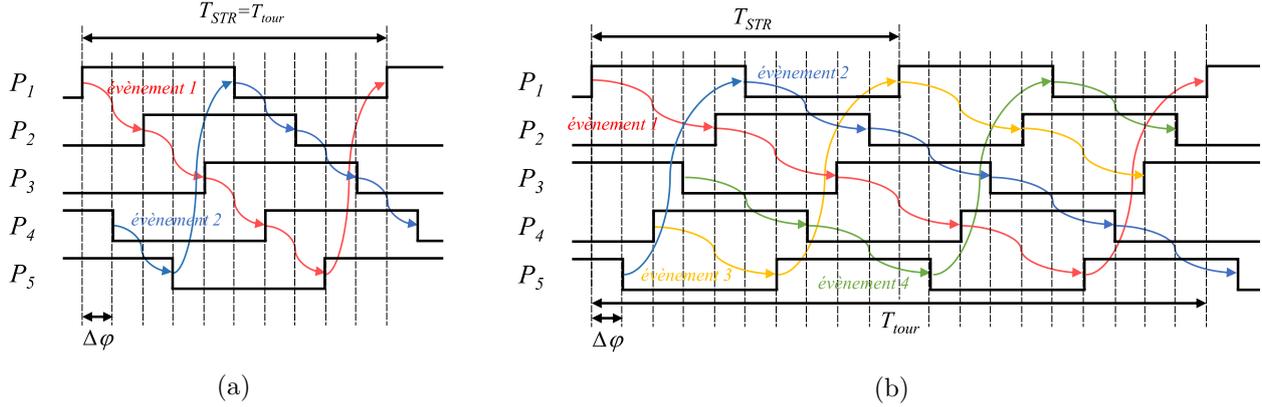


FIGURE 4.6 – Chronogramme des phases d'un STR de 5 étages. Propagation de (a) 2 évènements et (b) 4 évènements.

Prenons l'exemple d'un STR à N étages dans lequel circule L évènements. Les chronogrammes des figures 4.6a et 4.6b illustrent un tel circuit pour $L = 5$ et respectivement $N = 2$ ou $N = 4$. Nous notons T_{STR} la période d'oscillation qui est commune à chacune des phases de ce circuit. Pour faire un tour complet, un évènement se propageant dans cet anneau doit traverser successivement les N étages. Si l'on note D_{eff} le temps de propagation effectif au travers d'un étage, ce temps T_{tour} est alors donné par l'équation (4.3) :

$$T_{tour} = L \times D_{eff} \quad (4.3)$$

Par ailleurs, le temps à cet évènement de faire le tour complet, l'étage de départ va voir défiler les $N - 1$ autres évènements, chacun étant séparé de la demi période d'oscillation $T_{STR}/2$ (par définition). Ainsi T_{tour} peut être exprimé :

$$T_{tour} = N \times \frac{T_{STR}}{2} \quad (4.4)$$

En combinant les équations (4.3) et (4.4), le temps D_{eff} de propagation au travers d'un étage peut donc être exprimé :

$$D_{eff} = \frac{N}{L} \times \frac{T_{STR}}{2} \quad (4.5)$$

Ce temps n'est autre que la séparation temporelle entre les phases de deux étages successifs, représentée par les flèches de couleur sur les figures 4.6a et 4.6b. Le déphasage entre deux étages distants de n étages est donc défini par :

$$\Phi_n = n \times \frac{N}{L} \times \frac{T_{STR}}{2} \quad (4.6)$$

Ou, exprimé en degrés :

$$\Phi_n = n \times \frac{N}{L} \times 180^\circ \quad (4.7)$$

La plus petite distance $\Delta\varphi$ entre deux des phases de l'anneau peut alors être définie par (avec $\{x\}$ la partie fractionnaire de x) :

$$\Delta\varphi = \min_{n=1}^{L-1} \left(\left\{ n \times \frac{N}{L} \right\} \times 180^\circ \right) \quad (4.8)$$

S'il existe un diviseur commun d entre N et L tel que $L = d \times x$ et $N = d \times y$ avec $x, y, d \in \mathbb{N}$, alors certains étages de l'anneau ont un déphasage nul, et l'anneau présente seulement x phases distinctes. Mais dans le cas où N et L sont premiers entre eux, l'anneau présente $2 \times L$ phases distinctes. La résolution du **STR**, la distance minimale entre deux phases de l'anneau, peut alors être exprimée dans le domaine temporel par l'équation (4.9) :

$$\Delta\varphi = \frac{T_{STR}}{2 \times L} \quad (4.9)$$

Ainsi, la résolution d'un **STR** dépend directement de la taille de l'anneau et il est possible d'en contrôler finement la valeur en adaptant le nombre d'étages, tout en conservant un taux d'occupation constant. La résolution d'un **STR** peut donc être rendue plus petite que le délai de propagation dans le plus petit inverseur réalisable. En d'autres termes, nous pouvons créer un signal virtuel⁹ en mélangeant, à l'aide d'une porte **XOR**, chacune des phases d'un **STR**. La fréquence d'oscillation de ce signal virtuel est plus grande que la fréquence que l'on peut obtenir avec le plus petit **IRO** fabricable dans la technologie cible.

4.2.5 Abstraction jetons/bulles

Afin de mieux appréhender le fonctionnement d'un **STR**, CHERKAoui [Che14] propose de modéliser la propagation des événements sous forme de jetons et de bulles. Il considère pour cela le **STR** comme un circuit asynchrone implémentant un protocole 2 phases (*Micropipeline*). D'autres, comme RENAUDIN [Ren00], ont préféré dériver un modèle en considérant le **STR** comme un circuit 4 phases. Comme nous l'avons vu dans la section 1.3.4.1, lorsque l'on ignore les éléments séquentiels, les contrôleurs *Micropipeline* et **WCHB** sont identiques. Les deux modèles sont donc équivalents. Le premier, dérivé du protocole 2 phases, sera préféré pour la compréhension du mode de propagation *evenly-spaced* du fait de sa correspondance avec la notion d'évènement. Quant au second, il facilite la modélisation des perturbations du fonctionnement du **STR** (voir section 5.3).

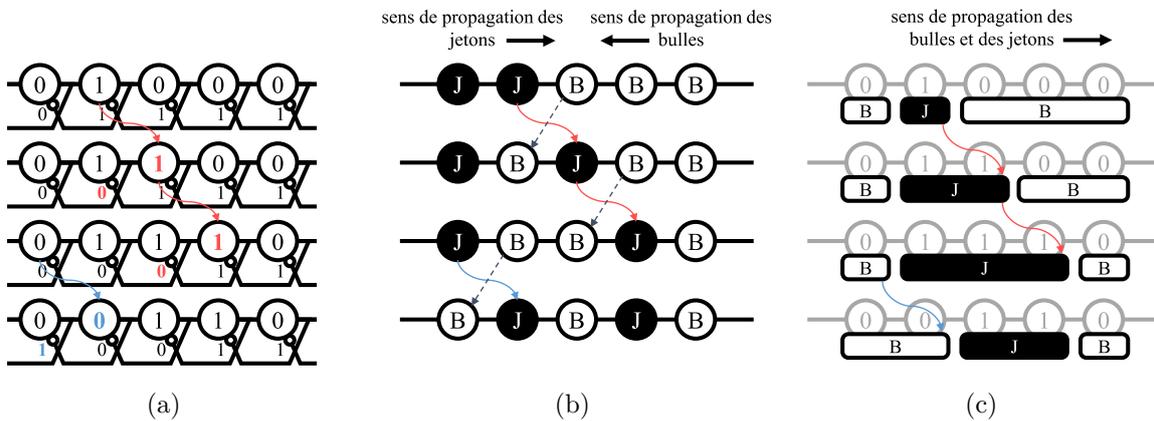


FIGURE 4.7 – Modèles d'abstraction jetons/bulles du **STR**. Illustration de la propagation d'évènements et de jetons/bulles dans une portion de 5 étages d'un **STR** aux niveaux (a) portes logiques, (b) abstraction jetons/bulles 2 phases et (c) abstraction jetons/bulles 4 phases.

La figure 4.7 présente différents niveaux d'abstraction permettant de décrire la propagation d'évènements dans un **STR**. La figure figure 4.7a donne quatre états successifs

9. Du fait de sa très haute fréquence, ce signal ne peut pas être physiquement entreteu.

d'une portion de cinq étages d'un anneau. Les flèches de couleurs mettent en avant les changements d'états des différents étages liés à la propagation des deux événements se propageant dans ce circuit (rouge et bleu).

La figure 4.7b donne la distribution des jetons et des bulles pour les cinq états du circuit décrits par la figure 4.7a, tels que définis par le modèle d'abstraction jetons/bulles 2 phases. Pour le protocole 2 phases, chaque transition représente une donnée. Ainsi un étage contient une donnée, appelé jeton, à partir du moment où la valeur de l'étage suivant est différente. A l'inverse, un étage présente une bulle dès lors que l'étage suivant contient la même valeur. Ainsi la présence de la séquence '01000' à l'état initial correspond à la succession de deux jetons et de trois bulles. Dans l'état suivant, la transition montante du signal de requête se propage du deuxième au troisième étage, ce dernier bascule de 0 vers 1. Cette propagation de la requête correspond au déplacement du jeton de l'étage 2 vers l'étage 3, dans le sens de propagation des données. Il peut aussi être compris comme le déplacement dans le sens opposé de la bulle présente dans l'étage 3 vers l'étage 2. Pour le modèle 2 phase, le nombre de jetons N_J et le nombre de bulles N_B reste donc constant, et leur somme $N_J + N_B$ est égale au nombre d'étages L de l'anneau. Comme nous l'avons vu, pour se propager un jeton doit échanger sa place avec une bulle. En d'autre terme, pour que l'anneau oscille, il faut qu'il contienne au minimum un jeton et une bulle. Par ailleurs, d'après la définition des jetons, leur nombre est nécessairement pair. Nous en déduisons que le plus petit STR réalisable possède 3 étages ($L = 3$), contient 2 jetons ($N_J = 2$) et 1 bulle ($N_B = 1$)¹⁰.

En considérant qu'un STR est un circuit 4 phases, un autre modèle jetons/bulles peut être proposé. Pour celui-ci un jeton est de taille variable et s'étale sur l'ensemble des étages consécutifs contenant simultanément la valeur 1. Ce modèle correspond donc bien à un protocole de communication 4 phases dans lequel une donnée reste valide tant que la phase de RZ n'a pas été finalisée. Cette donnée peut se propager vers l'avant et occuper plusieurs étages consécutifs si le retour d'acquiescement tarde. Les bulles sont aussi de taille variable. Elles s'étalent sur les étages successifs ayant achevé leur cycle de RZ. La figure 4.7c illustre ce phénomène. Initialement, un seul jeton est présent dans la portion de l'anneau représentée. Il est confiné sur le deuxième étage. Nous voyons par la suite ce jeton s'étaler successivement sur deux puis trois étages, pour finalement se rétracter sur les étages 3 et 4. Dans le même temps la bulle présente sur les trois derniers étages du circuit va se contracter pour être confinée sur le dernier étage, alors que la bulle à l'entrée du circuit va petit à petit progresser pour s'étendre sur les deux premiers étages. Pour ce modèle d'abstraction, jetons et bulles se propagent donc dans le même sens. Leur nombre est toujours constant, mais par définition le nombre de jetons est deux fois moindre que pour le modèle 2 phases. Un corollaire est qu'à l'exception des cas de blocage où tous les étages sont initialisés à la même valeur¹¹, le nombre de jetons N_J et le nombre de bulles N_B sont égaux. De plus, au moins un des jetons (ou des bulles) doit s'étaler sur deux étages (ou plus) pour que l'anneau oscille¹². Ainsi pour osciller, un STR doit avoir une taille $L > 2 \times N_J + 1$. Le plus petit anneau réalisable a donc une taille de $L = 3$ et contient 1 bulle et un jeton ($N_B = N_J = 1$). Ces hypothèses sont bien cohérentes avec celles obtenues pour le modèle 2 phases.

10. En réalité, il est possible d'ajouter un inverseur sur les signaux de requête et d'acquiescement entre deux étages (initialisés à 0) d'un STR afin d'insérer un seul jeton « virtuel ». De cette manière, il est possible de créer un STR à deux étages qui oscille. Cette structure n'est cependant pas régulière et nous ne l'avons donc pas retenue pour la suite de nos travaux.

11. Dans ce cas, l'anneau contient un seul jeton ou bulle et ne peut donc pas osciller.

12. Dans le cas contraire, il y a une situation d'inter-blocage. Chacun des jetons attendant la propagation du suivant pour avancer.

4.3 Générateur d'aléa dynamique : TRNG

Un générateur de nombres véritablement aléatoires (*True Random Number Generator*) (**TRNG**) est un dispositif permettant d'extraire de l'entropie à partir de phénomènes physiques réputés aléatoires. De nombreuses architectures ont été proposées dans la littérature pour répondre à cette définition. Cependant, toutes ne sont pas comparables en termes de performance et de sécurité. Dans l'idéal, en plus d'être imprévisible, la sortie d'un tel générateur se doit d'être non-manipulable. Nous verrons que ces deux prérequis ne sont pas toujours satisfaits. Dans cette section, nous définissons les concepts généraux et les principes à la base du fonctionnement des **TRNG**. Nous introduisons ainsi les notions d'imprédictibilité et de « non-manipulabilité ». Puis, nous présentons deux architectures de **TRNG** et donnons quelques chiffres quant à l'efficacité énergétique de ces dispositifs.

4.3.1 Principes de fonctionnement

La définition des **TRNG** donnée en introduction se doit d'être détaillée. S'il convient de définir plus précisément les termes *entropie* et *aléatoire*, il est aussi important de souligner la différence entre *extraction* et *génération*. Contrairement aux **PRNG** qui créent algorithmiquement des suites de nombres de manière déterministe, les **TRNG** sont en réalité des capteurs et non des générateurs. Ils viennent capturer des phénomènes physiques pour les convertir en suites de nombres qui reflètent le caractère intrinsèquement aléatoire des grandeurs observées.

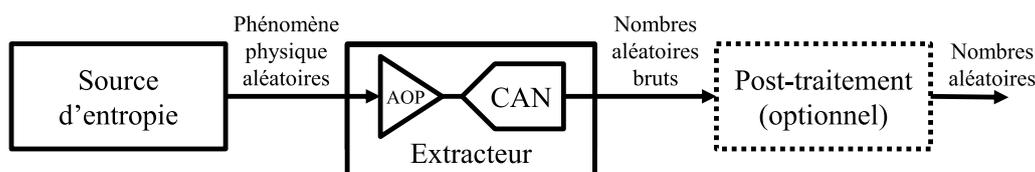


FIGURE 4.8 – Architecture générique d'un **TRNG**.

La figure 4.8 donne le schéma de principe de n'importe quel **TRNG**. La source d'entropie génère un phénomène aléatoire, qui peut s'exprimer par la fluctuation de grandeurs physiques observables telles que la tension ou le courant. L'extracteur va ensuite convertir ce phénomène physique en une suite de nombres aléatoires. Ce bloc peut être décomposé en une première partie – Amplificateur Opérationnel (**AOP**) – visant à accumuler et/ou amplifier le signal provenant de la source d'entropie, puis une deuxième partie – Convertisseur Analogique-Numérique (**CAN**) – permettant d'échantillonner ce signal pour le transformer en valeurs numériques. Enfin, un bloc de post-traitement applique des opérations arithmétiques ou algorithmiques permettant de corriger d'éventuels biais statistiques présents dans les séquences de bits. Ce dernier bloc est optionnel, mais il permet d'atteindre les plus hauts niveaux de certifications des **TRNG** (par exemple, la classe PTG.3 de la notes d'application et d'interprétations de schéma (*Anwendungshinweise und Interpretationen im Schema*) (**AIS**) 31 [KS01]). Il reste en effet intéressant même dans le cas d'un générateur parfait (sans biais). Grâce à la périodicité des algorithmes utilisés, il permet de garantir un fonctionnement aléatoire en sortie du générateur malgré la défaillance d'un des étages précédents. Il laisse ainsi le temps à d'éventuels alarmes et tests en ligne de détecter ce problème et d'agir en conséquence (blocage du circuit, invalidation des clés, destruction de données. . .).

Un phénomène *aléatoire* doit être différencié d'un phénomène *chaotique*. Ce dernier est purement déterministe et ses effets sont théoriquement prédictibles à condition d'en connaître toutes les causes. Ainsi, un système non-linéaire est dit chaotique à partir du

moment où une infime variation dans ses conditions initiales produit des comportements très différents dans les états suivants du système. L'imprévisibilité de tels systèmes repose donc uniquement sur la difficulté technique à mesurer avec une précision suffisante les conditions initiales. Même si des suites de nombres aléatoires avec de très bonnes propriétés statistiques peuvent être générées à partir de systèmes chaotiques, d'un point de vue sécurité, il reste difficile de garantir que les conditions initiales ne seront jamais mesurables. Pour extraire des nombres véritablement aléatoires, un TRNG doit se reposer sur un phénomène non-déterministe.

En réalité, il n'existe que très peu de véritables sources d'aléa dans les circuits intégrés. Une grande majorité des contributeurs au bruit apparent sur des grandeurs comme la tension ou le courant sont en fait des sources déterministes. Ce sont principalement des phénomènes de diaphonie par couplage capacitif ou par propagation de bruits au travers du substrat, des rails d'alimentation ou de rayonnements électromagnétiques, etc. La communauté scientifique s'accorde à dire que la seule source d'entropie fiable dans les circuits électroniques provient du bruit thermique qui apparaît dans tout conducteur électrique, indépendamment d'une quelconque tension, du fait de l'agitation thermique des porteurs de charges. Ce phénomène se traduit par un bruit blanc, et a donc un spectre plat sur toute la bande passante¹³. Le bruit de scintillement, ou bruit en excès, est aussi considéré comme une source aléatoire, cependant sa dépendance avec la fréquence (c'est un bruit rose avec une densité spectrale en $1/f^\alpha$) induit des phénomènes d'autocorrelation et le rend plus facilement manipulable. Il n'est donc généralement pas considéré pour la construction de TRNG.

4.3.2 *Jitter* dans les oscillateurs en anneau

Le bruit thermique a de nombreuses conséquences observables dans un circuit intégré. La mesure la plus directe consiste à quantifier la variation de tension aux bornes d'une résistance. Le bruit thermique peut aussi entraîner des variations dans le temps ou la valeur de résolution d'un état métastable, ou dans le temps de propagation au travers d'une porte logique. Très souvent, ce sont les variations de la période d'oscillation d'une horloge, le *jitter*, qui sont utilisées pour quantifier ce bruit.

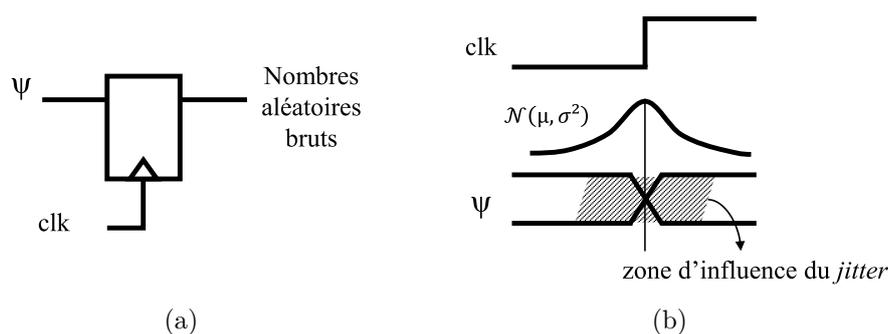


FIGURE 4.9 – Principe des techniques d'extraction du *jitter* basé sur la coïncidence de deux signaux. (a) Architecture et (b) chronogramme.

Les techniques d'extraction du *jitter* peuvent être classées en deux grandes catégories : par coïncidence et par accumulation. Les premières s'intéressent à une unique réalisation

13. Cette approximation est valable pour des basses fréquences jusqu'à quelques gigahertz, au-delà desquelles une approche quantique est nécessaire.

du *jitter*. Comme illustré en figure 4.9, elles tentent de faire coïncider l'instant d'échantillonnage d'une bascule D avec une transition du signal bruité. Les deuxièmes vont au contraire, chercher à accumuler suffisamment de *jitter* pour qu'un impact puisse être visible sur la valeur d'un bloc comptant le nombre d'oscillations du signal bruité sur une période donnée (figure 4.10).

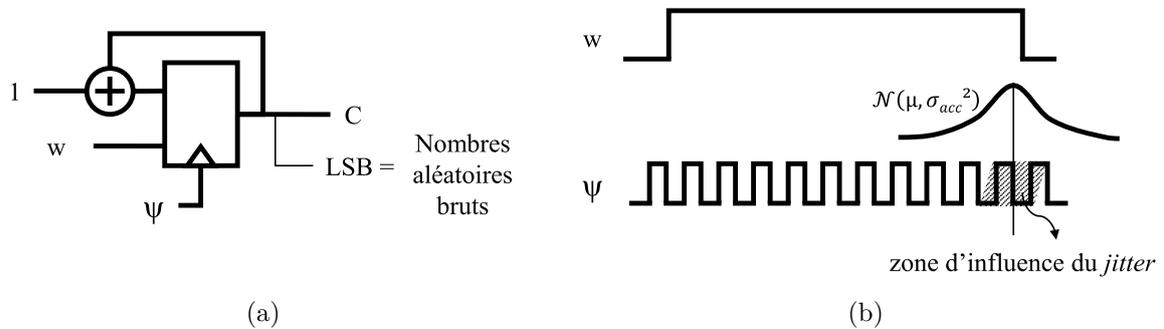


FIGURE 4.10 – Principe des techniques d'extraction du *jitter* basé sur l'accumulation. (a) Architecture et (b) chronogramme.

La période d'oscillation d'un anneau est traditionnellement modélisée par une loi normale, dont l'écart type σ_T permet de quantifier le *jitter*. Dans les RO, le *jitter* au niveau de la période est souvent décomposé en la somme des variations $\sigma_{\acute{e}tage}$ de délais apparaissant dans chacun des étages composant l'anneau [Fis+08]. Ces deux écarts types peuvent alors être reliés par l'équation (4.10) :

$$\sigma_T = \sqrt{2N} \times \sigma_{\acute{e}tage} \quad (4.10)$$

Par ailleurs, différents travaux se sont intéressés au *jitter* des STR [Che+12; Che14; Nou+18]. Ils montrent que ce type d'oscillateur présente un *jitter* Gaussien sur chacun de ses étages, qui peut être considéré indépendant et qui s'accumule de manière très similaire aux IRO classiques.

L'évaluation du *jitter* est une tâche complexe¹⁴. Les variations à mesurer sont généralement de l'ordre de la picoseconde et elles sont très vite masquées par le bruit de quantification et celui des appareils de mesure. Une approche statistique est nécessaire et le *jitter* est ainsi habituellement estimé au travers de l'écart type d'une distribution. Pour une meilleure précision, les mesures doivent être faites à l'aide de matériel embarqué directement dans la puce. Par ailleurs, il est difficile de discriminer la composante du *jitter* provenant uniquement du bruit thermique¹⁵. Pour plus de détails, nous redirigeons le lecteur vers quelques travaux s'étant penchés sur ce problème épineux [Val+08; LB15; Yan+17; Nou+18].

4.3.3 Imprédictibilité

La distinction entre générateur et extracteur¹⁶ pointe une des grandes problématiques liées à la conception des TRNG : pour prouver la qualité des nombres aléatoires extraits

14. Au moins aussi complexe que la construction d'un TRNG, ces derniers étant par définition des dispositifs permettant de faire cette mesure.

15. Les autres composantes, déterministes et/ou manipulables, ne doivent pas être prises en compte dans les modèles stochastiques pour assurer le plus haut niveau de sécurité.

16. Dans la littérature cette différence est rarement faite. Pour ne pas perturber le lecteur et pour être cohérent avec le terme TRNG nous emploierons indifféremment ces deux termes dans la suite de ce manuscrit.

par un tel dispositif, les tests statistiques ne sont pas suffisants. En effet les séquences de bits générés par un bon PRNG passent avec succès l'ensemble des tests statistiques, mais elles sont pourtant parfaitement prédictibles (à condition de connaître la graine et l'algorithme utilisé). Leur utilisation dans un système cryptographique est donc particulièrement hasardeuse.

A ce stade, il est important de préciser que si de nombreuses applications utilisent des nombres aléatoires, toutes ne demandent pas leur imprédictibilité. C'est le cas d'applications de simulation numérique (simulations Monte-Carlo), de calcul stochastique ou encore d'échantillonnage, pour lesquels une bonne qualité statistique est suffisante. Un TRNG est donc surdimensionné pour de telles applications qui peuvent généralement se contenter d'un PRNG.

La première qualité d'un TRNG est donc l'imprédictibilité de sa sortie. La garantie de ce caractère des TRNG repose sur deux types de preuves : une évaluation *a posteriori*, permettant de vérifier l'absence de défauts statistiques dans une suite de nombres aléatoires de grande taille, et une preuve *a priori*, basé sur un modèle stochastique, garantissant que l'aléa est bien extrait de phénomènes non-déterministes.

La figure 4.11 reprend l'architecture générique d'un TRNG (figure 4.8) en précisant les différents éléments permettant de garantir l'imprédictibilité. Les étapes d'évaluation statistique sont présentées en bleu, celles liées à la modélisation de l'entropie en rouge.

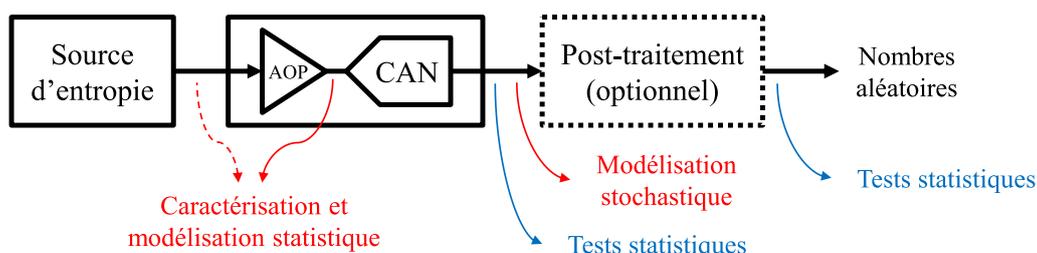


FIGURE 4.11 – Modèle de TRNG garantissant l'imprédictibilité.

4.3.3.1 Évaluation statistique

Évaluer le caractère aléatoire d'une suite n'est pas aisé. Par définition, une suite aléatoire est infinie et toute séquence de bits a une certaine probabilité d'y apparaître, même la plus improbable d'entre elles¹⁷. Il n'est donc pas possible de se baser sur l'apparition de certaines séquences pour déterminer la présence de défauts. Il est d'ailleurs techniquement impossible d'analyser une suite infinie de bits. Une approche statistique est nécessaire. Celle-ci va s'intéresser aux fréquences d'apparition de séquences dans une suite finie mais suffisamment longue¹⁸ et vérifier qu'elles correspondent à ce que l'on peut attendre d'une suite issue d'un générateur idéal. Ce type de tests dits « statistiques », va s'intéresser à la crédibilité d'une hypothèse H_0 , appelée hypothèse nulle. De manière générale, H_0 correspondra à « la suite analysée est aléatoire » ou plus précisément « la caractéristique X de la suite analysée est comparable à celle d'une suite issue d'un générateur idéal ». Cette hypothèse sera évaluée en fonction d'un risque seuil α , et sera rejetée dans le cas où elle

17. Dans une suite parfaitement aléatoire, la séquence '00000000' est tout aussi probable qu'une autre séquence qui serait intuitivement plus aléatoire comme '01001110'. L'absence de longue séquence de bits à la même valeur est d'ailleurs souvent caractéristique d'une suite binaire générée par l'homme[Rév05].

18. Ce paramètre dépend du test en lui-même et de la précision recherchée, c'est-à-dire le taux de faux-positifs et de faux-négatifs que l'on accepte.

est considérée improbable. Dans les faits, un ensemble de tests est constitué, chacun d'eux s'intéressant à une caractéristique statistique particulière de la suite.

TABLE 4.1 – Liste des tests de la suite SP800-22.

Nom du test	Paramètres	Description
Frequency (monobit) test	$n \geq 100$	La proportion de 0 et de 1 dans la suite binaire de taille n doit être proche $1/2$
Frequency test within a block	$n \geq 100$ et $M > 0.01n$	La proportion de 1 dans des blocs de taille M de la suite binaire de taille n doit être proche de $1/2$.
Runs test	$n \geq 100$	Analyse du nombre de transitions 0->1 et 1->0 dans la suite binaire de taille n et détecte si ces oscillations sont trop rapides ou trop lentes.
Longest run of ones test	$\{n, M\} = \{128, 8\}$ ou $\{6272, 128\}$ ou $\{750000, 10^4\}$	Analyse du nombre et de la taille des séquences de '1' consécutifs dans les blocs de taille M de la suite binaire de taille n .
Binary matrix rank test	$n \geq 38912$ et $Q = 32$	Vérifie qu'il n'y a pas de dépendance linéaire entre des séquences consécutives de Q bits dans la suite binaire de taille n .
Discrete FFT test	$n \geq 1000$	Vérifie qu'il n'y a pas de motif périodique dans la suite binaire de taille n à l'aide d'une transformée de Fourier discrète.
Non-overlapping template matching test	indéfini	Vérifier que certains motifs prédéfinis ne se répètent pas de manière non-périodique dans la suite binaire de taille n . Les motifs ne se recouvrent pas.
Overlapping template matching test	$n \geq 10^6$	Vérifier que certains motifs prédéfinis ne se répètent pas de manière non-périodique dans la suite binaire de taille n . Les motifs peuvent se recouvrir.
Maurer's "universal statistical" test	$n \geq 387840$	Vérifie que le taux de compression possible de la suite binaire de taille n est petit.
Linear complexity test	$n \geq 10^6$ et $5000 > M > 500$	Vérifie que la taille moyenne des LFSR équivalent, permettant de générer les différentes séquences de M bits de la suite binaire de taille n , est assez grande.
Serial test	$m < \lfloor \log_2 n \rfloor - 2$	Vérifie que toutes les séquences de taille jusqu'à m sont équiprobables dans la suite binaire de taille n .
Approximate entropy test	$m < \lfloor \log_2 n \rfloor - 5$	Compare les fréquences de toutes les séquences de taille m et $m + 1$ dans la suite binaire de taille n .
Cumulative sums test	$n \geq 100$	Vérifie que le <i>random walk</i> décrit par la suite binaire de n n'a pas une excursion maximale trop faible ou trop importante.
Random excursions test	$n \geq 10^6$	Analyse des cycles dans le <i>random walk</i> décrit par la suite binaire de n .
Random excursions variant test	$n \geq 10^6$	Analyse du nombre de visites de chaque état du <i>random walk</i> décrit par la suite binaire de n .

Une des suites de tests de référence est celle proposée par le *National Institute of Standard and Technology* (NIST) dans sa publication spéciale (*Special Publication*) (SP)800-22 [Ruk+10]¹⁹. Elle est composée de 15 tests plus ou moins complexes permettant de vérifier différentes caractéristiques statistiques d'une suite de bits. Pour être significatifs, certains de ces tests doivent être appliqués sur des suites de taille importante. Ainsi, une taille de plus de 10^9 bits est en général utilisée, ce qui représente des fichiers d'environ 125 Mo. Le tableau 4.1 présente la liste des différents tests, une courte description expliquant l'objectif visé et le nombre minimum de bits sur lesquels ils s'appliquent.

19. Le standard FIPS 140-2 est aussi souvent mentionné. Dans sa version initiale, il définissait une suite de 4 tests statistiques basiques : *monobit*, *poker*, *runs* et *long runs*. Mais les révisions successives de ce standard ont supprimé ces tests et pointent maintenant sur la SP800-22 qui reprend et complète ces tests.

Le NIST propose aussi, au travers de sa série de publications SP800-90 [BFW15; Tur+18; BK16], une série de recommandations sur la manière de construire et d'évaluer les générateurs de nombres aléatoires (PRNG et TRNG). En particulier la SP800-90B s'intéresse aux TRNG, appelés sources d'entropie dans ce document, et à la manière de vérifier la qualité des suites binaires générées. Elle définit pour cela une série de tests complémentaires dans le but d'établir si ces suites peuvent être assimilées à une variable Indépendante et Identiquement Distribuée (IID) et donnent ensuite différents algorithmes permettant d'évaluer l'entropie suivant la catégorie de variable aléatoire. Ces recommandations sont particulièrement utilisées par les organismes de certification (CAVP, CMVP, CCTL, etc.). En Europe, la Office fédéral de la sécurité des technologies de l'information (*Bundesamt für Sicherheit in der Informationstechnik*) (BSI) propose des recommandations similaires au travers de sa note AIS31 [KS01].

Même s'il était possible de construire une infinité de tests pour valider le caractère aléatoire de la sortie d'un TRNG, ceux-ci ne seraient cependant pas suffisants pour prouver l'imprédictibilité. Une illustration concrète de cette vérité apparaît en analysant les décimales d'un nombre irrationnel tel que π . Tous les tests statistiques s'accorderont à dire que ce nombre a toutes les propriétés d'un nombre véritablement aléatoire. Et pourtant les 31 415 926 535 897 premières décimales de π sont connues²⁰ et un TRNG basé sur cette séquence déterministe serait de piètre qualité. Comme rappelé par les auteurs des différents standards, les tests statistiques permettent uniquement de déceler des faiblesses statistiques qui seraient des preuves du caractère non aléatoire d'un TRNG. Ils ne peuvent cependant se substituer à une analyse *a priori* et à la construction d'un modèle stochastique.

4.3.3.2 Modélisation

L'entropie, telle que définie par SHANNON [Sha48], est une mesure de l'imprédictibilité d'une information délivrée par une source. Si l'on modélise cette source par une variable aléatoire discrète X ayant pour réalisations n valeurs ou symboles différents, l'entropie H de cette source est définie par l'équation (4.11) :

$$H(X) = - \sum_{i=1}^n P_i \log_2 P_i \quad (4.11)$$

avec $P_i = P(X = i)$ la probabilité que X prenne la valeur i . Intuitivement, on peut interpréter $H(X)$ comme le nombre de bits que la source d'information doit fournir en moyenne pour déterminer avec certitude un tirage de X . Prenons pour exemple de source d'entropie le tirage d'une pièce de monnaie. L'ensemble des états possibles de la pièce suite à un tirage peut être divisé en deux symboles a et b correspondant respectivement à : « la pièce est tombée sur la tranche » et « la pièce est sur pile ou face ». Cette traduction du phénomène physique vers un ensemble de symboles constitue l'extracteur d'entropie. Bien entendu la probabilité de a est négligeable par rapport à celle de b . L'entropie associée à ce couple source/extracteur sera presque nulle, puisqu'un récepteur n'aura besoin d'aucune information pour deviner le prochain symbole. Mais en répartissant différemment chacun des états de la pièce avec pour définition et probabilité :

- a la pièce est tombée sur face, $P_a = 0.4999999$
- b la pièce est tombée sur pile ou sur la tranche, $P_b = 0.5000001$

l'entropie de ce nouveau système sera très proche de 1 ($-P_a \log_2(P_a) - P_b \log_2(P_b) \approx 1$). La transmission d'un bit sera donc nécessaire pour être sûr de transmettre complètement

20. <https://blog.google/products/google-cloud/most-calculated-digits-pi/>

l'information d'état de la source. Autrement dit, il est impossible de prédire l'état de cette source avec moins de 50 % de chance de se tromper. Ainsi, l'entropie de chaque bit généré par un TRNG de qualité devra être la plus proche possible de 1. Par exemple, le standard AIS31 de la BSI fixe un seuil d'entropie $H(X) > 0.997$ [KS01]. L'objectif d'un modèle stochastique est de pouvoir estimer *a priori* l'entropie en sortie d'un TRNG. Pour pouvoir construire un tel modèle il faut être capable :

- 1) d'identifier clairement le phénomène physique utilisé comme source d'entropie,
- 2) de caractériser cette source d'entropie puis de la modéliser sous forme de variables aléatoires (en général à l'aide d'une loi Gaussienne d'espérance μ et de variance σ^2)
- 3) de construire un modèle décrivant le processus d'extraction et établissant le lien entre les variables aléatoires, les paramètres potentiellement ajustables du générateur et les probabilités des symboles de sortie
- 4) de calculer l'entropie équivalente

Du fait de la difficulté à décrire précisément le processus d'extraction et l'ensemble des contributeurs au caractère aléatoire d'un TRNG, une approche conservatrice est généralement utilisée. On calcule ainsi une entropie minimale, qui garantira un fonctionnement correct d'un système cryptographique dans le pire des cas.

4.3.3.3 Le contre-exemple du MURO-TRNG

La nécessité d'établir un modèle stochastique pour les TRNG peut être illustrée en étudiant une architecture de générateur très utilisé : le générateur de nombres véritablement aléatoires basé sur une multitude d'oscillateurs en anneau (*Multiple Ring Oscillators based TRNG*) (MURO-TRNG) [SMS07]. Ce générateur, dont l'architecture est présentée en la figure 4.12, utilise un grand nombre de RO²¹ identiques et indépendants, dont les sorties sont recombinaées à l'aide d'un arbre de cellules XOR. Le signal Ψ ainsi formé peut alors être échantillonné à relativement haute fréquence pour obtenir un flux binaire aléatoire. L'idée sous-jacente est d'accumuler de l'entropie simultanément dans un nombre important de capteurs (les ROs) pour être sûr que lors de l'échantillonnage, la valeur du signal Ψ soit totalement aléatoire. Bien que ce type d'oscillateur produise des suites binaires d'apparence aléatoire, les sources de cet aléa ne sont pas toutes non-déterministes.

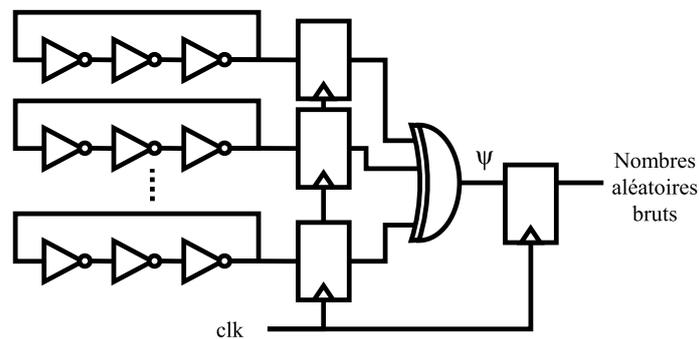


FIGURE 4.12 – Architecture d'un MURO-TRNG.

Pour mettre en évidence cette faille de sécurité, BOCHARD *et al.* [Boc+11] proposent un modèle permettant de simuler le comportement d'un tel circuit. En définissant des

21. Généralement des IRO sont utilisés [SMS07 ; WT08]

fréquences légèrement différentes pour chacun des oscillateurs²², ils ont pu générer des séquences binaires semblant aléatoires. Le modèle utilisé est pourtant purement comportemental et ne fait appel à aucune variable aléatoire. Ce comportement s'explique par le pseudo-aléa, généré par la dérive entre les différentes phases. Les auteurs montrent ainsi qu'avec seulement 18 IRO, cette architecture génère un aléa déterministe suffisant pour passer l'ensemble des tests SP800-22. Cette expérience ne remet pas en cause la validité de ce type de TRNG, mais montre clairement que son dimensionnement ne peut pas se baser sur des tests statistiques. Dans le cas contraire, l'extraction par un attaquant de la fréquence de chaque oscillateur – par exemple, en analysant le rayonnement électromagnétique [Bay+13] – pourrait donner suffisamment d'information sur l'aléa généré pour compromettre tout le système.

4.3.4 Non-manipulabilité

L'imprédictibilité des TRNG permet de construire des protocoles cryptographiques sûrs. Cependant, comme pour toute application sécurisée, les potentielles vulnérabilités matérielles doivent aussi être prises en compte. Pour assurer un niveau de sécurité élevé, il est ainsi nécessaire de s'assurer de la « non-manipulabilité » des TRNG. Sans quoi, un attaquant pourrait prendre la main sur le flux binaire aléatoire et, de cette manière, se faciliter la cryptanalyse des informations confidentielles. En dehors des attaques passives cherchant à récupérer directement les informations et qui ne sont pas spécifiques aux TRNG, une analyse précise et un modèle de menace doit être construit pour anticiper les attaques actives visant spécifiquement ces dispositifs. En particulier, la source d'entropie est un élément sensible qui doit être protégé.

4.3.4.1 Retour sur le MURO-TRNG

Le MURO-TRNG présenté précédemment, permet d'illustrer le concept de « manipulabilité ». Dans [Boc+11], les auteurs ont montré que près du quart des oscillateurs implémentés dans un réseau de portes programmables *in situ* (*Field-Programmable Gate Array*) (FPGA) se retrouvent, du fait qu'ils partagent les mêmes rails d'alimentation et de la présence de couplage capacitif et/ou électromagnétique, à osciller en phase. Ce phénomène remet en cause le principe même de ce TRNG, car le nombre d'oscillateurs effectivement indépendants est moindre qu'attendu. En effet, le modèle stochastique proposé par SUNAR, MARTIN et STINSON [SMS07] se base sur une hypothèse probabiliste au niveau des oscillateurs – « si suffisamment d'oscillateurs sont utilisés, l'échantillonnage du signal Ψ se fera nécessairement à proximité d'un front bruité » – qui n'est réaliste que si ceux-ci sont indépendants. Par ailleurs, BAYON *et al.* [Bay+12] ont montré que ce phénomène de verrouillage des oscillateurs pouvait être contrôlé par injection électromagnétique, et qu'il est donc possible de manipuler ce TRNG pour pouvoir en réduire l'entropie et prédire ses bits de sortie.

4.3.4.2 Tests en ligne et monitoring de la source

Une première approche pour se protéger des tentatives de manipulation se base sur l'utilisation de tests en ligne, ou *health tests*. Trois types de tests différents sont préconisés dans [Tur+18] : les tests de démarrage, les tests continus et les tests à la demande. Ils ont pour but de détecter aussi rapidement que possible une déviation du fonctionnement

22. Ce qui, du fait des variations de fabrication, est bien plus proche de la réalité que l'hypothèse de IRO strictement identiques.

nominal du générateur. Ces tests s'intéressent aux données binaires et vérifient leurs propriétés statistiques. Ils sont très proches de ceux utilisés dans le cadre de la validation et de la certification des TRNG. Ils sont cependant simplifiés pour pouvoir être intégrés directement dans le circuit avec un surcoût matériel raisonnable et pour s'adapter aux biais généralement inhérents à la source d'entropie et ainsi limiter le nombre d'alarmes faussement positives ou faussement négatives. Ainsi ces tests doivent être configurés au regard de l'entropie minimum H_{min} établie par le modèle stochastique.

Le NIST recommande deux tests continus dans sa SP800-90B [Tur+18] : un test de répétition et un test de proportion. Le premier cherche à détecter des séquences continues du même symbole alors que le second s'intéresse à un défaut dans la proportion de 1 et de 0. Ils sont les pendants des tests *monobit* et *longest runs* de SP800-22 [Ruk+10]. De son côté, l' AIS31 [KS01] propose d'autres procédures de test basées sur la qualité d'ajustement (test du χ^2 d'ajustement).

Une approche complémentaire se base sur des tests plus spécifiques, construits pour valider le bon fonctionnement de la source d'entropie. Ils ne s'intéressent plus aux flux binaires, mais directement à des propriétés bien particulières de la source d'entropie, qui sont caractéristiques de son fonctionnement (mesures de fréquence, estimation du jitter, etc.). L'idée est donc d'effectuer des mesures au plus proche de la source d'entropie de manière à détecter toutes défaillances avant même qu'elles n'affectent le flux de données aléatoires. En général, le phénomène physique aléatoire n'est pas observable directement, la mesure de celui-ci en sortie de l'étage d'amplification est donc préférée. Afin de limiter le surcoût matériel, ce monitoring peut avoir une précision réduite. Il doit cependant être capable de détecter des situations caractéristiques de défaillance et générer différentes alarmes avec les bons niveaux de criticité.

Par exemple, le monitoring de la source d'entropie du MURO-TRNG devra vérifier simultanément que les oscillateurs sont opérationnels, qu'ils oscillent à une fréquence similaire et qu'ils ne sont pas verrouillés entre eux. Cela peut être implémenté en premier lieu par l'ajout d'un fréquencemètre sur chacun des oscillateurs – un simple compteur du nombre d'oscillation sur une période donnée. La sortie de chacun de ces blocs peut alors être utilisée pour générer différentes alarmes. La détection de l'arrêt d'un ou plusieurs oscillateurs devra déclencher une alarme de défaillance totale. La détection d'une excursion de fréquence trop importante entre les différents oscillateurs pourra déclencher une alarme secondaire. La détection de fluctuations sur la fréquence d'un ou plusieurs oscillateurs pourra aussi être identifiée comme suspecte et déclencher une séquence de test à la demande. La détection du verrouillage est plus délicate à implémenter. Elle devra passer par une mesure du déphasage entre les différents oscillateurs. Un tel bloc pourra alors générer une alarme dès qu'un nombre trop important d'oscillateurs sont verrouillés²³. Bien sûr, chacune de ces alarmes doit être adaptées au fonctionnement du générateur et au modèle stochastique utilisé.

La figure 4.13 complète l'architecture présentée en figure 4.11 et précise les différents niveaux de test permettant de garantir la « non-manipulabilité » d'un TRNG.

4.3.5 Exemples de TRNG

De nombreuses architectures de TRNG ont été imaginées pour tenter de capter de l'entropie le plus efficacement possible. Métastabilité dans un verrou, tension aux bornes d'une résistance, jitter des RO, etc., chacune utilise l'une ou l'autre des conséquences

23. D'après [Boc+11], le verrouillage entre différents oscillateurs peut apparaître dans un fonctionnement nominal (sans attaque). La question est donc de savoir si le nombre d'oscillateurs non verrouillés est suffisant pour atteindre l'objectif d'entropie.

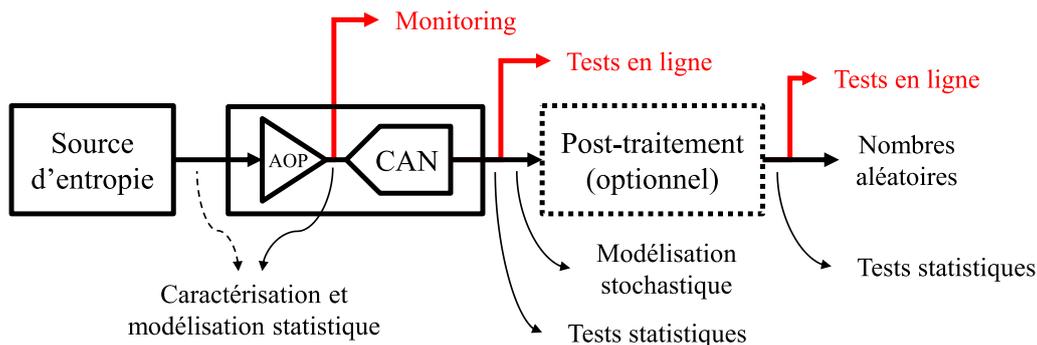


FIGURE 4.13 – Architecture de TRNG garantissant la non-manipulabilité.

physiques du bruit thermique. Mais force est de constater que les architectures à base de RO sont celles ayant le plus de succès du fait de leur facilité d'implémentation, de leur portabilité sur FPGA et de la modélisation aboutie de leur source d'entropie. PETURA *et al.* [Pet+16] recensent six architectures utilisant des oscillateurs et étant compatibles avec les recommandations de la note AIS31 [KS01]. Dans cette section nous présenterons deux de ces générateurs. Le premier est particulièrement intéressant par sa simplicité et par la robustesse du modèle stochastique l'accompagnant. Le deuxième se démarque par son efficacité et l'utilisation d'un oscillateur auto-séquenté (STR).

4.3.5.1 Le ERO-TRNG

Le premier TRNG implémenté dans un circuit intégré, le générateur de nombres véritablement aléatoires élémentaire basé sur des oscillateurs en anneau (*Elementary Ring Oscillators based TRNG*) (ERO-TRNG), a été initialement proposé par FAIRFIELD, MORTENSON et COULTHART [FMC85] du laboratoire Bell. Il utilise deux oscillateurs. Le premier, à basse fréquence, échantillonne le second oscillant à plus haute fréquence. L'échantillonnage est effectué avec une simple bascule D. Ce TRNG a été largement étudié et amélioré par la communauté scientifique. La figure 4.14a présente l'architecture de ce générateur telle que couramment utilisée. Les deux oscillateurs sont implémentés à l'aide de IRO identiques, permettant ainsi de réduire l'impact des sources de bruit globales (les deux oscillateurs sont impactés de la même manière). Afin d'obtenir un signal basse-fréquence, la sortie d'un des oscillateurs est connectée à un bloc de division de fréquence de facteur K_D . Intuitivement, l'idée est de venir accumuler du bruit dans l'oscillateur à basse fréquence, pour ensuite être sûr que l'échantillonnage du signal à haute fréquence ait autant de chance de capturer un 0 qu'un 1. Ce fonctionnement est illustré par la figure 4.14b : σ doit être suffisamment grand vis-à-vis de T_1 pour que l'aire cumulée des zones bleues et celle des zones rouges sous la Gaussienne soient équivalentes.

BAUDET *et al.* [Bau+11] proposent un modèle stochastique basé sur un processus de Wiener pouvant facilement être adapté au ERO-TRNG. Ce modèle permet alors de calculer une limite basse H_{min} à l'entropie par bit pouvant être extraite par ce dispositif (4.12). Elle dépend de l'écart type σ_N du jitter provenant du bruit thermique, de la période des oscillateurs T_1 et T_2 et du facteur de division K_D :

$$H_{min} \approx 1 - \frac{4}{\pi^2 \ln(2)} e^{-\frac{\pi^2 \sigma_N^2 K_D T_2}{T_1^3}} \quad (4.12)$$

Par ailleurs, malgré le faible débit de nombres aléatoires autorisé par cette architecture, sa « non-manipulabilité » peut être garantie en vérifiant seulement deux propriétés : les ROs oscillent et ne sont pas verrouillés entre eux.

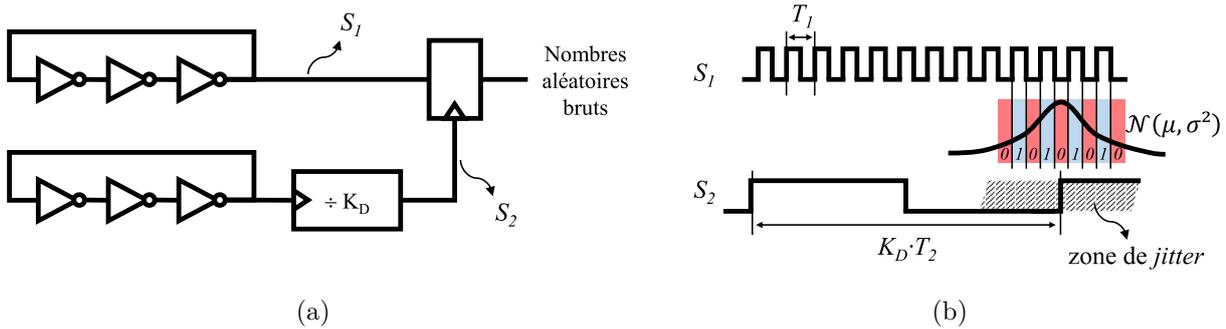


FIGURE 4.14 – Principe de fonctionnement du **ERO-TRNG**. (a) Architecture et (b) chronogramme.

4.3.5.2 Le STR-TRNG

Plus récemment, CHERKAOUI *et al.* [Che+13a] ont proposé un générateur de nombres véritablement aléatoires basé sur un oscillateur auto-séquence (*STR based TRNG*) (**STR-TRNG**). Cette architecture exploite la répartition uniforme des phases du **STR** afin d'extraire de l'entropie de manière très efficace. Elle est illustrée par la figure 4.15a. De manière similaire au **MURO-TRNG** (figure 4.12), les phases du **STR** regroupées à l'aide d'un arbre de **XOR**. Si le **STR** est dans son mode *evenly-spaced*, le signal Ψ ainsi construit peut être vu comme une horloge oscillant à la fréquence $F_{virtuelle} = 1/2\Delta\varphi$. En réalité, un tel signal ne peut se propager au travers de l'arbre de **XOR**. L'utilisation d'un premier étage de bascule D (en pointillés sur la figure 4.15a) permet de reconstruire ce signal après échantillonnage à plus basse fréquence.

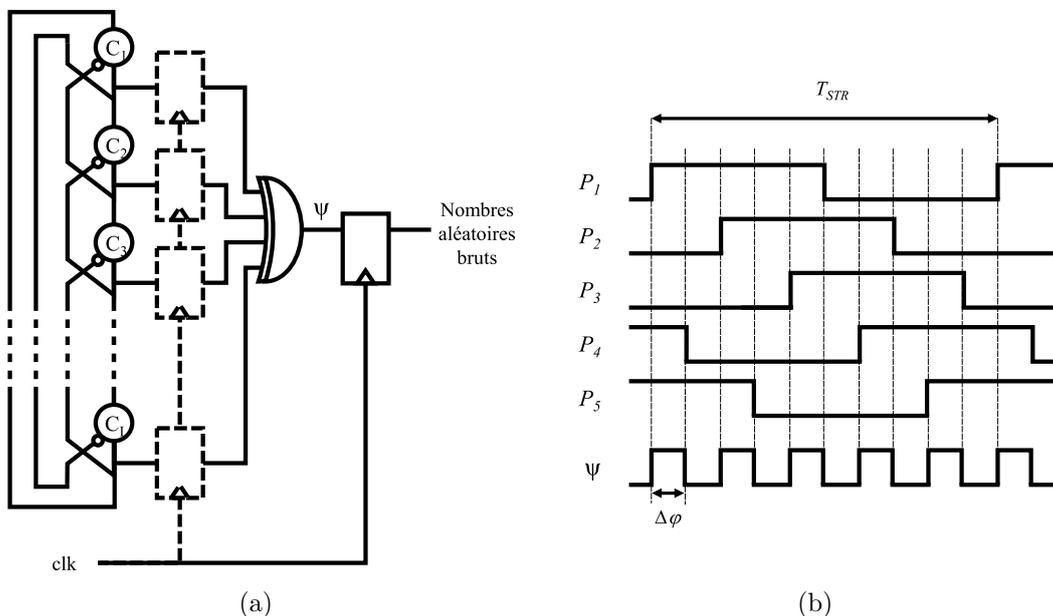


FIGURE 4.15 – Principe de fonctionnement du **STR-TRNG**. (a) Architecture et (b) chronogramme.

À première vue, cette architecture est donc très proche de celle du **MURO-TRNG**. La principale différence vient cependant de la manière dont est construit le signal Ψ . Alors qu'une approche probabiliste est utilisée pour construire ce signal bruité dans l'architec-

ture **MURO-TRNG**, le **STR-TRNG** s'appuie sur une répartition uniforme des phases par rétroaction électrique. Intuitivement le nombre de phases – d'étages du **STR** – à utiliser est donc moins important que le nombre d'oscillateurs de la solution **MURO-TRNG**. Le **STR** devrait donc permettre d'extraire de manière plus efficace l'entropie. Par ailleurs, nous avons vu que la résolution $\Delta\varphi$ d'un **STR** peut être finement contrôlée (4.9). En augmentant le nombre d'étages de l'anneau tout en gardant un taux d'occupation constant, la fréquence de Ψ peut donc être rendue aussi grande que souhaité.

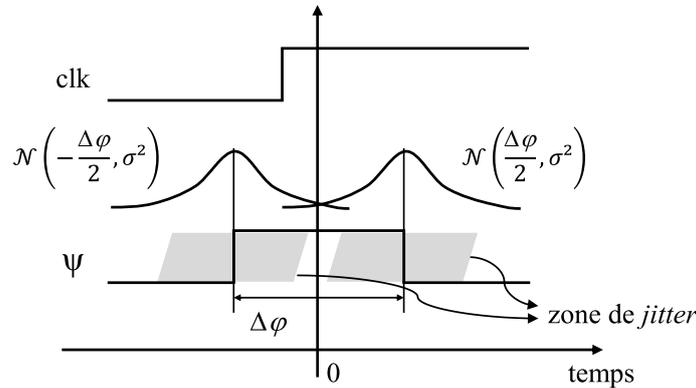


FIGURE 4.16 – Modèle de l'extraction d'entropie du **STR-TRNG** basé sur le *jitter* de chaque phase.

Deux modèles concurrents permettent d'illustrer le fonctionnement du **STR-TRNG**. Le premier, présenté dans [Che+13b; Che14] et illustré par la figure 4.16, considère que l'entropie provient du *jitter* présent dans le **STR**. Si la résolution $\Delta\varphi$ du **STR** est de l'ordre de grandeur de l'écart type σ_{STR} du *jitter* présent sur chacune des phases alors, non seulement le rapport cyclique de Ψ est variable, mais la succession des différents fronts de Ψ n'est plus garanti et certaines impulsions peuvent disparaître²⁴. Le signal Ψ est alors suffisamment bruité pour que son échantillonnage produise un bit qui soit aléatoire. En se basant sur les différentes probabilités d'échantillonnage, CHERKAOUI *et al.* [Che+13a] montrent que le pire cas pour l'entropie par bit apparaît lorsque l'instant d'échantillonnage est le plus éloigné des instants moyens des fronts (c'est-à-dire lorsqu'il se produit en $t = 0$ sur la figure 4.16). Ils proposent alors une limite basse de l'entropie de Shannon par bit H_{min} tel que définie par l'équation (4.13) :

$$H_{min} = -P_0 \log_2(P_0) - (1 - P_0) \log_2(1 - P_0) \quad (4.13)$$

$$\text{avec } P_0 = 1 - 2\Phi\left(\frac{T}{4L\sigma_{STR}}\right) + 2\left(\Phi\left(\frac{T}{4L\sigma_{STR}}\right)\right)^2 \quad (4.14)$$

$$\text{et } \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt, \quad x \in \mathbb{R} \quad (4.15)$$

Une deuxième approche consiste à s'intéresser à l'entropie extraite de l'accumulation de *jitter* qui s'opère entre deux instants d'échantillonnage. Le signal clk peut, par exemple, être issu d'un simple **IRO** de période T_{RO} . Dans le cas où l'hypothèse d'accumulation est valide²⁵, il est possible de modéliser ce générateur de la même manière que le **ERO-TRNG** (voir la figure 4.14 et l'équation (4.12)), en considérant $T_1 = \Delta\varphi = T_{STR}/L$, $K_D T_2 = T_{RO}$

24. Les fronts successifs de Ψ sont liés à des transitions de différents étages de l'anneau.

25. C'est-à-dire que le signal Ψ en sortie du **STR** accumule du bruit de manière similaire à un **IRO** classique. Cela n'est pas trivial du fait des dépendances qui existent entre les différentes phase du **STR**.

et $\sigma_N = \sigma_{STR}$.

$$H_{min} \approx 1 - \frac{4}{\pi^2 \ln(2)} e^{-\frac{\pi^2 \sigma_{STR}^2 T_{ROL}^3}{T_{STR}^3}} \quad (4.16)$$

Quoique différents, ces deux modèles convergent vers la même conclusion : plus le nombre d'étage du **STR** est important, plus l'entropie par bit en sortie de ce générateur est proche de 1.

Concernant les aspects de « non-manipulabilité » du **STR-TRNG**, nous présentons dans le [chapitre 5](#) un modèle de menace, des attaques ainsi qu'une série de contre-mesures visant à sécuriser cette architecture de **TRNG**.

4.3.6 Comparaison de différentes architectures

Pour donner une idée des performances atteignables pour chacune des trois architectures présentées précédemment, nous reprenons ici les résultats obtenus par *PETURA et al.* [[Pet+16](#)] dans leur travail de comparaison de **TRNG** compatible avec une implémentation sur **FPGA**. Le [tableau 4.2](#) donne ainsi les chiffres de débit et d'efficacité énergétique pour ces trois **TRNG** implémentés dans un **FPGA** Spartan 6 de Xilinx. Ce tableau donne aussi une estimation de l'entropie par bit telle que préconisée par **AIS31** ainsi que le débit d'entropie (produit de l'entropie par le débit).

TABLE 4.2 – Comparaison des performances de trois **TRNG**, extrait de [[Pet+16](#)].

Type de TRNG	Consommation [mW]	Débit [Mbits/s]	Efficacité [bits/ μ J]	Entropie par bit	Débit d'entropie [bits d'entropie/ μ J]
ERO-TRNG	2.16	0.0042	1.94	0.999	0.004
MURO-TRNG	54.72	2.57	46.9	0.999	2.567
STR-TRNG	65.9	154	2343.2	0.998	154.121

Comme anticipé, le **ERO-TRNG**, du fait de sa simplicité est aussi le générateur qui consomme le moins. Son débit d'entropie est par contre très faible. En multipliant le nombre de sources, le **MURO-TRNG** permet d'augmenter le débit d'entropie au prix d'une consommation plus élevée. De par son approche radicalement différente axée sur l'échantillonnage de chaque occurrence du *jitter* et non une accumulation de celui-ci, le **STR-TRNG** est le plus efficace (avec un facteur proche de 50 vis-à-vis du **MURO-TRNG** !) mais est aussi celui qui consomme le plus.

4.4 Générateur d'aléa statique : PUF

Une fonction physique non-clonable (*Physical Unclonable Function*) (**PUF**) peut être appréhendée comme une mémoire stockant un secret inhérent au circuit dans lequel il est embarqué. La particularité de cette mémoire vient du fait que son contenu n'est accessible que dynamiquement. Le secret n'est stocké ni électriquement ni de manière visible²⁶, mais il peut être dynamiquement extrait en mesurant les micro variations résultant des phénomènes aléatoires qui perturbent et modifient le processus de fabrication. Ainsi ces dispositifs s'appuient sur le principe suivant : les disparités du processus de fabrication – le secret stocké physiquement – ne peuvent être mesurées autrement qu'avec le **PUF**. Toutes tentatives d'outrepasser ce principe, doit corrompre ce secret. Dans les circuits intégrés, ce

26. Par visible, nous entendons : qui peut être mesuré de manière précise depuis l'extérieur du circuit.

postulat est raisonnable au regard de l'état de l'art de l'analyse de la structure cristalline du silicium. Mais la validité de cette hypothèse doit être régulièrement reconsidérée par la communauté scientifique.

4.4.1 Principes de fonctionnement

Un PUF peut être décomposé en trois parties (figure 4.17) :

- 1) Un ensemble de sources d'entropie, permettant de capturer les variations aléatoires et incontrôlables du processus de fabrication. Lorsqu'elles sont stimulées par un challenge C , une ou plusieurs de ces sources génèrent un signal analogique dont au moins l'une des grandeurs caractéristiques reflète les variations de fabrication.
- 2) Un bloc d'extraction composé d'une partie amplification et d'une partie échantillonnage²⁷. Leurs objectifs sont d'augmenter le rapport signal à bruit (*Signal-to-Noise Ratio*) (SNR), en combinant, accumulant ou filtrant le signal en provenance de la source et de faire la conversion, dans le domaine numérique, de la caractéristique analogique exploitée.
- 3) Un bloc de post-traitement mettant en forme les mesures pour construire la réponse R du PUF tout en augmentant sa stabilité.

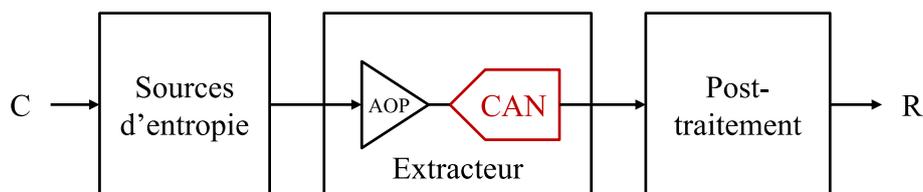


FIGURE 4.17 – Architecture générique d'un PUF.

Cette décomposition est très similaire à celle faite pour les TRNG (figure 4.8). La principale différence se situe au niveau des sources d'entropie. Alors qu'un TRNG va s'intéresser aux variations dues au bruit d'une unique source²⁸, un PUF va comparer un grand nombre de sources pour extraire de l'aléa statique – le bruit qui a été figé dans le circuit par le processus de fabrication. Mais de manière générale, les sources d'entropies utilisées dans les PUF sont les mêmes que celles utilisées par les TRNG. En faisant l'hypothèse réaliste que ces mesures ne sont pas parfaitement stables²⁹, il est intéressant de remarquer qu'un PUF pourra très souvent être utilisé dans un mode de fonctionnement TRNG. Des nombres aléatoires pourront être dérivés des variations entre les réponses fournies lorsqu'un même challenge est répété de nombreuses fois.

Par ailleurs, les PUF sont par essence sensibles aux variations des conditions d'opération PVT. Pourtant, seules les disparités de fabrication (le « P ») doivent être extraites. Les autres (le « V » et le « T ») viennent perturber les mesures et introduire de l'instabilité dans les réponses. Pour s'en prémunir, des mesures différentielles sont généralement faites. Les réponses à un challenge sont alors construites en comparant plusieurs sources d'entropie. Il est cependant difficile de s'affranchir totalement de cette instabilité. Certaines variations de fabrication ont plus d'effet à basse température, d'autres augmentent

27. Dans les différentes architectures de PUF présentées dans la suite de cette section, le bloc d'échantillonnage sera toujours représenté en rouge.

28. Cette unique source peut être une composition de plusieurs autres, comme c'est le cas pour le MURO-TRNG

29. Elles sont au minimum influencées par les bruits non-déterministes (thermique, flicker, etc.).

avec la tension d'alimentation, et il n'est généralement pas possible de les discriminer. Cela peut conduire à des effets de non-linéarité et d'inversion des grandeurs physiques mesurées. Par exemple, la différence entre les fréquences d'oscillation de deux RO pourra être négative dans certaines conditions et positive dans d'autres. Afin de remédier à ce problème, une calibration des PUF est habituellement faite. Seules les sources étant très impactées par les variations de fabrication seront utilisées. Le processus de calibration permettra de les identifier et les paires de sources à comparer³⁰ seront stockées dans une NVM du circuit. D'autres approches utilisant des codes correcteurs d'erreurs (*Error Correction Codes*) (ECC) sont aussi largement utilisées [Bös+08; MTV09; YD10; MVV12; Che+17] car elles permettent de limiter la taille du *helper data* à stocker. Il apparaît cependant que ces solutions peuvent faciliter le travail d'un attaquant [Koe+14; DV14; SFP20].

Suivant les méthodes d'extraction et de post-traitement utilisées, chacune de ces sources d'entropie peut être caractérisée de manière binaire ou non, permettant de générer un ou plusieurs bits de réponse. La première approche permet d'obtenir des réponses globalement plus stables, alors que la deuxième permet de réduire la taille des PUF et d'augmenter leur efficacité (nombre de bits extrait par unité de surface ou d'énergie).

4.4.2 Évaluation

La qualité d'un PUF peut être évaluée de différentes manières [MGS13]. Généralement trois caractéristiques sont particulièrement regardées :

L'unicité – la capacité d'extraire une réponse unique pour chaque circuit.

La stabilité – la capacité de produire des réponses stables malgré le bruit, ou les variations PVT.

L'imprédictibilité – la capacité de générer des réponses aléatoires avec une entropie par bit la plus grande possible.

Chacune de ces caractéristiques peut être mesurée à l'aide d'une métrique dédiée. Si l'on suppose un ensemble de N circuits $(d_i)_{1 \leq i \leq N}$. Pour chaque circuit, une réponse de n bits est extraite L fois. On note alors r_i^l la l -ième réponse du circuit d_i , avec $1 \leq i \leq N$ and $1 \leq l \leq L$.

4.4.2.1 Évaluer l'unicité

Les réponses de deux circuits différents d_i et d_j doivent avoir une très forte probabilité d'être différentes. C'est à dire, $r_i^l \neq r_j^m, \forall (l, m) \in [1; T]$ et $\forall (i, j) \in [1; N]$ avec $i \neq j$. Une métrique usuelle pour mesurer l'unicité U se base sur la distance de Hamming inter-circuit. Elle peut être calculée avec l'équation suivante :

$$U = \frac{1}{N(N-1)L} \sum_{i=1}^N \sum_{k=1, k \neq i}^N \sum_{l=1}^L \frac{HD(r_i^l, \bar{r}_k)}{n} \quad (4.17)$$

avec \bar{r}_k la valeur moyenne des L réponses extraites du PUF k , n le nombre de bits des réponses et HD la distance de Hamming entre deux vecteurs. Dans l'idéal, cette métrique doit être 0.5.

4.4.2.2 Évaluer la stabilité

Pour un challenge donné, répété L fois dans n'importe quelles conditions de tension, de température ou de vieillissement, la réponse d'un circuit d_i doit toujours être la même :

30. Si ce schéma de post-traitement est utilisé.

r_i^l ne doit pas varier dans le temps. Une métrique habituellement utilisée pour évaluer la stabilité S_i d'un PUF i est la distance de Hamming intra-circuit, qui peut être calculée avec l'équation suivante :

$$S_i = \frac{1}{L} \sum_{l=1}^L \frac{HD(r_i^l, \bar{r}_i)}{n} \quad (4.18)$$

Idéalement, sa valeur doit être 0.

4.4.2.3 Évaluer l'imprédictibilité

Les réponses provenant de N circuits doivent être imprévisibles et uniformément réparties. Comme pour tout générateur de nombres aléatoires (*Random Number Generator*) (RNG), cela peut être évalué en utilisant les tests statistiques de la suite SP800-22. Les réponses de tous les PUF sont regroupées en une suite de bits sur laquelle les tests sont appliqués. Le taux de succès de ces tests est alors utilisé comme métrique. Généralement, pour limiter les dépendances liées au bruit, les valeurs moyennes \bar{r}_i sont utilisées.

4.4.2.4 Autres paramètres

La consommation dynamique des PUF n'est pas très étudiée dans la littérature. Ces primitives ne sont pas utilisées fréquemment dans un fonctionnement normal et leur consommation dynamique est donc généralement négligeable vis-à-vis du système. La consommation statique (courant de fuite) peut être plus problématique, surtout pour les circuits passant beaucoup de temps en mode veille et pour ceux implémentés dans une technologie avancée. Ainsi, la surface³¹ occupée par ces circuits, ou le nombre de transistors, est une caractéristique à regarder lors de la comparaison des différentes architectures. Pour relier cette caractéristique au besoin de sécurité, on peut plus particulièrement s'intéresser à l'efficacité des PUF que nous définissons comme le nombre de bits d'entropie pouvant être générés par transistor des sources.

La latence de réponse des PUF est un autre paramètre peu étudié. Il nous semble pourtant être une caractéristique sensible de ces primitives. Une latence importante mène à une plus grande sensibilité aux variations basses fréquences de la température ou au bruit de scintillement, et dégrade la stabilité des réponses. Mais plus important encore, un long temps de réponse peut entraîner des comportements hasardeux de la part des utilisateurs. Pour s'affranchir de la perte de temps et d'énergie associée à la lecture du PUF, ceux-ci pourront être tentés de stocker le secret dans une mémoire plus facilement accessible, mais non-sécurisée. Un tel comportement anéantirait alors les avantages de ces primitives et pourrait compromettre toute la sécurité du système.

4.4.3 Exemple de PUF

Comme remarqué par MCGRATH *et al.* [McG+19], de très nombreux types de PUF pouvant être implémentés dans un circuit intégré ont été imaginés ces vingt dernières années. Récemment, une architecture basée sur la probabilité de création d'un contact sur une couche physique (métallique, VIA ou poly) a été proposée par JEON *et al.* [Jeo+15]. Les règles de dessin du DRM sont dérogées de manière à construire des structures ayant, du fait des variations du processus de fabrication, 50% de chance d'avoir un défaut : la moitié de ces structures sont équivalentes à un circuit ouvert, l'autre à un court-circuit. Ce

31. On peut raisonnablement considérer que les courants de fuites sont proportionnels à la surface.

principe peut être utilisé pour construire des sources ayant une parfaite stabilité³². Outre la connaissance approfondie du processus de fabrication nécessaire au développement de nouvelles règles de dessin, la non-clonabilité de ces solutions est questionnable. Comme nous pouvons le voir sur la [figure 4.18](#), le secret – l'état physique des connexions – est fixé dans le circuit de manière visible. Cette solution n'apporte donc pas de vraie garantie de sécurité supplémentaire par rapport à l'utilisation d'une simple [NVM](#).

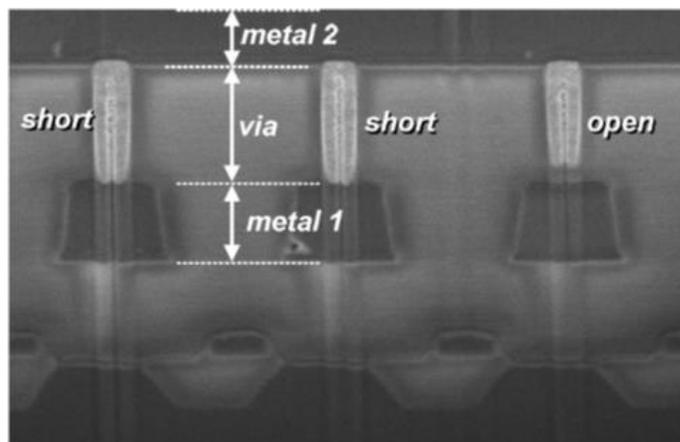


FIGURE 4.18 – Section d'un circuit intégré prise au [MEB](#) révélant le statut des connexions physiques ([VIA](#)). Extraite de [[Jeo+15](#)].

La grande majorité des autres [PUF](#) se basent sur une combinaison de dispositifs passifs (résistances métalliques, capacités parasites, etc.) et actifs (transistors) pour constituer leurs sources d'entropie³³. Bien que ces sources ne soient pas idéales, elles sont potentiellement plus efficaces car elles peuvent être caractérisées de manière non binaire. S'ils partagent tous des sources semblables, ces [PUF](#) se différencient essentiellement par les grandeurs physiques qu'ils utilisent et donc dans la manière dont ils extraient l'entropie. Dans la suite de cette section, nous présentons les trois grandeurs physiques les plus exploitées dans les circuits intégrés ainsi que l'architecture de [PUF](#) généralement associée.

4.4.3.1 Temps de propagation : le A-PUF

Le [PUF](#) à base d'Arbitre ([A-PUF](#)), introduit par [LEE et al.](#) [[Lee+04](#)], recueille de l'entropie en comparant les temps de propagation au travers de deux chemins, théoriquement identiques, formés par la mise en série d'éléments de délais. L'architecture de ce [PUF](#) est présentée en [figure 4.19](#). La conversion numérique est faite par un arbitre : généralement, un simple élément séquentiel (une bascule ou un verrou). L'un des chemins est utilisé comme horloge, l'autre sert de donnée. Du fait des variations de fabrication, l'un des deux chemins sera plus rapide : un 0 est capturé par l'extracteur si l'horloge arrive en premier, un 1 dans le cas contraire. Chaque challenge permet de configurer les deux chemins en intervertissant les éléments de délais entre eux. Plus le nombre de délais est important plus l'espace de challenge-réponse disponible sera grand. De plus, l'augmentation du nombre de délais permet d'accentuer la différence des temps de propagation, rendant l'arbitrage

32. Pour s'en assurer, une étape de vieillissement accéléré peut être utilisée pour forcer les connexions peu fiables à « griller » à la manière d'un fusible

33. L'augmentation des disparités lors de la fabrication est assurément une problématique clé de la construction de [PUF](#) fiables. Mais ce sujet est en dehors du périmètre de ce manuscrit.

plus fiable³⁴.

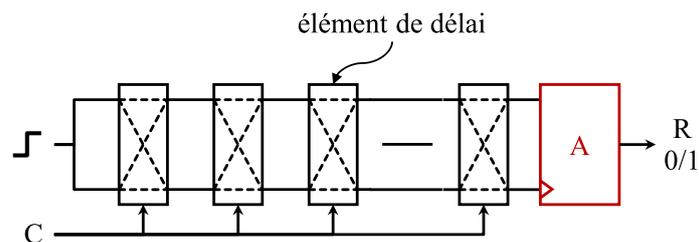


FIGURE 4.19 – Architecture de l'A-PUF.

D'après KATZENBEISSER *et al.* [Kat+12], le A-PUF a une faible entropie de sortie et est sensible aux conditions d'opérations. Ce PUF est aussi exposé aux attaques par *Machine Learning* (ML) du fait de l'existence de dépendances entre ses réponses. Son efficacité est assez faible car chaque challenge ne permet d'extraire qu'un seul bit de réponse. Par contre, sa latence est raisonnable. Elle dépend uniquement de la taille des chemins de propagation.

4.4.3.2 Fréquence : le RO-PUF

La première architecture totalement numérique³⁵, le PUF à base de RO (RO-PUF), avait été introduit quelques années plus tôt par GASSEND *et al.* [Gas+02]. La figure 4.20 illustre son fonctionnement.

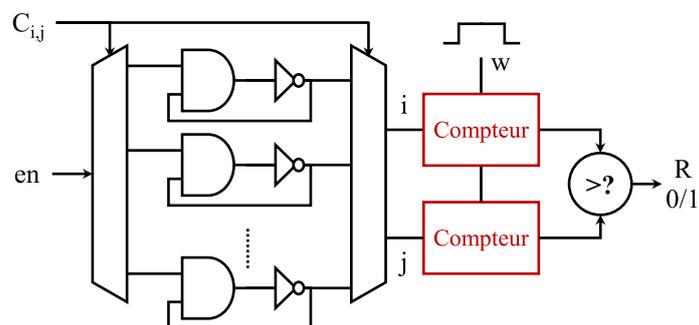


FIGURE 4.20 – Architecture du RO-PUF.

Il consiste aussi à comparer les temps de propagation à travers différents éléments de délai. Ces délais sont cependant arrangés pour former des RO permettant d'amplifier les variations de fabrication à mesurer³⁶. Une image des temps de propagation est extraite en mesurant la fréquence d'oscillation de chaque oscillateur. Un simple bloc comptant le nombre d'oscillations durant une fenêtre de temps prédéfinie (signal d'activation w des compteurs) joue le rôle de fréquencemètre. Ces mesures de fréquence peuvent alors être post-traitées. La comparaison simpliste de deux RO – lequel est le plus rapide ? – permet de générer un bit de réponse. Des post-traitements plus complexes permettent de générer plus de bits par RO [HG17]. Par exemple, ils peuvent être basés sur la valeur réelle de la

34. Les phénomènes de métastabilité lorsque l'horloge et les données arrivent simultanément peuvent fausser la mesure.

35. C'est-à-dire, qui peut être implémentée en utilisant des cellules standards uniquement. En comparaison, le tout premier PUF impliquait la construction de blocs analogiques spécifiques [LDT00].

36. Plus de détails sont donnés au chapitre 6

différence de fréquence de deux RO ou sur la comparaison avec la fréquence moyenne de tous les RO. Dans [Kat+12], les auteurs montrent que les réponses des RO-PUF sont de très bonne qualité (stables et aléatoires). Mais cette solution souffre d'une latence assez importante du fait de la méthode d'échantillonnage utilisée. En effet, pour avoir une bonne résolution, les mesures de fréquence doivent être faites sur de larges fenêtres de temps. La possibilité d'extraire plusieurs bits de réponse à partir de chaque RO permet d'augmenter l'efficacité de cette solution, même si cela vient généralement au détriment de la stabilité.

4.4.3.3 Initialisation d'éléments bistables : le SRAM-PUF

Le PUF à base de SRAM (SRAM-PUF) a été introduit plus tardivement par GUAJARDO *et al.* [Gua+07]. Leur idée est d'extraire un secret à partir des valeurs d'initialisation d'une mémoire vive statique (*Static Random Access Memory*) (SRAM) (figure 4.21). A la mise sous tension, beaucoup des points mémoires se stabilisent toujours à la même valeur, 0 ou 1, en fonction de la force respective des inverseurs les constituant. Chaque point mémoire joue simultanément le rôle de source d'entropie, d'amplificateur et de convertisseur.

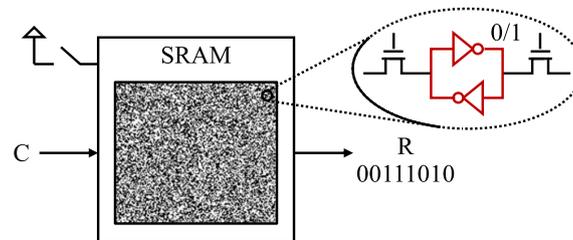


FIGURE 4.21 – Architecture du SRAM-PUF.

Ce type de PUF est très compact et a une latence courte car plusieurs bits sont extraits en un seul accès en lecture dans la SRAM. D'après [Kat+12], les réponses générées sont de bonne qualité. Cette architecture souffre cependant d'un manque de contrôle sur le processus d'extraction. Celui-ci est enclenché dès la mise sous tension de la mémoire. Ainsi, tant que la mémoire n'est pas réinitialisée, le secret est stocké électriquement dans le circuit et peut alors être sujet à des attaques par lecture laser [Ned+13; Hel+13]. Pour remédier à cette faille, des *power switches* peuvent être utilisés pour contrôler le démarrage et l'extinction de la mémoire. Mais dans ce cas, le temps d'allumage doit être pris en compte dans l'évaluation de la latence et de l'efficacité de cette architecture. D'après ROELKE et STAN [RS16], le SRAM-PUF est aussi sensible aux attaques par vieillissement.

Cette architecture est intéressante d'un point de vue efficacité. Même si le rendement (le taux de points mémoire stables) n'est pas toujours élevé, les points mémoires sont composés de seulement 6 transistors et permettent d'extraire un bit d'entropie.

Dans cette même famille de PUF, nous pouvons aussi mentionner le PUF à base de TERO (TERO-PUF) [Bos+14] qui utilise aussi un circuit bistable. Dans cette architecture, l'état stable est atteint après une série d'oscillations transitoires qui peuvent être comptées pour caractériser les variations de fabrication de manière non-binaire.

4.5 Conclusion

L'aléa a toujours posé problème aux concepteurs de circuits intégrés. Il perturbe le processus de fabrication et limite les rendements. Il est aussi source d'imprécision dans les

mesures et limite les performances. Mais cet aléa est aussi indispensable à la construction de circuits sécurisés.

Lorsqu'il est figé dans le silicium au travers des processus de fabrication, l'aléa devient statique. Il permet alors d'identifier de manière unique chaque circuit et permet de stocker, de manière inviolable, des secrets qui peuvent servir de *root of trust*. Une fonction physique non-clonable (*Physical Unclonable Function*) (PUF) est le dispositif permettant d'extraire ces secrets. Nous avons vu qu'il existe différents types de PUF, chacun s'intéressant à différentes grandeurs influencées par les disparités du processus de fabrication. Ces différents procédés d'extraction peuvent être comparés vis-à-vis de propriétés spécifiques aux PUF – stabilité, unicité et imprédictibilité – qui sont évalués avec des métriques bien définies. Les différentes architectures peuvent aussi être comparées en terme de latence de réponse et d'efficacité (nombre de bit de réponse par transistor).

L'aléa dynamique permet, quant à lui, de générer des nombres véritablement aléatoires. Ils sont essentiels à la plupart des algorithmes de chiffrement, aux protocoles d'identification et aux diverses contremesures servant à protéger les circuits. Nous avons vu que seul le bruit thermique est une source fiable de bruit permettant de générer des nombres aléatoires. Les TRNG sont des dispositifs capables d'extraire ce bruit pour le convertir en suite binaire aléatoire. Nous avons vu que leur principale caractéristique est l'imprédictibilité – l'impossibilité pour un attaquant d'anticiper le prochain bit généré même s'il a la connaissance de tous les précédents. Cette propriété peut être évaluée à l'aide de tests statistiques, mais ceux-ci ne sont pas suffisants. Un modèle stochastique est nécessaire pour prouver, *a priori*, le niveau d'entropie des bits de sortie. Nous avons aussi vu que la sécurité de ces dispositifs repose sur une deuxième qualité, la « non-manipulabilité ». Cette propriété s'appuie sur des tests en ligne qui vérifient la qualité statistique des suites de nombres aléatoires, et des détecteurs spécifiques à chaque source d'entropie qui viennent vérifier leur bon fonctionnement. Parmi les nombreuses architectures de TRNG, nous avons vu que le STR-TRNG est particulièrement attractif de par le nombre de bits d'entropie qu'il est capable de générer par unité d'énergie. Ce générateur utilise un anneau auto-séquence pour capturer chaque réalisation du bruit thermique sans avoir besoin d'attendre qu'il s'accumule.

Nous avons aussi présenté le principe de fonctionnement de ces STR, qui sont des RO construits à partir de circuit asynchrones. Ils ont la particularité de pouvoir propager plusieurs événements en leur sein, sans collision. Ils peuvent alors osciller dans un mode stable *evenly-spaced* pour lequel les différents événements se répartissent uniformément autour de l'anneau. La résolution de ces anneaux, le temps séparant chacune des phases, peut être ajustée en choisissant soigneusement le nombre d'étages et le nombre d'événements. Plus le STR sera grand, plus il pourra avoir une résolution fine, et possiblement inférieure au plus petit délai atteignable avec une porte logique. Si les propriétés de ces anneaux ont pu être utilisées pour construire des extracteurs d'aléa dynamique très efficaces, il semble intéressant de s'intéresser à leur utilisation pour extraire de l'aléa statique. Nous aborderons ce sujet dans le chapitre 6. Par ailleurs, les potentielles vulnérabilités liées à l'utilisation de ces STR au sein de TRNG sont une problématique essentielle vis-à-vis de la sécurité et de leur « non-manipulabilité ». Nous aborderons cette question en détail dans le chapitre 5.

Chapitre 5

Optimisation et sécurisation du STR-TRNG

Sommaire

5.1	Introduction	116
5.2	Modèle de menaces	117
5.3	Attaques	118
5.3.1	Injection et suppression de jetons et de bulles	119
5.3.2	Modification du temps de propagation	120
5.4	Contremesures	120
5.4.1	Choix du nombre d'étages	121
5.4.2	Utilisation du mode interne	121
5.4.3	Moniteur de jetons	122
5.4.4	Fréquencemètre	124
5.5	Simulations	125
5.5.1	Simulations mixtes	125
5.5.1.1	Suppression de jetons	126
5.5.1.2	Modification de délai	128
5.5.2	Simulations de haut-niveau	128
5.6	Implémentation ASIC	131
5.6.1	Architecture du circuit	131
5.6.2	Résultats	133
5.6.3	Discussion	135
5.7	Nouveau modèle stochastique basé sur le bruit du signal d'échantillonnage	135
5.7.1	Probabilité et entropie	136
5.7.2	Ordre du modèle	137
5.7.3	Limite basse de l'entropie	138
5.7.4	Comparaison des deux modèles	139
5.8	Conclusion et perspectives	140

5.1 Introduction

Le STR-TRNG est un générateur de nombres aléatoires particulièrement attractif. Avec une efficacité énergétique dépassant de plusieurs ordres de grandeur les solutions concurrentes (section 4.3.6, [Pet+16]), il semble particulièrement bien adapté aux circuits ayant de fortes contraintes de consommation. Par ailleurs, les résultats obtenus sur FPGA et ASIC montrent un très bon comportement statistique et corroborent le modèle stochastique accompagnant ce générateur [Che+13a; Che+13b]. L'imprévisibilité de ce TRNG est ainsi assurée.

Étonnamment, très peu de travaux se sont penchés sur la « non-manipulabilité » du STR-TRNG. CHERKAOUI [Che14] a anticipé des attaques potentielles et suggéré des pistes à explorer pour créer des contremesures adaptées. Mais seuls MARTÍN *et al.* [Mar+15] ont mené de véritables attaques. Ils ont stressé ce générateur de quatre manières différentes et ont analysé l'impact de ces manipulations sur les séquences aléatoires générées. Tout d'abord, ils ont tenté de manipuler l'environnement, en jouant sur la température et la tension d'alimentation. Ces deux premières attaques ont montré un faible impact sur les bits de sortie. En augmentant la température, la fréquence d'oscillation de STR diminue, augmentant du même coup la résolution de phase. Mais le bruit thermique augmente aussi avec la température. Ces deux phénomènes ont donc tendance à se compenser et la qualité des séquences n'est que peu impactée. De la même manière la fréquence de l'anneau diminue en même temps que la tension d'alimentation. Bien que l'impact sur la résolution est plus perceptible que pour l'attaque en température, la perte d'entropie peut facilement être compensée par un surdimensionnement du système ou l'utilisation d'un post-traitement arithmétique [Mar+15].

MARTÍN *et al.* [Mar+15] ont ensuite attaqué le STR-TRNG en injectant des impulsions sur l'alimentation ou l'horloge. Ces attaques affectent les sorties du TRNG et y introduisent un biais observable. En réalité les deux attaques ont un effet très similaire sur le circuit. En réduisant brièvement la tension d'alimentation ou en injectant des impulsions supplémentaires sur l'horloge, les auteurs ont mis à défaut l'extracteur et ont entraîné une violation temporelle à la sortie de l'arbre de XOR¹. Les auteurs proposent une contremesure élémentaire qui protège de ces attaques et montre son efficacité. Elle consiste à utiliser une version *pipelined* de l'arbre de XOR qui modère le chemin critique et réduit la menace d'une violation temporelle. Ils présentent aussi une contremesure légère basée sur un filtre permettant d'empêcher les manipulations de l'horloge².

Dans ce chapitre, nous cherchons à compléter ces premiers travaux en identifiant les différentes vulnérabilités du STR-TRNG (section 5.2). Nous nous intéressons tout particulièrement aux attaques actives³ visant le STR qui est le cœur même de ce générateur et lui confère toute sa spécificité. Dans la lignée des premières idées émises par CHERKAOUI [Che14], nous implémentons deux attaques (section 5.3) et proposons quatre contremesures pour sécuriser le générateur (section 5.4). Les sections 5.5 et 5.6 présentent les simulations et le circuit de test ASIC permettant de valider la pertinence de ces attaques et celle de nos contremesures.

Enfin, nous verrons dans ce chapitre que l'efficacité de ce générateur peut être encore améliorée (section 5.7). En particulier nous montrerons que le modèle stochastique utilisé

1. Soit en ralentissant la propagation dans l'arbre tout en gardant la même contrainte d'horloge, soit en augmentant la fréquence à l'aide d'impulsions sur l'horloge. Les deux pouvant être combinés.

2. En réalité, ces deux attaques ne sont pas spécifiques au STR-TRNG. Par exemple, la même structure de XOR est présente dans le MURO-TRNG. Mais plus généralement, n'importe quel circuit synchrone est sensible à ces attaques. Les deux contremesures sont donc aussi applicables à d'autres circuits.

3. Par opposition aux attaques passives qui cherchent à prédire les séquences de bits sans influencer le fonctionnement du générateur.

est pessimiste et que le **STR-TRNG** recueille plus d'entropie qu'initialement imaginé. Nous proposons alors un nouveau modèle permettant de dimensionner plus justement le **STR** pour réduire sa consommation, à performances constantes.

5.2 Modèle de menaces

L'imprévisibilité du **STR-TRNG** est garantie par la définition d'un modèle stochastique. Mais la sécurisation d'un tel générateur passe aussi par une étude de sa « manipulabilité ». Autrement dit, il est nécessaire d'analyser la capacité d'un agresseur à prendre le contrôle du **TRNG** pour définir ces séquences de sortie ou a minima en altérer la qualité statistique. Cette analyse commence par l'identification des différentes vulnérabilités du générateur et la construction d'un modèle de menace.

En nous basant sur une compréhension approfondie du fonctionnement du **STR-TRNG**, nous avons identifié six vulnérabilités. Deux menaces visent l'extracteur, l'arbre de **XOR**, alors que les quatre autres s'attaquent à la source d'entropie, le **STR**. Les menaces sur la source d'entropie peuvent être sous-divisées en deux catégories : 1) Les menaces qui modifient la fréquence de l'anneau sans compromettre la répartition *evenly-spaced* des événements et entraînent des **performances dégradées**. 2) Les menaces qui compromettent la répartition *evenly-spaced* et entraînent un **fonctionnement incorrect**.

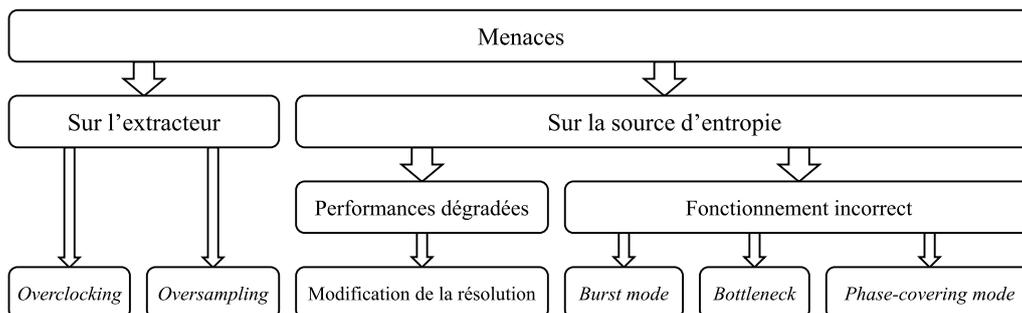


FIGURE 5.1 – Modèle de menaces du **STR-TRNG**.

La figure 5.1 présente ces différentes menaces selon les catégories précitées et leurs effets sur le fonctionnement du **TRNG**.

- **Overclocking** : En réduisant la période de l'horloge, il est possible de générer des violations des contraintes temporelles et d'induire un mauvais fonctionnement du circuit. Le **STR** appartient à la classe **DI** des circuits asynchrones. Il est donc immunisé contre ce type d'attaques. Mais l'extracteur peut être impacté par une telle attaque. Comme montré par MARTÍN *et al.* [Mar+15], elle peut donner le contrôle des bits de sortie à un agresseur. Le même comportement peut être obtenu en augmentant le temps de propagation dans le chemin de données tout en gardant la même fréquence d'horloge. Cette menace inclut ce type d'attaque.
- **Oversampling** : Le sur-échantillonnage consiste à forcer l'extracteur à extraire des valeurs ayant une certaine corrélation. Ce comportement peut être obtenu en modifiant le rapport de fréquence entre le **STR** et l'horloge d'échantillonnage. La dépendance entre les bits réduit l'entropie des séquences aléatoires et compromet la sécurité des applications les utilisant.
- **Modification de la résolution** : MARTÍN *et al.* [Mar+15] ont montré que le **STR-TRNG** est robuste aux variations de l'environnement (température et tension). La

fréquence du STR est plus stable que celle des RO classiques [Che+12]. Un dimensionnement conservateur du générateur permet de supporter de telles manipulations et de garder un très bon niveau d'entropie sur toute la plage d'opération. Cependant il est très probable que des attaques actives telles que les attaque par injection électromagnétique (*ElectroMagnetic Attack*) (EMA) ou les attaque par induction optique de fautes (*Optical Fault Induction Attack*) (OFIA) puissent modifier la fréquence du STR tout en gardant le fonctionnement *evenly-spaced*. D'après les équations (4.9) et (4.13), de telles attaques réduiraient la résolution de phase et entraîneraient une diminution de l'entropie.

- **Burst mode** : Une autre menace consiste à forcer le STR à passer dans le mode d'oscillation dit « en rafales ». Dans ce cas, le principe du TRNG est compromis. La limite basse d'entropie est fortement impactée⁴, et les séquences de bits générées deviennent prédictibles. Modifier le nombre d'évènements se propageant dans le STR est une attaque plausible. Les différents types d'attaques attaque par injection de fautes (*Fault Injection Attack*) (FIA) et plus particulièrement celle injectant des impulsions sur les signaux de *reset* doivent être considérées.
- **Phase covering mode** : Le mode de recouvrement de phase est un autre état critique du STR. Il est observable quand le nombre d'évènements N et le nombre d'étages L ne sont pas premiers entre eux. Il existe alors un diviseur commun X entre N et L . Dans ce cas, d'après l'équation (4.7), les évènements sont rassemblés en groupes de X phases. Il en résulte une résolution multipliée par le même facteur X et une entropie très fortement dégradée.
- **Bottleneck** : Si l'un des étages de l'anneau présente un temps de propagation bien plus important que les autres, il agit comme un goulot d'étranglement pour la propagation des évènements. Le temps de séparation entre les entrées de cet étage est trop important et l'effet *Charlie* (section 4.2.3) devient négligeable. Les évènements ont alors tendance à se regrouper à l'entrée de cet étage, formant une file dans laquelle la propagation du signal d'acquiescement limite le débit. Au contraire, juste après l'étage incriminé, les évènements se propagent librement jusqu'à atteindre le dernier évènement de la file. Lorsque l'anneau opère en mode *bottleneck*, sa période d'oscillation dépend uniquement du délai de l'étage le plus lent. Comme chaque phase présente alors des oscillations régulières, ce mode peut être confondu avec le mode *evenly-spaced*. Cependant, chaque phase ayant son propre rapport cyclique, ce comportement ne garantit plus le fonctionnement du générateur tel que décrit par le modèle stochastique. Le cas où un attaquant est capable d'arrêter le STR peut être considéré comme un cas particulier de *bottleneck* pour lequel le délai d'un étage devient infini.

5.3 Attaques

À partir du modèle de menaces, nous avons imaginé deux attaques visant la source d'entropie. Chacune d'elles active un ou plusieurs des comportements identifiés. Ces attaques ne visent pas à contrôler directement la valeur des bits en sortie du TRNG comme c'est le cas pour l'attaque par manipulation de l'horloge proposée par MARTÍN *et al.* [Mar+15]. Elles cherchent plutôt à réduire l'entropie par bit et à rendre ainsi plus prédictibles les suites de bits générées.

4. Il faut alors prendre en compte le temps séparant deux rafales au lieu de la résolution $\Delta\varphi$ dans le calcul de l'entropie (4.13).

5.3.1 Injection et suppression de jetons et de bulles

La première attaque consiste à modifier le nombre d'évènements se propageant dans le circuit. Lorsqu'un nombre limité d'évènements sont ajoutés ou enlevés, le STR peut conserver son mode de propagation *evenly-spaced*. La fréquence d'oscillation et la résolution de phase peuvent être légèrement dégradées, mais le générateur conservera une entropie suffisante en sortie. En revanche si l'attaque force l'anneau à fonctionner en dehors de la plage de distribution uniforme – en mode *burst* ou *phase covering* – la qualité des bits aléatoires peut chuter drastiquement. Suivant la taille du STR et la force de l'effet *Charlie*, de nombreux évènements doivent être retirés ou ajoutés pour provoquer un mode *burst*. Au contraire, si le ratio N/L est favorable, alors une très petite altération du nombre d'évènements peut être suffisante pour déclencher un mode de recouvrement de phase. Par exemple, en enlevant deux évènements d'un STR de 25 étages contenant 12 évènements, on augmente $\Delta\varphi$ d'un facteur 5 (5 étant le Plus Grand Commun Diviseur (PGCD) entre 25 et 10).

Le modèle d'abstraction jetons/bulles 4 phases est propice à la description de cette attaque. Comme nous l'avons vu dans section 4.2.5, la position à un instant donné des jetons et des bulles dans un STR peut être trouvée en capturant la valeur présente dans chaque étage. Ainsi la séquence '001110100' contenu dans les 9 étages d'un anneau peut être interprétée comme la présence de deux bulles et de deux jetons tel qu'illustré par la figure 5.2a.

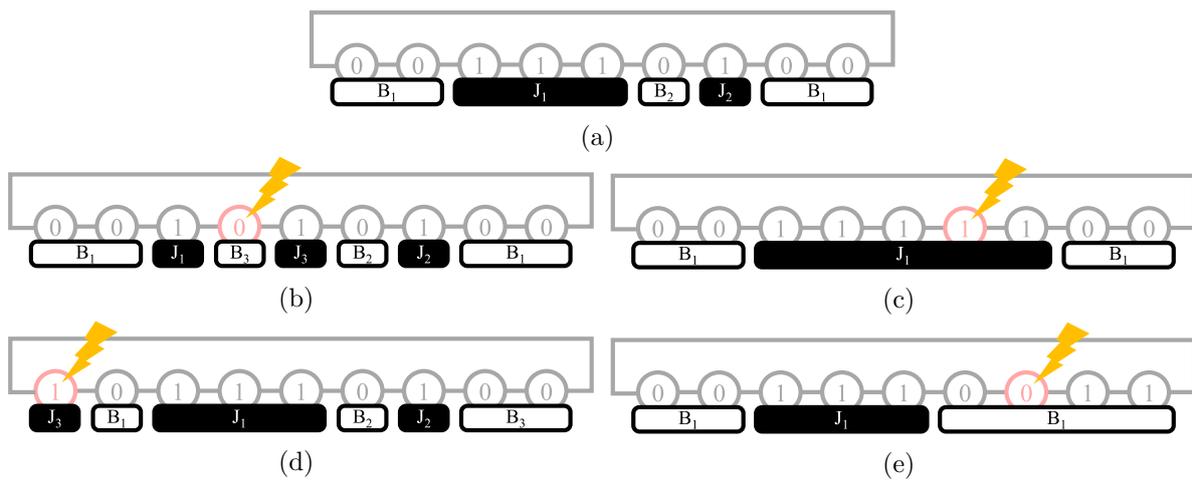


FIGURE 5.2 – Exemple d'ajout ou de suppression de jetons et de bulles dans un STR. (a) état initial de l'anneau, (b) ajout d'une bulle, (c) suppression d'une bulle, (d) ajout d'un jeton et (e) suppression d'un jeton.

Dans le modèle employé, le nombre de jetons et le nombre de bulles sont toujours équivalents (à l'exception des cas où tous les étages contiennent la même valeur). L'ajout d'un jeton dans cet anneau va donc indirectement insérer une bulle et la suppression d'un jeton retire indirectement une bulle⁵. Nous différencions donc les ajouts et suppressions directs, qui concernent l'étage impacté par l'attaque, des modifications indirectes qui viennent rééquilibrer mathématiquement le nombre de jetons et de bulles.

La figure 5.2b illustre l'ajout d'une bulle. En forçant la valeur du quatrième étage de 1 vers 0, on insère une bulle B_3 sur cet étage tout en scindant le jetons J_1 en deux pour faire

5. Et inversement : l'ajout d'une bulle insère indirectement un jeton alors que la suppression d'une bulle retire indirectement un jeton.

apparaître le jeton J_3 . L'opération inverse est illustrée par la [figure 5.2c](#). La modification de l'étage 6 de 0 vers 1 entraîne la suppression de la bulle B_2 et le regroupement des jetons J_1 et J_2 . De la même manière, l'ajout et la suppression d'un jeton sont illustrés par les [figures 5.2d](#) et [5.2e](#). Plus généralement, la suppression d'un jeton est faite en transformant le motif '0[1]*0' en un motif '0[0]*0' alors que la suppression d'une bulle est faite en remplaçant le motif '1[0]*1' en '1[1]*1'⁶. L'ajout d'un jeton ou d'une bulle se fait par les opérations inverses.

Différentes techniques peuvent être utilisées pour effectuer ces attaques. Si le vecteur d'initialisation du STR, qui définit l'état de chaque étage à l'initialisation de l'anneau, est accessible à l'agresseur, celui-ci peut facilement changer sa configuration. L'injection d'impulsions directement sur les signaux d'initialisation peut aussi entraîner la suppression ou l'ajout de jetons. Ces attaques par injection de fautes peuvent donner des résultats reproductibles si l'attaquant est capable de cibler des parties spécifiques du STR ou de forcer l'état de chaque étage individuellement. Un laser peut être utilisé pour injecter de telles fautes dans l'anneau. Cependant, une attaque moins invasive utilisant des impulsions électromagnétiques pourrait être tout aussi efficace et moins coûteuse.

5.3.2 Modification du temps de propagation

La deuxième attaque cherche à modifier le temps de propagation d'un ou plusieurs étages de l'anneau. Elle ne change pas le nombre d'événements circulant dans le STR, mais diminue la fréquence d'oscillation. Si l'attaquant ajoute un petit délai, le STR peut garder sa distribution équidistante des phases. Dans ce cas, la baisse de fréquence entraîne une augmentation de la résolution et, par conséquent, une entropie plus faible. Cependant, cette modification peut ne pas être suffisante pour dégrader réellement la qualité statistique des séquences générées, en particulier lorsqu'un post-traitement est utilisé. *A contrario*, lorsque le temps de propagation d'un étage est considérablement augmenté, un *bottleneck* se crée. Le fonctionnement du générateur n'est plus garanti et la limite basse d'entropie par bit diminue drastiquement. La [figure 5.3](#) illustre cette attaque lorsque le délai du septième étage est impacté : les jetons s'accumulent en amont de cet étage.

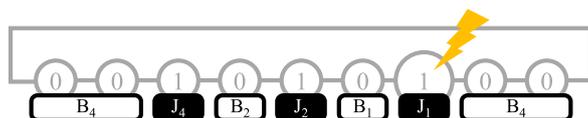


FIGURE 5.3 – Illustration d'une attaque par modification du temps de propagation.

Plusieurs techniques peuvent être exploitées pour insérer un tel délai dans une porte *Complementary Metal–Oxide–Semiconductor* (CMOS). Les manipulations environnementales (tension, température) sont les plus simples, mais elles ont un effet global : elles ont un impact à la fois sur la fréquence et sur le *jitter* et ne compromettent pas la propagation régulière des événements. Par contre, des attaques électromagnétiques ou avec un laser sont susceptibles de créer ces délais [[Deh+12](#)] et doivent donc être considérées.

5.4 Contremesures

Nous proposons quatre contremesures pour détecter ou se prémunir de la plupart des menaces identifiées. Les deux premières sont des règles à respecter lors de la conception. Les

6. La notation $[X]^*$ correspond à un nombre arbitraire de valeurs X successives.

deux autres se basent sur l'ajout de moniteurs permettant de vérifier le bon fonctionnement de la source d'entropie.

5.4.1 Choix du nombre d'étages

Si la première contremesure est élémentaire, elle permet pourtant de se prémunir de toutes les attaques visant à provoquer un mode *phase covering*. Elle consiste à choisir, lors de la conception, un nombre d'étages L du STR tel que, quel que soit le nombre d'évènements N (avec $N \in 2\mathbb{N} \cap [2, L-2]$), la fraction L/N est irréductible. Cette condition est respectée si L est un nombre premier. Dans ce cas, le générateur est protégé les attaques modifiant le nombre de jetons tant que le STR reste dans son mode *evenly-spaced*⁷.

Bien sûr, cette contremesure ne protège pas contre les altérations de la structure de l'anneau. Il n'est pas déraisonnable de considérer qu'en utilisant un sonde ionique focalisée (*Focused Ion Beam*) (FIB) un attaquant puisse modifier physiquement le STR et enlever des étages. Il pourrait ainsi changer le ratio N/L . Mais une telle attaque est complexe et très coûteuse. Elle est aussi contrariée par les détecteurs d'attaque invasive qui peuvent être intégré au niveau du circuit.

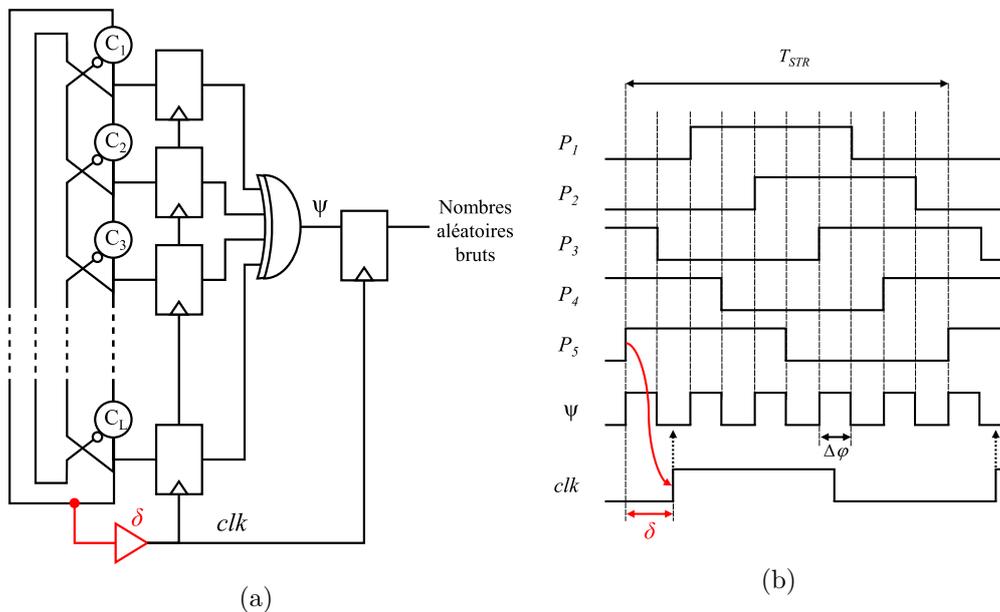


FIGURE 5.4 – Mode interne du STR-TRNG. (a) architecture et (b) chronogramme.

5.4.2 Utilisation du mode interne

Les standards tels que l'AIS31 [KS01] de la BSI ou la SP 800-90B [Tur+18] du NIST recommandent l'utilisation de tests embarqués pour évaluer en temps réel la qualité des suites de nombres aléatoires générés. Mais ces tests ne sont pas très précis et leur dimensionnement peut s'avérer difficile lorsque l'on veut limiter le nombre de faux-positifs et de faux-négatifs. De plus, ces tests sont statistiques et vérifient uniquement que les bits de sortie sont uniformément répartis. Ils ne sont donc pas capables de différencier un pseudo-aléa d'un véritable aléa non-déterministe. En d'autres termes, une défaillance de la source

7. Bien entendu, cela présuppose que le générateur ait été dimensionné vis-à-vis du pire cas de son fonctionnement *evenly-spaced*.

d'entropie et une violation du modèle stochastique pourraient ne pas être détectés si le processus d'extraction du TRNG produit suffisamment de pseudo-aléa pour leurrer les tests en ligne. Dans le cas du STR-TRNG, il est possible que la dérive entre les phases du STR et l'horloge d'échantillonnage soit suffisante pour valider ces tests.

Pour s'affranchir de cette limitation, nous proposons d'utiliser le générateur dans un mode interne : l'une des phases du STR est utilisée comme signal d'échantillonnage tel qu'illustré par la figure 5.4. Un délai δ est ajouté de manière à écarter le front d'échantillonnage de l'événement qui l'a généré et de limiter ainsi les dépendances. L'extraction de l'aléa est donc faite de manière synchrone et aucun pseudo-aléa n'est généré car il n'y a plus de dérive de phase déterministe. Dans ce cas, toute attaque réduisant le rapport entre le bruit et la résolution de l'anneau⁸ va générer un biais grossier en sortie du générateur. Celui-ci va avoir tendance à échantillonner toujours la même valeur⁹. Ce biais est alors facilement détectable avec les tests en ligne : le taux de faux-négatif est largement réduit et la probabilité de détecter une vraie baisse d'entropie est plus forte¹⁰.

5.4.3 Moniteur de jetons

Même si le mode *phase covering* est évité en choisissant un nombre premier pour dimensionner le nombre d'étages, la modification du nombre d'événements circulant dans le STR peut encore diminuer la limite basse d'entropie. Ce sera le cas si la résolution de phase est suffisamment affectée par le changement du nombre d'événements et tout particulièrement si un mode *burst* est provoqué. Cependant, en fonctionnement normal, le nombre d'événements circulant dans un STR ne peut pas fluctuer¹¹. Ce nombre est fixé lors de l'initialisation de l'anneau.

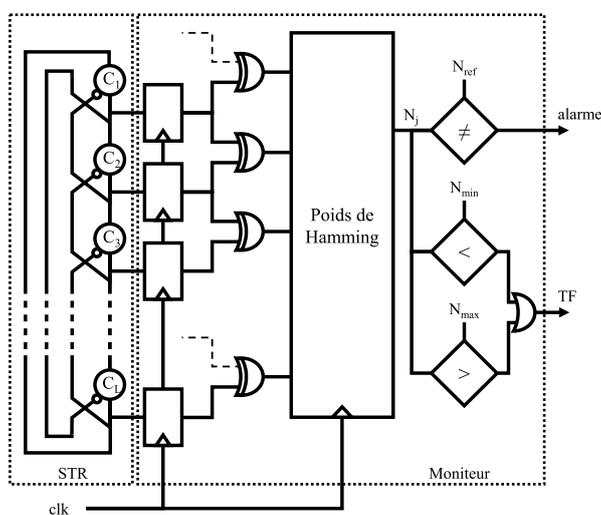


FIGURE 5.5 – Architecture du moniteur de jetons.

Une troisième contremesure peut être construite sur la base de ce principe. Elle consiste à ajouter un bloc vérifiant en temps réel (ou à la demande) que le nombre de jetons dans

8. Par exemple, en diminuant la tension ce qui aura tendance à réduire la fréquence de l'anneau.

9. S'il n'y avait pas de bruit du tout, la sortie du générateur serait toujours à la même valeur.

10. On peut néanmoins noter que, dans le cas de circuits avec des exigences de sécurité moindres, le pseudo-aléa lié à la dérive entre le STR et l'horloge d'échantillonnage pourrait être considéré comme une contremesure intéressante et à faible coût.

11. Il n'y a pas de collision entre les événements grâce au protocole de poignée-de-main implémenté par chaque étage

l'anneau est bien celui escompté. La [figure 5.5](#) présente l'architecture de ce moniteur. D'après le modèle d'abstraction jetons/bulles 2 phases, un évènement est présent dans un étage si sa valeur est différente de celle contenue dans l'étage suivant. Il est donc possible de détecter les jetons en comparant deux à deux, à l'aide d'une simple porte **XOR**, les valeurs de sortie de chaque étage. Si un jeton est présent dans l'étage, le **XOR** associé délivre un 1. Le nombre d'évènements peut alors être extrait en calculant le poids de Hamming du vecteur composé des sorties des L portes **XOR**.

Un tel moniteur émet une alarme dès que le nombre de jetons comptabilisé N_j est différent de la consigne N_{ref} . Cette condition peut être interprétée comme une tentative d'intrusion. Une réaction appropriée¹² peut alors être prise au niveau système. C'est le cas si le nombre de jetons sort de la plage permettant d'obtenir un mode *evenly-spaced* du **STR**. Cette plage peut être caractérisée au préalable et les valeurs N_{min} et N_{max} , les bornes basse et haute de cette plage, peuvent être données en paramètres au moniteur. Or suivant le cas d'utilisation, une réinitialisation peut être excessive. Par exemple, lorsqu'un bloc de post-traitement cryptographique est utilisé, les séquences aléatoires du générateur peuvent encore être utilisées à condition que le temps de défaillance de la source d'entropie n'excède pas la périodicité du bloc de post-traitement. Mais si l'alarme de bas niveau est déclenchée pendant un laps de temps trop important, celle-ci doit être transformée en une alarme de *Total Failure* (**TF**).

Le concept de jetons n'est pas spécifique aux **STR**. Il est en réalité à la base de tout circuit asynchrone. Ceux-ci sont donc susceptibles à des attaques très similaires à celles visant le **STR-TRNG**. En particulier, dans les circuits **BD**, la cohérence entre les jetons circulant dans le circuit de contrôle et les données se propageant dans la logique est primordiale. L'ajout ou la suppression d'un jeton est une attaque qui, même si elle mènera souvent à un blocage du circuit, pourra aussi générer des fautes « utiles » à un agresseur. Ainsi, il semble intéressant de sécuriser n'importe quel circuit asynchrone en vérifiant que le nombre de jetons en circulation est bien celui attendu.

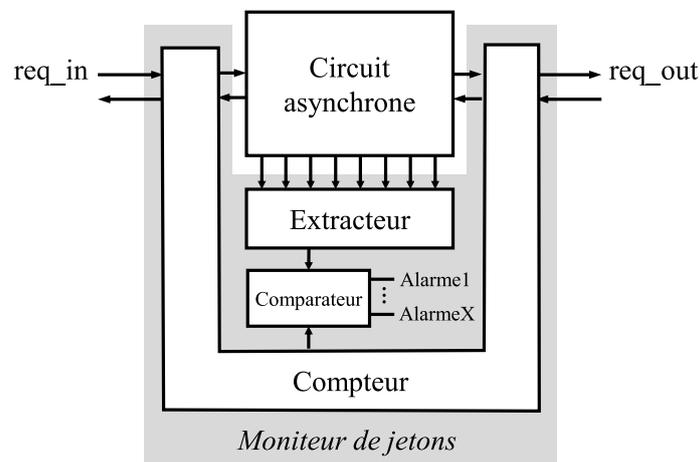


FIGURE 5.6 – Principe de fonctionnement d'un compteur de jetons générique pour circuits asynchrones.

Contrairement aux anneaux auto-séquenceés qui sont des circuits rebouclés et qui ont donc un fonctionnement « iso-jetons », la plupart de ces circuits sont ouverts. Leur nombre de jetons fluctue dans le temps suivant l'arrivée ou non de données. Pour adapter le détecteur à ce cas général, il est nécessaire de lui associer un bloc permettant de comptabiliser le

12. Par exemple, la réinitialisation du générateur.

nombre de jetons théoriquement contenu dans le circuit, ou la partie de circuit, à protéger. La [figure 5.6](#) présente l'architecture générale d'un tel détecteur. Le bloc d'extraction correspond au compteur de jetons proposé plus haut. Il doit être adapté au schéma de circuit asynchrone (4 phases ou 2 phases). Ensuite, le bloc compteur analyse les transactions sur les canaux d'entrée et de sortie. Il permet de calculer le nombre de jetons dans le circuit en faisant la différence entre le nombre de jetons entrant et ceux sortant du circuit (il doit aussi avoir la connaissance du nombre de jetons initialement présent). La comparaison des sorties de ces deux blocs permet de générer des alarmes dans le cas où une différence est constatée. Le principe illustré par cette figure a fait l'objet d'un dépôt de brevet [[Fes+18](#)] couvrant ce cas générique et le cas particulier dédié au [STR-TRNG](#).

5.4.4 Fréquencemètre

Si le compteur de jetons permet de protéger le [STR-TRNG](#) contre une partie des menaces, ce détecteur est incapable de déceler les attaques par insertion de délai. Pour nous prémunir de ce type d'attaque, il faut nous assurer que le [STR](#) oscille et qu'il conserve la fréquence attendue. Pour cela, nous proposons d'utiliser le fréquencemètre décrit en [figure 5.7](#). Celui-ci est basé sur le comptage des oscillations de l'anneau pendant une période W donnée. Le ratio C/W donne la fréquence mesurée par ce bloc. Suivant ces mesures, différentes alarmes peuvent être déclenchées. Une comparaison avec des seuils minimum et maximum permet de délimiter une plage de fonctionnement correcte du générateur. Pour simplifier ce circuit, les comparaisons peuvent être faites directement sur la valeur du compteur C . Dans ce cas, les seuils sont dépendants de la valeur de W .

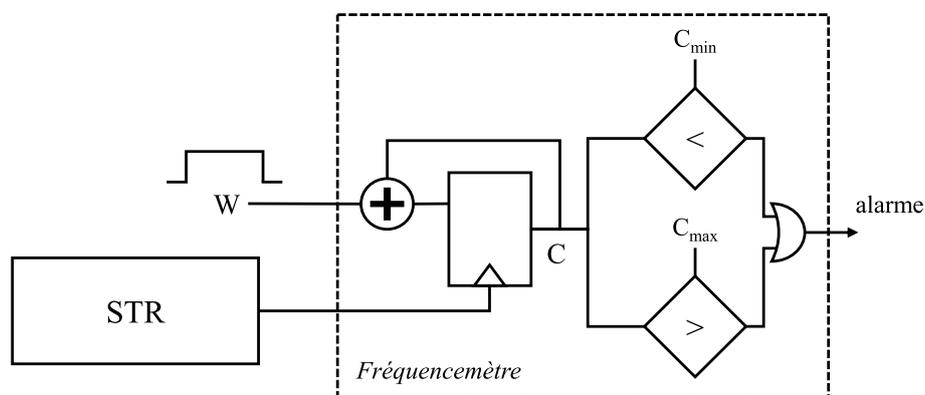


FIGURE 5.7 – Architecture d'un fréquencemètre élémentaire.

Quelques points doivent être pris en compte lors du développement d'un tel détecteur :

- La précision de la mesure de fréquence dépend de la taille de la fenêtre de mesure : l'erreur de mesure $\epsilon = T_{STR}/W$ est d'autant plus petite que la fenêtre de mesure est grande.
- Pour pouvoir détecter une attaque ayant un faible impact sur la fréquence, il faut mesurer la fréquence sur une large période. Cela peut augmenter considérablement la latence de détection et il faut donc veiller à ce que cette fenêtre ne dépasse pas la taille de défaillance acceptable de la source d'entropie (au regard de la périodicité du traitement algorithmique qui serait utilisé).
- Du fait des disparités de fabrication, la fréquence d'oscillation du [STR](#) peut être très différente d'un circuit à un autre. Pour que la détection d'attaque soit efficace, une étape de calibration du système et de définition des seuils est indispensable.

- Les variations des conditions d’opération introduisent aussi des variations dans la fréquence du **STR**. Le dimensionnement du circuit et la définition des seuils devra nécessairement prendre en compte ce phénomène.

5.5 Simulations

De manière à évaluer la pertinence des attaques et des contremesures proposées, nous avons développé un environnement de simulation mixte en technologie **ULP** 55 nm de **UMC** et basé sur l’outil **SMASH** de Dolphin Integration. Cet environnement, présenté en la [figure 5.8](#) peut être utilisé de deux manières. Dans un premier temps un modèle analogique du **STR** est utilisé pour émuler précisément les attaques. Ensuite, le modèle logiciel de simulation de circuits électroniques analogiques (*Simulation Program with Integrated Circuit Emphasis*) (**SPICE**) du **STR** peut être remplacé par un modèle comportemental basé sur la bibliothèque **TAL** développées au sein de mon laboratoire d’accueil. Ces bibliothèques modélisent les effets *Charlie* et *drafting* [Ham+08] et permettent de reproduire, dans des simulations numériques, les différents modes de propagation d’un **STR**. Elles prennent aussi en compte l’influence du bruit thermique en faisant varier, au cours de l’exécution et de manière indépendante, le temps de propagation de chaque porte logique constituant la source d’entropie. L’aléa injecté suit une loi normale dont la valeur moyenne et la déviation standard sont paramétrables. En utilisant ce modèle haut-niveau, le comportement du **STR-TRNG** peut être simulé très rapidement et un grand nombre de bits aléatoires peuvent être extraits pour alimenter les tests statistiques¹³ et évaluer l’influence des différentes attaques.

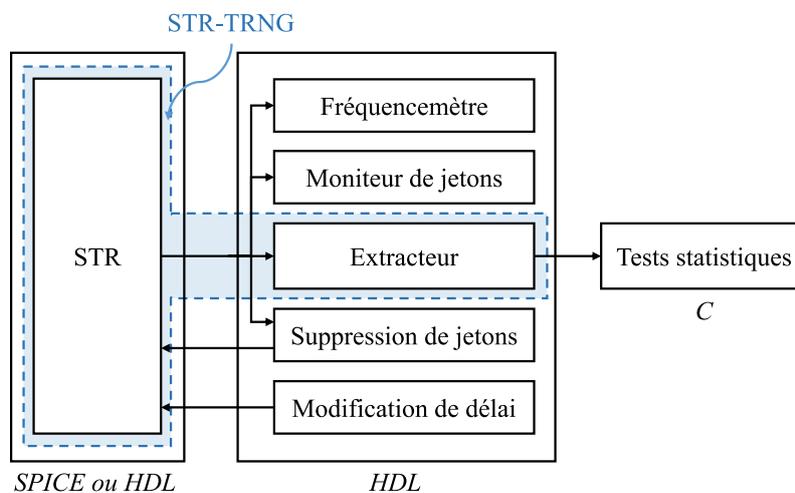


FIGURE 5.8 – Environnement de simulation développé pour évaluer les attaques et les contremesures.

5.5.1 Simulations mixtes

Les attaques sont dans un premier temps simulées en utilisant un modèle **SPICE** de la source d’entropie. L’extracteur, le moniteur de jetons ainsi que les blocs réalisant les différentes attaques sont développés en langage de description de matériel (*Hardware*

13. Dans l’environnement développé, les tests sont effectués séparément à l’aide d’un petit programme élaboré en langage C.

Description Language) (HDL). Ainsi les différentes attaques peuvent être facilement séquencées au niveau RTL et leurs impacts précisément évalués au niveau transistors. Pour les besoins de la démonstration, un STR de 125 étages est construit et initialisé avec 62 jetons.

5.5.1.1 Suppression de jetons

Le principe de fonctionnement de la simulation de suppression de jetons est illustré par la figure 5.9. Elle se base sur l’hypothèse qu’un attaquant est capable à la fois d’observer les différentes phases du STR et de contrôler la réinitialisation de quelques-uns de ses étages. Ainsi, les quatre derniers étages de l’anneau (121 à 124) sont observés. Dès que la séquence ‘1001’ est détectée, une impulsion est générée sur le signal d’initialisation de ces quatre étages pour transformer ce motif en ‘1111’¹⁴. Les éléments de retard δ_1 et δ_2 permettent respectivement de contrôler la position de l’attaque et la largeur de l’impulsion générée.

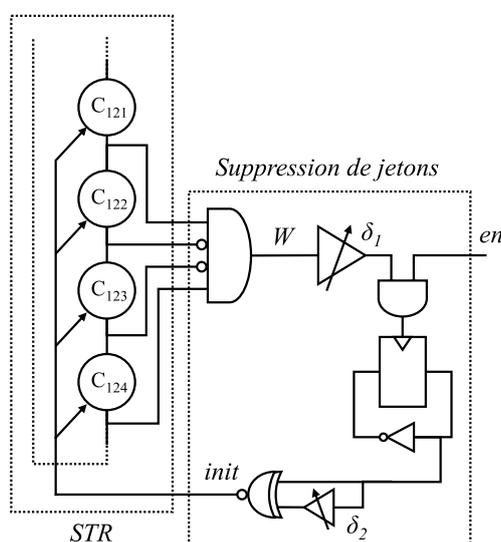


FIGURE 5.9 – Principe de fonctionnement du bloc de suppression de jetons.

La figure 5.10 présente les traces de la simulation de cette attaque. Nous pouvons voir son impact sur les phases du STR. Suite à l’impulsion sur le signal d’initialisation *init*, la sortie du compteur de jetons passe comme attendu de 62 à 60. L’attaque est donc bien fonctionnelle. La fenêtre pendant laquelle elle peut être réalisée – lorsque $W = 1$ (identifiée en rouge sur la figure) – est très courte. Elle s’avère être large de 80 ps dans notre cas. Une fois l’ensemble des parasites Résistance et Capacité (RC) pris en compte, cette fenêtre devrait s’élargir quelque peu car la fréquence du STR sera légèrement plus faible. En réduisant la température ou la tension d’alimentation cette fenêtre doit pouvoir être encore agrandie. Il semble cependant complexe et coûteux pour un agresseur d’observer simultanément plusieurs étages et de synchroniser l’impulsion d’initialisation sur une si courte fenêtre.

Des attaques plus simples, ne nécessitant pas de synchronisation avec les phases de l’anneau, sont possibles. Celles-ci sont par contre moins précises. En effet, suivant le nombre de jetons en circulation dans l’anneau certains motifs sont plus ou moins probables. Lorsque

14. Ce qui correspond, vis-à-vis du modèle d’abstraction 2 phases, au retrait d’une bulle et à une suppression *indirecte* d’un jeton section 5.3.1.

de nombreux évènements circulent, les motifs '0101010'¹⁵ et '0110011'¹⁶ sont majoritaires. Une attaque initialisant le contenu de quatre étages successifs aura dans ce cas tendance à supprimer des jetons, en transformant ces motifs en '0111110'. Mais lorsque le nombre de jetons diminue, les motifs '111111' et '000000' deviennent petit à petit prédominants. La même attaque sera alors sans effet (le motif '111111' reste inchangé) ou aura l'effet contraire et réintroduira des jetons dans l'anneau (le motif '000000' est transformé en '011110'). De manière générale, le nombre d'étages impactés par l'attaque définit la configuration de l'anneau qui sera en moyenne atteinte après un certain nombre d'attaques successives. Plus le nombre d'étages impactés est petit plus on aura tendance à injecter des jetons. A l'inverse plus le nombre d'étages impactés est grand plus on aura tendance à retirer des jetons¹⁷.

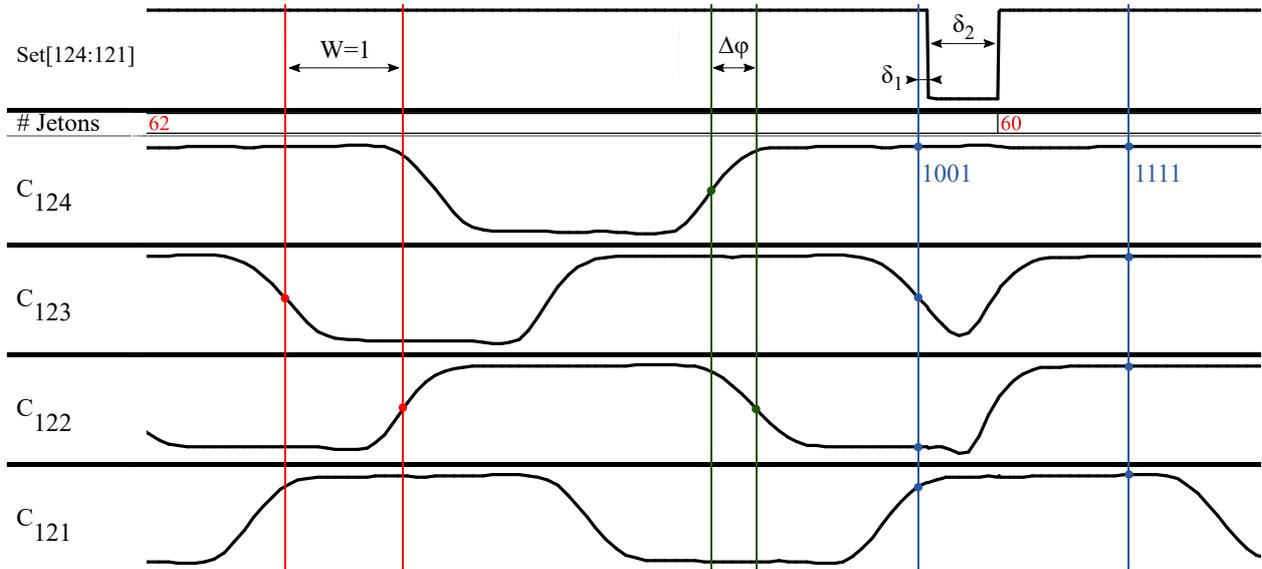


FIGURE 5.10 – Traces d'une attaque par suppression de jetons.

Dans le cas de l'attaque présentée en figure 5.10, une impulsion de 50 ps (δ_2) est injectée. Cette largeur est suffisante pour que l'initialisation soit effective dans les quatre étages visés. Il n'y a cependant pas de limite haute à la taille de cette impulsion. Lors de nos différents essais, nous avons constaté qu'une impulsion plus large permet de réaliser l'attaque tout en bloquant temporairement la propagation des jetons. Le STR reprend son fonctionnement normal (avec un jeton de moins) dès que le signal d'initialisation est relâché.

Ce constat révèle une autre vulnérabilité du STR-TRNG particulièrement critique. Il est possible de bloquer temporairement le STR en forçant la valeur d'un de ses étages, et d'ainsi prendre le contrôle de la sortie du générateur. Pour se prémunir d'une telle manipulation il est donc primordial de vérifier que le STR oscille correctement. Le fréquencemètre répond à cette problématique.

15. Le nombre maximum, $N/2$, de jetons 4 phases est atteint.

16. Le nombre de jetons est proche de $N/4$.

17. Ce constat prend pour postulat qu'une attaque produit le même effet avec chacun des étages (soit forcer la valeur à 1, soit à 0).

5.5.1.2 Modification de délai

Afin de simuler l'attaque par modification du temps de propagation, nous avons inséré une cellule de délai sur le chemin de requête entre le dernier et le premier étage du STR (voir figure 5.11). Ce type de cellule est habituellement utilisé pour corriger des violations de *hold* dans les circuits numériques.

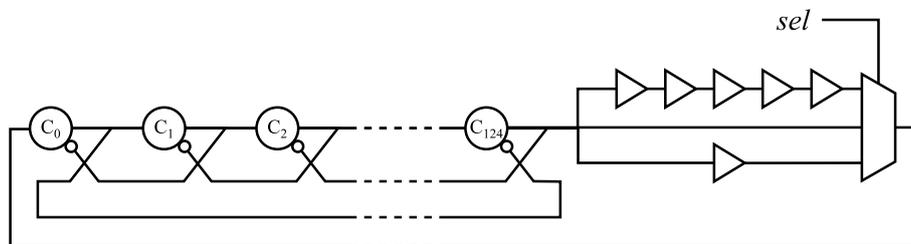


FIGURE 5.11 – Principe de fonctionnement de l'attaque par modification de temps de propagation.

D'autres approches basées sur l'ajout de capacités parasites ou la modification du modèle d'une des portes de Muller peuvent être adoptées. Mais la cellule de délai est la méthode donnant les résultats les plus prédictibles car la caractérisation des temps de propagations est accessible directement dans les modèles *Liberty*. Nous avons implémenté deux attaques. Pour la première, nous utilisons un retard équivalent au temps de propagation d'une porte de Muller. Comme attendu, ce premier cas ne modifie pas le mode d'oscillation du STR : même si la fréquence est légèrement réduite, l'anneau conserve sa distribution uniforme des événements. L'effet *Charlie* de chaque étage contribue à compenser le délai ajouté. Pour la deuxième, un délai approximativement 5 fois plus grand est ajouté. L'effet *Charlie* de chacun des étages n'étant plus suffisant pour compenser le retard introduit dans l'anneau, cette attaque provoque avec succès un fonctionnement en *bottleneck*. Comme nous pouvons le voir sur les traces de cette simulation (figure 5.12), les événements se propagent de deux manières différentes lorsque ce mode de fonctionnement est déclenché. Une partie des jetons font la queue devant le goulot d'étranglement : les étages 66 à 124 présentent des oscillations avec un rapport cyclique proche de 50 %, et une alternance de jetons et de bulles avec une taille minimale (un étage) peut être observée. Dans le reste de l'anneau, entre les étages 0 et 65, les jetons se propagent de manière libre jusqu'à atteindre le dernier élément de la file.

5.5.2 Simulations de haut-niveau

Dans le même environnement de simulation, le modèle *SPICE* du STR peut être remplacé par son modèle comportemental. Nous pouvons alors effectuer des simulations haut-niveau en utilisant un modèle moins complexe qui réduit les temps d'exécution et permet de générer des séquences aléatoires de taille suffisante pour les différents tests statistiques. Ce modèle est écrit en *VHDL* et intègre les effets *Charlie* et *drafting*, ainsi que le *jitter* créé localement par les cellules de Muller. Il décrit donc un comportement temporel permettant de modéliser le fonctionnement analogique de chacun des étages du STR. Nous avons configuré ce modèle de la manière suivante : le temps de propagation moyen d'un étage est fixé à 500 ps avec une déviation standard de 10 ps pour prendre en compte le *jitter*.

Le *testbench* des simulations mixtes est conservé à l'identique et nous avons ainsi effectué les mêmes attaques de suppression de jetons et de modification du temps de propagation. Pour chaque attaque, nous mesurons la période d'oscillation de l'anneau qui

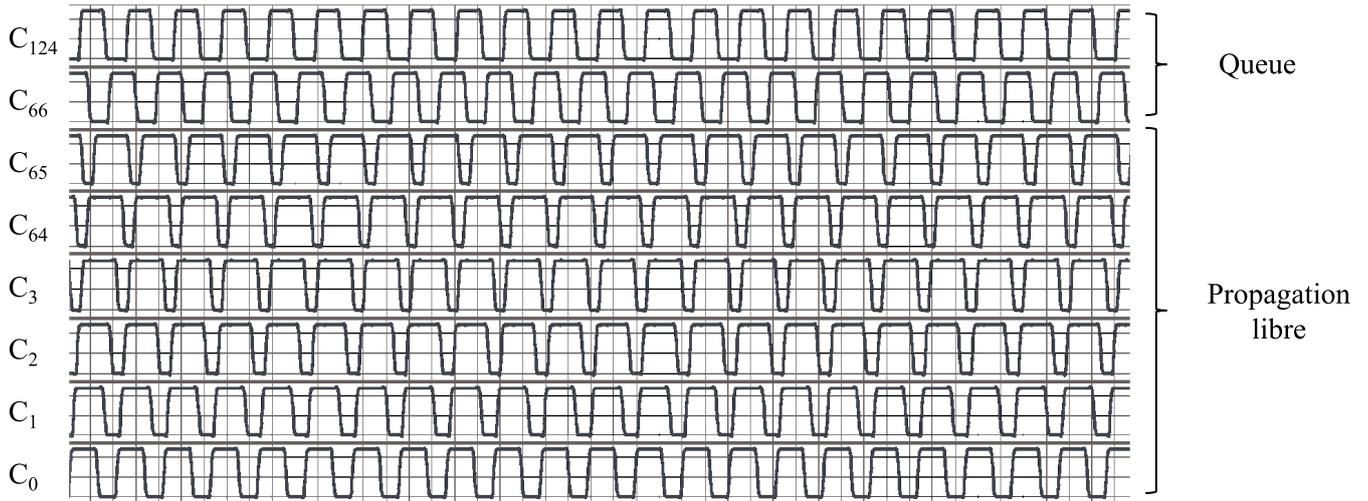


FIGURE 5.12 – Traces d’une attaque par modification de temps de propagation. Un délai 5 fois plus important que le temps de propagation d’une cellule de Muller est ajouté entre les étages 124 et 0.

nous permet de retrouver $\Delta\varphi$ et de calculer la limite basse d’entropie théorique (4.7) et (4.13). Enfin, nous pouvons extraire 10 séquences de 20000 bits que nous évaluons avec les tests initialement décrits par le standard FIPS 140-1 [F94] – *monobit*, *poker*, *runs* et *long runs* – qui peuvent facilement être utilisés comme test en ligne.

Les résultats de ces simulations haut-niveau sont présentés dans le tableau 5.1. Il donne pour chaque attaque : l’effet escompté, la période d’oscillation mesurée T_{STR} , la limite basse d’entropie calculée H_m , le nombre de séquences passant avec succès les tests statistiques ainsi que l’état des deux détecteurs (fréquence et compteur de jetons).

Dans le scénario de référence, lorsqu’aucune attaque n’est effectuée, le STR de 125 étages oscille avec une période de 2,13 ns. La limite basse d’entropie est de 0,991¹⁸. Cela n’empêche pas les 10 séquences de passer avec succès les tests statistiques. Les deux détecteurs ne remontent pas d’alarme.

Nous avons dans un premier temps effectué une attaque ajoutant un délai de 100 ps (le cinquième du temps de propagation d’un étage). Le fonctionnement de l’anneau n’est pas affecté. Au début et à la fin de l’attaque, une très courte période de temps apparaît avec un régime transitoire pendant laquelle les événements se réorganisent uniformément dans l’anneau. Comme attendu, la fréquence d’oscillation et le niveau d’entropie par bit théorique diminuent légèrement (0,986). D’après les tests statistiques, les séquences sont toujours uniformément réparties. Mais cette attaque est détectée par le fréquencemètre qui observe le changement de fréquence et génère une alarme.

En augmentant la taille du retard inséré dans le STR à 1 ns (deux fois le temps de propagation d’un étage), nous parvenons à provoquer un *bottleneck*. Les événements ne sont plus répartis de manière uniforme et la séparation temporelle maximale entre deux phases de l’anneau devient conséquente (> 100 ps) au regard de la résolution dans le scénario de référence ($\approx 8,5$ ps). En conséquence, la limite basse d’entropie est largement impactée – elle est très proche de 0 – et tous les tests statistiques sont en échec. Cette

18. Ce qui est déjà en dessous des 0,997 préconisé par [KS01].

TABLE 5.1 – Résultats des simulations haut-niveau.

	Effet	Attaque	T_{str} (ns)	H_{min}	FIPS 140-1	Compteur de jetons	Fréquence- mètre
Référence	N/A	N/A	2,130	0,991	10/10	ok	ok
Performances dégradées	modification de $\Delta\varphi$	Supp. 10 jetons	2,394	0,987	10/10	alarme	alarme
	modification de $\Delta\varphi$	Ajout de 100 ps	2,414	0,986	10/10	ok	alarme
Fonctionnement incorrect	<i>Phase covering mode</i>	Supp. 2 jetons	2,160	0,196	10/10	alarme	ok
	<i>Phase covering mode</i>	Supp. 12 jetons	2,544	~ 0	0/10	alarme	alarme
	<i>Burst mode</i>	Supp. 52 jetons	12,522	~ 0	2/10	alarme	alarme
	<i>Bottleneck mode</i>	Ajout de 1 ns	5,993	~ 0	0/10	ok	alarme

attaque est aussi révélée par le fréquencemètre qui détecte le changement drastique de la période d'oscillation ($T_{STR} \approx 6$ ns).

Grâce au bloc de suppression de jetons, nous retirons 5 jetons (4 phases) du STR. L'anneau est toujours dans sa plage d'opération *evenly-spaced*, avec 52 évènements répartis dans les 125 étages. Par ailleurs, comme 52 et 125 sont premiers entre eux, il n'y a pas de recouvrement de phases. La période d'oscillation et l'entropie minimum par bit ne sont que légèrement impactés. Les tests statistiques ne sont pas assez précis pour détecter cette modification qui est par contre signalée par le moniteur de jetons et le fréquencemètre.

En retirant 2 ou 12 évènements (1 ou 6 jetons 4 phases) l'anneau conserve sa distribution uniforme mais nous déclenchons le mode de *phase covering*. Le nombre d'étages et le nombre d'évènements ne sont plus premiers entre eux et la résolution de phase du STR se retrouve respectivement multipliée par 5 puis 25. L'entropie minimale théorique en sortie du générateur est gravement réduite. Nous remarquons cependant que la précision du fréquencemètre ne lui permet pas de détecter l'attaque. Les tests statistiques sont par ailleurs tout aussi incapables de déceler cette perte d'entropie. Dans notre cas, comme le générateur fonctionne en mode interne, ce comportement s'explique par un déphasage favorable entre la phase qui échantillonne et celle qui est échantillonnée : l'échantillonnage est toujours fait dans une zone d'incertitude liée au *jitter*. Ce déphasage, qui est lié au délai δ comme décrit dans la figure 5.4, est dépendant de la température et de la tension. Suivant les conditions d'opération du circuit, il est donc possible que l'entropie soit beaucoup plus proche de la valeur minimale calculée théoriquement (0,196) et que les tests deviennent négatifs. Mais dans le cas plus critique où le générateur est utilisé en mode externe, il est aussi possible que la dérive entre le STR et l'horloge d'échantillonnage soit suffisante pour leurrer les tests statistiques. Dans ce cas, seul le compteur de jetons permet de détecter l'attaque.

Nous avons testé un dernier cas, pour lequel seuls 5 jetons (10 évènements) sont conservés dans l'anneau. Celui-ci rentre alors dans un mode *burst* et sa fréquence diminue énormément. La plus grande distance entre deux phases devient très importante et l'entropie minimale est nulle. Comme attendu les deux détecteurs d'attaque lèvent leur alarme alors que de manière anecdotique quelques séquences passent avec succès les tests.

Les différentes simulations haut niveau confirment la dangerosité des attaques que nous avons imaginées. Elles montrent aussi la pertinence des contremesures proposées. De manière intéressante, une redondance existe entre les trois détecteurs et la plupart des

attaques sont révélées par au moins deux d'entre eux. Si le **STR** est construit de manière à empêcher le mode *phase covering* (L est un nombre premier), l'attaque par insertion de délai reste la plus dangereuse : lorsque le délai ajouté est suffisamment petit, elle peut réduire l'entropie mais n'être détectée que par le fréquencemètre. Le dimensionnement de ce dernier est donc particulièrement critique. Il doit s'adapter aux variations de fréquence inhérentes aux différentes conditions d'opération, être assez précis pour détecter une variation de fréquence suspecte, mais aussi pouvoir réagir rapidement pour ne pas laisser passer des attaques très brèves. En pratique, ces contraintes sont très difficiles à concilier, et il peut s'avérer utile de surdimensionner le générateur afin qu'il accepte une plage de fréquence de fonctionnement plus importante. Les mesures de fréquence pourront alors être moins précises mais aussi beaucoup plus rapides. Afin de garantir le plus haut niveau de sécurité, le mode interne doit aussi être privilégié. Dans ce mode, le blocage temporaire de l'anneau stoppe aussi la génération des bits aléatoires. Un agresseur ne peut donc plus utiliser ce biais pour prendre le contrôle du générateur.

5.6 Implémentation ASIC

Pour compléter notre étude sur la « manipulabilité » du **STR-TRNG** nous avons réalisé un circuit de test en technologie **UMC 55nm ULP**. Ce circuit a plusieurs objectifs. En premier lieu, son but est de valider sur silicium l'attaque par suppression de jetons ainsi que le bloc de monitoring associé. Ensuite, nous souhaitons comparer deux types de cellule de Muller. La Muller conventionnelle a été utilisée lors de la construction des premiers **STR-TRNG** pour ASIC ([Che14]), cependant cette porte n'est généralement pas disponible dans les bibliothèques de cellules standard. Nous souhaitons donc construire pour la première fois un générateur utilisant une Muller de type analogique [FM04] qui, contrairement à ce que son nom sous-entend, peut être construite uniquement avec des cellules standards. Cette cellule est d'autant plus intéressante que les **STR** l'utilisant présentent un fonctionnement totalement symétrique avec un rapport cyclique très proche de 50%¹⁹. Enfin, avec ce testchip, nous souhaitons aussi analyser l'influence du placement et du routage automatique du **STR** sur les performances du générateur. Pour obtenir des anneaux performants, il est en effet nécessaire de placer et de router manuellement chacun des étages pour obtenir une structure la plus régulière possible. Nous souhaitons donc statuer sur la capacité des outils à construire automatiquement des **STR** équilibrés.

5.6.1 Architecture du circuit

L'architecture de ce circuit ainsi que son environnement de caractérisation sont présentés en figure 5.13. Le circuit embarque plusieurs générateurs avec différentes configurations de taille du **STR** et de nombre d'évènements²⁰. Pour chacun des générateurs, l'échantillonnage peut être fait en mode externe ou mode interne. Un bloc diviseur est utilisé pour réduire la fréquence d'échantillonnage d'un facteur 32 ou 128. En mode interne, cela rend possible la lecture des nombres aléatoires bruts directement sur un plot de sortie du circuit²¹.

19. Contrairement à ceux basés sur des Muller conventionnelle ou à base de porte majorité, qui héritent nécessairement du déséquilibre des transistors MOS de type P (**PMOS**) et MOS de type N (**NMOS**) et de la présence d'un inverseur supplémentaire sur les chemins d'acquiescement.

20. Ce nombre d'évènements est fixe pour chaque générateur. Cela a permis de simplifier la conception du circuit mais limite aussi les possibilités de caractérisation. Les courbes de fréquence en fonction de l'occupation de l'anneau (figure 4.4b) ne peuvent être extraites.

21. La fréquence moyenne de fonctionnement des **STR** se situe entre 2 et 6 GHz. Mais on considère habituellement que les plots standards ont une bande passante 125 MHz.

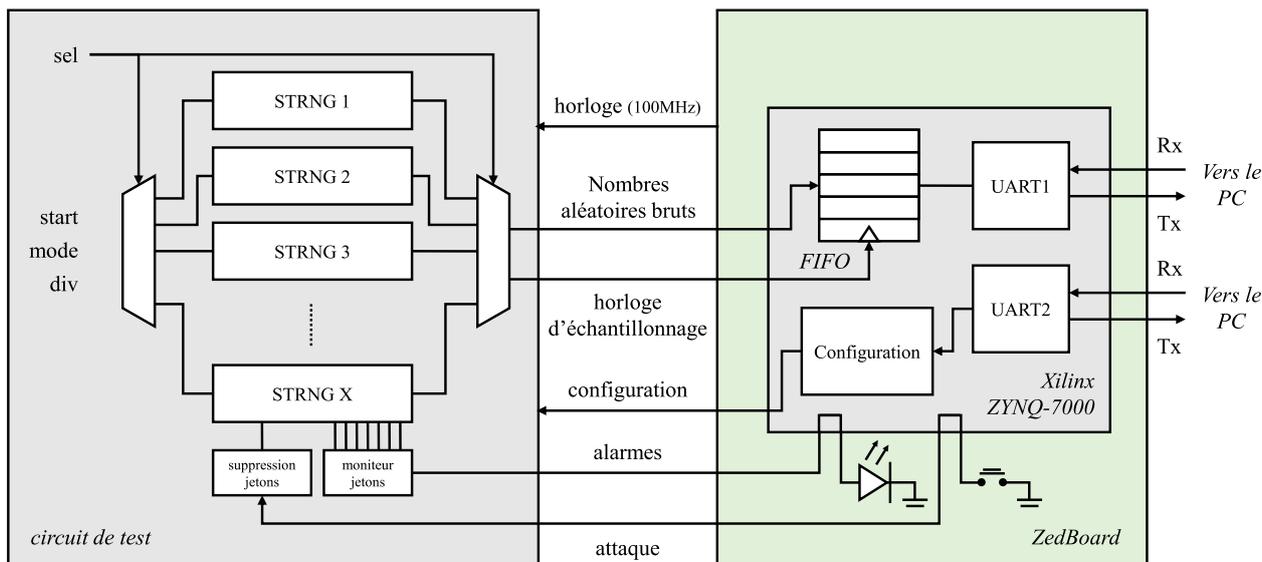


FIGURE 5.13 – Architecture du circuit de test et de l'environnement de caractérisation.

La moitié des **STR** est construite à partir de cellules de Muller conventionnelles développées par Dolphin Intégration. Nous avons été largement impliqués dans le dimensionnement de ces cellules de manière à optimiser l'effet *Charlie* et ainsi maximiser la plage de fonctionnement *evenly-spaced* des anneaux. L'autre moitié des **STR** se base sur une architecture de porte de Muller analogique qui utilise uniquement des cellules standards (voir figure 5.14). Des simulations analogiques nous ont permis de dimensionner au mieux ces anneaux en sélectionnant la taille des portes NON-ET et des inverseurs *tristate*.

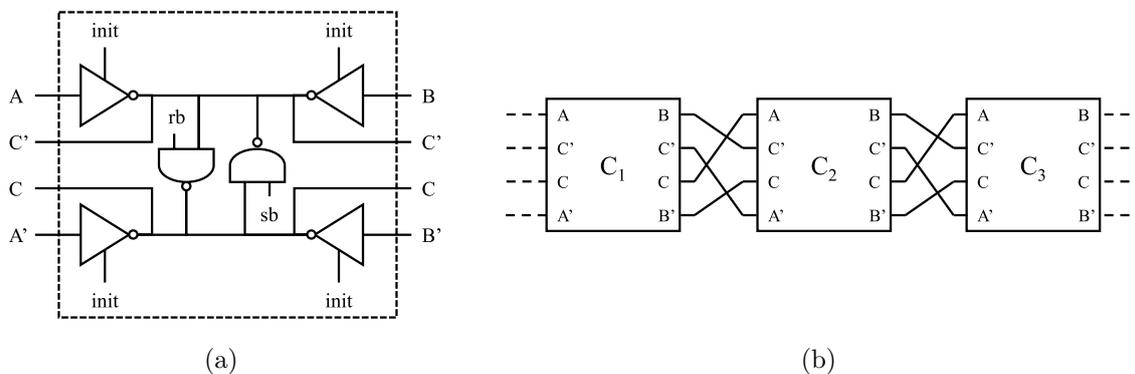


FIGURE 5.14 – Cellule de Muller analogique. (a) Architecture et (b) assemblage pour construire un **STR**.

Enfin, nous avons équipé l'un des générateurs du moniteur de jetons présenté en section 5.4.3 ainsi que d'un bloc permettant d'effectuer l'attaque par suppression de jetons. Contrairement à ce que nous avons pu faire en simulation (figure 5.9), le module d'attaque fonctionne de manière non-synchronisée. Il consiste simplement à générer une impulsion sur le signal d'initialisation de trois étages consécutifs dès que l'utilisateur en fait la demande²².

22. Et donc, sans viser une fenêtre d'attaque particulière.

TABLE 5.2 – Évaluation des séquences aléatoires de chacun des générateurs. L’ordre du filtre de parité nécessaire pour que les tests statistiques AIS31 soient positifs est donné. L’ordre maximal testé est 8, s’il n’est pas suffisant le test est considéré en échec.

Générateur	Étages (L)	Configuration		Mode externe		Mode interne	
		Évènements (N)	Commentaire	@0,8 V 25 MHz	@1,2 V 25 MHz	@0,8 V div128	@1,2 V div32
1	11	6		3	3	échec	échec
2	31	18		2	4	échec	échec
3	31	20		3	3	échec	échec
4	65	38		1	3	2	2
5	65	42		1	3	1	échec
6	135	64		1	7	7	échec
7	135	66		1	3	1	3
8	135	68		1	3	1	3
9	135	70		1	3	1	3
10	135	72		1	3	1	3
11	135	64	Placement automatique	1	3	1	3
12	135	66	Placement automatique	1	3	1	3
13	135	68	Placement automatique	1	3	1	3
14	135	70	Placement automatique	1	3	échec	1
15	135	72	Placement automatique	1	3	1	1
16	135	68	Avec moniteur de jetons	1	3	1	2

Nous avons aussi construit un environnement de test basé sur une carte de développement **FPGA** ZedBoard. Une première interface **UART** permet de contrôler la configuration du *testchip* en définissant les signaux de sélection des générateurs, d’activation, de mode ou de division de l’horloge d’échantillonnage. Le flux de nombres aléatoires bruts est capturé par le **FPGA** et stocké dans un *buffer* de 4 Mbit. Les séquences sont ensuite envoyées à un ordinateur à l’aide d’une deuxième interface **UART** à 230400 bauds. Les différents post-traitements peuvent alors être effectués de manière logicielle.

5.6.2 Résultats

Si les générateurs utilisant les cellules de Muller conventionnelles fonctionnent parfaitement, nous avons constaté une défaillance dans tous les **TRNG** à base de cellules de Muller analogiques. Après analyse, il apparaît que le **STR** de ces générateurs n’oscille pas à cause d’un problème dans la séquence d’initialisation. Ne n’avons pas trouvé de contournement à ce problème et les résultats présentés dans cette section se limiteront donc à l’analyse d’un seul type de générateur.

Nous avons testé chaque **STR-TRNG** fonctionnels, dans leur mode interne et externe et à deux tensions d’alimentation différentes (0,8 et 1,2 V). Pour chacune de ces configurations nous avons extrait 80 Mo de données aléatoires brutes que nous avons ensuite post-traitées à l’aide d’un filtre de parité fenêtré²³ avec un ordre variant de 1 à 8. Nous avons pu analyser chacune des séquences ainsi générées à l’aide de la suite statistique proposée par le standard AIS31 [KS01]. Les résultats de ces tests sont résumés dans le **tableau 5.2**. Ce tableau donne l’ordre du filtre nécessaire pour que les tests soient valides.

Plusieurs enseignements peuvent être tirés de l’analyse de ces résultats. Nous rappelons en préambule que si l’échec à un test statistique est un bon révélateur d’un générateur

23. Filtre effectuant une compression des données à l’aide du **XOR** de n bits consécutifs, n définissant l’ordre du filtre. Pour $n = 1$ le filtre est transparent.

biaisé générant des bits avec une composante déterministe, le succès ne permet pas de conclure sur la qualité du TRNG. Tout test statistique est en effet incapable de distinguer un pseudo-aléa d'un vrai aléa.

Tout d'abord, il apparaît que la diminution de la tension d'alimentation permet de mieux capter le bruit. La plupart des générateurs passent l'ensemble des tests sans post-traitement lorsque la tension est abaissée à 0,8 V alors que ce n'est pas le cas à tension nominale (1,2 V). En effet, le ratio entre le bruit thermique et la résolution du STR est plus favorable lorsque la tension diminue. Ce constat est en phase avec les observations faites par COUSTANS *et al.* [Cou+18]. Par ailleurs, dans les autres cas, un filtre de parité d'ordre raisonnable (≤ 4) est généralement suffisant pour permettre de passer les tests statistiques. Ce filtre a un coût certain en termes de débit et d'efficacité, mais il permet un fonctionnement sans avoir recours à un coûteux post-traitement cryptographique.

Comme attendu en théorie, les configurations avec peu d'étages sont moins performantes. Ceci est visible sur les générateurs avec 11 et 31 étages. En particulier, nous voyons que les séquences générées en mode interne ne parviennent pas à passer les tests statistiques. Comme mentionné précédemment (section 5.4.2), ce mode de fonctionnement permet de supprimer tout aléa déterministe lié à la dérive entre l'horloge d'échantillonnage et le STR. Il permet de déceler plus facilement un problème d'entropie dans le générateur. Il est donc fort probable qu'en mode externe, la présence d'une composante déterministe permette de faire passer les tests malgré le faible niveau d'entropie récolté par le générateur. Nous verrons cependant dans la section 5.7 que l'apport de bruit au travers de l'horloge d'échantillonnage peut aussi expliquer le meilleur comportement statistique de ces générateurs en mode externe.

De manière assez inattendue, le placement automatique des STR ne semble pas vraiment influencer sur la qualité des nombres aléatoires générés. Certaines configurations se comportent un peu mieux (64 et 72 événements), une autre légèrement moins bien (70 événements) sans que nous ayons de véritable explication. Malheureusement, nous n'avons pas la capacité de vérifier si le mode *evenly-spaced* est atteint pour chacun de ces anneaux. Nous ne préférons donc pas tirer de conclusion hâtive à ce sujet. Une étude plus approfondie est nécessaire, en commençant par la simulation analogique des différents anneaux.

Finalement, nous avons pu tester avec succès le module d'attaque et le moniteur de jetons. Du fait de sa non-synchronisation, l'attaque n'est effective qu'après un nombre variable de répétitions. Le moniteur permet de détecter les cas suivants : fonctionnement normal ($N_j = N_{ref}$), fonctionnement dégradé ($Seuil < N_j < N_{ref}$), fonctionnement *burst* ($N_j < Seuil$) et fonctionnement en recouvrement de phase ($pgcd(L, N_j) > 1$).

Après quelques attaques successives nous arrivons à déclencher une alarme de fonctionnement dégradé. En insistant plus longtemps, nous arrivons à générer un alarme de recouvrement de phase²⁴, puis à nouveau de fonctionnement dégradé, pour enfin générer une alarme de mode *burst*. Mais en continuant d'attaquer le générateur nous constatons que les alarmes s'arment et se désarment l'une après l'autre. Sans revenir jusqu'à un fonctionnement normal (toutes les alarmes désactivées), nous pouvons voir le STR osciller entre ses différents modes de fonctionnement altérés.

Cette observation confirme ce que nous pouvions prévoir théoriquement. Une attaque non-synchronisée avec les phases du STR peut retirer des jetons. Mais il faut pouvoir régler le nombre d'étages touchés par l'attaque pour pouvoir contrôler finement l'état moyen qu'atteindra l'anneau au bout d'un certain nombre de répétitions de l'attaque. Par ailleurs, le moniteur de jetons que nous proposons est fonctionnel. Il est un bon moyen pour monitorer de manière continue le nombre de jetons et détecter ce type d'attaque.

24. Dans notre cas, lorsque l'on supprime des jetons, le premier recouvrement de phase apparaît lorsque $N_j = 60$.

5.6.3 Discussion

Ce premier *testchip* nous a donné des confirmations sur la « manipulabilité » du **STR-TRNG** et les moyens de s'en prémunir. Mais il a aussi apporté de nombreuses questions et quelques frustrations. Une estimation du bruit thermique nous aurait permis de consolider nos résultats en les comparant au modèle stochastique. Nous n'avons cependant pas été en mesure de caractériser précisément le *jitter* des phases du **STR**. En effet, nous avons été confronté aux limites d'une mesure externe de ce *jitter* qui est parasitée par de nombreuses autres sources de bruit (sonde, oscilloscope, régulateur de tension, etc.). A l'instar de ce que propose [Val+10; LB15] ou [Yan+17], du matériel de mesure embarqué dans le circuit est indispensable pour cette caractérisation.

L'utilisation d'un nombre fixe d'évènements pour chaque **STR** nous a empêché de valider leur fonctionnement en mode *evenly-spaced*²⁵. Si le mode *burst* est identifiable en observant une seule phase de l'anneau, les modes *bottleneck* et *evenly-spaced* peuvent facilement être confondus. L'observation de toutes les phases de l'anneau aurait permis de faire cette distinction mais cela nous était matériellement impossible. En l'absence de cette validation, il est difficile de conclure définitivement sur les différents résultats obtenus. La comparaison des séquences aléatoires générées par les **TRNG** avec placements automatique et manuel est particulièrement intrigante du fait qu'elle ne montre presque pas de différence au niveau des tests statistiques. La possibilité qu'un mode *bottleneck* ou *burst* permette de capturer suffisamment d'entropie ne doit pas être écartée. Une étude plus poussée du comportement du générateur et une estimation de l'entropie récoltée dans ces modes serait intéressante.

Si le mode interne permet de facilement révéler un manque d'entropie, il ne garantit pas l'imprédictibilité du générateur quelles que soient les conditions d'opération. En effet, dans ce mode, l'entropie extraite dépend de la proximité entre le signal d'échantillonnage et les fronts du signal Ψ ²⁶. Un générateur mal dimensionné (avec $\Delta\varphi \gg \sigma_{STR}$) pourra capturer beaucoup de bruit dans certaines conditions car l'instant d'échantillonnage coïncidera avec un front de Ψ . Dans des conditions de tension ou de température différentes, le signal d'échantillonnage pourra se retrouver éloigné des fronts de Ψ et l'entropie extraite sera moindre. Ce phénomène pourrait expliquer l'exception du **STR** de 135 étages et 70 évènements placé automatiquement. Ce générateur est le seul pour lequel les séquences à basse tension sont moins bonnes qu'à tension nominale. Encore une fois la caractérisation du *jitter* et la vérification du mode de propagation du **STR** sont indispensables pour valider ces hypothèses.

Nous avons conçu un second circuit de test pour palier à ces différentes limitations²⁷, mais au jour de la rédaction de ce manuscrit celui-ci n'est toujours pas sorti de la fonderie. Ce deuxième circuit a aussi pour objectif de valider une optimisation du **STR-TRNG** qui découle du nouveau modèle stochastique que nous présentons dans la section suivante.

5.7 Nouveau modèle stochastique basé sur le bruit du signal d'échantillonnage

Le modèle stochastique proposé par CHERKAOUI [Che14] fait l'hypothèse que chacune des phases du **STR** présente un *jitter* Gaussien σ_{STR} qui ne dépend pas de la taille de l'anneau mais qui est généré localement au niveau de chaque étage. En diminuant la résolution

25. Validation qui est idéalement faite en construisant la courbe de fréquence de l'anneau en fonction du nombre d'évènements.

26. Ce déphasage est fixe car l'échantillonnage est fait de manière synchrone.

27. Plus de détails sur ce circuit sont donné en section 6.6.

de l'anneau tel que $\Delta\varphi \approx \sigma_{STR}$, les zones d'incertitude de chaque phase se recouvrent. Le signal Ψ , construit en combinant chacune des phases, est alors très bruité et son échantillonnage avec une horloge de référence idéale (sans bruit) donnera nécessairement un bit aléatoire. L'avantage de cette approche est que l'entropie ainsi extraite ne dépend pas de la fréquence d'échantillonnage (voir équation (4.13)). CHERKAOUI [Che14] donne quelques contraintes pour éviter les dépendances entre les échantillons successifs, mais de manière générale le débit du STR-TRNG peut être très important si on le compare aux autres architectures de TRNG. Cette singularité rend ce générateur très efficace et son débit supérieur permet de largement compenser sa consommation énergétique importante liée à l'utilisation d'un oscillateur plus complexe.

Cependant, dès qu'une fréquence d'échantillonnage plus basse est utilisée, par exemple du fait que l'application ne nécessite pas un débit important de nombres aléatoires, l'efficacité énergétique baisse considérablement. Pour garantir un bon niveau d'entropie il est en effet nécessaire de conserver le ratio $\sigma/\Delta\varphi$ le plus petit possible. En d'autres termes, la taille requise du STR (qui définit la résolution) ne dépend pas du débit mais uniquement du *jitter* intrinsèque de la technologie. Il n'est donc pas possible de réduire la taille de l'anneau afin de réduire la consommation²⁸ à moins de diminuer l'imprévisibilité.

5.7.1 Probabilité et entropie

Pour nous affranchir de cette limitation, nous proposons d'apporter du bruit dans ce générateur par un autre moyen. Nous considérons ainsi que le *jitter* présent sur le signal d'échantillonnage sera la seule source d'entropie du système²⁹. Dans le nouveau modèle stochastique qui en découle, le STR joue uniquement le rôle d'extracteur et nous ne prenons plus en compte le bruit généré localement par chacune de ses phases. Le signal Ψ est ainsi idéal, de rapport cyclique 50% et de période $2\Delta\varphi$. La figure 5.15 présente le principe de ce modèle.

Le signal idéal Ψ de période $2\Delta\varphi$ est échantillonné à l'aide d'une horloge bruitée clk . À chaque instant d'échantillonnage t correspond un bit b_i en sortie du générateur. Nous prenons comme origine du temps un front montant arbitraire du signal Ψ . Du fait de la périodicité de ce signal, chaque front t d'échantillonnage pourra toujours être ramené, par translation, dans l'intervalle $[0; 2\Delta\varphi[$. Par ailleurs, pour modéliser le bruit présent sur clk , cet instant d'échantillonnage peut être décrit par une variable aléatoire T suivant une loi normale de moyenne μ et d'écart type σ_{ech} .

Afin de calculer l'entropie extraite par ce générateur nous cherchons à estimer la valeur de $P(b_i = 0)$ et $P(b_i = 1)$, les probabilités respectives de capturer un 0 ou un 1 en sortie du générateur. Chacune de ces probabilités correspond à la somme des aires rouges et bleues sous la courbe de distribution de T en figure 5.15. Nous avons ainsi avec $n \in \mathbb{Z}$:

$$P(b_i = 1) = \sum_{n=-\infty}^{+\infty} P(2n\Delta\varphi \leq T \leq (2n+1)\Delta\varphi) \quad (5.1)$$

$$P(b_i = 0) = 1 - P(b_i = 1) \quad (5.2)$$

28. À noter que la consommation dynamique d'un RO dépend du nombre d'évènements circulant en son sein. Pour un IRO cette consommation est fixe car il ne peut contenir qu'un seul évènement. Dans le cas d'un STR la consommation va dépendre directement de sa taille si l'on cherche à minimiser sa résolution en gardant un taux d'occupation constant.

29. Pour rappel, le modèle initial, ne considère qu'un bruit local, associé à chaque étage du STR. En réalité, un bruit global s'accumule certainement dans le STR à l'instar de ce que l'on observe pour les IRO. Dans cette section, nous ne préjugeons pas de la nature locale ou globale du *jitter*. Il peut tout aussi bien être une juxtaposition du bruit des différentes structures ramené sur l'horloge d'échantillonnage seule.

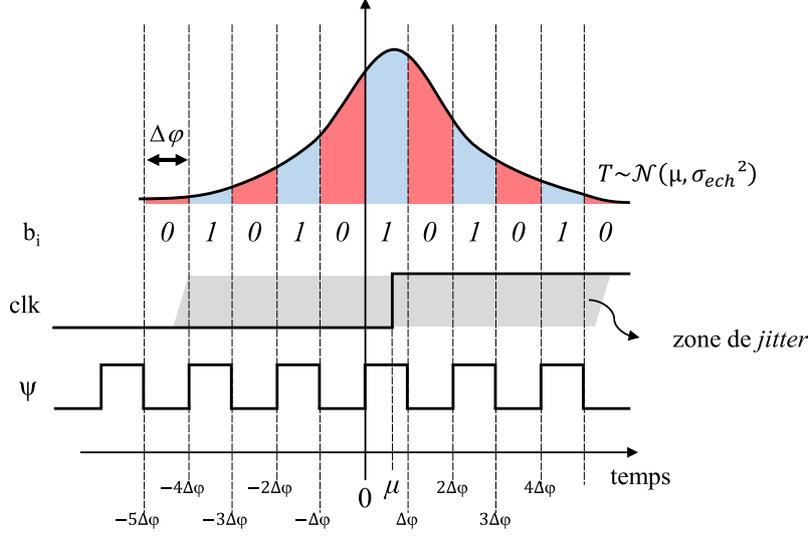


FIGURE 5.15 – Modèle stochastique du STR-TRNG basé sur le *jitter* de l'horloge d'échantillonnage.

En utilisant la fonction de répartition $\Phi(x)$ de loi normale centrée réduite $X = \frac{T-\mu}{\sigma_{ech}}$, la probabilité d'extraire un 1 peuvent alors s'exprimer :

$$P(b_i = 1) = \sum_{n=-\infty}^{+\infty} \Phi\left(\frac{(2n+1)\Delta\varphi - \mu}{\sigma_{ech}}\right) - \Phi\left(\frac{2n\Delta\varphi - \mu}{\sigma_{ech}}\right) \quad (5.3)$$

avec

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt \quad , \quad x \in \mathbb{R} \quad (5.4)$$

L'entropie par bit extraite par le générateur est alors simplement calculée avec l'équation :

$$H(b_i) = -P(b_i = 1)\log_2(P(b_i = 1)) - (1 - P(b_i = 1))\log_2(1 - P(b_i = 1)) \quad (5.5)$$

5.7.2 Ordre du modèle

Pour simplifier le calcul numérique de ces probabilités nous pouvons limiter l'ordre de ce modèle. Nous définissons l'ordre $\omega \in \mathbb{R}^+$ tel que $n \in [-\omega + 1; \omega - 1]$, et nous pouvons alors dériver de l'équation (5.3) la probabilité à l'ordre ω d'extraire un 1 $P(b_i = 1)_\omega$ et l'entropie à l'ordre ω , $H(b_i)_\omega$, qui en découle :

$$P(b_i = 1)_\omega = \sum_{n=-\omega+1}^{\omega-1} \Phi\left(\frac{(2n+1)\Delta\varphi - \mu}{\sigma_{ech}}\right) - \Phi\left(\frac{2n\Delta\varphi - \mu}{\sigma_{ech}}\right) \quad (5.6)$$

$$H(b_i)_\omega = -P(b_i = 1)_\omega \log_2(P(b_i = 1)_\omega) - (1 - P(b_i = 1)_\omega) \log_2(1 - P(b_i = 1)_\omega) \quad (5.7)$$

En considérant un intervalle de confiance de $\pm 3\sigma_{ech}$, nous pouvons déduire l'inégalité suivante garantissant la précision du modèle ³⁰ :

$$(2(\omega - 1) + 1)\Delta\varphi - (2(-\omega + 1))\Delta\varphi \geq 6\sigma_{ech} \quad (5.8)$$

30. L'intervalle de temps intégré sous la courbe de distribution – la différence entre la borne max et la borne min utilisé dans la somme – doit être plus grand que $6\sigma_{ech}$.

Cette inégalité peut finalement être réduite en une contrainte au niveau de l'ordre ω :

$$\omega \geq \left\lfloor \frac{3}{2} \frac{\sigma_{ech}}{\Delta\varphi} + \frac{3}{4} \right\rfloor \quad (5.9)$$

Dans le modèle initial, le dimensionnement du STR-TRNG est fait de façon à ce la résolution de l'anneau soit proche de bruit thermique que l'on cherche à extraire ($\sigma_{STR} \approx \Delta\varphi$). Pour notre nouveau modèle, si l'on considère $\sigma_{STR} = \sigma_{ech} \approx \Delta\varphi$ ³¹, l'ordre 3 est suffisant pour modéliser le générateur. Cela sera toujours valable tant que la condition $\Delta\varphi \geq \sigma_{ech}$ est respectée.

5.7.3 Limite basse de l'entropie

Les figures 5.16a et 5.16b présentent respectivement la probabilité d'extraction d'un 1 $P(b_i = 1)_{\omega=3}$ et l'entropie par bit $H(b_i)_{\omega=3}$ en sortie du générateur en fonction du déphasage μ et de différentes valeurs du ratio $\sigma_{ech}/\Delta\varphi$. Nous pouvons constater que ces courbes présentent un maximum en $\Delta\varphi/2$: plus l'instant d'échantillonnage s'éloigne des fronts du signal Ψ , plus la probabilité d'extraire un 1 augmente et l'entropie s'approche de 0. Le minimum d'entropie étant atteint lorsque l'échantillonnage est fait à équidistance de deux fronts successifs de Ψ . A l'inverse, lorsque les fronts coïncident, l'entropie est maximale.

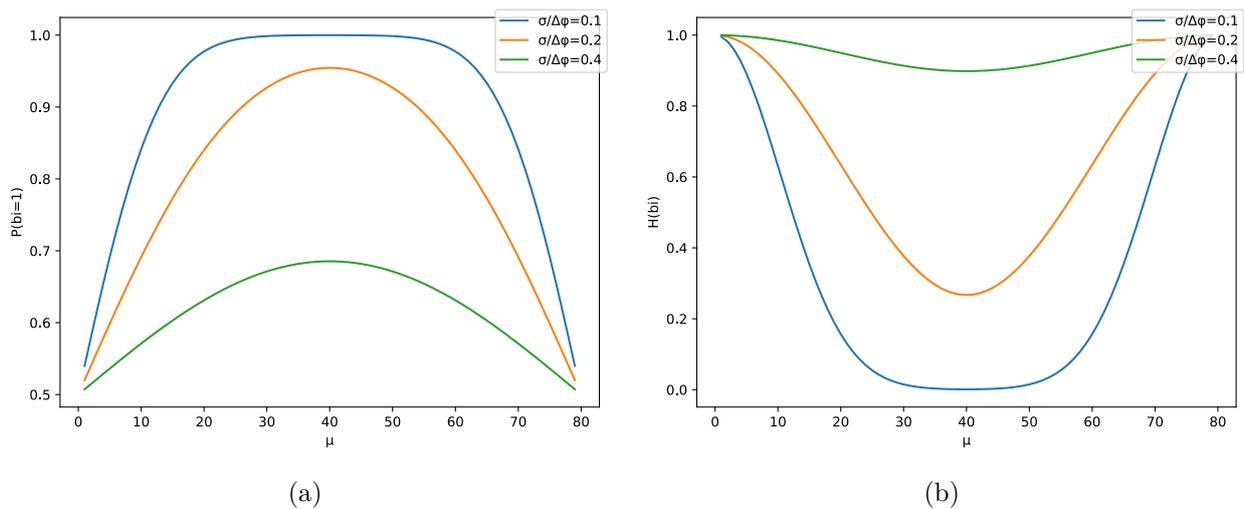


FIGURE 5.16 – Probabilité d'extraire un 1 et entropie en sortie du générateur en fonction du déphasage μ et pour différents ratio $\frac{\sigma_{ech}}{\Delta\varphi}$ (avec $\Delta\varphi = 80$ et $\omega = 3$).

Ainsi, $P(1)$, la probabilité maximale d'échantillonner un 1 lorsque $\mu = \frac{\Delta\varphi}{2}$, peut se calculer avec l'équation :

$$P(1)_{\omega} = \max_{\mu} (P(b_i = 1)_{\omega}) = P\left(b_i = 1 \mid \mu = \frac{\Delta\varphi}{2}\right)_{\omega} \quad (5.10)$$

$$P(1)_{\omega} = \sum_{n=-\omega+1}^{\omega-1} \Phi\left((4n+1)\frac{\Delta\varphi}{2\sigma_{ech}}\right) - \Phi\left((4n-1)\frac{\Delta\varphi}{2\sigma_{ech}}\right) \quad (5.11)$$

31. Ce qui est vraisemblablement un pire cas pour notre nouveau modèle, puisque le bruit d'échantillonnage peut provenir d'une accumulation sur un temps plus important que le temps de traversée d'un unique étage du STR.

Cette probabilité permet de dériver un seuil bas d'entropie H_m^ω , qui peut être exprimé en fonction de l'ordre du modèle ω , du bruit d'échantillonnage σ_{ech} , du nombre d'étages L et de la fréquence de l'anneau T_{STR} :

$$H_m^\omega = -P(1)_\omega \log_2(P(1)_\omega) - (1 - P(1)_\omega) \log_2(1 - P(1)_\omega) \quad (5.12)$$

avec

$$P(1)_\omega = \sum_{n=-\omega+1}^{\omega-1} \Phi \left((4n+1) \frac{T_{STR}}{4L\sigma_{ech}} \right) - \Phi \left((4n-1) \frac{T_{STR}}{4L\sigma_{ech}} \right) \quad (5.13)$$

5.7.4 Comparaison des deux modèles

La figure 5.17a montre l'évolution du seuil d'entropie en fonction du nombre d'étages du STR pour le modèle initial, basé sur le bruit de chacune des phases du STR, et le modèle proposé, basé sur le bruit du signal d'échantillonnage. Si le comportement des deux modèles est similaire lorsque le nombre d'étages est faible (et donc $\Delta\varphi$ est grand), nous pouvons voir que notre modèle converge plus rapidement vers 1. Ainsi, comme nous pouvons le voir sur le zoom présenté par la figure 5.17b, le seuil recommandé par le standard AIS31 est atteint pour seulement 97 étages, alors que le modèle initial prédit un anneau deux fois plus grand (193 étages).

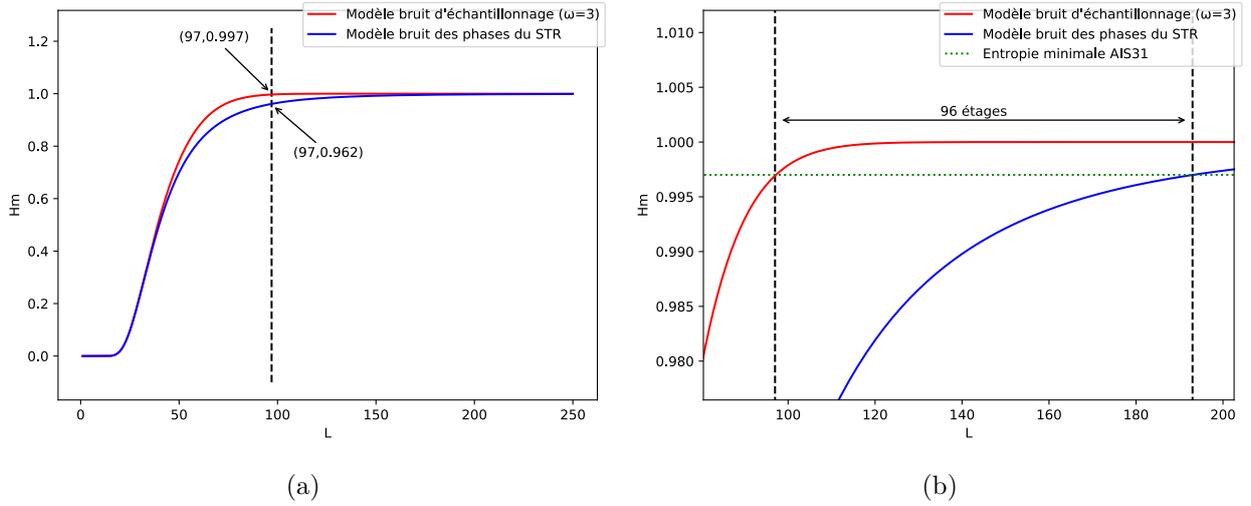


FIGURE 5.17 – Comparaison des modèles basés sur le bruit des phases et le bruit du signal d'échantillonnage. L'entropie minimum en sortie du STR-TRNG en fonction du nombre d'étages est présentée avec $T_{STR} = 1$ ns, $\sigma_{ech} = \sigma_{STR} = 4$ ps et $\omega = 3$.

Le gain associé à cette nouvelle modélisation n'est pas négligeable. En effet, le nombre de registres (L), la profondeur de l'arbre de XOR ($\log_2(L)$) et le nombre de XOR ($L - 1$) dépendent tous du nombre d'étages. Cela se répercute indubitablement sur la consommation du générateur qui a ainsi une efficacité en bits d'entropie par unité d'énergie encore plus favorable.

Les figures 5.17a et 5.17b illustrent un cas particulier où le *jitter* du STR et celui du signal d'échantillonnage sont identiques. Cela correspond par exemple au mode interne du générateur pour lequel l'une des phases est utilisée pour faire l'échantillonnage. Nous pouvons cependant dériver d'autres architectures exploitant au mieux notre nouveau modèle. En effet, outre un gain brut sur la consommation du générateur, ce modèle permet aussi

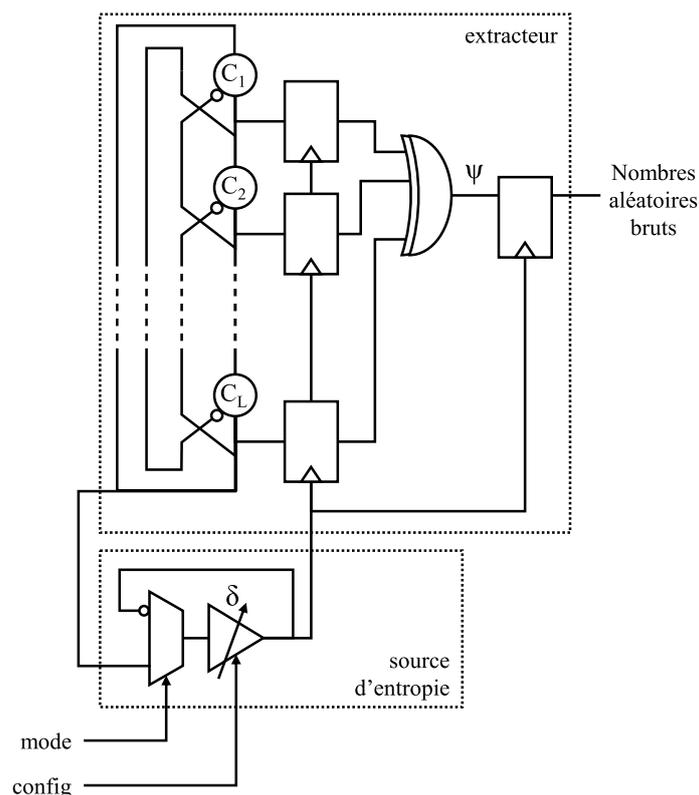


FIGURE 5.18 – Évolution de l'architecture du STR-TRNG pour exploiter le bruit d'échantillonnage.

de décorrélérer le dimensionnement de l'extracteur de celui de la source de bruit. Ainsi il est possible de modifier indépendamment σ_{ech} et $\Delta\varphi$.

Tout en conservant un ratio $\sigma_{ech}/\Delta\varphi$ constant, nous pouvons, par exemple, imaginer un générateur utilisant un oscillateur très bruité pour l'échantillonnage et un STR beaucoup plus petit pour l'extraction. La problématique est alors de trouver le meilleur rapport entre la taille de cette source d'entropie et celle du STR vis-à-vis de leur consommation respective. Une autre approche consisterait à conserver le mode interne du TRNG, pour profiter de ses avantages au niveau de la « non-manipulabilité » (sections 5.3 et 5.4), tout en injectant du bruit supplémentaire sur le signal d'échantillonnage à l'aide d'une ligne à retard dont chacun des éléments de délai contribue à ajouter du bruit. La figure 5.18 réunit ces deux propositions dans une architecture optimisée du STR-TRNG.

Finalement, notre proposition de modèle pointe de nouvelles possibilités d'optimisation du STR-TRNG. Il appelle aussi à une étude plus approfondie. En particulier, la comparaison de différentes structures sensibles au bruit thermique, pour identifier la plus à même d'apporter efficacement du bruit dans le système, est une perspective très intéressante dans le but d'améliorer ce générateur.

5.8 Conclusion et perspectives

Comme toute primitive de sécurité le STR-TRNG est susceptible d'être la cible d'attaques. Un agresseur pouvant réduire la qualité des nombres aléatoires générés, aura une entrée dans le système sécurisé pour élaborer des attaques plus sophistiquées. S'il arrive à prendre le contrôle du flux de sortie du générateur, il pourra très facilement déjouer les protocoles de communication ou les contremesures s'appuyant sur des nombres aléatoires

de qualité. Nous avons montré dans ce chapitre que de telles attaques sont possibles et qu'il est indispensable de prendre en compte la problématique de « manipulabilité » des **TRNG** dès leur conception.

En proposant un modèle de menace, nous avons identifié les principales vulnérabilités du **STR-TRNG**. Nous avons alors pu construire deux attaques spécifiques, visant le cœur de ce générateur. Ces attaques ont pu être simulées au niveau transistors pour valider leur effet sur le **STR** et aussi au niveau comportemental pour voir l'impact qu'elles peuvent avoir sur les suites de nombres générés. L'attaque par suppression de jetons a aussi pu être testée sur silicium, et nous avons montré qu'elle pouvait être efficace même si elle n'est effectuée que de manière grossière, sans être précisément synchronisée avec les phases de l'anneau.

Notre modèle de menace nous a aussi permis de construire des contremesures dédiées permettant de combler la plupart des vulnérabilités identifiées – soit en détectant, soit en empêchant les effets qu'elles exploitent. Nous avons montré l'efficacité de ces différentes contremesures à l'aide de simulations analogiques et haut-niveau. Ainsi, la combinaison de règles de conception, de tests en ligne, du mode interne et de détecteurs permet de sécuriser ce générateur. L'un des détecteurs, le moniteur de jetons, a été embarqué dans le circuit de test et s'est montré totalement fonctionnel. Ce bloc s'assure en temps réel que le nombre de jetons circulant dans le **STR** n'est pas modifié. Nous avons aussi généralisé l'utilisation de ce détecteur. Il peut être utile à tout circuit asynchrone que l'on souhaiterait protéger contre des attaques similaires. La généralisation du moniteur de jetons aux circuits asynchrones a également donné lieu à un dépôt de brevet.

Finalement, nous avons montré au travers d'un nouveau modèle stochastique que les performances du **STR-TRNG** sont sûrement sous-estimées. En comptabilisant le bruit provenant de l'horloge d'échantillonnage, le générateur peut être dimensionné plus justement, et son efficacité énergétique peut être encore améliorée. Nous avons donc proposé une modification de l'architecture de ce **TRNG** permettant d'exploiter ce nouveau modèle, en décorrélant la partie extraction – le réglage de la résolution de l'anneau auto-séquence – de la source de bruit qui est amenée par le signal d'échantillonnage.

Nous voyons de nombreuses perspectives pour faire suite à ces travaux. Nous donnons ici une liste, non exhaustive, résumant les principales pistes de recherche que nous avons identifiées :

- Notre approche pour valider nos attaques consiste à simuler le comportement que pourrait avoir le circuit lorsqu'il subit une injection de faute. Pour confirmer nos résultats il serait intéressant d'effectuer réellement ces attaques à l'aide de lasers ou de sondes électromagnétiques et d'ainsi valider les hypothèses que nous avons fait sur leurs effets.
- Nos premiers résultats silicium ont besoin d'être consolidés. La caractérisation du *jitter* et la validation du mode de fonctionnement du **STR** sont les deux éléments essentiels permettant de s'assurer que le générateur se comporte comme prédit par le modèle stochastique. Un deuxième *testchip* a été conçu dans ce sens. Ce circuit nous permettra aussi de valider le nouveau modèle proposé. Nous pourrions ainsi vérifier que la nouvelle architecture qui en découle permet effectivement d'augmenter l'efficacité énergétique du générateur.
- Toujours dans une optique d'optimisation, il serait intéressant d'étendre le modèle stochastique pour prendre en compte à la fois le bruit du **STR** et celui du signal d'échantillonnage. Le générateur résultant de ce cumul des deux modèles pourra être plus petit et moins consommant.
- Enfin, il serait intéressant de travailler sur la source d'entropie pour essayer de maximiser le ratio $\sigma/\Delta\varphi$. Nous avons vu que l'abaissement de la tension avait un effet

bénéfique sur σ , et il est nécessaire d'approfondir ce point. L'utilisation de cellules ou de structures particulièrement sensibles au bruit thermique peut aussi être envisagée.

Les travaux présentés dans ce chapitre ont été en partie publiés dans une conférence internationale et ont fait l'objet d'un dépôt de brevet :

- G. GIMENEZ *et al.* « Self-timed Ring based True Random Number Generator : Threat model and countermeasures ». In : *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. 2017, p. 31-38
- Laurent FESQUET *et al.* « Circuit and method for protecting asynchronous circuits ». Brev. amér. WO2020008229. 3 juil. 2018

Chapitre 6

Fonction physique non-clonable à base d'anneau auto-séquencé

Sommaire

6.1	Introduction	144
6.2	Modèles stochastiques	145
6.2.1	PUF basé sur la mesure de fréquence	145
6.2.2	PUF basé sur la mesure de temps	147
6.2.3	Mesurer le temps et non la fréquence	149
6.3	TDC à base de STR	149
6.4	STR-PUF	150
6.4.1	Architecture	151
6.4.2	Principes de fonctionnement	152
6.5	Implémentation sur FPGA	153
6.5.1	Détails d'implémentation	153
6.5.2	Qualité des réponses brutes	154
6.5.3	Évaluation	155
6.5.4	A propos de la latence	157
6.6	Circuit de test	158
6.6.1	Objectifs	158
6.6.2	Architecture	159
6.6.3	Implémentation	160
6.7	Conclusion et perspectives	161

6.1 Introduction

À l’instar d’une empreinte digitale, les disparités du processus de fabrication permettent de caractériser de manière unique un circuit intégré. En mesurant les effets qu’elles peuvent avoir sur les différents composants d’un ensemble de circuits, on retrouve généralement des distributions de type Gaussiennes, caractéristiques de l’addition d’un grand nombre de petites perturbations aléatoires¹. Ainsi le processus de fabrication lui-même peut être vu comme une source d’aléa ; un aléa statique, figé dans le circuit, qui peut être raisonnablement considéré comme incontrôlable². Nous l’avons vu au [chapitre 4](#), il existe de nombreuses manières d’extraire cet aléa. Les plus prisées restent la mesure de la variation de fréquence d’un oscillateur, de la variation d’un temps de propagation ou bien des variations dans la résolution d’un état métastable. Les trois principales architectures de **PUF** qui en découlent sont comparées dans la [figure 6.1](#). Cette figure positionne de manière relative ces **PUF** suivant leur efficacité et leur latence.

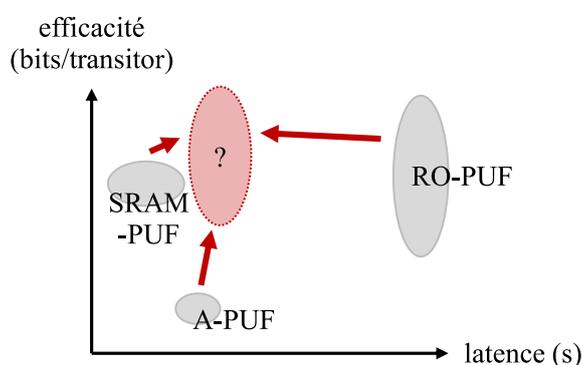


FIGURE 6.1 – Comparaison de la latence et de l’efficacité des trois principales architectures de PUF.

Dans ce chapitre, nous nous inspirons de chacune de ces architectures pour imaginer un nouveau **PUF** numérique qui soit efficace, réactif et résilient. Nous nous intéressons particulièrement à la partie de conversion analogique-numérique qui doit être capable d’extraire des valeurs non binaires avec un haut niveau d’entropie à la manière du **RO-PUF**, dans un temps réduit à l’instar du **A-PUF** et sans faille évidente, contrairement au **SRAM-PUF**. Ce **PUF** recherché est positionné en rouge sur la [figure 6.1](#).

Nos travaux commencent avec le constat suivant : parmi les trois architectures dominantes, les **PUF** utilisant le délai de propagation sont à notre connaissance les seuls n’ayant été associés qu’avec un extracteur binaire. Les deux autres architectures, n’ayant pas cette limitation [[KL15](#) ; [Bar+18](#) ; [Bos+14](#) ; [HG17](#)], sont potentiellement plus efficaces car pouvant extraire plusieurs bits d’entropie par source. Notre intuition est donc de venir optimiser le fonctionnement des **PUF** à base de délais en les assortissant d’un extracteur plus précis, permettant de mesurer finement les temps de propagation. La [section 6.3](#), présente un tel extracteur – un convertisseur temps-numérique (*Time-to-Digital Converter*) (**TDC**) – qui utilise la très petite résolution de phase d’un **STR** pour mesurer précisément des temps de propagation. Dans la [section 6.4](#), nous proposons une nouvelle architecture de **PUF** utilisant ce **TDC** et détaillons son principe de fonctionnement. Les résultats d’une

1. D’après le théorème central limite.

2. Depuis plus de trente ans, les industriels luttent pour minimiser ces disparités qui limitent les performances des circuits. Celles-ci ne font pourtant qu’augmenter avec la réduction des dimensions de fabrication.

implémentation sur **FPGA** et une analyse de ceux-ci sont présentés dans la [section 6.5](#). Finalement, le circuit de test **ASIC** que nous avons développé est présenté en [section 6.6](#).

Mais pour commencer, il est intéressant de noter que les techniques d'extraction des aléas dynamique et statique sont très proches. Les mêmes grandeurs physiques (fréquence, temps de propagation, métastabilité) et blocs d'extraction (**RO**, élément de retard, éléments séquentiels) sont globalement utilisés et, bien souvent, ces deux primitives sont associées dans un même bloc **IP**. A l'instar des **TRNG**, il est donc pertinent de construire un modèle stochastique pour anticiper les performances et comparer les différents **PUF**. Nous abordons ce sujet dans la [section 6.2](#).

6.2 Modèles stochastiques

Nous proposons de modéliser le caractère aléatoire d'un **PUF**. À cette fin, nous ne ferons pas de distinction entre l'entropie inter-puces et l'entropie intra-puces. Ces deux notions partagent la même source d'incertitude : les disparités du processus de fabrication. Cependant, chacune d'entre elles s'intéresse à l'estimation d'un biais systématique différent. La première s'attache à une déviation du processus de fabrication que l'on peut distinguer au niveau d'un *wafer* ou entre différents *wafer*. Le second est sensible à un problème local principalement lié à un biais de conception ou de dessin des circuits (mauvais équilibrages de chemins, différences dans les environnements des sources d'entropie, etc.). Les deux modèles stochastiques proposés dans cette section sont donc basés sur un paramètre qui estime la variabilité globale du processus de fabrication (σ_p). Bien entendu, ces modèles ne sont pas suffisants pour garantir le bon fonctionnement et la sécurité d'un **PUF**. Ils restent pourtant utiles pour comparer qualitativement différentes architectures. Nous modélisons deux **PUF** implémentant deux méthodes de mesure :

- 1) la mesure de la fréquence en comptant le nombre d'impulsions N_f du signal mesuré de période T_x sur une fenêtre de temps W . La fréquence est alors simplement retrouvée en calculant le ratio N_f/W . Et
- 2) la mesure d'un temps en comptant le nombre d'impulsions N_t d'une base de temps fixe de période Δ pendant le temps T_x à mesurer. Ce temps est alors retrouvé en multipliant N_t par Δ .

Afin de comparer ces deux approches, les mêmes sources d'entropie sont utilisées, des **RO** formés à partir d'éléments de délai rebouclés sur eux-mêmes. Les deux **PUF** s'intéressent donc respectivement à caractériser la fréquence ou la période de ces oscillateurs. Par ailleurs, dans un souci de simplification, nous faisons les approximations suivantes :

- La conversion analogique-numérique est considérée idéale, sans métastabilité ni bruit.
- L'erreur de quantification est prise en compte au niveau du post-traitement au lieu de l'étape d'extraction.
- Les sources d'entropie sont indépendantes. Même si cette hypothèse est raisonnable, elle est difficile à obtenir et impossible à prouver en réalité.

6.2.1 PUF basé sur la mesure de fréquence

Soit P la période d'un **RO**. Ses variations dues au processus de fabrication peuvent être modélisées à l'aide d'une loi normale (6.1) :

$$P \sim \mathcal{N}(\mu, \sigma_p) \tag{6.1}$$

La sortie C du compteur cadencé par ce même RO et activé pendant une fenêtre $W \in \mathbb{R}^+$, peut être approchée³ par une autre loi normale (6.2) :

$$\frac{W}{P} \approx C \sim \mathcal{N}\left(\frac{W}{\mu}, \frac{W\sigma_p}{\mu^2}\right) \quad (6.2)$$

Soient C_1 et C_2 deux variables aléatoires indépendantes de C (la mesure de fréquence de deux RO indépendants). Alors :

$$X = C_1 - C_2 \sim \mathcal{N}\left(0, \frac{\sqrt{2}W\sigma_p}{\mu^2}\right) \quad (6.3)$$

Enfin, soit F une fonction modélisant le bloc de post-traitement tel que :

$$F = \begin{cases} 1 & \text{if } C_1 - C_2 > 1 \\ 0 & \text{otherwise} \end{cases} \quad (6.4)$$

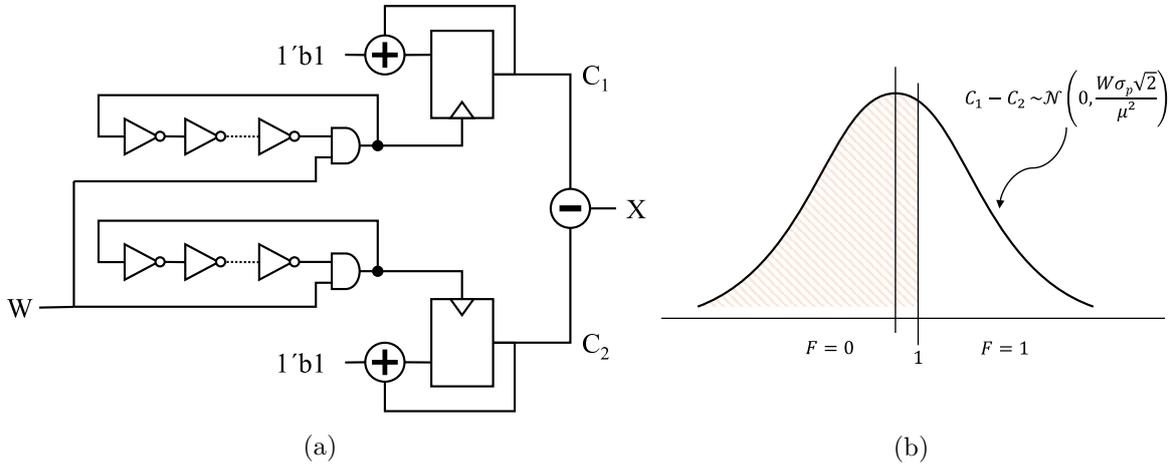


FIGURE 6.2 – Principe de fonctionnement d'un PUF basé sur la fréquence de RO. (a) architecture et (b) modèle stochastique.

Comparer la fréquence de deux RO permet d'améliorer la stabilité des réponses du PUF en limitant les effets des conditions d'opération. Lorsque le premier oscillateur est plus rapide, un 1 est extrait, un 0 dans le cas contraire. Mais dans le cas où $0 < C_1 - C_2 < 1$, les deux oscillateurs paraissent avoir la même fréquence, donnant un 0 en sortie du bloc de post-traitement. C'est l'erreur de quantification. La figure 6.2 illustre ce fonctionnement. L'aire hachurée sous la distribution de X correspond à la probabilité d'obtenir un 0. Elle peut s'exprimer :

$$\begin{aligned} P(F = 0) &= P(X \leq 1) = P\left(X \frac{\mu^2}{\sqrt{2}W\sigma_p} \leq \frac{\mu^2}{\sqrt{2}W\sigma_p}\right) \\ &= \Phi\left(\frac{\mu^2}{\sqrt{2}W\sigma_p}\right) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{\mu^2}{2W\sigma_p}\right) \end{aligned} \quad (6.5)$$

3. Cette approximation est intuitive et peut facilement être simulée. Une preuve est présentée dans [DR13]. Elle n'est valide que pour $|\mu| \gg \sigma$. En pratique, $|\mu| > 10\sigma$ apparaît être un seuil raisonnable. Dans notre cas, cette limite est globalement toujours respectée.

Et par complémentarité, la probabilité d'obtenir un 1 s'écrit :

$$P(F = 1) = 1 - P(F = 0) = \frac{1}{2} - \frac{1}{2} \operatorname{erf}\left(\frac{\mu^2}{2W\sigma_p}\right) \quad (6.6)$$

Nous pouvons alors, comme pour les **TRNG**, calculer une entropie minimum par bit H_m :

$$H_m = -P(F = 0) \log_2(P(F = 0)) - P(F = 1) \log_2(P(F = 1)) \quad (6.7)$$

La **figure 6.3** donne l'évolution de cette entropie minimum en fonction de la taille de la fenêtre de mesure et pour différents ratios de variabilité du processus de fabrication.

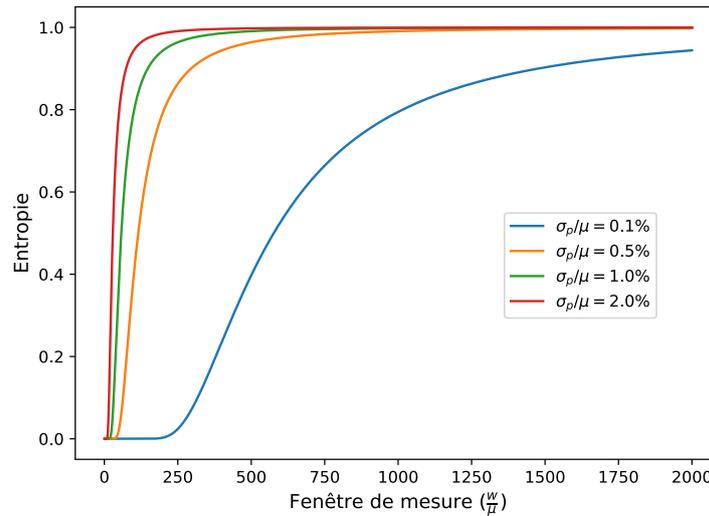


FIGURE 6.3 – Entropie d'un bit d'un PUF basé sur la fréquence en fonction de la taille de la fenêtre de mesure et de la variabilité du processus de fabrication.

6.2.2 PUF basé sur la mesure de temps

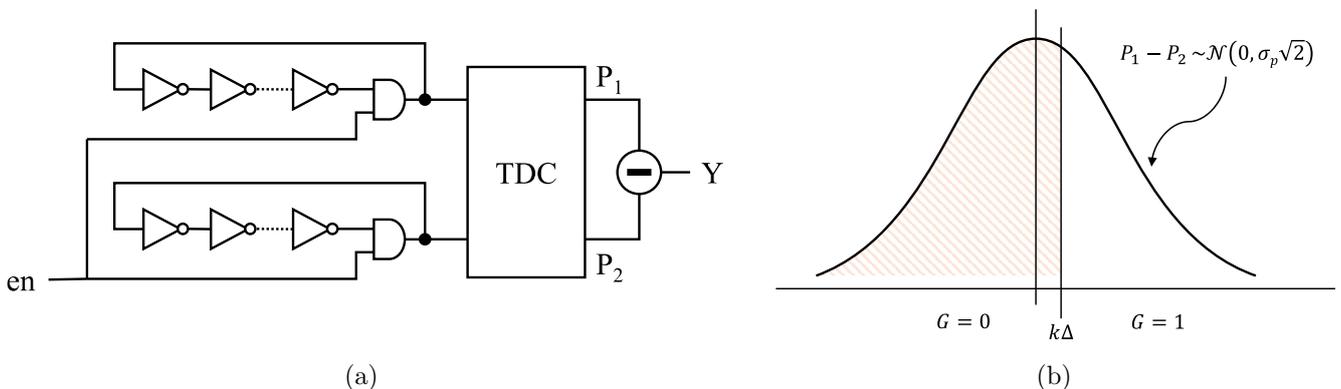


FIGURE 6.4 – Principe de fonctionnement d'un PUF basé sur la période de RO. (a) architecture et (b) modèle stochastique.

L'architecture et le modèle stochastique d'un **PUF** se basant sur la mesure de la période du même **RO** sont présentés en **figure 6.4**. L'espérance μ de P (voir **équation (6.1)**),

directement mesurée à l'aide d'un **TDC** de résolution $\Delta \in \mathbb{R}^+$, peut être exprimée ainsi :

$$\mu = (N + k) \Delta \quad (6.8)$$

avec $N \in \mathbb{N}$ le résultat en sortie du **TDC** et $k \in [0; 1[$ le facteur d'erreur. Soient P_1 et P_2 deux variables aléatoires de P indépendantes. La comparaison des périodes de deux **RO** peut s'exprimer à l'aide d'une autre variable aléatoire telle que :

$$Y = P_1 - P_2 \sim \mathcal{N}\left(0, \sqrt{2}\sigma_p\right) \quad (6.9)$$

De manière similaire à l'équation (6.4), nous définissons G la fonction de post-traitement telle que :

$$G = \begin{cases} 1 & \text{if } P_1 - P_2 > k\Delta \\ 0 & \text{otherwise} \end{cases} \quad (6.10)$$

Les probabilités d'extraire un 0 ou un 1 s'expriment en fonction de l'erreur k :

$$P_k(G = 0) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{k\Delta}{2\sigma_p}\right) \quad (6.11)$$

et

$$P_k(G = 1) = \frac{1}{2} - \frac{1}{2} \operatorname{erf}\left(\frac{k\Delta}{2\sigma_p}\right) \quad (6.12)$$

L'entropie par bit en sortie de ce **PUF** utilisant un **TDC** est minimum lorsque l'erreur est la plus grande. Elle peut donc être calculée ainsi :

$$H_m = -P_{k=1}(G = 0) \log_2(P_{k=1}(G = 0)) - P_{k=1}(G = 1) \log_2(P_{k=1}(G = 1)) \quad (6.13)$$

La figure 6.5 donne l'évolution de l'entropie minimum en fonction de la résolution du **TDC** et pour différents ratios de variabilité du processus de fabrication.

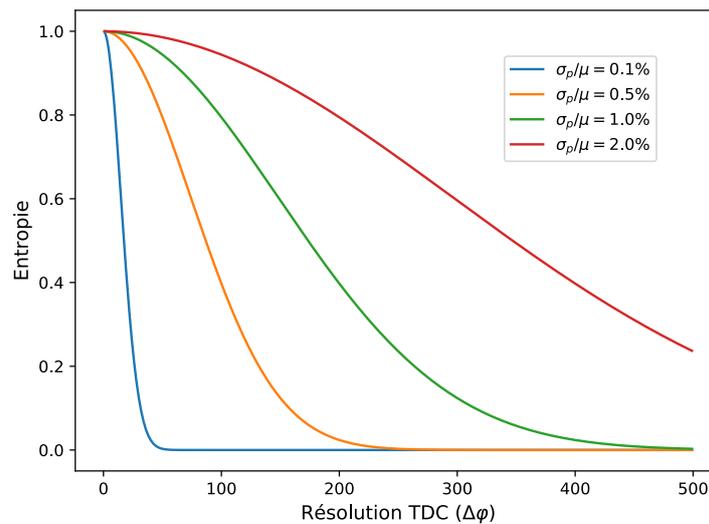


FIGURE 6.5 – Entropie d'un bit d'un PUF basé sur la période en fonction de la résolution du TDC et de la variabilité du processus de fabrication.

6.2.3 Mesurer le temps et non la fréquence

Quelques remarques intéressantes peuvent être faites en comparant ces deux modèles. Dans les deux cas, l'entropie par bit augmente logiquement avec la variabilité du processus de fabrication. Les figures 6.3 et 6.5 illustrent très bien ce phénomène. Mais cette disparité est difficile à contrôler. Elle est très dépendante de la technologie et repose essentiellement sur la maîtrise du processus de fabrication des fondeurs. Augmenter artificiellement cette variabilité est donc une piste de recherche intéressante, qui doit être suivie en étroite collaboration avec les fonderies.

Comme attendu, nous pouvons déduire des équations (6.5) à (6.7) que l'entropie par bit du modèle basé sur la fréquence augmente avec W (c'est-à-dire, lorsque $\frac{\mu^2}{2W\sigma_p}$ tend vers 0). En d'autres termes, plus la fenêtre de mesure est grande, plus il sera facile de distinguer les RO les uns des autres et plus l'entropie extraite sera importante. Mais il est aussi intéressant de voir que le modèle basé sur la fréquence dépend exponentiellement de la période du RO. Cela sous-entend que les RO avec peu d'étages ont une meilleure efficacité. A l'inverse, l'entropie du modèle basé sur la mesure du temps dépend de la résolution de l'extracteur. Plus Δ est petit vis-à-vis de la déviation standard du processus de fabrication plus l'entropie sera grande.

Les deux modèles proposés sont en ligne avec des résultats déjà bien connus. Pour avoir le même niveau d'entropie entre les deux PUF, il faut respecter l'égalité $\Delta = \mu^2/W$. En réalité cette relation vient directement du calcul de l'erreur de quantification de chacune de ces méthodes. Nous l'avons vu, nos modèles ne prenant pas en compte d'autres sources de bruit, seule cette erreur de mesure vient impacter l'entropie extraite. En d'autres termes, s'il n'y a pas d'erreur de mesure, l'entropie est toujours maximale. Pour la modèle caractérisant la fréquence, il est bien connu que l'erreur de mesure faite est $\varepsilon = T_x/W$. Plus la fenêtre de mesure est grande plus la précision augmente. Cette méthode de mesure est donc très lente. Pour la mesure du temps, l'erreur s'exprime quant à elle par $\varepsilon = \Delta/T_x$. Dans ce cas, la précision de la mesure dépend donc de la fréquence de la base de temps. Naturellement nous retrouvons l'égalité précédente.

Même si les principes de mesure du temps et de mesure de la fréquence sont très proches, ils introduisent en pratique des contraintes très différentes. Si augmenter la fenêtre de mesure W est trivial, augmenter la résolution Δ de la technique de mesure du temps est bien plus difficile. Pour résoudre ce problème nous proposons d'utiliser le STR qui peut être vu comme un générateur polyphasé élaboré par concaténation de plusieurs bases de temps plus lentes.

6.3 TDC à base de STR

Une approche naïve pour augmenter la résolution d'un TDC à base de compteurs, comme modéliser par l'équation (6.8), consiste à utiliser une base de temps la plus rapide possible. Comme nous l'avons vu au chapitre 4, le signal virtuel Ψ composé de l'agrégation de chacune des phases d'un STR semble à première vue un bon candidat. La fréquence de ce signal peut être très élevée et permettra d'avoir une résolution du TDC plus fine que la période minimale atteignable avec n'importe quel RO à base de cellules standards. Mais c'est cette propriété même qui empêche d'entretenir électriquement et d'utiliser le signal Ψ dans un circuit. En effet, aucun élément séquentiel ne peut fonctionner à cette vitesse et, par ailleurs, la moindre cellule logique traversée par ce signal jouera le rôle de filtre passe-bas⁴. Une autre approche est donc nécessaire.

4. Les oscillations de Ψ sont si courtes que la sortie de la cellule n'a pas le temps de s'établir avant qu'un événement inverse ne se présente en entrée.

EL-HADBI *et al.* [El-+17] proposent de voir le STR comme une agrégation de plusieurs bases de temps plus lentes, fonctionnant de manière synchrone mais déphasée. La figure 6.6 présente l’architecture de TDC imaginée à partir de ce constat. Chacune des phases du STR est utilisée pour cadencer un compteur⁵ qui est démarré et arrêté par le signal à mesurer. Ainsi, chaque compteur caractérise la période de temps à mesurer à la résolution T_{STR} , la période d’oscillation du STR. Les valeurs de chacun des compteurs peuvent alors être recombinaées pour atteindre la résolution $\Delta\varphi$.

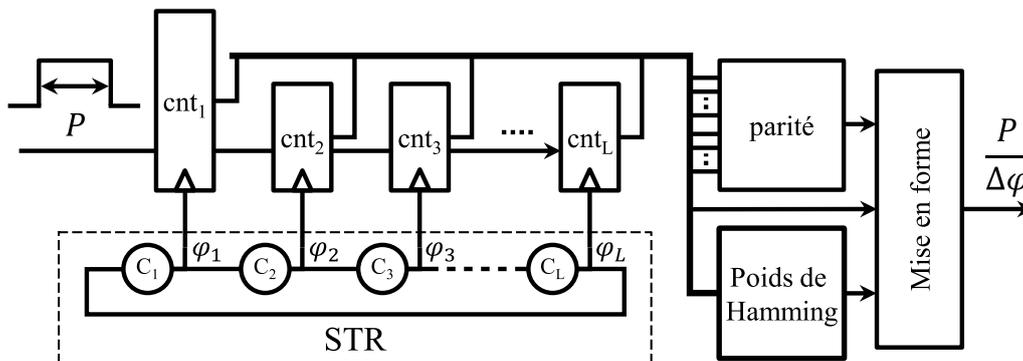


FIGURE 6.6 – Architecture d’un TDC à base de STR.

Une impulsion P appliquée en entrée de ce TDC peut être décomposée ainsi :

$$P = M \times T_{STR} + k\Delta\varphi = (M \times L + k)\Delta\varphi \quad (6.14)$$

avec $M \in \mathbb{N}$ et $k \in [0, L[$. En pratique, le résultat N_v d’un seul des compteurs permet de déterminer le nombre M d’intervalles T_{STR} dans l’impulsion mesurée P . Les autres compteurs peuvent être limités à deux bits, ce qui est suffisant pour retrouver le nombre k d’intervalles $\Delta\varphi$ restants. Ce calcul peut être fait à l’aide d’un algorithme simple utilisant le poids de Hamming du vecteur composé des bits de poids faible de chacun des compteurs et un bloc calculant la parité de M à partir des bits de rang 1 des compteurs [El-+17].

D’autres architectures de TDC peuvent être utilisées pour construire notre PUF [EEF19]. En particulier, une architecture à base de Vernier est très souvent utilisée afin d’atteindre une résolution très fine [DSH00 ; RB06 ; LAL13]. Nous préférons le TDC à base de STR car il a l’avantage d’être auto-calibré (le mode *evenly-spaced* du STR est atteint naturellement) et d’être facilement configurable en dimensionnant le nombre d’étages de l’anneau et le nombre d’évènements y circulant.

6.4 STR-PUF

Nous proposons de construire un PUF à partir du TDC présenté en section 6.3. L’architecture de ce nouveau PUF est donnée en figure 6.7.

5. Une résolution deux fois plus petite ($T_{STR}/2$) est initialement proposée dans [El-+17]. Cela requière l’utilisation de bascules sensibles sur les fronts montants et descendants qui n’existent nativement pas dans les bibliothèques de cellules standards. Pour simplifier la conception nous préférons dans nos travaux utiliser des bascules simple-front, multipliant ainsi la résolution par 2. Ce choix est aussi motivé par la difficulté à obtenir des $\Delta\varphi$ équilibrés dans un STR : à cause du déséquilibre PMOS-NMOS intrinsèque à toutes technologies CMOS, il est difficile d’obtenir un signal Ψ avec un rapport cyclique d’exactement 50 %.

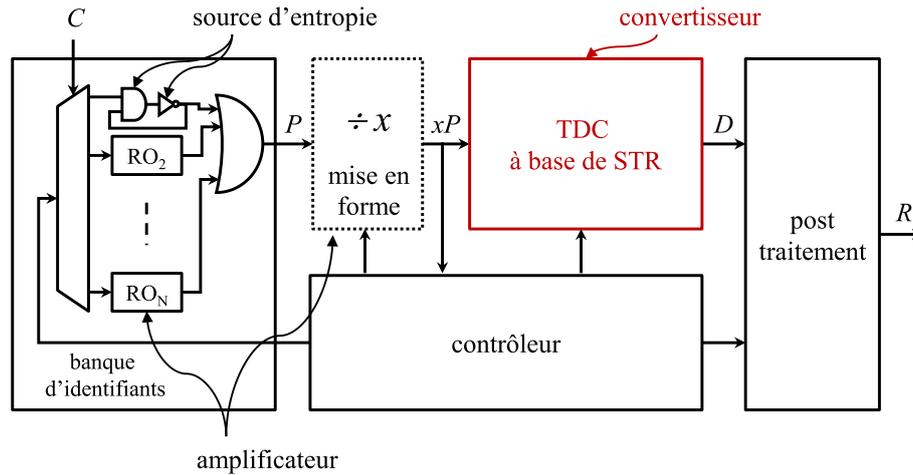


FIGURE 6.7 – Architecture du STR-PUF.

6.4.1 Architecture

Le fonction physique non-clonable basée sur un oscillateur auto-séquence (*STR based PUF*) (**STR-PUF**) collecte de l'entropie de manière conventionnelle, en utilisant des cellules standards comme des capteurs de variabilité du processus de fabrication. A l'instar du **A-PUF**, deux chemins de propagation concurrents peuvent être utilisés pour créer une impulsion à mesurer. Nous préférons cependant utiliser des **RO** car ceux-ci sont plus compacts et permettent d'amplifier le signal. De très petits anneaux peuvent ainsi être utilisés pour limiter la taille de la banque d'identifiants. Leur fréquence d'oscillation peut ensuite être réduite par divisions successives, afin d'augmenter le **SNR**⁶. Cette tâche est effectuée par un bloc de mise en forme, composé de simples bascules montées en diviseur de fréquence.

Le signal analogique caractéristique est généré de la manière suivante. Le challenge C sélectionne l'un des **RO** à caractériser. Lorsqu'il est activé, ce dernier génère une série d'impulsions de largeur P . Le bloc de mise en forme multiplie ce signal et extrait une seule impulsion de largeur xP . La largeur de l'impulsion générée peut être mesurée grâce au **TDC**. Il donne un résultat D en nombre de $\Delta\varphi$, la résolution de phase du **STR**.

$$D = \lfloor xP/\Delta\varphi \rfloor \quad (6.15)$$

Comme le **STR** et les **RO** sont impactés de manière similaire par les conditions d'opération, le résultat du **TDC** est plutôt stable et peut être utilisé directement pour construire la réponse du **PUF**. Mais un bloc de post-traitement peut être adjoint pour assurer une meilleure stabilité des réponses. Globalement, trois principales approches existent : comparaison, moyenne ou encodage [HG17]. Leur fonctionnement n'est pas détaillé dans ce manuscrit. Dans nos travaux, nous utilisons la version la plus simple : la comparaison de paires de **RO**. Un bit de réponse 0 ou 1 est extrait en fonction de l'oscillateur se trouvant être le plus rapide. Cette solution n'est pas la plus efficace car elle permet d'extraire uniquement un nombre de bits moitié du nombre de sources. Mais elle assure théoriquement l'indépendance des bits de réponse.

6. Plus de détails sont donnés dans la section 6.4.2.

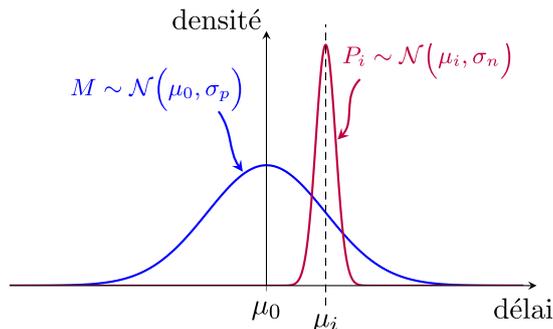


FIGURE 6.8 – Modèle probabiliste de la largeur d'impulsion.

6.4.2 Principes de fonctionnement

Bien que cette nouvelle architecture soit plus complexe que celle des PUF classiques, elle permet d'optimiser l'extraction d'entropie en ajustant seulement quelques paramètres. Deux leviers sont disponibles pour adapter l'efficacité de ce PUF : 1) la résolution $\Delta\varphi$ du TDC et 2) le facteur multiplicatif x du bloc de mise en forme. Il existe des preuves bien documentées [PDW89] que le temps de propagation au travers d'éléments de délai peut être modélisé comme une variable aléatoire indépendante et identiquement distribuée (i.i.d.) sélectionnée lors de la fabrication. Nous pouvons donc modéliser la largeur des impulsions générées par toutes les sources d'un ensemble de PUF par une distribution Gaussienne $M \sim \mathcal{N}(\mu_0, \sigma_p)$.

Comme tout composant en électronique, les éléments de délai sont sujets au bruit thermique. Une autre variable aléatoire δ doit être ajoutée à chaque réalisation μ_i de M . Comme proposé par DAIHYUN LIM *et al.* [Dai+05], nous modélisons ce bruit comme une distribution normale centrée de déviation standard σ_n . Le modèle P_i des impulsions générées par une source unique i peut alors s'écrire :

$$P_i = \mu_i + \delta \sim \mathcal{N}(\mu_i, \sigma_n) \quad (6.16)$$

Comme proposé par SCHAUB *et al.* [Sch+18], le SNR de cette impulsion peut être défini comme suivant :

$$SNR = \frac{\mathbb{E}[M^2]}{\mathbb{E}[\delta^2]} = \frac{\sigma_p^2}{\sigma_n^2} \quad (6.17)$$

La figure 6.8 présente les deux distributions M et P_i . Pour que notre PUF soit efficace, l'extracteur doit capturer un maximum de variabilité mais rejeter le plus possible le bruit. De manière intuitive, plus la résolution du TDC est petite, plus la différenciation de deux impulsions est facile et plus la disparité du processus de fabrication peut être capturée. En même temps, une résolution plus importante permet de limiter l'influence du bruit sur les mesures. Un compromis doit donc être trouvé lors du réglage de $\Delta\varphi$ du TDC.

Le ratio des déviations standards joue un rôle majeur dans cet équilibre. Mais comme σ_p et σ_n sont tous deux fixés par la technologie employée, il est intéressant de trouver un autre paramètre de conception permettant d'améliorer l'efficacité de l'extracteur. C'est l'objectif du bloc de mise en forme. Il multiplie la période d'oscillation du RO sélectionné par un facteur fixe x . Cela impacte de manières différentes les deux distributions M et P_i . La largeur d'impulsion étant fixée lors de la fabrication, la distribution M évolue linéairement avec x . Par contre, chaque oscillation du RO est sujette au bruit. Le bloc de mise en forme accumule ainsi le bruit proportionnellement à la racine carrée du facteur de multiplication ($\propto \sqrt{x}$). La sortie de ce bloc peut alors être modélisée par la distribution

suivante :

$$xP_i \sim \mathcal{N}(x\mu_i, \sqrt{x}\sigma_n) \quad (6.18)$$

Et le SNR résultant est :

$$SNR_x = x \times \frac{\sigma_p^2}{\sigma_n^2} \quad (6.19)$$

La mise en forme du signal permet ainsi d’ajuster le SNR vu par le TDC. Il donne une flexibilité supplémentaire pour optimiser l’entropie récoltée par le STR-PUF tout en maintenant la stabilité à un niveau raisonnable.

6.5 Implémentation sur FPGA

Nous avons implémenté le STR-PUF sur 8 FPGA Xilinx Zynq7000. Pour chaque PUF, la banque d’entropie est composée de 1024 RO. Le bloc de mise en forme permet de multiplier la taille de l’impulsion générée d’un facteur 16, 32, 64 ou 128. Nous avons testé trois configurations différentes de STR : 9, 31 et 63 étages pour respectivement 4, 16 et 32 événements. La résolution atteinte est approximativement de 301 ps, 87 ps et 43 ps. Nous avons effectué une campagne de mesures incluant 1000 extractions de largeur d’impulsion pour chacune des 8192 sources et pour les 12 configurations de PUF. Au total, plus de 98 million de mesures ont été faites. Le circuit est contrôlé depuis un ordinateur au travers d’une interface UART permettant d’envoyer un challenge (l’adresse du RO à caractériser) et de recevoir la réponse de la part du PUF (la période en nombre de $\Delta\varphi$). Pour accélérer l’extraction, un mode rafale permet de mesurer séquentiellement chacune des sources de la banque d’entropie. L’opération de post-traitement – dans notre cas la comparaison de la période de deux RO – est effectuée *a posteriori*, de manière logicielle à l’aide de routines écrites en Python.

6.5.1 Détails d’implémentation

La banque d’entropie est constituée de RO formés d’une porte ET et de trois inverseurs. Afin de minimiser tout biais systématique, le placement et le routage de ces sources est répliqué automatiquement en assignant chaque cellule et chaque fil à des ressources spécifiques de la matrice du FPGA. Chaque RO est ainsi placé sur 4 *LookUp Table* (LUT)⁷ appartenant au même SLICE⁸. Comme préconisé par [Hes+18], nous utilisons uniquement des SLICE de même type (SLICEL dans notre cas) afin d’éviter les légères différences qui existent au niveau des ressources de routage et dans les temps de propagation de ces différents éléments. Par ailleurs, le placement dans un même SLICE permet de conserver un routage local entre chaque cellules d’un anneau, limitant l’influence de l’environnement sur la fréquence d’oscillation.

Pour garantir un fonctionnement optimal du STR, le placement et le routage de chacun de ses étages requièrent aussi une attention particulière. Comme nous l’avons vu aux chapitres 4 et 5, le mode *evenly-spaced* ne peut être atteint que si les différences de temps de propagation entre les étages sont suffisamment petites pour être couvertes par l’effet *Charlie*. Il est donc nécessaire de placer méticuleusement chaque cellule et de s’assurer que les routages entre les étages – les chemins de requête et d’acquittement – soient les plus

7. Un LUT par cellule

8. Il existe plusieurs types de SLICE dans les FPGA de Xilinx, mais tous embarquent au moins 4 LUT et 8 registres.

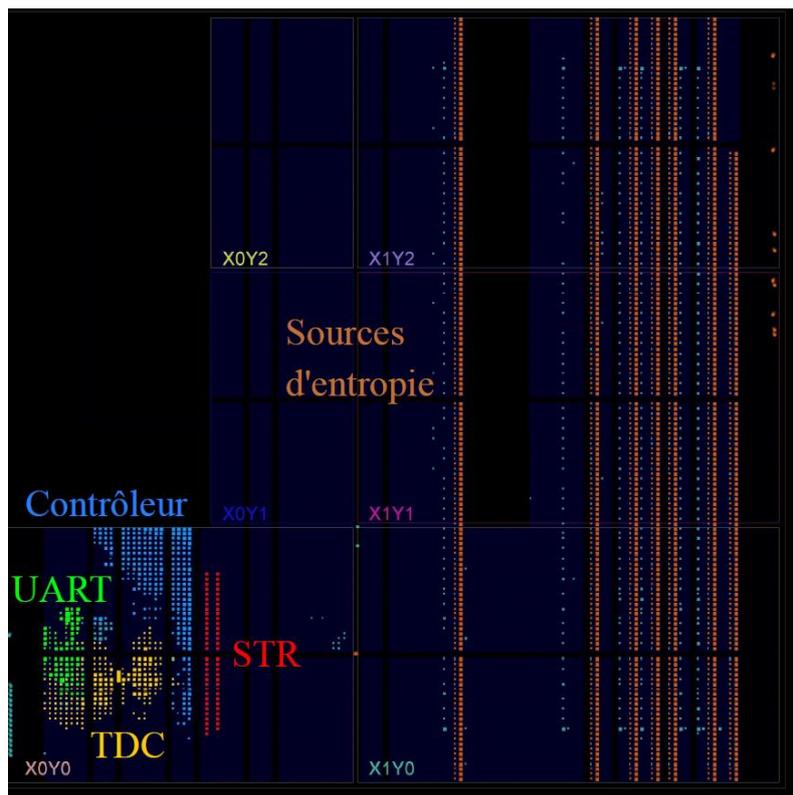


FIGURE 6.9 – Floorplan du STR-PUF dans la matrice FPGA.

similaires possibles. Pour répondre à ces exigences, nous avons choisi de réserver un SLICE pour chaque étage. La cellule de Muller est implémentée dans un seul LUT (A6LUT). Comme pour les RO nous profitons des capacités avancées de contrôle du routage de l’outil de conception Vivado pour reproduire le même routage entre chacun des étages⁹.

La figure 6.9 présente le placement des différents éléments du circuit de test du STR-PUF. La logique de contrôle et de mesure ainsi que le STR sont placés le plus loin possible de la banque d’entropie afin de ne pas influencer le fonctionnement des RO. Pour limiter un éventuel biais déterministe dans la réponse du PUF, durant le post-traitement nous comparons des oscillateurs placés côte à côte. De cette manière nous assurons un environnement similaire pour chacun d’eux. Dans cette perspective, il serait intéressant d’analyser l’impact de la logique avoisinante sur les réponses du PUF. Comme proposé dans [Hes+18], cela peut être fait en ajoutant un bloc de perturbateur proche de la banque d’entropie.

6.5.2 Qualité des réponses brutes

La figure 6.10a présente une cartographie de la matrice du FPGA détaillant la largeur d’impulsion de chaque source d’entropie. Malgré toutes les précautions prises, nous pouvons clairement distinguer un motif répétitif sur cette carte. Les RO situés dans les lignes 24, 74 et 124 se trouvent avoir une période d’oscillation plus importante. Ce phénomène a déjà été reporté dans [Hes+18]. Bien que les mêmes ressources de routage soient utilisées, il apparaît que la présence de canaux dédiés au transport de signaux d’horloge proches

9. Comme nous pouvons le voir sur la figure 6.9, les étages du STR sont placés sur deux colonnes. Les étages en haut et en bas de ces deux colonnes, qui forment les « virages » de l’anneau, ont un routage légèrement différent. Mais cette différence est suffisamment limitée pour ne pas affecter le fonctionnement de l’anneau.

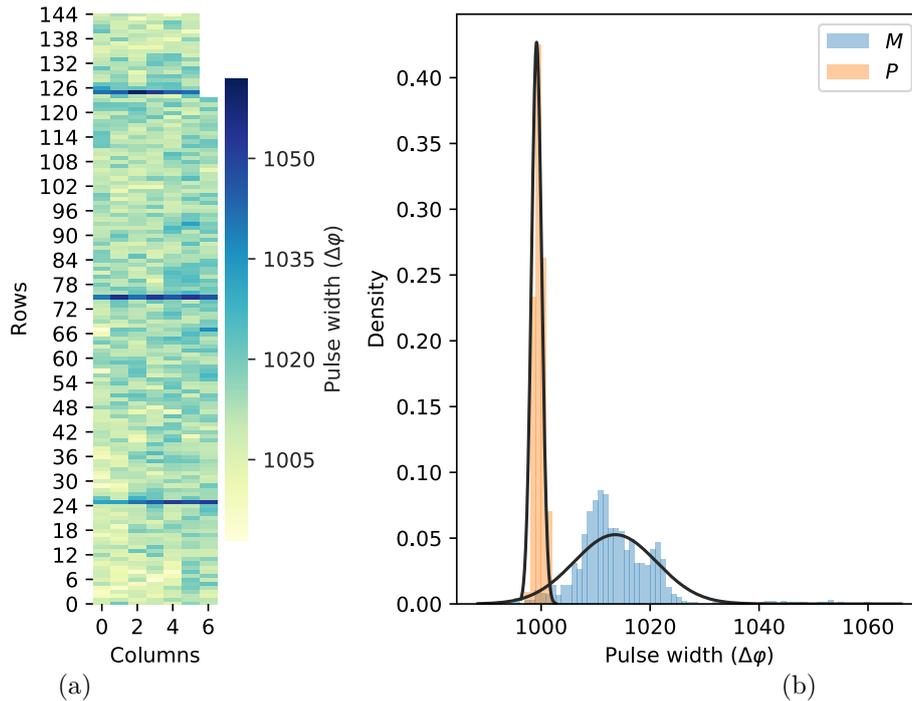


FIGURE 6.10 – Réponses brutes des 1024 sources d’un PUF, avec $x = 16$ et $\Delta\varphi = 43$ ps. (a) cartographie des largeurs d’impulsion (en nombre de $\Delta\varphi$) et (b) distribution des largeurs d’impulsion pour chaque source (bleu) et pour 1000 mesures de la même source (orange).

de ces lignes entraîne une légère augmentation de la longueur d’un des fils de ces RO. De manière intéressante, ce motif est aussi visible même avec la configuration de PUF la plus relâchée (c’est-à-dire, avec la plus grande résolution $\Delta\varphi$ et le plus petit facteur multiplicatif x). Cela confirme la bonne sensibilité de notre PUF, même avec des temps de mesures très faibles. Nous ignorons ces sources biaisées lors de l’analyse de la qualité des réponses du PUF.

La fonction de masse de la largeur des impulsions mesurées avec l’un des PUF est tracée sur la figure 6.10b. Cette figure présente aussi la distribution de 1000 mesures d’une même source d’entropie de ce même PUF. Les courbes de tendance Gaussiennes sont aussi reportées sur ce graphique. Elles sont très similaires aux courbes théoriques présentées en figure 6.8. Les déviations standards de ces distributions sont $\sigma_p = 7,61\Delta\varphi$ et $\sigma_n = 0,93\Delta\varphi$. D’après l’équation (6.17), cela correspond à un SNR d’approximativement 18 dB (ou 6 dB une fois normalisé par le facteur x).

6.5.3 Évaluation

Nous évaluons la qualité des réponses du STR-PUF à l’aide des différentes métriques définies à la section 4.4.2. La figure 6.11 présente les résultats du calcul de SNR (6.17) et (6.19), d’unicité (4.17) et de stabilité (4.18) pour les 12 configurations des 8 implémentations FPGA du STR-PUF. Les trois métriques présentent la dépendance attendue vis-à-vis du facteur de multiplication x .

La figure 6.11a montre clairement la relation linéaire entre le SNR et le logarithme du facteur multiplicatif. Toutes les configurations du PUF ont une pente proche de +3 dB/octave qui correspond exactement au modèle proposé (6.19). Le SNR normalisé est aussi reporté sur le graphique. Comme attendu, il est presque constant, avec une valeur moyenne autour de 8,26 dB.

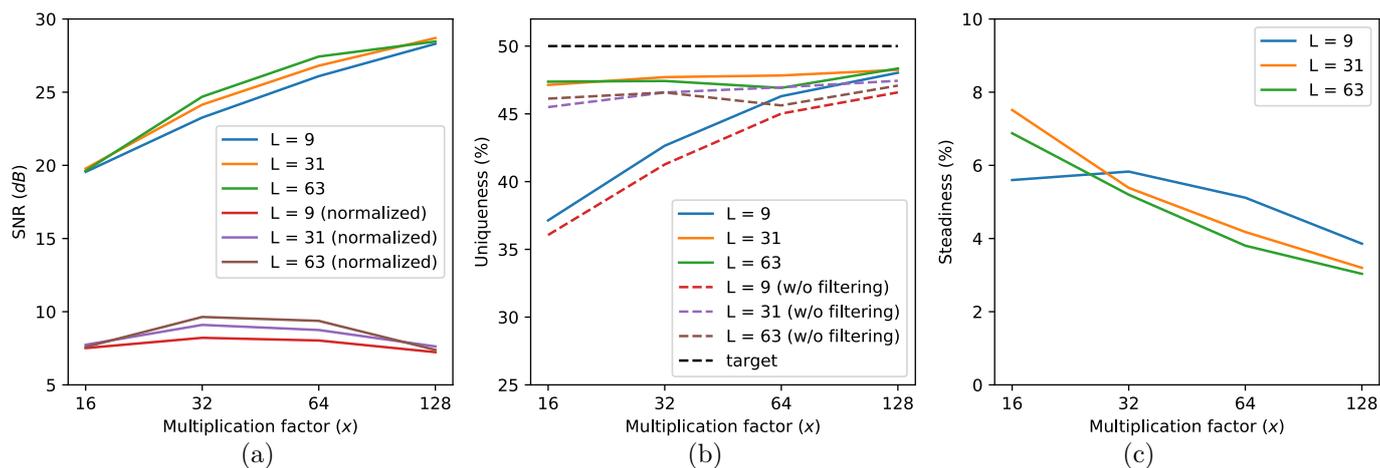


FIGURE 6.11 – Métriques évaluant la qualité des 8 STR-PUF suivant le facteur multiplicatif x et pour trois tailles de STR différentes. (a) SNR et SNR normalisé, (b) unicité et (c) stabilité moyenne.

La métrique caractérisant l’unicité est présentée en figure 6.11b. Les résultats avec ou sans filtrage des sources biaisées sont reportés. L’effet négatif des sources incriminées est bien visible (courbe en pointillés). Il réduit l’unicité de plus de 1 point. Comme attendu, nous voyons que l’unicité augmente avec le facteur multiplicatif. En revanche, quelle que soit la configuration, celle-ci tend asymptotiquement vers une valeur de 48,3 %, qui n’est pas totalement idéale. Bien que cela ne soit pas fatal pour le fonctionnement du PUF, il semble qu’un autre biais soit présent dans les sources d’entropie (par exemple, lié à la structure de la matrice des FPGA ou à des problèmes de routage ou de diaphonie). Ce phénomène peut aussi être lié au faible nombre de PUF testés. Une étude de plus large envergure est donc nécessaire pour trouver l’explication exacte de ce comportement.

Finalement, la courbe de stabilité, présentée dans la figure 6.11c, montre aussi une évolution cohérente vis-à-vis du facteur de multiplication. Mais elle ne semble pas être affectée par la variation de la résolution $\Delta\varphi$ comme nous pouvions intuitivement le prédire. La tendance inverse semble même émerger, la stabilité étant globalement meilleure lorsque la taille du STR augmente. Bien que nous n’ayons pas une explication arrêtée pour ce phénomène, il montre que le modèle de bruit utilisé n’est pas très précis. Notamment, nous ne considérons pas le bruit de quantification. Pour le STR-PUF, à cause des capacités de placement et de routage limité des FPGA, ce bruit de mesure peut pourtant être très important comparativement aux bruits électroniques. Cette hypothèse semble confirmée par la faible valeur de SNR de notre PUF vis-à-vis de celle observée par HESSELBARTH *et al.* [Hes+18].

La suite de tests NIST SP 800-22 est appliquée aux réponses du PUF. Pour chaque configuration, les réponses des 8 instances du PUF sont regroupées une seule séquence de 4016 bits¹⁰. Cette séquence est divisée en deux blocs de 2008 bits, qui sont suffisamment larges pour alimenter la majorité des tests de la suite. Les résultats de chacun de ces tests sont présentés dans le tableau 6.1. En analysant ces résultats, nous constatons que le nombre de tests en échec diminue alors que le facteur de multiplication augmente ou que la résolution diminue. Malgré la faible puissance statistique lié à l’étude d’un nombre limité de PUF, cette tendance est cohérente avec les observations faites pour l’unicité et

10. Chaque PUF a 1024 sources parmi lesquelles 20 sont ignorées car biaisées. Après comparaison des paires d’oscillateurs, nous obtenons $(1024 - 20)/2 = 502$ bits par PUF.

TABLE 6.1 – Évaluation de l'imprédictibilité des réponses des STR-PUF à l'aide de la suite de tests NIST SP800-22.

Taille du STR ($\Delta\varphi$) Facteur multiplicatif. x	L9 (301 ps)				L31 (87 ps)				L63 (43 ps)			
	16	32	64	128	16	32	64	128	16	32	64	128
<i>Frequency</i>	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	0/2	2/2	2/2	2/2
<i>BlockFrequency</i>	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	0/2	2/2	2/2	2/2
<i>CumulativeSums</i>	0/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	4/4	0/4	4/4	4/4
<i>Runs</i>	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	0/2	1/2	2/2	2/2
<i>LongestRun</i>	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	1/2	2/2	2/2
<i>FFT</i>	0/2	1/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	1/2	2/2
<i>NonOverlappingTemplate</i>	272/296	270/296	272/296	283/296	271/296	277/296	284/296	288/296	278/296	286/296	285/296	286/296
<i>ApproximateEntropy</i>	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	1/2	2/2	2/2	2/2
<i>Serial</i>	0/4	3/4	2/4	4/4	4/4	4/4	4/4	4/4	2/4	2/4	2/4	4/4
Taux de succès	86,3%	86,6%	87,3%	91,6%	87,9%	91,0%	95,3%	96,3%	91,0%	96,3%	95,7%	96,9%

confirme la bonne qualité de l'entropie extraite par le STR-PUF.

6.5.4 A propos de la latence

Comme nous le mentionnons initialement, l'un des principaux objectifs de nos travaux était d'optimiser le temps de réponse des RO-PUF. Dans cette architecture, la latence de réponse L_{ROPUF} dépend directement de la taille de la fenêtre de mesure¹¹ :

$$L_{ROPUF} \approx T_W \quad (6.20)$$

Comme mentionné en section 6.2.3, cette fenêtre de mesure est variable et peut être adaptée pour optimiser la qualité des réponses. Mais même pour des oscillateurs de petite taille, implémentés dans des FPGA identiques à ceux que nous utilisons, le temps de mesure optimal recommandé par HESSELBARTH *et al.* [Hes+18] reste élevé, autour de 70 μ s.

En revanche, la latence L_{STRPUF} de l'architecture que nous proposons est liée à la fois à la période d'oscillation moyenne T_{RO} des sources d'entropie et au facteur de multiplication x ¹¹ :

$$L_{STRPUF} \approx x \times T_{RO} \quad (6.21)$$

Cette latence peut facilement être retrouvée en multipliant les résultats bruts moyens du STR-PUF par la résolution estimée du TDC employé (6.14). La période moyenne des RO peut alors être calculée à partir de ces valeurs de latence et de l'équation (6.21). Ces résultats sont présentés dans le tableau 6.2. Nous remarquons que la période moyenne décroît légèrement lorsque le facteur de multiplication augmente. Cet artefact de mesure est lié à l'augmentation de la précision de la mesure. Finalement, la comparaison du point de vue de la latence des deux architectures PUF montre que le STR-PUF améliore les performances du RO-PUF d'au moins un facteur 200.

La réduction de latence apportée par cette nouvelle architecture est donc particulièrement intéressante. En particulier, elle a un coût très raisonnable. Le STR-PUF à 63 étages ne requière que 13% de ressources du FPGA de plus que le RO-PUF. Pour les autres configurations, l'incrément matériel est limité à 3% et 6% (respectivement pour un STR de 9 et 31 étages).

11. Le temps de réponse pour la mesure d'une unique source d'entropie est exprimé. C'est la latence minimale atteignable quand toutes les sources sont lues en parallèle (sans prendre en compte le temps de post-traitement).

TABLE 6.2 – Latence du STR-PUF et périodes moyennes des RO calculées suivant les différentes valeurs du facteur multiplicatif.

Facteur multiplicatif x	16	32	64	128
Latence (ns)	41,4	81,6	162,0	322,7
Période moyenne des RO (ns)	2,59	2,55	2,53	2,52

Une latence réduite est aussi un avantage majeur en terme de sécurité. Cela permet de réduire la fenêtre des attaques par canaux cachés qui pourraient tenter d’extraire la fréquence d’oscillation des RO. Ces attaques peuvent même être complètement prévenues en utilisant des générateurs d’impulsion non oscillants (au prix d’un peu plus de surface silicium).

Bien sûr, une analyse plus approfondie est nécessaire pour comparer équitablement ces deux architectures en termes de qualité de réponse, de résistance aux attaques et de consommation d’énergie. Cela est prévu avec une version ASIC du STR-PUF que nous avons pu envoyer en fabrication mais qui n’est, à l’heure où nous écrivons ces lignes, pas encore sorti des lignes de fabrication.

6.6 Circuit de test

Nous avons réalisé un circuit de test en technologie CMOS065 de STMicroelectronics. Ce circuit en cours de fabrication, et devrait nous être livré au début de l’année 2021. Il doit permettre de tester et de valider le STR-PUF dans sa version ASIC. Pour l’occasion, nous avons aussi intégré dans ce circuit la nouvelle architecture de STR-TRNG proposée en section 5.7. Nous avons adapté les deux primitives afin qu’elles partagent le même STR. Ainsi nous pouvons proposer un bloc IP dual – générant de l’aléa statique et dynamique – optimisé en taille et en consommation.

6.6.1 Objectifs

En plus de son rôle de démonstrateur pour nos deux primitives, ce circuit sert les objectifs suivants :

- Caractériser la technologie afin de retrouver les paramètres alimentant les différents modèles stochastiques : la variance du *jitter* ainsi que la fréquence des différents éléments oscillants (STR, RO utilisé comme source d’entropie du PUF ou comme horloge d’échantillonnage du TRNG).
- Étudier différentes architectures de STR avec des cellules de Muller de type analogique (figure 5.14) ou à base de porte majorité (figure 1.7f) et identifier leur plage d’oscillation *evenly-spaced* en fonction du nombre d’évènements. À noter que ces deux architectures utilisent uniquement des cellules standards et ont été implémentées dans un flot numérique classique.
- Étendre notre étude sur des gammes de tension et de température cohérentes avec les conditions d’utilisation des deux primitives. En particulier, nous souhaitons étudier la stabilité des réponses du PUF dans ses conditions limite d’opération. Nous cherchons aussi à vérifier que la diminution de la tension permet d’améliorer le ratio σ_{noise}/T_{STR} et donc l’entropie extraite par le STR-TRNG. Nous souhaitons aussi vérifier le comportement des deux primitives après vieillissement accéléré.
- Étudier le comportement de chacune des primitives en présence de perturbations venant d’un bloc logique voisin.

- Réaliser une étude statistique à plus large échelle sur le **STR-PUF**, pour conforter les premiers résultats obtenus sur **FPGA**.
- Comparer le **STR-PUF** et le **RO-PUF** selon leurs performances (latence, consommation) et la qualité de leurs réponses (stabilité, unicité, imprédictibilité).
- Observer, si possible de manière directe, la résolution de phase $\Delta\varphi$ des **STR**.

6.6.2 Architecture

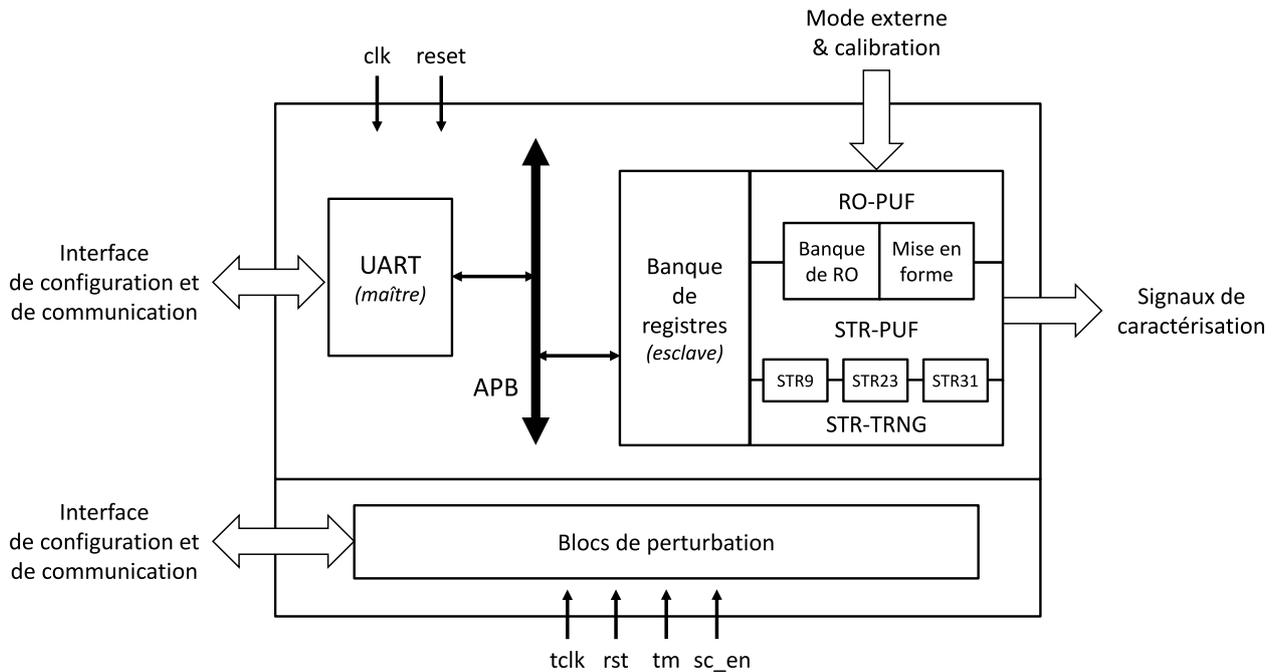


FIGURE 6.12 – Architecture du circuit de test ST CMOS065.

L'architecture du circuit de test est présentée en [figure 6.12](#). Elle intègre trois **STR** différents. Les deux premiers ont une architecture analogique et comportent respectivement 9 et 23 étages. Le dernier possède 31 étages et est formé avec des cellules de Muller à porte majorité. Ces trois **STR** sont utilisés pour construire trois **TRNG** indépendants pouvant fonctionner en mode externe¹², en mode interne¹³ ou alors en mode échantillonnage par un **RO**. À chaque fois, un bloc de délais variables permet de configurer le nombre d'éléments de retard que le signal d'échantillonnage doit traverser. La quantité de bruit capturée par ce signal peut ainsi être adaptée de manière à optimiser l'entropie extraite.

Les trois **STR** servent aussi de base de temps pour les **TDC** de trois **STR-PUF**. Ces **PUF** partagent la même banque d'entropie constituée de 5888 **RO**. Au total, 19 types différents de **RO** sont embarqués. Ils varient selon leur taille, le type de cellules standards ou la variante technologique utilisés (tension seuil, largeur de canal, cellules de délai ou simples inverseurs, etc.). Un bloc de mise en forme permet de diviser la fréquence de la source d'entropie sélectionnée. Il contient aussi un mécanisme de verrouillage permettant de couper la source d'entropie dès qu'une impulsion a été générée. De cette manière, les **RO** n'oscillent que pendant un temps très court, limitant la consommation et les fuites d'informations. La banque d'entropie et le bloc de mise en forme sont partagés avec un

12. Échantillonnage fait par l'horloge externe *clk*.

13. Échantillonnage fait par une des phases du **STR**.

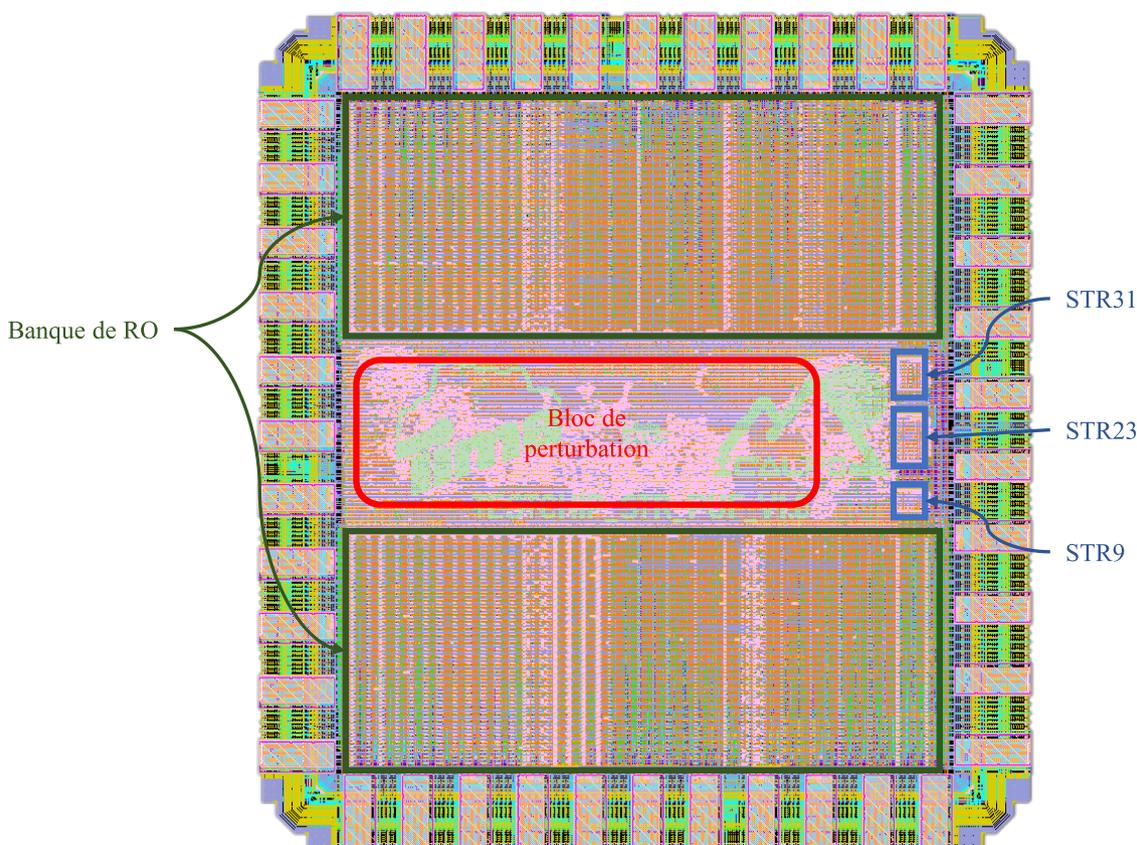


FIGURE 6.13 – Dessin du circuit de test.

RO-PUF classique. La taille de la fenêtre de comptage de ce PUF peut être configurée entre 0 et 2^{16} périodes d'oscillation de l'horloge principale¹⁴.

Chacun de ces blocs est configuré à l'aide d'une même banque de registres connectée sur un bus APB standard. Une interface UART permet de générer des trames sur ce bus et ainsi lire et écrire dans les différents registres de configuration et de statut. Par ailleurs, un mode externe permet d'accéder directement à certains points d'observation (phase des STR, sortie des différents RO, sortie brute des TRNG, etc.) au travers de plots entrées-sorties du circuit. Il permet aussi d'injecter des signaux pour calibrer les TDC ou le circuit de mesure de la résolution de STR. Finalement un dernier bloc permet de générer de l'activité dans le circuit afin de créer des perturbations. Ce bloc partage la même alimentation que le reste du circuit, mais a une interface de configuration et des signaux de contrôle dédiés.

6.6.3 Implémentation

Le circuit a une taille de $1,36 \text{ mm}^2$. Il possède 43 entrées-sorties dont 16 sont réservées à l'alimentation. Le dessin des masques du circuit de test est donné en figure 6.13.

Afin de garantir des performances optimales au niveau des anneaux auto-séquéncés¹⁵, les cellules composant chacun des étages sont placées et routées de manière strictement identique. Une zone d'isolation, sans cellule standard ni routage, est aussi réservée autour

14. Cette horloge est prévue pour fonctionner à 100 MHz, ce qui représente une fenêtre de comptage maximale de $655 \mu\text{s}$.

15. C'est-à-dire, l'étendue de la plage de fonctionnement *evenly-spaced*.

de ces **STR** pour limiter la pollution liée à l'activité de la logique avoisinante. Un procédé similaire est utilisé pour dessiner les sources d'entropie. Par ailleurs, afin de limiter les phénomènes de diaphonie, un blindage connecté à la masse est ajouté au dessus de chacun des **RO**.

La [figure 6.13](#) montre la localisation de chacun de ces blocs. La banque d'entropie est séparée en deux parties positionnée en haut et en bas du circuit. Les **STR** sont placés sur le coté droit. L'espace laissé au centre est occupé par le reste de la logique qui est majoritairement constitué du bloc de perturbation.

6.7 Conclusion et perspectives

Dans ce chapitre nous avons présenté le **STR-PUF**, une nouvelle architecture de **PUF**. Elle exploite la résolution très fine d'un oscillateur asynchrone – le **STR** – pour extraire efficacement l'entropie du processus de fabrication et construire un secret unique à un circuit. Nous avons réalisé une campagne de mesure incluant 8 instances de ce **PUF** implémentées sur **FPGA** et configurées de 12 manières différentes. Nous avons montré que ce nouveau **PUF** peut facilement être dimensionné : en modifiant le nombre d'étages du **STR** pour optimiser la résolution et en modifiant le facteur de division pour optimiser le **SNR**. Il peut ainsi extraire des réponses de qualité dans un temps très court. Ainsi, le **STR-PUF** présente des caractéristiques d'unicité, de stabilité et d'imprédictibilité comparables à l'état de l'art, tout en surpassant les traditionnels **RO-PUF** quant au temps de réponse. Cette latence réduite permet d'améliorer la sécurité des **PUF** en évitant des comportements hasardeux de la part des utilisateurs mais aussi en réduisant la fenêtre d'exposition des sources d'entropie à des attaques par canaux cachés. Elle a aussi certainement un impact sur l'efficacité, faisant de ce **PUF** un bon candidat pour les applications à forte contraintes énergétiques. Nous avons cependant remarqué certaines limitations de ce **PUF** lorsqu'il est implémenté sur **FPGA**. De plus, les sources d'entropie et le **STR** sont négativement impactés par les ressources limitées de placement et de routage de ces circuits. Nous avons donc construit un circuit de test **ASIC** qui nous permettra d'avoir encore de meilleurs résultats.

De nombreuses pistes de recherche existent pour enrichir ces premiers travaux sur cette nouvelle architecture de **PUF**. Nous donnons ici une liste de quelques-unes de ces perspectives que nous jugeons particulièrement intéressantes :

- Nous avons vu que la qualité des réponses d'un **PUF** dépendait avant tout du ratio entre les disparités de fabrication et le bruit, le **SNR**. Or, au niveau conception, il n'y a que très peu de leviers pour optimiser l'une et l'autre de ces distributions. Un travail de concert avec les fondeurs pourrait apporter de nouvelles pistes d'optimisation, par exemple en dégradant volontairement le processus de fabrication lors de la création des sources d'entropie¹⁶.
- Lors de la conception des sources d'entropie, certaines constructions peuvent avoir un meilleur **SNR** que d'autre. L'utilisation de cellules avec différentes tensions de seuil ou différents drives de sortie peut légèrement modifier ce ratio. L'étude de l'évolution du **SNR** en fonction de la tension d'alimentation pourrait aussi révéler des solutions simples pour optimiser les source d'entropie. Le circuit **ASIC** que nous avons conçu devrait nous permettre de poursuivre nos travaux dans ce sens.
- Les modèles stochastiques proposés pourraient être étendus de manière à prendre en compte simultanément le bruit de quantification et le bruit non-déterministe (ther-

16. Cela viendrait certainement au prix de masques supplémentaires.

mique ou autre). Cela permettrait de déterminer *a priori* l’entropie et la stabilité, et permettrait de dimensionner au mieux le PUF.

- Si nous avons préféré un TDC à base de STR, d’autres méthodes de mesure du temps pourraient être utilisées. Le choix d’un TDC à base de Vernier, même s’il nécessite une étape de calibration et que le temps de mesure peut-être long, pourrait réduire la complexité du PUF et améliorer son efficacité.

Les travaux présentés dans ce chapitre ont été en partie publiés dans une conférence internationale et ont fait l’objet d’un dépôt de brevet :

- G. GIMENEZ, A. CHERKAOUI et L. FESQUET. « A Self-Timed Ring based PUF ». In : *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2020
- Laurent FESQUET, Abdelkarim CHERKAOUI et Grégoire GIMENEZ. « Procédé pour générer une donnée unique propre à un circuit intégré en silicium ». Brev. franç. 2000491. 17 jan. 2020

Conclusion

La réduction de la consommation des circuits intégrés et leur sécurisation sont deux problématiques récurrentes auxquelles font face industriels et académiques. L’essor du nomadisme numérique et du monde du tout connecté renforce d’autant plus ces enjeux. Ceux-ci sont pourtant très souvent en opposition, l’un ne pouvant être adressé sans détériorer l’autre. Par ailleurs les limites physiques et économiques viennent petit à petit à bout des différentes lois prédisant une amélioration continue de l’efficacité énergétique des circuits. Dans ce contexte, un regain d’intérêt est porté à d’anciennes solutions qui avaient été initialement écartées à cause de leur complexité. C’est le cas des circuits sans horloge qui sont restés marginaux malgré leurs qualités généralement reconnues et leur capacité à répondre à ces deux problématiques. Dans ce manuscrit nous avons présenté différents éléments en faveur des circuits asynchrones qui, mis bout à bout, pourront contribuer à une adoption plus large de ce type de circuits. Nous avons ainsi proposé un flot d’implémentation des circuits à données groupées (*Bundled-data*) (**BD**) se basant uniquement sur des outils dédiés aux circuits synchrones, favorisant l’intégration simultanée de ces deux approches (**partie I**). Nous avons aussi présenté deux exemples tirant parti des propriétés des circuits asynchrones et plus particulièrement du **STR** pour construire des primitives de sécurité efficaces (**partie II**). Dans la suite de ce chapitre, nous rappelons les différentes contributions apportées tout au long de ce manuscrit. Bien sûr, nous ne prétendons pas avoir couvert l’ensemble de la problématique. Nous complétons cette conclusion avec une liste, non-exhaustive, de travaux complémentaires et de perspectives de recherches pouvant à leur tour plaider pour une intégration progressive de la logique auto-séquencée pour les dispositifs sensibles à la consommation et à la sécurité.

Contributions

La première partie de ce manuscrit, rassemblant les trois premiers chapitres, s’est concentrée sur le développement d’un flot permettant d’implémenter des circuits asynchrones avec des outils utilisés et développés dans le paradigme synchrone.

Dans le **chapitre 1** nous avons présenté un bref état de l’art des circuits sans horloge. Nous avons mis en évidence l’étendue du monde asynchrone et la diversité des classes de circuits qui le compose, esquissant ainsi la complexité de ce type de logique. Nous avons aussi présenté les différents avantages communément attribués à ces circuits et révélé leur pertinence pour construire des dispositifs à fois sécurisés et basse consommation. Nous avons remarqué la perméabilité, souvent ignorée, de la frontière entre synchrone et asynchrone, et montré que ces différents circuits peuvent finalement être différenciés au travers des hypothèses temporelles qu’ils utilisent. Nous avons ainsi rappelé la continuité qui existe entre ces deux mondes et le formalisme commun, les **RTC**, permettant de les appréhender de manière similaire.

Nous avons analysé le paradoxe apparent entre les avantages et la faible adoption des circuits asynchrones dans le **chapitre 2**. Après avoir détaillé les différents blocages – économiques, sociologiques et techniques – pouvant expliquer cette situation, nous avons

identifié les circuits **BD**, comme un candidat idéal pour faciliter la transition entre ces deux mondes. De par leur proximité avec les circuits synchrones, ils répondent en grande partie aux différentes réticences. Nous avons ensuite discerné un des principaux freins restant : les flots de conception actuels décrivent de manière partielle les contraintes temporelles de ces circuits, ce qui a pour conséquence de limiter leur analyse. Ils ne garantissent alors le fonctionnement de ces circuits qu’au prix de marges grevant les gains normalement atteignables. Nous avons aussi montré les limites des outils existants qui sont développés dans la perspective synchrone et ne supportent pas nativement la description de la propagation d’évènements non-périodiques dans un contrôleur asynchrone. Nous avons proposé une classification de ces contraintes illustrant bien la complexité de cette tâche mais donnant aussi une meilleure compréhension des enjeux. Cette classification insiste à nouveau sur les similitudes entre les contraintes s’appliquant aux circuits synchrones et celles s’appliquant aux circuits asynchrones.

Pour décrire les **RTC** s’appliquant sur tous circuits **BD** nous avons proposé une nouvelle méthode – la méthode des séries d’horloges locales (*Local Clock Sets*) (**LCS**) – s’appuyant sur une sémantique synchrone : les horloges. Nous avons modélisé un circuit à données groupées comme un assemblage de différents oscillateurs et nous nous sommes attachés à décrire ces oscillateurs à l’aide de commandes standards décrivant des signaux périodiques. Nous avons ainsi révélé quelques subtilités des outils de **STA** permettant d’utiliser une succession d’horloges pour décrire la propagation des évènements dans la partie de contrôle d’un circuit **BD**. Nous avons ensuite étendu cette méthode pour identifier toutes les **RTC**, leurs **POD** et **POC**, qu’un circuit doit respecter en se basant sur un modèle formel tel que le **STG**. Nous avons ainsi fait le lien entre ce modèle **STG** et les **LCS** et proposé de construire des fichiers de règles permettant de générer automatiquement les fichiers de contraintes temporelles (**SDC**). Ces fichiers peuvent être utilisés de manière standard dans les outils de **CAO** habituels. Nous avons alors construit ces règles **LCS** pour différents schémas de contrôleur **BD**, démontrant la versatilité de notre approche. Enfin, nous avons présenté les conséquences de l’utilisation de ces **LCS** sur les différentes étapes du flot d’implémentation physique, montrant aussi l’adéquation de notre méthode avec les outils commerciaux. Ceci a été confirmé en comparant l’impact sur les temps d’exécution et l’empreinte mémoire de notre nouvelle méthodologie par rapport à l’approche utilisée habituellement (approche min/max) et vis-à-vis d’un flot synchrone classique.

La deuxième partie de ce manuscrit s’est focalisée sur deux primitives de sécurité, un générateur de nombres véritablement aléatoires (*True Random Number Generator*) (**TRNG**) et une fonction physique non-clonable (*Physical Unclonable Function*) (**PUF**), qui peuvent être construites en utilisant les propriétés d’un circuit asynchrone, le **STR**.

Dans le **chapitre 4** nous donnons un état de l’art de la génération d’aléa dans les circuits logiques. L’aléa est primordial dans l’optique de la sécurisation des informations. Lorsqu’il est dynamique, il est nécessaire à la plupart des protocoles de chiffrement et à la construction de contremesures visant à masquer le fonctionnement du circuit. Lorsqu’il est statique, il peut être utilisé comme un secret permettant d’identifier de manière unique un circuit ou de stocker des clés de chiffrement de manière inviolable. Nous avons donc présenté le concept de **TRNG**, dispositif ayant pour but d’extraire des nombres véritablement aléatoires à partir d’une source de bruit non-déterministe. Nous avons rappelé que le bruit thermique est la seule source d’entropie réellement fiable et que, s’il a de nombreuses conséquences observables, son influence sur le *jitter* d’un oscillateur est le phénomène majoritairement utilisé pour le caractériser. Nous avons aussi présenté les enjeux de construction d’un bon **TRNG**. Il doit être à la fois imprédictible et non-manipulable. La première propriété doit être à la fois validée *a priori* à l’aide d’un modèle stochastique et *a posteriori* à l’aide d’une suite de tests statistiques détectant de potentiels biais dans les séquences de bit générées. La deuxième propriété doit, quant à elle, être assurée par

l'utilisation de tests en ligne et de blocs monitorant continuellement le bon fonctionnement du générateur. Nous avons alors donné deux exemples d'architecture : le **ERO-TRNG** et le **STR-TRNG**. Nous avons vu que ce dernier est basé, non pas sur l'accumulation de *jitter*, mais sur l'échantillonnage de chaque réalisation de celui-ci – principe qui rend ce générateur particulièrement efficace du point de vue énergétique. L'une des particularités de ce générateur est aussi l'utilisation d'un oscillateur en anneau asynchrone : le STR. Dans ce chapitre nous avons donc présenté au préalable le fonctionnement de ce type bien particulier d'oscillateur en insistant sur une de ses caractéristiques principales : sa résolution de phase $\Delta\varphi$ configurable et pouvant être plus petite que le temps de propagation minimum au travers d'une porte logique. Pour finir, ce chapitre a présenté le concept de fonction physique non-clonable (*Physical Unclonable Function*) (**PUF**), permettant d'extraire un secret intrinsèque à un circuit en mesurant les variations du processus de fabrication. Nous avons rappelé les différentes métriques utilisées habituellement pour analyser la qualité de ces primitives, à savoir : unicité, stabilité et imprédictibilité. Nous avons ensuite présenté les trois grandes catégories de **PUF** intégrables dans un flot de conception numérique. Ils caractérisent les disparités de fabrication en analysant soit des temps de propagation, des fréquences d'oscillation ou des états d'initialisation de points mémoires.

Une étude détaillée de la « non-manipulabilité » du **STR-TRNG** a été présentée dans le **chapitre 5**. Nous avons proposé un modèle de menace identifiant les principales vulnérabilités de ce générateur et les comportements anormaux pouvant être forcés par un agresseur souhaitant réduire l'entropie ou directement prendre le contrôle de la sortie du générateur. Nous avons présenté deux attaques visant à générer l'un ou l'autre de ces fonctionnements dégradés. La première cherche à modifier le nombre d'évènements circulant dans l'anneau alors que la seconde injecte un délai supplémentaire dans un des étages pour ralentir l'anneau voire générer un mode *bottleneck*. Nous avons ensuite proposé diverses contremesures permettant de se prémunir de ces attaques. La pertinence de ces attaques et des contremesures a été validée par des simulations au niveau transistor et à l'aide d'un modèle comportemental du STR incluant les effets *Charlie* et *drafting*. Nous avons aussi présenté le résultat de la validation sur silicium de l'attaque par suppression de jetons et du moniteur permettant de s'en protéger. Ce même moniteur peut être utilisé pour protéger un circuit **BD** (ou une partie de circuit) en vérifiant de manière continue ou par intermittence que le nombre de jetons contenu dans le bloc de contrôle est cohérent avec celui attendu. Pour finir nous avons proposé un nouveau modèle stochastique du **STR-TRNG**. Celui-ci permet de réduire le pessimisme introduit par le modèle utilisé précédemment [Che14] et d'ainsi améliorer l'efficacité énergétique de ce générateur. Ce nouveau modèle peut être utilisé pour dériver une nouvelle architecture où le bruit n'est plus apporté par le STR mais directement par l'horloge d'échantillonnage. De cette manière nous avons ajouté un degré de configuration supplémentaire en décorrélant le réglage de la quantité de bruit σ_n et de la résolution de phase $\Delta\varphi$.

Pour finir, le **chapitre 6** a proposé un nouveau type de **PUF** s'appuyant à son tour sur la très petite résolution de phase d'un STR. Dans cette architecture, l'anneau asynchrone est utilisé pour construire un convertisseur temps-numérique (*Time-to-Digital Converter*) (**TDC**) avec une très petite résolution. Ce convertisseur permet de mesurer les temps de propagation de manière très précise et d'ainsi distinguer les variations de fabrication garantissant les propriétés d'unicité et de stabilité. De plus, l'une des motivations pour choisir cette architecture était de réduire le temps de mesure pour diminuer la latence de réponse des **RO-PUF** et ainsi limiter le temps d'exposition de la primitive à d'éventuelles attaques. Dans un premier temps nous avons proposé une modélisation stochastique simple permettant de comparer les deux méthodes d'extraction d'entropie dans un **PUF** : le principe de mesure de la fréquence et celui de mesure de la période des mêmes **RO**. Pour ces deux approches, nous avons montré le lien entre la résolution de mesure et l'entropie

extraite en considérant, de manière simpliste, que l'erreur de mesure est l'unique source de dégradation de l'entropie. Nous avons aussi justifié comment l'utilisation d'un diviseur en sortie des sources d'entropie (les RO) permet théoriquement d'augmenter le SNR et d'ainsi améliorer la stabilité et l'unicité des réponses du STR-PUF. Ces hypothèses ont été validées avec une implémentation sur FPGA de notre PUF. Les résultats obtenus ont confirmé la faisabilité de cette nouvelle architecture, donnant des chiffres de stabilité et d'unicité décentés tout en divisant la latence de réponse par près de 200. Afin de confirmer ces résultats prometteurs, nous avons conçu un circuit de test pour étendre l'étude à un plus grand nombre de pièces tout en analysant l'impact des variations de la température et de la tension d'alimentation. Ce circuit permettra aussi d'étudier le comportement du PUF en présence de blocs perturbateurs à proximité du PUF.

Travaux complémentaires et perspectives

La description LCS permet de donner aux outils d'implémentation une connaissance complète des RTC s'appliquant à un circuit à données groupées. En utilisant une telle description, les nombreuses marges conservatrices habituellement utilisées ne sont plus nécessaires. Nous pouvons ainsi prétendre construire des circuits asynchrones plus performants et moins gourmands en énergie. Cette connaissance des temps relatifs permet aussi d'exploiter toute l'étendue des fonctionnalités et des capacités d'optimisation des outils commerciaux. La possibilité d'utiliser les algorithmes d'optimisation concurrente de l'arbre d'horloge et du chemin de données (*Concurrent Clock and Data Optimization*) (CCDO) pour insérer automatiquement les délais *matchés* est certainement une des pistes d'amélioration des plus intéressantes pour la méthode LCS. La création des fichiers de règles pour de nouveaux protocoles est aussi une perspective de recherche pertinente. Mais plus encore, en uniformisant ces fichiers de règles, il est envisageable de facilement mixer les différents protocoles. Ainsi, le contrôleur le plus adapté à chaque partie du circuit pourrait être choisi automatiquement lors de la synthèse logique. Pour compléter ces travaux, il nous semble important de développer une méthode de vérification formelle permettant de contrevérifier la cohérence et la complétude des fichiers de contraintes temporelles générés par le flot LCS. De manière générale, l'industrie de la microélectronique s'appuie toujours sur des outils de vérification pour s'assurer de l'équivalence fonctionnelle tout au long du cycle de développement d'un circuit – les outils de preuve d'équivalence comme Formality ou Conformal, viennent ainsi s'assurer que la fonctionnalité du circuit n'a pas été affectée par les étapes de synthèse et de placement routage, les outils de *Layout Versus Schematic* (LVS) s'assurent que le dessin du circuit (GDSII) est bien équivalent avec la *netlist* précédemment validée, etc. Vérifier que le fichier de contraintes SDC et la *netlist* implémentent correctement le protocole décrit dans un langage formel (STG ou autres) est un autre sujet important à traiter dans l'objectif de convaincre l'industrie d'adopter plus largement les circuits asynchrones. A ce titre, étudier la compatibilité du flot LCS avec des approches de désynchronisation telles que [ZZC18] ou de synthèse de haut niveau [Sim17] est aussi une piste de recherche très intéressante. Mais pour convaincre, la communauté asynchrone doit avant tout identifier les applications ou partie d'un circuit pour lesquels la logique auto-séquentée est pleinement valorisable. Il nous semble que la triple contrainte de très faible consommation, de réveil rapide et de sécurité des zones d'un circuit alimentées en continu est une l'une de ces applications qui bénéficierait grandement de l'asynchrone.

Le STR-TRNG a montré toutes les propriétés d'un vrai générateur d'aléa dynamique. La disponibilité d'un modèle stochastique garantit son imprédictibilité alors que sa non-manipulabilité peut être assurée à l'aide de moniteurs et de tests en ligne. Par ailleurs les tests statistiques exécutés sur les séquences brutes de ce générateur confortent le modèle et ne détectent que peu, voire pas de biais. Pour autant, la dépendance qui peut exister entre

les bits générés successivement mériterait d'être étudiée plus en détails. Les simulations faites par CHERKAoui [Che14] soutiennent l'hypothèse d'indépendance. La création d'un modèle basé sur une chaîne de Markov et intégrant les effets *Charlie* et *drafting* permettrait de les conforter davantage. Par ailleurs, les performances de ce générateur peuvent sûrement être encore optimisées. Nous avons vu que celles-ci dépendaient essentiellement du rapport $\sigma/\Delta\varphi$. Il serait intéressant d'analyser le comportement de ce ratio en fonction de différents paramètres : options technologiques, tension de seuil et largeurs de canal des transistors, architectures des cellules de Muller, tension d'alimentation. Cette étude pourrait être complétée en tentant de corréliser ses résultats avec ceux obtenus en simulation et en utilisant les paramètres de bruit inclus dans les modèles de transistor. Bien sûr cette étude devra être complétée par une caractérisation précise du bruit thermique, avec du matériel embarqué directement dans le circuit. De nombreux travaux s'attachent déjà à ce sujet [Val+10; LB15; Yan+17]. Finalement, l'une des hypothèses sur laquelle est basé le STR-TRNG est le mode de fonctionnement *evenly-spaced* du STR. Si ce mode d'oscillation est facilement observable en simulation, il n'y a, à notre connaissance, pas de travaux ayant pu observer directement la résolution de phase. Celle-ci est généralement déduite suite à la caractérisation du STR et en utilisant l'équation $\Delta\varphi = T_{STR}/2L$. La confirmation du fonctionnement en mode *evenly-spaced* est, quant à elle, déduite de l'observation d'une seule phase, en analysant son rapport cyclique et en observant la variation de sa fréquence en fonction du nombre d'évènements. Une observation plus directe de $\Delta\varphi$ permettrait de valider les différentes hypothèses et d'étudier son comportement lorsque l'anneau est perturbé.

Des mesures sur un nombre de circuits plus important sont primordiales pour confirmer les premiers résultats obtenus sur FPGA pour le STR-PUF. Mais il nous semble aussi intéressant d'approfondir le modèle que nous avons présenté. A l'instar de ce que propose SCHAUB *et al.* [Sch+18], il serait intéressant d'étendre notre modèle pour intégrer le bruit présent dans le circuit (quel qu'il soit) et pour ainsi relier variabilité de fabrication, résolution de mesure et bruit aux métriques usuelles d'un PUF : stabilité, unicité, mais aussi entropie et SNR. De manière plus générale, le travail sur les sources d'entropie, dans l'idéal en étroite collaboration avec les fondeurs, est un axe de recherche qui permettrait d'optimiser n'importe quel PUF. Outre le choix des structures les plus sensibles à la disparité de fabrication, la possibilité d'accentuer cette variation localement au niveau des sources d'entropie permettrait de construire des PUF plus efficaces et plus stables. Plus spécifiquement à notre architecture de PUF, nous pouvons aussi imaginer d'utiliser des sources qui ne soient pas oscillantes. Ainsi des générateurs d'impulsion plus ou moins complexes peuvent être imaginés. Ils auraient l'avantage d'être moins sensibles à des attaques par canaux cachés, même s'ils occuperont certainement plus de surface silicium.

Les circuits asynchrones sont réputés plus sécurisés que leurs homologues synchrones. A première vue, il est vrai que l'absence d'horloge réduit la surface d'attaque utilisable par un agresseur. Mais si les circuits QDI ont été plébiscités pour leur résistance intrinsèque aux attaques par canaux cachés (l'utilisation de logique double rails permet de limiter les fuites d'information), aucune étude n'existe, à notre connaissance, à ce sujet pour les circuits BD. Suite à quelques essais que nous avons pu mener, nos premiers résultats sont plutôt défavorables. Ils montrent que si le problème de synchronisation est ignoré, les circuits BD sont bien moins résistants que les circuits synchrones. En effet dans les circuits avec horloge, la commutation des registres du circuit se fait de manière simultanée. Ce phénomène tend à masquer l'information utile qu'un agresseur cherche à retrouver, rendant une analyse par *Correlation Power Analysis* (CPA) plus difficile (plus de mesures sont nécessaires pour retrouver l'information). A l'inverse, dans les circuits BD, chaque opération est faite plus ou moins indépendamment des autres, et si un agresseur est capable de synchroniser les traces de consommation avec l'opération de l'étage qu'il souhaite attaquer, celles-ci sont beaucoup

moins bruitées et une attaque par CPA sera plus aisée (un facteur 100 est apparu dans nos essais). Pour pouvoir comparer ces circuits d'un point de vue sécurité il faut donc être capable d'évaluer la complexité de cette étape de synchronisation des traces. Des travaux complémentaires sont donc nécessaires pour s'assurer que les circuits BD n'introduisent pas de failles de sécurité. Si cela était avéré, une solution serait l'utilisation de logique double rail avec un schéma BD. Cependant, cette solution a un coût non négligeable et peut par ailleurs aussi être utilisée avec les circuits synchrones. Par contre, les circuits BD étant plus robustes aux variations de tension, une contremesure basée sur l'ajout de bruit aléatoire (ou mieux, corrélé avec les données traitées) sur l'alimentation serait certainement plus facile à mettre en œuvre et potentiellement plus efficace.

Bibliographie

- [Sha48] Claude E SHANNON. « A mathematical theory of communication ». In : *The Bell system technical journal* 27.3 (1948), p. 379-423.
- [Huf54] D.A. HUFFMAN. « The synthesis of sequential switching circuits ». In : *Journal of the Franklin Institute* 257.3 (1954), p. 161-190. ISSN : 0016-0032. DOI : [https://doi.org/10.1016/0016-0032\(54\)90574-8](https://doi.org/10.1016/0016-0032(54)90574-8). URL : <http://www.sciencedirect.com/science/article/pii/0016003254905748>.
- [MB59] D. E. MULLER et W. S. BARTKY. « A theory of asynchronous circuits ». In : *Digital Computer Laboratory* 78 (1959).
- [Moo65] G. E. MOORE. « Cramming more components onto integrated circuits. » In : *Electronics* 38.8 (1965), 114 ff.
- [Pet66] Carl Adam PETRI. « Communication with automata ». In : (1966).
- [Den+74] R. H. DENNARD *et al.* « Design of ion-implanted MOSFET's with very small physical dimensions ». In : *IEEE Journal of Solid-State Circuits* 9.5 (1974), p. 256-268.
- [Hol82] HOLLAAR. « Direct Implementation of Asynchronous Control Units ». In : *IEEE Transactions on Computers* C-31.12 (1982), p. 1133-1141.
- [LRS83] Charles E. LEISERSON, Flavio M. ROSE et James B. SAXE. « Optimizing Synchronous Circuitry by Retiming (Preliminary Version) ». In : *Third Caltech Conference on Very Large Scale Integration*. Sous la dir. de Randal BRYANT. Berlin, Heidelberg : Springer Berlin Heidelberg, 1983, p. 87-116. ISBN : 978-3-642-95432-0.
- [FMC85] R. C. FAIRFIELD, R. L. MORTENSON et K. B. COULTHART. « An LSI Random Number Generator (RNG) ». In : *Advances in Cryptology*. Sous la dir. de George Robert BLAKLEY et David CHAUM. Berlin, Heidelberg : Springer Berlin Heidelberg, 1985, p. 203-230. ISBN : 978-3-540-39568-3.
- [Udd86] Jan Tijmen UDDING. « A formal model for defining and classifying delay-insensitive circuits and systems ». In : *Distributed Computing* 1.4 (1986), p. 197-204.
- [Chu87] Tam-Anh CHU. « Synthesis of self-timed VLSI circuits from graph-theoretic specifications ». Thèse de doct. Massachusetts Institute of Technology, 1987.
- [Mil89] R. MILNER. *Communication and Concurrency*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1989. ISBN : 0-13-115007-3.
- [PDW89] M. J. M. PELGROM, A. C. J. DUINMAIJER et A. P. G. WELBERS. « Matching properties of MOS transistors ». In : *IEEE Journal of Solid-State Circuits* 24.5 (oct. 1989), p. 1433-1439. ISSN : 1558-173X. DOI : [10.1109/JSSC.1989.572629](https://doi.org/10.1109/JSSC.1989.572629).

- [Sut89] I. E. SUTHERLAND. « Micropipelines ». In : *Commun. ACM* 32.6 (juin 1989), p. 720-738. ISSN : 0001-0782. DOI : [10.1145/63526.63532](https://doi.org/10.1145/63526.63532). URL : <https://doi.org/10.1145/63526.63532>.
- [Mar90] Alain J. MARTIN. « The limitations to delay-insensitivity in asynchronous circuits ». In : *Beauty is our business*. Springer, 1990, p. 302-311.
- [MBM91] T. H. -. MENG, R. W. BRODERSEN et D. G. MESSERSCHMITT. « Asynchronous design for programmable digital signal processors ». In : *IEEE Transactions on Signal Processing* 39.4 (1991), p. 939-952.
- [Fur93] S.B. FURBER. « Breaking step : the return of asynchronous logic ». In : *IEE Review* 39.4 (1993), p. 159-162.
- [F94] *FIPS 140-1 (archived) – Security Requirements for Cryptographic Modules*. Rapp. tech. NIST, 1994.
- [Kis+94] Michael KISHINEVSKY, Alex KONDRATYEV, Alexander TAUBIN et Victor VARSHAVSKY. *Concurrent hardware : the theory and practice of self-timed design*. John Wiley & Sons, Inc., 1994.
- [MM95] Rajit MANOHAR et Alain J. MARTIN. *Quasi-Delay-Insensitive Circuits Are Turing-Complete*. Rapp. tech. USA, 1995.
- [CS96] V. CHANDRAMOULI et K. A. SAKALLAH. « Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time ». In : *33rd Design Automation Conference Proceedings, 1996*. 1996, p. 617-622.
- [FD96] S. B. FURBER et P. DAY. « Four-phase micropipeline latch control circuits ». In : *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 4.2 (1996), p. 247-253.
- [YBA96] K. Y. YUN, P. A. BEEREL et J. ARCEO. « High-performance asynchronous pipeline circuits ». In : *Proceedings Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 1996, p. 17-28.
- [SKK97] T. SAKURAI, H. KAWAGUCHI et T. KURODA. « Low-power CMOS design through V/sub TH/ control and low-swing circuits ». In : *Proceedings of 1997 International Symposium on Low Power Electronics and Design*. 1997, p. 1-6.
- [CYD98] S. CHAKRABORTY, K. Y. YUN et D. L. DILL. « Practical timing analysis of asynchronous circuits using time separation of events ». In : *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No.98CH36143)*. 1998, p. 455-458.
- [Tiw+98] V. TIWARI *et al.* « Reducing power in high-performance microprocessors ». In : *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*. 1998, p. 732-737.
- [SGR99] K. STEVENS, R. GINOSAR et S. ROTEM. « Relative timing ». In : *Proceedings. Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 1999, p. 208-218.
- [Ble00] Daniel BLEICHENBACHER. « On the generation of one-time keys in DL signature schemes ». In : *Presentation at IEEE P1363 working group meeting*. 2000, p. 81.
- [DSH00] P. DUDEK, S. SZCZEPANSKI et J. V. HATFIELD. « A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line ». In : *IEEE Journal of Solid-State Circuits* 35.2 (2000), p. 240-247.

-
- [LDT00] K. LOFSTROM, W. R. DAASCH et D. TAYLOR. « IC identification circuit using device mismatch ». In : *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*. Fév. 2000, p. 372-373. DOI : [10.1109/ISSCC.2000.839821](https://doi.org/10.1109/ISSCC.2000.839821).
- [Ren00] Marc RENAUDIN. *Etat de l'art sur la conception des circuits asynchrones : perspectives pour l'intégration des systèmes complexes*. Rapp. tech. TIMA-RR-02/12-02-FR. TIMA Laboratory, jan. 2000. URL : http://tima.univ-grenoble-alpes.fr/publications/files/rr/eac_168.pdf.
- [HS01] Nick A HOWGRAVE-GRAHAM et Nigel P. SMART. « Lattice attacks on digital signature schemes ». In : *Designs, Codes and Cryptography* 23.3 (2001), p. 283-290.
- [KS01] Wolfgang KILLMANN et Werner SCHINDLER. « AIS 31 : Functionality classes and evaluation methodology for true (physical) random number generators, version 3.1 ». In : *Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn* (2001).
- [Nor01] K. NORMOYLE. « Where are the async millionaires? » In : *2001 Seventh International Symposium on Asynchronous Circuits and Systems*. IEEE. 2001, p. 138-.
- [SN01] M. SINGH et S. M. NOWICK. « MOUSETRAP : ultra-high-speed transition-signaling asynchronous pipelines ». In : *Proceedings 2001 IEEE International Conference on Computer Design : VLSI in Computers and Processors. ICCD 2001*. 2001, p. 9-17.
- [Tri01] Claire TRISTRAM. *It's Time for Clockless Chips*. 2001. URL : <https://www.technologyreview.com/2001/10/01/41224/its-time-for-clockless-chips/> (visité le 31/08/2020).
- [WG01] Anthony WINSTANLEY et Mark GREENSTREET. « Temporal Properties of Self-Timed Rings ». In : *Correct Hardware Design and Verification Methods*. Sous la dir. de Tiziana MARGARIA et Tom MELHAM. Berlin, Heidelberg : Springer Berlin Heidelberg, 2001, p. 140-154. ISBN : 978-3-540-44798-6.
- [Gas+02] Blaise GASSEND, Dwaine CLARKE, Marten van DIJK et Srinivas DEVADAS. « Silicon Physical Random Functions ». In : *Proceedings of the 9th ACM Conference on Computer and Communications Security. CCS '02*. Washington, DC, USA : ACM, 2002, p. 148-160. ISBN : 1-58113-612-9. DOI : [10.1145/586110.586132](https://doi.org/10.1145/586110.586132). URL : <http://doi.acm.org/10.1145/586110.586132>.
- [KL02] A. KONDRATYEV et K. LWIN. « Design of asynchronous circuits by synchronous CAD tools ». In : *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*. 2002, p. 411-414.
- [MHR02] Jesper MØLLER, Henrik HULGAARD et Henrik REIF ANDERSEN. « Timed Verification of Asynchronous Circuits ». In : *Concurrency and Hardware Design : Advances in Petri Nets*. Sous la dir. de Jordi CORTADELLA, Alex YAKOVLEV et Grzegorz ROZENBERG. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002, p. 274-312. ISBN : 978-3-540-36190-9. DOI : [10.1007/3-540-36190-1_8](https://doi.org/10.1007/3-540-36190-1_8). URL : https://doi.org/10.1007/3-540-36190-1_8.
- [NS02] Phong Q NGUYEN et Igor E SHPARLINSKI. « The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. » In : *Journal of Cryptology* 15.3 (2002).
-

- [SK02] Werner SCHINDLER et Wolfgang KILLMANN. « Evaluation criteria for true (physical) random number generators used in cryptographic applications ». In : *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2002, p. 431-449.
- [WGG02] A. J. WINSTANLEY, A. GARIVIER et M. R. GREENSTREET. « An event spacing experiment ». In : *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*. 2002, p. 47-56.
- [Bor03] Shekhar BORKAR. *Does asynchronous logic design really have a future ?* 2003. URL : <https://www.eetimes.com/does-asynchronous-logic-design-really-have-a-future/> (visité le 31/08/2020).
- [FM04] S. FAIRBANKS et S. MOORE. « Analog micropipeline rings for high precision timing ». In : *10th International Symposium on Asynchronous Circuits and Systems, 2004. Proceedings*. 2004, p. 41-50.
- [Lee+04] J. W. LEE *et al.* « A technique to build a secret key in integrated circuits for identification and authentication applications ». In : *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*. Juin 2004, p. 176-179. DOI : [10.1109/VLSIC.2004.1346548](https://doi.org/10.1109/VLSIC.2004.1346548).
- [Dai+05] DAIHYUN LIM *et al.* « Extracting secret keys from integrated circuits ». In : *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.10 (oct. 2005), p. 1200-1205. ISSN : 1557-9999. DOI : [10.1109/TVLSI.2005.859470](https://doi.org/10.1109/TVLSI.2005.859470).
- [MHO05] Takahito MIYAZAKI, Masanori HASHIMOTO et Hidetoshi ONODERA. « A Performance prediction of clock generation PLLs : A ring oscillator based PLL and an LC Oscillator based PLL ». In : *IEICE transactions on electronics* 88.3 (2005), p. 437-444.
- [Rév05] Pál RÉVÉSZ. *Random Walk in Random and Non-Random Environments*. 2nd. WORLD SCIENTIFIC, 2005. DOI : [10.1142/5847](https://doi.org/10.1142/5847). eprint : <https://www.worldscientific.com/doi/pdf/10.1142/5847>. URL : <https://www.worldscientific.com/doi/abs/10.1142/5847>.
- [Cor+06] J. CORTADELLA, A. KONDRATYEV, L. LAVAGNO et C. P. SOTIRIOU. « Desynchronization : Synthesis of Asynchronous Circuits From Synchronous Specifications ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.10 (2006), p. 1904-1921.
- [FB06] M. FERRETTI et P. A. BEEREL. « High performance asynchronous design using single-track full-buffer standard cells ». In : *IEEE Journal of Solid-State Circuits* 41.6 (2006), p. 1444-1454.
- [RB06] V. RAMAKRISHNAN et P. T. BALSARA. « A wide-range, high-resolution, compact, CMOS time to digital converter ». In : *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*. 2006, 6 ff.
- [And+07] N. ANDRIKOS, L. LAVAGNO, D. PANDINI et C. P. SOTIRIOU. « A Fully-Automated Desynchronization Flow for Synchronous Circuits ». In : *2007 44th ACM/IEEE Design Automation Conference*. 2007, p. 982-985.
- [Gua+07] Jorge GUAJARDO, Sandeep S. KUMAR, Geert-Jan SCHRIJEN et Pim TUYLS. « FPGA Intrinsic PUFs and Their Use for IP Protection ». In : *Cryptographic Hardware and Embedded Systems - CHES 2007*. Sous la dir. de Pascal PAILLIER et Ingrid VERBAUWHEDE. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 63-80. ISBN : 978-3-540-74735-2.

-
- [Mon07] Y. MONNET. « Etude et modélisation de circuits résistants aux attaques non intrusives par injection de fautes ». Theses. Institut National Polytechnique de Grenoble - INPG, avr. 2007. URL : <https://tel.archives-ouvertes.fr/tel-00163817>.
- [SMS07] B. SUNAR, W. J. MARTIN et D. R. STINSON. « A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks ». In : *IEEE Transactions on Computers* 56.1 (2007), p. 109-119.
- [Bös+08] Christoph BÖSCH, Jorge GUAJARDO, Ahmad-Reza SADEGHI, Jamshid SHOKROLLAHI et Pim TUYLS. « Efficient Helper Data Key Extractor on FPGAs ». In : *Cryptographic Hardware and Embedded Systems – CHES 2008*. Sous la dir. d'Elisabeth OSWALD et Pankaj ROHATGI. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 181-197. ISBN : 978-3-540-85053-3.
- [DL08] Aurélie DELEMARLE et Philippe LARÉDO. In : Oxford University Press, 2008. Chap. Breakthrough Innovation and the Shaping of New Markets : The Role of Communities of Practice.
- [Fis+08] V. FISCHER, F. BERNARD, N. BOCHARD et M. VARCHOLA. « Enhancing security of ring oscillator-based trng implemented in FPGA ». In : *2008 International Conference on Field Programmable Logic and Applications*. 2008, p. 245-250.
- [Ham+08] J. HAMON, L. FESQUET, B. MISCOPEIN et M. RENAUDIN. « High-Level Time-Accurate Model for the Design of Self-Timed Ring Oscillators ». In : *2008 14th IEEE International Symposium on Asynchronous Circuits and Systems*. 2008, p. 29-38.
- [MY08] C. MANNAKARA et T. YONEDA. « Asynchronous pipeline controller based on early acknowledgement protocol ». In : *2008 8th International Conference on Application of Concurrency to System Design*. 2008, p. 118-127.
- [Pee08] Ad PEETERS. « Asynchronous Circuit Technology is on the Market ». In : *2008 14th IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE. 2008, p. xiv-xiv.
- [Val+08] B. VALTCHANOV, A. AUBERT, F. BERNARD et V. FISCHER. « Modeling and observing the jitter in ring oscillators implemented in FPGAs ». In : *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. 2008, p. 1-6.
- [WT08] K. WOLD et C. H. TAN. « Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings ». In : *2008 International Conference on Reconfigurable Computing and FPGAs*. 2008, p. 385-390.
- [Dum+09] C. L. G. DUMAGUING, A. K. K. KHAN, M. A. D. PARUNGAO, A. B. ALVAREZ et J. A. P. REYES. « An Asynchronous Implementation of a 32-bit DLX Microprocessor ». In : *2009 International Conference on Information and Multimedia Technology*. 2009, p. 503-506.
- [Fai09] Scott FAIRBANKS. *High precision timing using self-timed circuits*. Rapp. tech. UCAM-CL-TR-738. University of Cambridge, Computer Laboratory, jan. 2009. URL : <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-738.pdf>.
- [Ham09] J. HAMON. « Oscillateurs et architectures asynchrones pour le traitement des signaux radio impulsionnelle UWB ». ISBN : 978-2-84813-138-2. Theses. Institut National Polytechnique de Grenoble - INPG, oct. 2009. URL : <https://tel.archives-ouvertes.fr/tel-00481841>.
-

- [MTV09] R. MAES, P. TUYLS et I. VERBAUWHEDE. « A soft decision helper data algorithm for SRAM PUFs ». In : *2009 IEEE International Symposium on Information Theory*. 2009, p. 2101-2105.
- [SXV09] K. S. STEVENS, Y. XU et V. VIJ. « Characterization of Asynchronous Templates for Integration into Clocked CAD Flows ». In : *2009 15th IEEE Symposium on Asynchronous Circuits and Systems*. 2009, p. 151-161.
- [Pee+10] A. PEETERS, F. t. BEEST, M. d. WIT et W. MALLON. « Click Elements : An Implementation Style for Data-Driven Compilation ». In : *2010 IEEE Symposium on Asynchronous Circuits and Systems*. 2010, p. 3-14.
- [Ruk+10] A RUKHIN *et al.* *SP 800-22 Rev. 1a. – A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Rapp. tech. National Institute of Standards and Technology, 2010.
- [Val+10] B. VALTCHANOV, V. FISCHER, A. AUBERT et F. BERNARD. « Characterization of randomness sources in ring oscillator-based true random number generators in FPGAs ». In : *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. 2010, p. 48-53.
- [VD10] Michal VARCHOLA et Milos DRUTAROVSKY. « New high entropy element for FPGA based true random number generators ». In : *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2010, p. 351-365.
- [YD10] M. YU et S. DEVADAS. « Secure and robust error correction for physical unclonable functions ». In : *IEEE Design Test of Computers* 27.1 (2010), p. 48-65.
- [Bau+11] Mathieu BAUDET, David LUBICZ, Julien MICOLOD et André TASSIAUX. « On the Security of Oscillator-Based Random Number Generators ». In : *Journal of Cryptology* 24.2 (2011), p. 398-425. DOI : [10.1007/s00145-010-9089-3](https://doi.org/10.1007/s00145-010-9089-3). URL : <https://hal.archives-ouvertes.fr/hal-00584405>.
- [Boc+11] Nathalie BOCHARD, Florent BERNARD, Viktor FISCHER et Boyan VALTCHANOV. « True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators ». In : *International Journal of Reconfigurable Computing* 2010 (mar. 2011). Sous la dir. de Lionel TORRES, p. 879281. ISSN : 1687-7195. DOI : [10.1155/2010/879281](https://doi.org/10.1155/2010/879281). URL : <https://doi.org/10.1155/2010/879281>.
- [Eli11] Oussama EL ISSATI. « Oscillateurs asynchrones en anneau : de la théorie à la pratique ». Theses. Université de Grenoble, sept. 2011. URL : <https://tel.archives-ouvertes.fr/tel-00683174>.
- [Koo+11] J. KOOMEY, S. BERARD, M. SANCHEZ et H. WONG. « Implications of Historical Trends in the Electrical Efficiency of Computing ». In : *IEEE Annals of the History of Computing* 33.3 (2011), p. 46-54.
- [Wan+11] E. WANDERLEY *et al.* « Security FPGA Analysis ». In : *Security Trends for FPGAs : From Secured to Secure Reconfigurable Systems*. Sous la dir. de Benoit BADRIGNANS, Jean Luc DANGER, Viktor FISCHER, Guy GOGNIAT et Lionel TORRES. Dordrecht : Springer Netherlands, 2011, p. 7-46. ISBN : 978-94-007-1338-3. DOI : [10.1007/978-94-007-1338-3_2](https://doi.org/10.1007/978-94-007-1338-3_2). URL : https://doi.org/10.1007/978-94-007-1338-3_2.

-
- [Bay+12] Pierre BAYON *et al.* « Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator ». In : *COSADE : Constructive Side-Channel Analysis and Secure Design*. Sous la dir. de W. SCHINDLER et S. A. HUSS. T. LNCS. Constructive Side-Channel Analysis and Secure Design 7275. Darmstadt, Germany, mai 2012, p. 151-166. DOI : [10.1007/978-3-642-29912-4_12](https://doi.org/10.1007/978-3-642-29912-4_12). URL : <https://hal-ujm.archives-ouvertes.fr/ujm-00699618>.
- [Che+12] Abdelkarim CHERKAoui, Viktor FISCHER, Alain AUBERT et Laurent FESQUET. « Comparison of Self-Timed Ring and Inverter Ring Oscillators as Entropy Sources in FPGAs ». In : *Design Automation and Test in Europe (DATE 2012)*. 11.4_2. Dresden, Germany, mar. 2012, p. 1-6. URL : <https://hal-ujm.archives-ouvertes.fr/ujm-00667639>.
- [Deh+12] A. DEHBAoui, J. DUTERTRE, B. ROBISSON et A. TRIA. « Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES ». In : *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2012, p. 7-15.
- [Kat+12] Stefan KATZENBEISSER *et al.* « PUFs : Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon ». In : *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems. CHES'12*. Leuven, Belgium : Springer-Verlag, 2012, p. 283-301. ISBN : 978-3-642-33026-1. DOI : [10.1007/978-3-642-33027-8_17](https://doi.org/10.1007/978-3-642-33027-8_17). URL : http://dx.doi.org/10.1007/978-3-642-33027-8_17.
- [MvV12] Roel MAES, Anthony VAN HERREWEGE et Ingrid VERBAUWHEDE. « PUFKY : A Fully Functional PUF-Based Cryptographic Key Generator ». In : *Cryptographic Hardware and Embedded Systems – CHES 2012*. Sous la dir. d'Emmanuel PROUFF et Patrick SCHAUMONT. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 302-319. ISBN : 978-3-642-33027-8.
- [Sut12] Ivan SUTHERLAND. « The Tyranny of the Clock ». In : *Commun. ACM* 55.10 (oct. 2012), p. 35-36. ISSN : 0001-0782. DOI : [10.1145/2347736.2347749](https://doi.org/10.1145/2347736.2347749). URL : <https://doi.org/10.1145/2347736.2347749>.
- [Bay+13] P. BAYON, L. BOSSUET, A. AUBERT et V. FISCHER. « Electromagnetic analysis on ring oscillator-based true random number generators ». In : *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2013, p. 1954-1957.
- [Che+13a] A. CHERKAoui, V. FISCHER, A. AUBERT et L. FESQUET. « A Self-Timed Ring Based True Random Number Generator ». In : *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*. 2013, p. 99-106.
- [Che+13b] Abdelkarim CHERKAoui, Viktor FISCHER, Laurent FESQUET et Alain AUBERT. « A Very High Speed True Random Number Generator with Entropy Assessment ». In : *Cryptographic Hardware and Embedded Systems – CHES 2013 15th International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2013*. T. 8086. Security and Cryptology. Santa Barbara, California, United States : Springer, août 2013, p. 179-196. URL : <https://hal-ujm.archives-ouvertes.fr/ujm-00859906>.
- [DR13] Eloísa DÍAZ-FRANCÉS et Francisco J. RUBIO. « On the existence of a normal approximation to the distribution of the ratio of two independent normal random variables ». In : *Statistical Papers* 54.2 (mai 2013), p. 309-323. ISSN : 1613-9798. DOI : [10.1007/s00362-012-0429-2](https://doi.org/10.1007/s00362-012-0429-2). URL : <https://doi.org/10.1007/s00362-012-0429-2>.
-

- [Hel+13] C. HELFMEIER, C. BOIT, D. NEDOSPASOV et J. SEIFERT. « Cloning Physically Unclonable Functions ». In : *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. Juin 2013, p. 1-6. DOI : [10.1109/HST.2013.6581556](https://doi.org/10.1109/HST.2013.6581556).
- [LAL13] P. LU, P. ANDREANI et A. LISCIDINI. « A 2-D GRO vernier time-to-digital converter with large input range and small latency ». In : *2013 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*. 2013, p. 151-154.
- [MGS13] Abhranil MAITI, Vikash GUNREDDY et Patrick SCHAUMONT. « A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions ». In : *Embedded Systems Design with FPGAs*. Sous la dir. de Peter ATHANAS, Dionisios PNEVMATIKATOS et Nicolas SKLAVOS. New York, NY : Springer New York, 2013, p. 245-267. ISBN : 978-1-4614-1362-2. DOI : [10.1007/978-1-4614-1362-2_11](https://doi.org/10.1007/978-1-4614-1362-2_11). URL : https://doi.org/10.1007/978-1-4614-1362-2_11.
- [Ned+13] D. NEDOSPASOV, J. SEIFERT, C. HELFMEIER et C. BOIT. « Invasive PUF Analysis ». In : *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. Août 2013, p. 30-38. DOI : [10.1109/FDTC.2013.19](https://doi.org/10.1109/FDTC.2013.19).
- [Bos+14] L. BOSSUET, X. T. NGO, Z. CHERIF et V. FISCHER. « A PUF Based on a Transient Effect Ring Oscillator and Insensitive to Locking Phenomenon ». In : *IEEE Transactions on Emerging Topics in Computing* 2.1 (mar. 2014), p. 30-36. ISSN : 2376-4562. DOI : [10.1109/TETC.2013.2287182](https://doi.org/10.1109/TETC.2013.2287182).
- [Che14] Abdelkarim CHERKAoui. « Ring oscillator based true random number generators : design, characterization and security ». Theses. Université Jean Monnet - Saint-Etienne, juin 2014. URL : <https://tel.archives-ouvertes.fr/tel-01171002>.
- [DV14] Jeroen DELVAUX et Ingrid VERBAUWHEDE. « Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation ». In : *Topics in Cryptology – CT-RSA 2014*. Sous la dir. de Josh BENALOH. Cham : Springer International Publishing, 2014, p. 106-131. ISBN : 978-3-319-04852-9.
- [Koe+14] P. KOEBERL, J. LI, A. RAJAN et W. WU. « Entropy loss in PUF-based key generation schemes : The repetition code pitfall ». In : *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2014, p. 44-49.
- [Mer+14] Paul A MEROLLA *et al.* « A million spiking-neuron integrated circuit with a scalable communication network and interface ». In : *Science* 345.6197 (2014), p. 668-673.
- [BFW15] Elaine BARKER, Larry FELDMAN et Gregory WITTE. *SP 800-90A Rev. 1 – Recommendation for random number generation using deterministic random bit generators*. Rapp. tech. National Institute of Standards and Technology, 2015.
- [GMC15] M. GIBILUKA, M. T. MOREIRA et N. L. V. CALAZANS. « A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework ». In : *2015 Euromicro Conference on Digital System Design*. 2015, p. 79-86.
- [Han+15] D. HAND *et al.* « Blade – A Timing Violation Resilient Asynchronous Template ». In : *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*. 2015, p. 21-28.

-
- [Jeo+15] D. JEON, J. H. BAEK, D. K. KIM et B. CHOI. « Towards Zero Bit-Error-Rate Physical Unclonable Function : Mismatch-Based vs. Physical-Based Approaches in Standard CMOS Technology ». In : *2015 Euromicro Conference on Digital System Design*. Août 2015, p. 407-414. DOI : [10.1109/DSD.2015.57](https://doi.org/10.1109/DSD.2015.57).
- [KL15] F. KODÝTEK et R. LÓRENCZ. « A Design of Ring Oscillator Based PUF on FPGA ». In : *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits Systems*. Avr. 2015, p. 37-42. DOI : [10.1109/DDECS.2015.21](https://doi.org/10.1109/DDECS.2015.21).
- [LB15] D. LUBICZ et N. BOCHARD. « Towards an Oscillator Based TRNG with a Certified Entropy Rate ». In : *IEEE Transactions on Computers* 64.4 (2015), p. 1191-1200.
- [Mar+15] H. MARTÍN, T. KORAK, E. S. MILLÁN et M. HUTTER. « Fault Attacks on STRNGs : Impact of Glitches, Temperature, and Underpowering on Randomness ». In : *IEEE Transactions on Information Forensics and Security* 10.2 (2015), p. 266-277.
- [Pan15] Barry PANGRLE. *Asynchronous Design : Is It Time Yet ?* 2015. URL : <https://semiengineering.com/asynchronous-design-is-it-time-yet/> (visité le 31/08/2020).
- [PB15] Mallika PRAKASH et Peter A BEEREL. *Static timing analysis of template-based asynchronous circuits*. US Patent 8,972,915. Mar. 2015.
- [BK16] Elaine BARKER et John KELSEY. *SP 800-90C (draft) – Recommendation for Random Bit Generator (RBG) Constructions*. Rapp. tech. National Institute of Standards and Technology, 2016.
- [Kan+16] J. KANG *et al.* « A System-on-Chip Solution for Point-of-Care Ultrasound Imaging Systems : Architecture and ASIC Implementation ». In : *IEEE Transactions on Biomedical Circuits and Systems* 10.2 (2016), p. 412-423.
- [MS16] J. V. MANORANJAN et K. S. STEVENS. « Qualifying Relative Timing Constraints for Asynchronous Circuits ». In : *2016 22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2016, p. 91-98.
- [Par+16] Hoon PARK, Anping HE, Marly RONCKEN, Xiaoyu SONG et Ivan SUTHERLAND. « Modular timing constraints for delay-insensitive systems ». In : *Journal of Computer Science and Technology* 31.1 (2016), p. 77-106.
- [Pet+16] O. PETURA, U. MUREDDU, N. BOCHARD, V. FISCHER et L. BOSSUET. « A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices ». In : *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 2016, p. 1-10.
- [RS16] A. ROELKE et M. R. STAN. « Attacking an SRAM-Based PUF through Wearout ». In : *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Juil. 2016, p. 206-211. DOI : [10.1109/ISVLSI.2016.68](https://doi.org/10.1109/ISVLSI.2016.68).
- [Sim+16] J. SIMATIC, A. CHERKAoui, R. P. BASTOS et L. FESQUET. « New asynchronous protocols for enhancing area and throughput in bundled-data pipelines ». In : *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*. 2016, p. 1-6.
- [Sit+16] C. SITIK, W. LIU, B. TASKIN et E. SALMAN. « Design Methodology for Voltage-Scaled Clock Distribution Networks ». In : *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.10 (2016), p. 3080-3093.
-

- [Tar16] Hubert TARDIEU. « La troisième révolution digitale. Agilité et fragilité ». FR. In : *Études* Octobre.10 (2016), p. 31-42. DOI : [10.3917/etu.4231.0031](https://doi.org/10.3917/etu.4231.0031). URL : <https://www.cairn.info/revue-etudes-2016-10-page-31.htm>.
- [Ber+17] F. BERTRAND, A. CHERKAOUI, J. SIMATIC, A. MAURE et L. FESQUET. « CAR : On the highway towards de-synchronization ». In : *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2017, p. 339-343.
- [Che+17] B. CHEN *et al.* « A Robust SRAM-PUF Key Generation Scheme Based on Polar Codes ». In : *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 2017, p. 1-6.
- [Gim+17] G. GIMENEZ, A. CHERKAOUI, R. FRISCH et L. FESQUET. « Self-timed Ring based True Random Number Generator : Threat model and countermeasures ». In : *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. 2017, p. 31-38.
- [HG17] Bilal HABIB et Kris GAJ. « A comprehensive set of schemes for PUF response generation ». In : *Microprocessors and Microsystems* 51 (juin 2017), p. 239-251. DOI : [10.1016/j.micpro.2017.05.001](https://doi.org/10.1016/j.micpro.2017.05.001).
- [El-+17] A. EL-HADBI, A. CHERKAOUI, O. ELISSATI, J. SIMATIC et L. FESQUET. « On-the-fly and sub-gate-delay resolution TDC based on self-timed ring : A proof of concept ». In : *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*. Juin 2017, p. 305-308. DOI : [10.1109/NEWCAS.2017.8010166](https://doi.org/10.1109/NEWCAS.2017.8010166).
- [Sim+17] J. SIMATIC, A. CHERKAOUI, F. BERTRAND, R. P. BASTOS et L. FESQUET. « A Practical Framework for Specification, Verification, and Design of Self-Timed Pipelines ». In : *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2017, p. 65-72.
- [Sim17] Jean SIMATIC. « Design flow for ultra-low power : non-uniform sampling and asynchronous circuits ». Theses. Université Grenoble Alpes, déc. 2017. URL : <https://tel.archives-ouvertes.fr/tel-01758184>.
- [Yan+17] B. YANG, V. ROŽIĆ, M. GRUJIĆ, N. MENTENS et I. VERBAUWHEDE. « On-chip jitter measurement for true random number generators ». In : *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 2017, p. 91-96.
- [Bar+18] M. BARBARESCHI, G. DI NATALE, L. TORRES et A. MAZZEO. « A Ring Oscillator-Based Identification Mechanism Immune to Aging and External Working Conditions ». In : *IEEE Transactions on Circuits and Systems I : Regular Papers* 65.2 (fév. 2018), p. 700-711. ISSN : 1558-0806. DOI : [10.1109/TCSI.2017.2727546](https://doi.org/10.1109/TCSI.2017.2727546).
- [Cou+18] M. COUSTANS *et al.* « Subthreshold logic for low-area and energy efficient true random number generator ». In : *2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. 2018, p. 1-3.
- [Dav+18] M. DAVIES *et al.* « Loihi : A Neuromorphic Manycore Processor with On-Chip Learning ». In : *IEEE Micro* 38.1 (2018), p. 82-99.
- [Fes+18] Laurent FESQUET, Abdelkarim CHERKAOUI, Grégoire GIMENEZ et Raphaël FRISCH. « Circuit and method for protecting asynchronous circuits ». Brev. amér. WO2020008229. 3 juil. 2018.

-
- [Gim+18] G. GIMENEZ, A. CHERKAOUI, G. COGNIARD et L. FESQUET. « Static Timing Analysis of Asynchronous Bundled-Data Circuits ». In : *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2018, p. 110-118. (*Best Paper Award*).
- [Hes+18] R. HESSELBARTH, F. WILDE, C. GU et N. HANLEY. « Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs ». In : *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Avr. 2018, p. 126-133. DOI : [10.1109/HST.2018.8383900](https://doi.org/10.1109/HST.2018.8383900).
- [Mor+18] S. MORADI, N. QIAO, F. STEFANINI et G. INDIVERI. « A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuro-morphic Asynchronous Processors (DYNAPs) ». In : *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (2018), p. 106-122.
- [Nou+18] Elie NOUMON ALLINI *et al.* « Evaluation and Monitoring of Free Running Oscillators Serving as Source of Randomness ». In : *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (août 2018), p. 214-242. DOI : [10.13154/tches.v2018.i3.214-242](https://doi.org/10.13154/tches.v2018.i3.214-242). URL : <https://tches.iacr.org/index.php/TCHES/article/view/7274>.
- [Sch+18] Alexander SCHAUB, Jean-Luc DANGER, Sylvain GUILLEY et Olivier RIOUL. « An Improved Analysis of Reliability and Entropy for Delay PUFs ». In : *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, août 2018. DOI : [10.1109/dsd.2018.00096](https://doi.org/10.1109/dsd.2018.00096).
- [SYK18] P. SRIVASTVA, P. K. YADAV et P. KUMAR MISRA. « Design of 32 bit Asynchronous RISC CPU Using Micropipeline ». In : *2018 Conference on Information and Communication Technology (CICT)*. 2018, p. 1-5.
- [Tur+18] Meltem Sönmez TURAN *et al.* *SP 800-90B – Recommendation for the Entropy Sources Used for Random Bit Generation*. Rapp. tech. National Institute of Standards and Technology, 2018.
- [Zha+18] Y. ZHANG *et al.* « Challenges in Building an Open-Source Flow from RTL to Bundled-Data Design ». In : *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2018, p. 26-27.
- [ZZC18] Yang ZHANG, Haipeng ZHA et Huimei CHENG. *Edge 1.0.5*. Available from <https://github.com/nobodybutyou1/Edge>. 2018.
- [Cor+19] Jordi CORTADELLA *et al.* *Petrify : a tool for synthesis of petri nets and asynchronous circuits*. Version 5.2. 2019. URL : <https://www.cs.upc.edu/~jordicf/petrify/distrib/home.html>.
- [F-S19] F-SECURE. *Attack Landscape H1 2019*. 2019. URL : https://s3-eu-central-1.amazonaws.com/evermade-fsecure-assets/wp-content/uploads/2019/09/12093807/2019_attack_landscape_report.pdf (visité le 24/09/2020).
- [Foj+19] M. FOJTIK *et al.* « A Fine-Grained GALS SoC with Pausible Adaptive Clocking in 16 nm FinFET ». In : *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2019, p. 27-35.
- [GSF19] G. GIMENEZ, J. SIMATIC et L. FESQUET. « From Signal Transition Graphs to Timing Closure : Application to Bundled-Data Circuits ». In : *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2019, p. 86-95.
-

- [Gim19] Grégoire GIMENEZ. *Local Clock Set Benchmark*. Version 1.0. 2019. URL : <http://tima.univ-grenoble-alpes.fr/tima/en/mediatheque/freetools.html>.
- [EEF19] A. EL-HADBI, O. ELISSATI et L. FESQUET. « Time-to-Digital Converters : A Literature Review and New Perspectives ». In : *2019 5th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*. Mai 2019, p. 1-8. DOI : [10.1109/EBCCSP.2019.8836857](https://doi.org/10.1109/EBCCSP.2019.8836857).
- [McG+19] Thomas MCGRATH, Ibrahim E. BAGCI, Zhiming M. WANG, Utz ROEDIG et Robert J. YOUNG. « A PUF taxonomy ». In : *Applied Physics Reviews* 6.1 (mar. 2019), p. 011303. DOI : [10.1063/1.5079407](https://doi.org/10.1063/1.5079407).
- [Xir+19] N. XIROMERITIS, S. SIMOGLU, C. SOTIRIOU et N. SKETOPOULOS. « Graph-Based STA for Asynchronous Controllers ». In : *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2019, p. 9-16.
- [Ara+20] Diego F. ARANHA, Felipe Rodrigues NOVAES, Akira TAKAHASHI, Mehdi TIBOUCHI et Yuval YAROM. *LadderLeak : Breaking ECDSA With Less Than One Bit Of Nonce Leakage*. Cryptology ePrint Archive, Report 2020/615. <https://eprint.iacr.org/2020/615>. 2020.
- [EMP20] Maik ENDER, Amir MORADI et Christof PAAR. « The Unpatchable Silicon : A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs ». In : *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, août 2020, p. 1803-1819. ISBN : 978-1-939133-17-5. URL : <https://www.usenix.org/conference/usenixsecurity20/presentation/ender>.
- [FCG20] Laurent FESQUET, Abdelkarim CHERKAOUI et Grégoire GIMENEZ. « Procédé pour générer une donnée unique propre à un circuit intégré en silicium ». Brev. franç. 2000491. 17 jan. 2020.
- [GCF20] G. GIMENEZ, A. CHERKAOUI et L. FESQUET. « A Self-Timed Ring based PUF ». In : *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 2020.
- [SFP20] Emanuele STRIEDER, Christoph FRISCH et Michael PEHL. *Machine Learning of Physical Unclonable Functions using Helper Data - Revealing a Pitfall in the Fuzzy Commitment Scheme*. Cryptology ePrint Archive, Report 2020/888. <https://eprint.iacr.org/2020/888>. 2020.

Annexe A

Liste des acronymes

- AES** *Advanced Encryption Standard* 72, 73
- AFSM** machine à états asynchrones (*Asynchronous Finite State Machine*) 26
- AHLS** synthèse de haut niveau asynchrone (*High-Level Synthesis*) 26
- AIS** notes d'application et d'interprétations de schéma (*Anwendungshinweise und Interpretationen im Schema*) 95, 100, 101, 103, 104, 107, 121, 133, 139
- ANoC** réseau sur puce asynchrone (*Asynchronous Network on Chip*) 21
- AOP** Amplificateur Opérationnel 95
- APB** *Advanced Peripheral Bus* 160
- A-PUF** PUF à base d'Arbitre x, 111, 112, 144, 151
- ASIC** circuit intégré spécialisé (*Application-Specific Integrated Circuit*) 116, 145, 158, 161
- BD** à données groupées (*Bundled-data*) xiii, 21–23, 32, 38, 44, 123, 163–165, 167, 168
- BSI** Office fédéral de la sécurité des technologies de l'information (*Bundesamt für Sicherheit in der Informationstechnik*) 100, 101, 121
- CAN** Convertisseur Analogique-Numérique 95
- CAO** Conception Assistée par Ordinateur 6, 32, 37–39, 41, 52, 68, 71, 78, 164
- CAVP** *Cryptographic Algorithm Validation Program* 100
- CCDO** optimisation concurrente de l'arbre d'horloge et du chemin de données (*Concurrent Clock and Data Optimization*) 80, 166
- CCTL** *Common Criteria Testing Laboratory* 100
- CHP** *Communicating Hardware Processes* 30
- CIME** Centre Inter-universitaire de Micro-Électronique et nanotechnologies iii
- CMOS** *Complementary Metal-Oxide-Semiconductor* 120, 150
- CMVP** *Cryptographic Module Validation Program* 100
- CPA** *Correlation Power Analysis* 167, 168
- CPU** *Central Processing Unit* 36
- CRPR** réduction du pessimisme lié aux reconvergences d'horloges (*Clock Reconvergence Pessimism Removal*) 63
- CSP** *Communicating Sequential Processes* 30
- DDoS** *Distributed Denial of Service* 5
- DFT** *Design For Test* 24

- DI** insensible aux délais (*Delay Insensitive*) 21, 22, 87, 117
- DRAM** mémoire vive dynamique (*Dynamic Random Access Memory*) 5
- DSA** algorithme de signature numérique (*Digital Signature Algorithm*) 86
- DVFS** adaptation dynamique de la tension et de la fréquence (*Dynamic Voltage and Frequency Scaling*) 8, 9
- ECC** code correcteur d'erreurs (*Error Correction Code*) 109
- EMA** attaque par injection électromagnétique (*ElectroMagnetic Attack*) 118
- EPC** horloge de propagation d'évènements (*Event Propagation Clock*) 54, 56, 57, 65, 67, 70, 73
- ERO-TRNG** générateur de nombres véritablement aléatoires élémentaire basé sur des oscillateurs en anneau (*Elementary Ring Oscillators based TRNG*) x, 104–107, 165
- FIA** attaque par injection de fautes (*Fault Injection Attack*) 118
- FIB** sonde ionique focalisée (*Focused Ion Beam*) 121
- FIPS** standards développés et publiés par le gouvernement des États-Unis (*Federal Information Processing Standard*) 129
- FPGA** réseau de portes programmables *in situ* (*Field-Programmable Gate Array*) xi, 5, 11, 102, 104, 107, 116, 133, 145, 153–157, 159, 161, 166, 167, VI
- FSM** machine à états finie (*Finite State Machine*) 25, 26
- GALS** Globalement Asynchrones et Localement Synchrones 6, 16, 21
- GDSII** *Graphic Design System 2* 166
- HDL** langage de description de matériel (*Hardware Description Language*) 125
- HLS** synthèse de haut niveau (*High-Level Synthesis*) 26
- IID** Indépendante et Identiquement Distribuée 100
- IoT** internet des objets (*Internet of Things*) 1
- IP** propriété intellectuelle (*Intellectual Property*) 1, 36, 37, 145, 158
- IRO** oscillateur en anneau à inverseurs (*Inverter Ring Oscillator*) 87, 91, 93, 97, 101, 102, 104, 106, 136
- IV** vecteur d'initialisation (*Initialization Vector*) 86
- LCS** série d'horloges locales (*Local Clock Set*) ix, x, 52, 53, 55–57, 59, 63–68, 70–81, 164, 166
- LFSR** registre à décalage à rétroaction linéaire (*Linear Feedback Shift Register*) 99
- LUT** *LookUp Table* 153, 154
- LVS** *Layout Versus Schematic* 166
- MEB** Microscope Électronique à Balayage x, 111
- ML** *Machine Learning* 112
- MOS** *Metal-Oxide-Semiconductor* III
- MURO-TRNG** générateur de nombres véritablement aléatoires basé sur une multitude d'oscillateurs en anneau (*MUltiple Ring Oscillators based TRNG*) x, 101–103, 105–108, 116

-
- NIST** *National Institute of Standard and Technology* 99, 100, 103, 121, 156
- NMOS** MOS de type N 131, 150
- NRZ** non retour-à-zéro (*Non Return to Zero*) 17
- NVM** mémoire non volatile (*Non-Volatile Memory*) 87, 109, 111
- OCV** variation à l'intérieur du circuit (*On-Chip Variation*) 27, 79
- OFIA** attaque par induction optique de fautes (*Optical Fault Induction Attack*) 118
- OTP** mémoire programmable une seule fois (*One Time Programmable*) 87
- PGCD** Plus Grand Commun Diviseur 119
- PLL** *Phase-Locked Loop* 20, 28, 87
- PMOS** MOS de type P 131, 150
- POC** point de convergence 43, 44, 47, 48, 53, 57, 62, 65, 164
- POD** point de divergence 44, 47, 53, 55, 59, 61–63, 65, 68, 164
- PRNG** générateur de nombres pseudo-aléatoires (*Pseudo Random Number Generator*) 86, 95, 98, 100
- PUF** fonction physique non-clonable (*Physical Unclonable Function*) x, xi, 10, 11, 87, 107–114, 144–162, 164–167, I, III–V
- PVT** processus de fabrication, tension et température (*Process, Voltage, Temperature*) 20, 41, 67, 72, 74, 87, 108, 109
- QDI** quasi-insensible aux délais (*Quasi-Delay Insensitive*) ix, xiii, 6, 10, 21, 22, 32, 37, 38, 167
- RC** Résistance et Capacité 126
- RNG** générateur de nombres aléatoires (*Random Number Generator*) 110
- RO** oscillateur en anneau (*Ring Oscillator*) xiii, 87, 88, 90, 91, 97, 101, 103, 104, 109, 112–114, 118, 136, 145–149, 151–155, 157–161, 165, 166, III
- RO-PUF** PUF à base de RO x, 112, 113, 144, 157, 159–161, 165
- RTC** contrainte de temps relatifs (*Relative Timing Constraint*) ix, x, 31, 32, 39–44, 46–49, 52–68, 70–76, 79, 80, 163, 164, 166
- RTL** niveau transferts de registres (*Register-Transfer Level*) 74, 126
- RZ** retour-à-zéro (*Return to Zero*) 17, 22, 23, 25, 29, 70, 72, 94
- SDC** *Synopsys Design Constraints* 31, 40, 43, 55, 57, 59, 67, 70–74, 79, 164, 166
- SI** indépendant de la vitesse (*Speed Independent*) 22
- SNN** réseau de neurones à impulsion (*Spiking Neural Network*) 6
- SNR** rapport signal à bruit (*Signal-to-Noise Ratio*) 108, 151–153, 155, 156, 161, 166, 167
- SP** publication spéciale (*Special Publication*) xiii, 99, 100, 102, 103, 110, 121, 156
- SPICE** logiciel de simulation de circuits électroniques analogiques (*Simulation Program with Integrated Circuit Emphasis*) 125, 128
- SRAM** mémoire vive statique (*Static Random Access Memory*) 113, III
- SRAM-PUF** PUF à base de SRAM x, 113, 144
- STA** analyse temporelle statique (*Static Timing Analysis*) 23, 24, 31, 39–41, 48–50, 53–55, 58, 59, 63, 65, 66, 70, 71, 73, 74, 78, 164, VI
-

- STG** graphe de transitions de signaux (*Signal Transition Graph*) 29, 30, 52, 59–62, 65, 68, 79, 164, 166
- STR** oscillateur auto-séquence (*Self-Timed Ring oscillator*) x, xi, 10, 11, 87, 89–94, 97, 104–107, 114, 116–136, 138–141, 144, 149–151, 153, 154, 156–165, 167, IV–VI
- STR-PUF** fonction physique non-clonable basée sur un oscillateur auto-séquence (*STR based PUF*) xi, xiii, 151, 153–159, 161, 166, 167
- STR-TRNG** générateur de nombres véritablement aléatoires basé sur un oscillateur auto-séquence (*STR based TRNG*) x, xi, 11, 105–107, 114, 116, 117, 121–125, 127, 131, 133, 135–141, 158, 165–167
- TAL** bibliothèque de cellules standard pour l’asynchrone du laboratoire TIMA (*TIMA Asynchronous Library*) 125
- TDC** convertisseur temps-numérique (*Time-to-Digital Converter*) xi, 144, 148–153, 157, 159, 160, 162, 165
- TERO** oscillateur en anneau à effet transitoire (*Transient Effect Ring Oscillator*) 88, 90, IV
- TERO-PUF** PUF à base de TERO 113
- TF** *Total Failure* 123
- TLM** description au niveau transactionnel (*Transaction Level Model*) 29
- TRNG** générateur de nombres véritablement aléatoires (*True Random Number Generator*) x, xiii, 10, 11, 86, 87, 95–98, 100–105, 107, 108, 114, 116–118, 122, 133–136, 140, 141, 145, 147, 158–160, 164, II, IV
- UART** émetteur-récepteur asynchrone universel (*Universal Asynchronous Receiver-Transmitter*) 18, 133, 153, 160
- ULP** *Ultra Low Power* 125, 131
- UMC** *United Microelectronics Corporation* 125, 131
- VCO** oscillateur commandé en tension (*Voltage Controlled Oscillator*) 87
- VHDL** langage de description de matériel développé par le VHSIC (*VHSIC Hardware Description Language*) 128
- VHSIC** initiative du département de la défense des États-Unis (*Very High Speed Integrated Circuit*) IV
- WCHB** *Weak Condition Half-Buffer* 25, 70–72, 93
- XOR** OU-exclusif 23, 24, 48, 49, 93, 101, 105, 116, 117, 123, 133, 139

Annexe B

Glossaire

bottleneck (mode d'oscillation d'un STR) avec goulot d'étranglement lié a un ou plusieurs étages ayant un temps de propagation important. 90, 118, 120, 128–130, 135, 165

buffer mémoire tampon. 133

burst (mode d'oscillation d'un STR) en rafale d'évènements. 89, 90, 118, 119, 122, 130, 134, 135

Charlie (effet) Phénomène analogique spécifique aux portes de Muller ayant pour effet d'allonger le temps de propagation lorsque le temps de séparant les commutations de ses entrées est court. 90, 91, 118, 119, 125, 128, 132, 153, 165, 167

clock-gating Méthode de réduction de la consommation dynamique d'un circuit consistant à couper le signal d'horloge d'une partie du circuit lorsqu'elle est inactive. ix, 37, 46, 57–59

drafting (effet) Phénomène analogique ayant pour effet de raccourcir le temps de propagation dans une porte combinatoire plus le délai séparant deux commutations successives de sa sortie est court. 90, 91, 125, 128, 165, 167

DRM Ensemble des règle de conception (dessin, électrique, etc.) pour une technologie donnée (*Design Rule Manual*) 110

evenly-spaced (mode d'oscillation d'un STR) avec répartition uniforme des évènements dans l'anneau. 89–91, 93, 105, 114, 117–119, 121, 123, 130, 132, 134, 135, 150, 153, 158, 160, 167

helper data données permettant reconstituer ou de corriger la réponse d'un PUF. 109

hold Temps pendant lequel une donnée doit être stable après avoir été échantillonnée. 32, 40, 43–45, 55, 65, 71, 75, 76, 128

jitter (ou gigue) Phénomène de fluctuation d'un signal. x, 96, 97, 104, 106, 107, 120, 128, 130, 135, 136, 139, 141, 158, 164, 165

Liberty Format de fichier d'abstraction temporelle développé par Synopsys 32, 44, 48, 128

matché (délai) apparié. Ce dit des délais insérés dans le chemin de contrôle pour compenser le pire temps de propagation dans le chemin de données. 23, 25, 32, 33, 38, 45, 49, 54–56, 71, 74–77, 79–81, 166

Monte-Carlo méthode de calcul d'une valeur numérique utilisant des techniques probabilistes. 98

- netlist** Fichier de description d'un circuit électronique définissant les différents blocs (portes logique ou sous-blocs) ainsi que leurs interconnexions. 61, 62, 65, 67, 68, 79, 166
- overclocking** augmentation la fréquence du signal d'horloge afin d'induire un mauvais fonctionnement. 117
- oversampling** augmentation la fréquence du signal d'échantillonnage afin de capturer plusieurs fois la même valeur. 117
- phase covering** (mode d'oscillation d'un STR) avec recouvrement de phase du fait de que le nombre d'étages et le nombre d'évènement ne sont pas premier entre eux. 118, 119, 121, 122, 130, 131
- pipelined** ensemble d'éléments de calcul connectés en série : la sortie d'un élément est l'entrée du suivant. 116
- power switch** interrupteur d'alimentation créé à l'aide d'un transistor. 113
- reset** Signal d'initialisation d'un circuit. 44, 118
- retiming** Méthode d'optimisation d'un circuit consistant à modifier la position de ses éléments séquentiels [LRS83]. 25, 37
- setup** Temps pendant lequel une donnée doit être stable avant d'être échantillonnée. ix, 32, 40, 42–45, 54–56, 65, 71, 75
- skew** Plus grande différence entre les temps d'arrivée d'une horloge sur chacun de ses éléments séquentiels. 37, 76, 78, 81
- slack** Marge temporelle associée à une contrainte de temps relatifs. Elle peut être positive ou négative suivant que la contrainte est respectée ou non. 54, 55
- SLICE** bloc élémentaire des matrices des FPGA proposées par Xilinx. 153, 154
- testbench** (litt.) banc d'essai, modélisation comportementale du système intégrant le circuit à valider (génération des stimuli, horloge, initialisation, etc.). 128
- testchip** circuit de test. 133, 135, 141
- time borrowing** Capacité des outils de STA d'utiliser les phases de transparence des verrous pour équilibrer les chemins temporels. 71
- tristate** (logique) à trois états, permettant à une sortie d'avoir un état haute-impédance en plus des états 0 et 1. 132
- unate** Capacité d'une fonction booléenne à faire passer une transition d'une de ses entrées vers sa sortie (quel que soit la valeur de ses autres entrées). 58
- VIA** trou métallisé permettant d'établir une liaison électrique entre deux couches métalliques (*Vertical Interconnect Access*) 110, 111
- wafer** (litt. gaufrette) tranche de matériau semi-conducteur monocristalin utilisée pour fabriquer des composants de micro-électronique. 145

Concevoir des circuits sécurisés à très faible consommation :

une alternative basée sur l'asynchrone

Résumé

Depuis ses débuts, l'industrie micro-électronique est confrontée au diptyque réduction des coûts et amélioration des performances. Mais la 3^e révolution digitale et l'avènement du « tous, tout, partout, tout le temps connectés » ont subversivement redéfini les enjeux et les défis auxquels cette industrie fait face. Ainsi la consommation et la sécurité sont des problématiques, hier encore limitées à des applications très spécifiques, qui aujourd'hui gagnent en importance et viennent contraindre la majorité des circuits intégrés. Si pendant des années la loi de Moore et l'échelle de Dennard ont observé une augmentation continue du nombre de calculs par unité d'énergie, celles-ci s'essouffent et les limites physiques et économiques semblent avoir pris le dessus. Par ailleurs, ces deux enjeux sont très souvent en opposition, et la sécurité ne se fait généralement qu'au détriment de la consommation. Dans ce manuscrit nous proposons d'étudier une alternative se basant sur de la logique dîtes asynchrone, ou sans horloge. Nous nous attachons particulièrement à montrer que le principal *a priori* vis-à-vis de ces circuits – leur complexité de conception – peut être contourné en utilisant les outils de CAO standards, faisant d'eux une solution accessible aux concepteurs synchrones. Nous montrons aussi que ces circuits permettent de construire des primitives de sécurité – PUF et TRNG – très performantes. Nous étudions leurs failles et proposons des contremesures adaptées. Finalement, sans prêcher un changement radical vers les circuits auto-séquenceés, nous plaçons pour une intégration progressive de la logique asynchrone en commençant par les éléments sensibles à la consommation et à la sécurité.

Mots-clés : circuit intégrés, basse consommation, sécurité, asynchrone, TRNG, PUF

Abstract

Microelectronics industry has always been confronted with the twofold challenge of reducing costs and improving performances. But the 3rd digital revolution and the emergence of the "everyone, everything, everywhere, everytime connected" have subversively redefined the challenges this industry faces. Consumption and security are issues that were once limited to very specific applications, but which are now gaining in importance and constraining the majority of integrated circuits. For years, Moore's law and Dennard's scale have observed a continuous increase in the number of operations per unit of energy, yet these are running out of steam and physical and economic limits seem to prevail. Moreover, these two issues are very often opposed, and security is generally only achieved at the expense of additional consumption. In this manuscript we propose to study an alternative based on so-called asynchronous (or clockless) logic. We show that the main *a priori* with respect to these circuits – their design complexity – can be circumvented using standard CAD tools, making them an accessible solution for synchronous designers. We also show that these circuits can be used to build high-performance security primitives – PUF and TRNG. We study their flaws and propose adapted countermeasures. Finally, without preaching a radical change towards self-sequencing circuits, we argue for a progressive integration of asynchronous logic, starting by circuits with low-power and security concerns.

Keywords : integrated circuit, low power, security, asynchronous, TRNG, PUF

