Vers des calculateurs Bayésiens compacts, à faible énergie et à précision ajustable

Jérémy Belot

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano-électronique et nanotechnologies**

Arrêtée ministériel : 25 mai 2016

Présentée par

Jérémy BELOT

Thèse dirigée par **Laurent FESQUET (directeur)**
et co-encadrée par **Abdelkarim CHERKAOUI (co-encadrant)**
et par **Raphaël LAURENT (co-encadrant)**

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

Vers des calculateurs Bayésiens compacts, à faible énergie et à précision ajustable

Towards compact, low power and with adjustable accuracy Bayesian computers

Thèse soutenue publiquement le **12 octobre 2022**,
devant le jury composé de :

François Pécheux Professeur des universités, LIP6 - Sorbonne Université,	**Président**
Dominique Dallet Professeur des universités, IMS - Bordeaux INP,	**Rapporteur**
Olivier Sentieys Professeur des universités, ENSSAT - Université de Rennes,	**Rapporteur**
Frédéric Pétrot Professeur des universités, TIMA - Université Grenoble Alpes,	**Examinateur**
Laurent FESQUET Maître de conférences, TIMA - Université Grenoble Alpes,	**Directeur de thèse**
Abdelkarim CHERKAOUI Docteur en sciences, HawAI.tech,	**Co-Encadrant**
Raphaël LAURENT Docteur en sciences, HawAI.tech,	**Co-Encadrant**

"À ma famille et mes amis dont j'ose espérer qu'ils liront jusqu'ici."

Remerciements

J'aimerais par ces quelques lignes remercier tous ceux qui, à leur manière, ont contribué à faire de ces 3 années de thèse une expérience hors du commun.

Tout d'abord un immense merci à Laurent, Karim et Raphaël pour votre engagement, votre rigueur et votre confiance, ils ont eu un impact très appréciable sur la qualité de la thèse. Merci aussi pour votre disponibilité et pour avoir su faire preuve de souplesse avec des agendas pas toujours très concordants.

Merci aux membres du jury pour avoir accepté de faire partie de cette thèse, merci pour vos retours des plus précieux.

Merci à tous ceux qui font ou ont fait partie de l'équipe CDSI et à ceux qui se sont perdus à ce premier étage du TIMA pour les discussions passionnées autour d'un café, d'un maté, ou de croissants. Merci à tous pour l'ambiance au travail comme en dehors. Merci à Laurent, Katell, Stéphane, Sylvain, Assia, Sophie, Otto, Ricardo, Julie, Grégoire, Nils, Mehdi, Yasser, Rodrigo, Yoan, Mohamed, Renato, Ankush, Lucas, Nicolas, Florent, Olivier, Marco, Xavier, Hasan, Rosalie, Damiano, Diana, Leticia, Vinicius, Cristiano et toute la relève que je n'ai pas eu la chance de rencontrer.

Merci à HawAI.tech pour le soutien tant moral et technique que financier, et pour avoir accepté de relever ce défi un peu fou d'engager une thèse CIFRE dans une start-up âgée de quelques mois seulement à l'époque ! Merci à Marvin, Raphael, Jean, Karim, Raphaël, Manu, Jacques, Pierre, Damien, Marie, Jon, Jérôme, Marie-Lou, Antoine. C'est un réel plaisir de continuer l'aventure Bayésienne avec vous tous.

Merci enfin à mes proches pour votre soutien indéfectible et pour avoir fait corps le jour de ma soutenance de thèse, ce moment si spécial pour moi. Merci à ma famille Monique, Gilles, Jeanne-Antide, Gilbert, Denise, Georges, Fanny, Yédir, Benjamin, Marie-Charlotte, Jean-Michel, Céline, Annie, Christine et tous les oncles, tantes, cousins, cousines et amis qui l'ont suivie en ligne. Merci à mes amis de Grenoble ou d'ailleurs, Marvin, Ket, Gaëtan, Jean, Luana, Raphael, Vania, Adrien, Seb, Aurélien. Merci à Mickaël et Jordan pour votre loyauté et votre amitié inconditionnelle malgré le temps et la distance. Merci à Liége pour ta confiance, ton soutien sans faille et ta patience pendant ces derniers mois un peu agités.

Merci à tous ceux que j'oublie.

Table des matières

Introduction

L'intelligence artificielle (IA) est sans doute l'un des domaines scientifiques qui a connu les plus grandes avancées technologiques de ce début de siècle. Elle montre des résultats spectaculaires et très médiatisés dans des situations où elle semble surpasser les capacités humaines. En mai 1997 déjà, IBM réalisait le tour de force de battre le grand maître Gary Kasparov aux échecs avec son supercalculateur *Deep Blue* [1, 2], remportant une victoire symbolique à une époque où les échecs étaient considérés comme l'apanage de l'intelligence humaine. Connue pour ses sorties médiatiques sensationnelles [3, 4], l'entreprise américaine récidive 14 ans plus tard en présentant son superordinateur Watson au jeu télévisé *Jeopardy !* [1] [5] qui remporte la finale face à Ken Jennings et Brad Rutter, deux des plus grands champions américains de l'époque. Une telle victoire est d'autant plus marquante que ce jeu nécessite des qualités habituellement attribuées à l'intelligence humaine, comme la compréhension du langage naturel, son interprétation et même l'expression orale (par synthèse vocale) d'une réponse compréhensible par l'être humain. Enfin, en mars 2016, c'est au tour de Google DeepMind et de son algorithme AlphaGo [6] de créer la surprise en battant au jeu de Go Lee Sedol, l'un des meilleurs joueurs mondiaux. Pour mettre en contexte la prouesse d'AlphaGo, rappelons que le jeu de Go est un problème autrement plus complexe que celui des échecs, avec un arbre de décision beaucoup plus vaste et donc un effet d'horizon plus important. C'est pourquoi le jeu de Go était encore considéré comme l'un des bastions de l'hégémonie humaine sur la machine dans le domaine du jeu, d'où le retentissement de l'exploit du programme du géant américain. D'une manière générale, la frontière s'affine chaque jour entre ce qu'est capable de produire un être humain et ce que pourrait produire une machine, que ce soit en termes de génération de texte [7], de code [8], d'image [9, 10], ou encore de musique [11]. Le fait que ces productions artificielles soient proches de convaincre, ou convainquent déjà [12] un public humain peut être considéré comme une réussite à ce qui peut s'apparenter à un test de Turing [13], supposant ainsi d'une forme de reconnaissance d'une certaine intelligence de la machine.

1. *Jeopardy !* est un jeu télévisé américain, semblable à "Questions pour un champion" en France, dont le but est de deviner un mot ou un personnage à partir de sa définition.

Les limites de l'IA aujourd'hui

Nous distinguons trois grandes classes d'IA : l'IA symbolique, l'IA connexionniste et l'IA probabiliste.

L'IA symbolique

L'IA symbolique est une intelligence basée sur des représentations haut niveau de la connaissance. Elle utilise la logique et le raisonnement pour prendre des décisions. Les systèmes experts en sont un exemple, modélisant le raisonnement d'un expert d'un certain domaine, ce sont ceux qui ont permis à Deep Blue de vaincre Kasparov en 1997. L'IA symbolique est en ce sens explicable car définie par des règles claires et univoques.

Cependant, "les éléphants ne jouent pas aux échecs" [14], doit-on pour autant en conclure qu'ils ne sont pas intelligents ? Cela semble injustifié, et de nombreux scientifiques ont commencé dans les années 1990 à se détourner de l'approche symbolique de l'IA, arguant que le monde étant son modèle le plus pur, il suffit de le mesurer de manière assez précise et assez fréquente plutôt que de le modéliser. De plus, l'IA symbolique demeure rigide, peu robuste à des données incertaines ou incomplètes. Les décisions qu'elle prend sont limitées à un problème défini et elle ne peut pas être généralisée à d'autres problèmes en dehors de son champ d'application.

L'IA connexionniste

Les rapides avancées de l'IA de ces 15 dernières années, dont celle de Watson et AlphaGo, sont à attribuer à l'IA connexionniste, notamment grâce au développement de l'apprentissage profond dont le Français Yann Le Cun est l'un des pionniers [15, 16]. L'apprentissage profond est une méthode d'apprentissage automatique caractérisée par l'utilisation de réseaux de neurones de grande profondeur, permettant à l'utilisateur de ne pas avoir à extraire les caractéristiques des données utiles au calcul : les données d'entrée du système peuvent être brutes. Pour ce faire, cette méthode requiert une grande quantité de données et d'annotations, ainsi qu'une puissance de calcul très importante.

Ainsi, bien que de tels algorithmes permettent des performances surpassant parfois les capacités humaines, ils possèdent des limites qui sont un frein à leur expansion dans tous les domaines. D'abord, le volume important de données annotées nécessaire à l'apprentissage profond le cantonne à des applications pour lesquelles les jeux de données sont abondants et diversifiés comme la vision par ordinateur ou le traitement de texte, mais cela devient plus complexe pour des applications où les données sont par essence plus rares comme la maintenance prédictive [17]. De plus, collecter, stocker et intégrer une très grande quantité d'informations, avec une forte puissance de calcul nécessaire pour l'entraînement de ces IA et, dans une moindre mesure, pour leur utilisation pour l'inférence, entraîne une importante consommation énergétique. Enfin, le fait que de tels réseaux de neurones possèdent un grand nombre de couches cachées rend leurs résultats difficilement explicables, avec un

comportement de type "boîte noire", ce qui peut poser un problème dans des applications critiques comme la finance, l'aéronautique ou encore la défense [18].

La solution probabiliste

L'IA probabiliste, que l'on associe aux modèles Bayésiens [19, 20], propose de son côté de tirer parti des avantages du symbolisme et du connexionnisme. D'une part, tout comme l'IA symbolique, elle s'appuie sur une représentation de haut niveau d'abstraction du monde, intégrant une connaissance experte *a priori* [2]. D'autre part, elle utilise un apprentissage par les données, permettant au modèle de s'affiner au fur et à mesure, comme l'IA connexionniste.

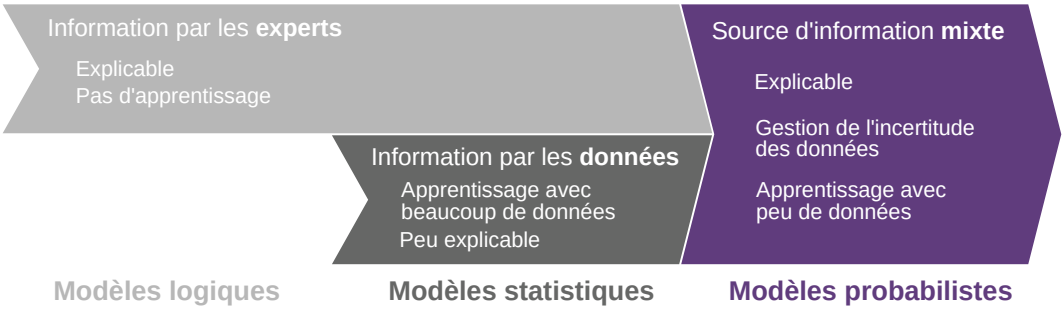

FIGURE 1 – *Les 3 vagues de l'IA (figure adaptée de documents HawAI.tech internes et publics [21]).*

Cette approche probabiliste adresse les principaux goulots d'étranglements des deux dernières classes d'IA.

Intérêts de l'IA probabiliste

Premièrement, elle est capable de raisonner et d'interagir dans un monde ouvert, dont les données sont incertaines voire incomplètes, contrairement à l'IA symbolique qui requiert un environnement très contrôlé. En effet, les modèles d'IA probabilistes intègrent, de manière intrinsèque, l'incertitude dans leur chaîne de traitement. Celle-ci est propagée à travers les différentes couches du modèle, de la perception jusqu'à la prédiction. La prise de décision en sortie du modèle peut ainsi être adaptée en fonction de l'incertitude à tous les niveaux.

De plus, l'approche probabiliste permet d'intégrer des connaissances expertes *a priori*, permettant l'économie d'un apprentissage de phénomènes déjà connus. Il est en effet superflu et souvent difficile d'essayer de réapprendre les lois de la physique ou le code de la route à partir d'une base de données à la manière d'un réseau de neurones profond. Les modèles d'IA probabilistes se basent, eux, sur

2. Ici le terme *a priori* signifie "avant tout calcul", il fait référence au *prior* Bayésien.

une décomposition du problème suffisamment fine pour permettre d'intégrer des lois et règles dans des sous-instances du modèle. Ils permettent ainsi de concilier l'apprentissage et l'utilisation de connaissances expertes *a priori* pour modéliser au mieux le phénomène d'intérêt.

Enfin, l'IA probabiliste est capable de rendre des résultats explicables et justifiés, par opposition aux réseaux de neurones profonds, pour lesquels il est difficile de déterminer les entrées et les étapes intermédiaires qui ont donné lieu au résultat. Les modèles d'IA probabilistes sont certes plus difficiles à construire, mais permettent dans certains cas de déduire, connaissant les résultats, les données d'entrées ou de tout niveau de raisonnement intermédiaire. Il est par conséquent possible d'identifier les facteurs ayant contribué à la décision finale. Ceci permet, par exemple, de diagnostiquer des pannes ou incohérences des capteurs en entrée du modèle.

Les modèles d'IA probabilistes nécessitent toutefois d'effectuer des calculs intensifs sur des distributions de probabilité dans des espaces de grandes dimensions. Aujourd'hui, le matériel existant ne permet pas d'effectuer ces calculs de manière efficace.

Le besoin d'une accélération matérielle dédiée

Les deux dernières vagues d'IA se sont reposées sur des architectures matérielles particulières. L'IA symbolique a été largement implémentée et déployée sur des processeurs (CPUs - *Central Processing Units*). La deuxième vague de l'IA, notamment basée sur les modèles d'apprentissage profond, a su profiter du développement des processeurs graphiques (GPUs - *Graphical Processing Unit*) [22] pour accélérer les calculs dans les réseaux de neurones artificiels. Ensuite, des accélérateurs spécifiques pour l'apprentissage profond, et notamment des Intelligence Processing Units (IPU) [23] sont apparus ces dernières années pour traiter des flots de données de plus en plus importants.

La Figure 2 montre l'évolution des architectures des accélérateurs matériels de l'IA développés pour traiter des objets de nature différente : vecteurs, tenseurs, graphes ou distributions de probabilité.

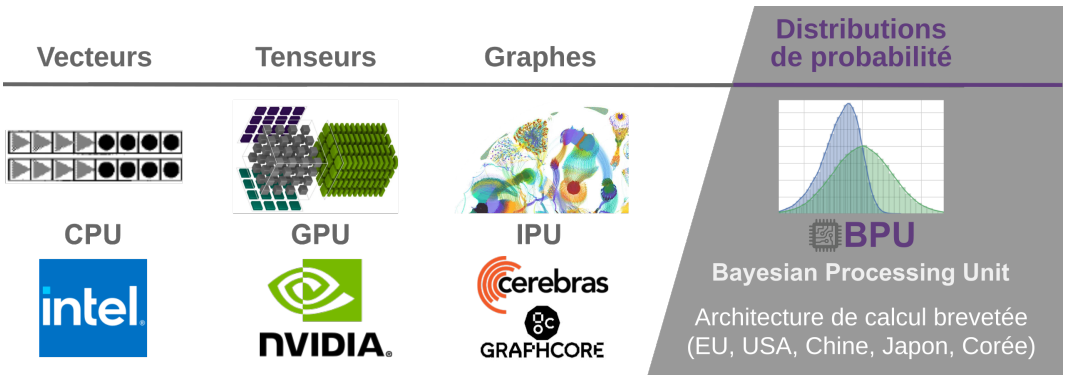

FIGURE 2 – *Évolution du paradigme d'IA accompagné de l'évolution de l'opération au cœur du calculateur (figure adaptée de documents HawAI.tech internes).*

Ainsi, pour être pleinement efficace énergétiquement et en termes de performances, l'IA probabiliste doit elle aussi être associée à une accélération matérielle via des circuits électroniques dédiés : c'est l'ambition de la startup HawAI.tech [24] qui développe le BPU (*Bayesian Processing Unit*) et qui a rendu ces travaux de thèse possibles en fournissant financement et encadrement.

Replacer l'intelligence au coeur du capteur

Avec l'avènement de l'Internet des Objets (IoT) [25], des milliards de dispositifs captent, collectent et redistribuent de grands volumes de données, posant des problèmes de bande-passante et de consommation d'énergie pour la communication avec le *cloud*. Ces dispositifs fonctionnant pour la plupart sur batteries, ils se doivent de consommer le moins possible pour réduire les coûts de maintenance de recharge ou de remplacement de celles-ci. L'un des enjeux majeurs de ce domaine d'application en plein essor est donc de rapprocher le calcul au plus près des capteurs, sur sa couche de perception : c'est ce qu'on appelle l'*edge computing* [26].

De cette manière, au moyen de méthodes comme la fusion de capteurs, le volume de données envoyées au *cloud* est beaucoup plus restreint. Ce ne sont plus les données brutes mais seulement les informations pertinentes qui sont transmises, permettant des gains importants en bande-passante et donc en énergie. Dans ce contexte, réaliser la fusion de capteurs via des modèles d'inférence Bayésienne simples offre explicabilité et gestion de l'incertitude alors que les données issues des dispositifs IoT sont souvent peu précises et fortement bruitées.

Pour ces applications contraintes en énergie et de faibles précisions, l'arithmétique stochastique apparaît comme un bon candidat pour représenter et manipuler des valeurs de probabilités à moindre coût logique.

Le calcul stochastique dédié à l'inférence Bayésienne

Le calcul stochastique représente les nombres par la fréquence d'apparition de bits à '1' sur une chaîne de bits, il permet de ce fait de simplifier des opérateurs complexes comme la multiplication par une simple porte logique ET. Cette propriété et le fait que les nombres représentés correspondent déjà intrinsèquement à des probabilités en font une piste particulièrement intéressante pour implémenter des circuits dédiés à l'IA probabiliste et à l'inférence Bayésienne en particulier.

Les projets de recherche BAMBI [27] et MicroBayes [28] ont posé les bases d'architectures stochastiques dédiées à l'inférence Bayésienne. Celles-ci intègrent des mémoires de distributions de probabilité du modèle d'inférence, des générateurs de nombres aléatoires (RNGs) pour produire les nombres stochastiques, ainsi qu'une matrice de multiplications à base de portes ET. Les premiers travaux de

deux thèses [29, 30] montrent des résultats intéressants lorsque les dimensions du problème et la précision nécessaire sont faibles. Cependant, ils mettent en exergue deux principaux goulots d'étranglement lors du passage à l'échelle :

— les distributions de probabilité du modèle d'inférence nécessitent de grandes quantités de mémoire dont la part de la surface sur le circuit total devient rapidement dominante lorsque la précision augmente,

— la consommation dynamique du circuit est majoritairement due à la génération de nombres stochastiques,

De plus, la convergence lente du calcul stochastique (en $1/\sqrt{n}$ selon le théorème central limite) implique un temps de calcul élevé et donc une consommation énergétique significative, dont il n'est pas toujours évident qu'elle soit inférieure à celle d'un calcul équivalent réalisé en virgule flottante (ou fixe) lorsque la précision requise est importante.

Contributions originales de ces travaux de thèse

Les travaux de cette thèse adressent ces trois problématiques :

— en repensant la manière de stocker les distributions de probabilité,

— en étudiant la qualité des nombres aléatoires nécessaires au calcul stochastique pour une précision donnée et en proposant une manière de générer les nombres stochastiques à faible coût énergétique,

— et en introduisant une méthode permettant de réduire le temps de calcul stochastique afin d'économiser de l'énergie.

De plus, nos travaux ont permis :

— de caractériser par des mesures un premier circuit de calcul d'inférences Bayésiennes avec des opérateurs stochastiques,

— de comparer, par des simulations rétro-annotées, les performances d'une nouvelle architecture améliorée par nos contributions avec : les performances de ce premier circuit caractérisé, celles d'un microcontrôleur ultra basse consommation ainsi que celles d'un circuit dédié intégrant une multiplication flottante non standard.

Plan de lecture

Le premier chapitre introduit le contexte théorique nécessaire à la bonne compréhension de ces travaux de thèse, en développant les notions de fusion de capteurs, d'inférence Bayésienne et de calcul stochastique. Un fil rouge à suivre tout le long de cette thèse, l'exemple de la localisation du bateau, est proposé afin de joindre à chaque notion théorique un exemple concret et de permettre une compréhension que nous espérons plus aisée.

Le chapitre 2 développe l'état de l'art de différentes méthodes que nous utilisons ou auxquelles nous comparons nos contributions. Il traite premièrement d'architectures stochastiques pour la fusion Bayésienne de capteurs. Ensuite, il établit un état de l'art sur les méthodes pour réduire, d'une part, le coût en consommation et en ressources logiques de la génération de nombres stochastiques et, d'autre part, l'énergie des circuits stochastiques. Enfin, il présente différents RNGs de la littérature scientifique qui peuvent être utilisés pour le calcul stochastique.

Les chapitres suivants abordent les contributions de cette thèse.

Le chapitre 3 traite du test et de la caractérisation d'un circuit de fusion Bayésienne de capteurs en représentation stochastique. Son architecture est celle présentée dans l'état de l'art. Notons que la conception et l'implémentation physique de ce circuit sont antérieures au début de la présente thèse, et s'appuient sur des bases d'architecture déjà décrites dans [31] et [32]. Une étude de la précision en fonction du temps de calcul stochastique est réalisée en fonction de différents jeux de données afin d'évaluer l'énergie consommée pour ces différents cas. La consommation énergétique est alors comparée à celle d'un microcontrôleur Renesas RE01. Cette étude confirme l'intérêt du calcul stochastique pour l'inférence Bayésienne lorsque la précision est faible, mais souligne également les trois goulots d'étranglements que sont les mémoires, la génération de nombres stochastiques et le temps de calcul élevé induisant une grande consommation d'énergie.

Le chapitre 4 introduit des solutions pour réduire la taille des mémoires de distribution des modèles d'inférence. Nous proposons tout d'abord une réorganisation de la manière de stocker ces distributions notamment en séquentialisant une partie du calcul, puis une méthode de compression mémoire permettant de diviser par 4 le nombre de mots à stocker.

Le chapitre 5 aborde différentes méthodes pour réduire l'impact de la génération de nombres stochastiques sur la surface et la consommation du circuit. Premièrement, une adaptation de l'architecture originale à l'état de l'art stochastique est effectuée. Puis une étude comparative de différents RNGs sur la précision du circuit est effectuée afin d'étudier la qualité d'aléa suffisante pour une précision donnée. Enfin, nous introduisons le *Shift Register Isolator*, une architecture partageant un seul RNG sur tout le circuit, permettant ainsi des gains en surface et en consommation électrique.

Le chapitre 6 fait part d'une proposition pour réduire la forte consommation énergétique des circuits stochastiques : le calcul multirail. Il s'agit d'une architecture stochastique parallèle semblable à celles développées dans l'état de l'art. Celle-ci s'appuie cependant sur les différentes optimisations du chapitre 5, notamment du *Shift Register Isolator*, ainsi que sur des permutations, permettant de générer de manière optimale les nombres aléatoires nécessaires aux différents rails.

Le chapitre 7 fait le bilan des différentes contributions de la thèse et compare, en termes de surface et d'énergie consommée, l'architecture proposée à l'architecture originale de l'état de l'art, au microcontrôleur RE01 et à une multiplication flottante non standard.

1

Contexte théorique

Ce chapitre présente les différentes notions théoriques nécessaires à la bonne compréhension des enjeux de ce manuscrit : la fusion de capteurs, l'inférence Bayésienne, le calcul stochastique et les métriques utilisées pour l'évaluation de la précision.

Sommaire

1.1 Fusion de capteurs

La fusion de données multi-capteurs, abrégée en fusion de capteurs, est une méthode d'intégration de données provenant de plusieurs sources afin de les synthétiser en une information plus pertinente. Ce principe est montré en figure 1.1.

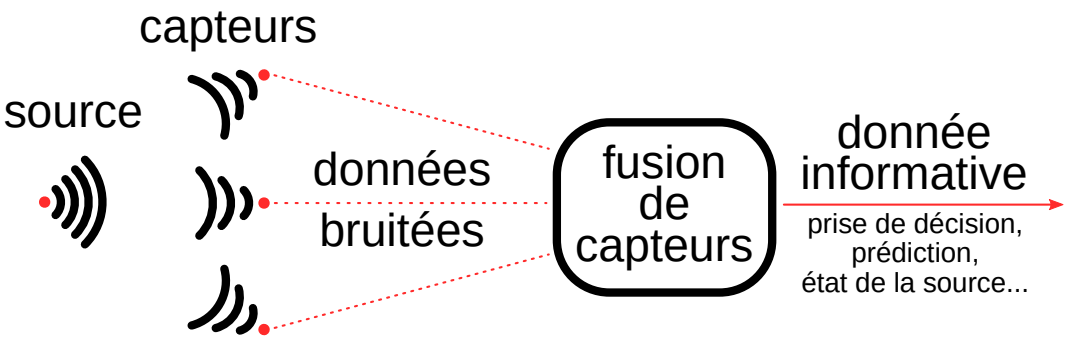

FIGURE 1.1 – *Principe de la fusion de capteurs.*

Elle fait partie intégrante de notre perception. Pour localiser une source sonore par exemple, nous utilisons notre audition binaurale pour mesurer à la fois la différence de temps (*Interaural Time Difference*), mais aussi la différence d'intensité (*Interaural Level Difference*) entre nos deux oreilles [33]. Ces informations fusionnées donnent une plus grande certitude quant à la position de la source. Si à cela s'ajoute la reconnaissance par la vue d'une forme susceptible d'être la source dans la direction présumée, la confiance en la localisation en sera d'autant plus grande.

La fusion de capteurs connaît un intérêt grandissant à l'heure de l'IoT (*Internet of Things*) où les données, bien que souvent peu précises, abondent, et qu'il est essentiel de synthétiser en informations exploitables. De plus, du fait des limites matérielles des dispositifs connectés, contraints en énergie, la fusion de capteurs permet un gain certain en bande-passante [34]. En effet, en réalisant le calcul au plus près du capteur, ce qu'on appelle l'*edge computing*, ces systèmes n'envoient au *cloud* que peu d'informations mais de meilleure qualité.

La fusion de capteurs est utilisée à des fins diverses comme le raffinement de données, la prise de décision ou la prédiction de comportements, et dans de nombreux domaines comme le diagnostic médical [35], la robotique [36], les véhicules autonomes [37], la domotique [38] ou les *smart cities* [39].

Notons qu'il existe une différence de terminologie avec la notion de *fusion de données*, qui englobe celle de *fusion de capteurs*, mais n'implique pas nécessairement l'intégration de données de capteurs différents, ni même de données capteurs brutes. C'est le cas par exemple pour des applications de raffinement de données qui peuvent intégrer plusieurs échantillons du même capteur pour réduire son bruit. Puisque la différentiation de telles notions importe peu dans le cadre de la thèse, nous nous permettons de substituer les termes dans ce manuscrit.

Du fait de l'intérêt grandissant de la fusion de capteurs, de nombreux chercheurs ont proposé une classification de tels algorithmes dont [40] fait état, en fonction de l'architecture de calcul, des types de données manipulées (niveaux d'abstraction) ou encore de la relation entre les données et les sources. Dans [41], Sasiadek pro-

pose une analyse en trois catégories suivant la nature de l'algorithme utilisé, qui se rapproche de notre classification de l'IA en introduction :

— la fusion par méthode des moindres carrés, qui peut être utilisée par exemple pour les filtres de Kalman [42], et que l'on peut classifier comme une approche du groupe de l'IA symbolique ;

— la fusion "intelligente" que l'on peut interpréter comme celle des algorithmes connexionnistes, comme les réseaux neuronaux ;

— la fusion probabiliste, via l'inférence Bayésienne, permettant la synthèse d'informations pertinentes, explicables et avec peu de données.

Pour ses qualités de frugalité en données et d'explicabilité des résultats, nous nous intéressons en particulier à la fusion probabiliste par inférence Bayésienne.

1.2 Inférence Bayésienne

Une inférence est un processus consistant à tirer une conclusion à partir de propositions appelées prémisses, considérées comme vraies. Elle se pose ainsi comme la base du raisonnement logique et de la théorie de la connaissance. En algèbre de Boole, où chaque proposition ne peut prendre que deux valeurs, soit Vrai soit Faux, la vérité des prémisses entraîne nécessairement la vérité de la conclusion : c'est l'inférence déductive (*e.g.* la démonstration mathématique). Cependant, ce cadre Booléen peut s'avérer parfois trop rigide, notamment lorsque l'on fait face à l'incomplétude des données, au bruit physique et aux erreurs de mesures. Il est alors nécessaire d'introduire la notion de probabilité d'un évènement. Mais qu'est-ce qu'une probabilité ?

Quand en 1950, un économiste chargé du budget de recherche pour l'U.S. Air Force demanda à David Blackwell, un éminent statisticien, quelle était la probabilité qu'un conflit majeur puisse avoir lieu dans les 5 ans à venir, ce dernier répondit :
«Cette question n'a pas de sens. La probabilité s'applique à une longue séquence d'évènements répétables, et ceci est clairement une situation unique. La probabilité est soit 0 soit 1, mais nous ne le saurons pas avant cinq ans.»
Ce à quoi l'économiste répondit :
«J'avais peur que vous disiez cela. J'ai parlé à plusieurs autres statisticiens, et ils m'ont tous dit la même chose.» [43]
Bien que Blackwell soit un statisticien des plus reconnus et célébrés, sa réponse paraît très insatisfaisante. La mésentente réside ici dans l'interprétation de ce qu'est une probabilité.

Pour Blackwell, une probabilité est la fréquence limite d'apparition d'un évènement lorsque l'expérience est réalisée à l'infini : c'est l'approche fréquentiste [1]. Une

1. Notons que c'est le point de vue de Blackwell à l'époque, mais il s'est "converti au Bayésianisme" selon ses mots durant la fin de sa carrière.

probabilité est alors considérée comme une vérité objective naturelle vers laquelle on tend par répétition d'expériences. C'est l'interprétation majoritaire, et celle à l'origine de l'outil statistique de la valeur-p, très utilisée en science. Celle-ci correspond à la vraisemblance $\mathbb{P}(D|H_0)$ des données observées (ou plutôt leur invraisemblance, la p-valeur étant espérée faible) en supposant vraie une hypothèse nulle H_0 (hypothèse représentant la norme). En dessous d'un certain seuil, la valeur-p permet de rejeter H_0 en procédant en quelque sorte à une extension aux probabilités de la preuve par l'absurde. Le principe de réfutation est le suivant : si les données sont extrêmement invraisemblables en supposant H_0 vraie, alors on doit rejeter H_0. Notons que quelle que soit la valeur de la p-valeur, elle ne permet pas de connaître la probabilité de l'hypothèse sachant les données : $\mathbb{P}(D|H_0) \neq \mathbb{P}(H_0|D)$.

Pour l'économiste, les probabilités se conçoivent comme la notion intuitive de plausibilité, un degré de croyance à donner à une proposition qui est révisé et affiné à mesure que de nouvelles observations sont réalisées : c'est l'approche Bayésienne. Une probabilité est donc de fait subjective de par toutes les expériences passées du sujet, mais permet une réponse même avec très peu de données répétables. L'inférence Bayésienne consiste ainsi à déterminer la probabilité de la conclusion en fonction de celle des prémisses, ou en d'autres termes, à évaluer la plausibilité d'une théorie à partir de données $\mathbb{P}(T|D)$, là où le fréquentisme s'intéresse à la vraisemblance des données en supposant la théorie vraie $\mathbb{P}(D|T)$. Cette approche Bayésienne, bien que plus intuitive, est peu utilisée en épistémologie car considérée comme trop subjective [2] et complexe à calculer car toutes les hypothèses doivent être envisagées (ou du moins les plus crédibles *a priori*).

Néanmoins, pour des applications concrètes, moyennant de potentielles simplifications du modèle probabiliste, elle offre de nombreux avantages comme :

— la gestion de l'incertitude des données,

— l'explicabilité des résultats, par opposition aux modèles de type boîte noire tels que les réseaux neuronaux,

— l'intégration des connaissances préalables sur le système étudié,

— la frugalité en données nécessaires à l'apprentissage.

L'inférence Bayésienne est ainsi utilisée en intelligence artificielle dans des domaines variés tels que le *data-mining* [44], les problèmes de diagnostic [45], de recommandation [46], ou encore pour la fusion de données capteurs [31, 32].

1.2.1 Principes fondamentaux des probabilités

Nous présentons ici brièvement les bases mathématiques de la théorie des probabilités utilisées dans ce manuscrit.

2. bien que le choix du seuil de réfutabilité pour la p-valeur puisse aussi être considéré comme arbitraire

Expérience aléatoire et évènement

Un modèle probabiliste est constitué d'un ensemble de résultats possibles pour une expérience aléatoire, appelé ensemble univers, noté Ω. Un exemple classique d'expérience aléatoire est le jet d'une pièce de monnaie, ou bien d'un dé à 6 faces, dont les ensembles univers respectifs sont :
$\Omega_{\text{pièce}} = \{\text{pile},\ \text{face}\}$ et $\Omega_{\text{dé}} = \{1,2,3,4,5,6\}$.

Un évènement correspond à un sous-ensemble de Ω. Par exemple, la proposition "Tirer un nombre premier au jet du dé" correspond à l'évènement $\{2,3,5\}$, et la probabilité de cet évènement, si le dé est équilibré, vaut $\frac{\text{card}(\{2,3,5\})}{\text{card}(\Omega)} = \frac{3}{6} = 0.5$.

Variable aléatoire

Une variable aléatoire X prend comme valeur $x \in \Omega$ avec la probabilité $\mathbb{P}(X = x) \in [0,\ 1]$. L'ensemble des valeurs de probabilité possibles de X : $\mathbb{P}(X)$ est appelé distribution de probabilité de X. Dans l'exemple d'un dé à 6 faces équilibré, une telle variable aléatoire peut représenter le résultat du dé, avec $\forall i \in [\![1,\ 6]\!]$, $\mathbb{P}(X = i) = \frac{1}{6}$. Dans ce cas, Ω est fini ou dénombrable, la variable aléatoire est dite discrète. X vérifie la propriété de normalisation :

$$\sum_{x \in \Omega} P(X = x) = 1$$

Dans le cas où l'ensemble univers est indénombrable, comme par exemple lorsque $\Omega = \mathbb{R}$, la variable est dite continue. On introduit alors la notion de densité de probabilité f_X de X définie telle que :

$$\forall a, b \in \Omega \mid a < b, \quad \mathbb{P}(a < X < b) = \int_a^b f_X(x)\,\mathrm{d}x$$

La propriété de normalisation devient alors :

$$\int_{x \in \Omega} f_X(x)\,\mathrm{d}x = 1$$

Les variables continues réelles suivent souvent des lois de probabilités usuelles, comme la loi uniforme sur $[a,b]$, notée $\mathscr{U}(a,b)$ dont la fonction de densité vaut :

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{si } x \in [a,b] \\ 0 & \text{sinon} \end{cases}$$

Une autre loi usuelle est la loi normale d'espérance μ et d'écart-type σ notée $\mathscr{N}(\mu,\sigma)$, dont la fonction de densité vaut :

$$\forall x \in \mathbb{R}, \quad f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \times \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Cette loi est très utilisée pour modéliser des phénomènes physiques (comme le bruit de mesure) que l'on estime être la conséquence de plusieurs facteurs car elle peut être considérée, selon le théorème central limite, comme la limite de la somme de variables aléatoires indépendantes dont aucune n'est prépondérante.

Conjonction

La distribution conjointe de deux variables aléatoires X et Y est notée $\mathbb{P}(X \wedge Y)$ et représente la distribution de probabilité lorsque les deux variables sont réalisées simultanément. Son cardinal est la multiplication des cardinaux des variables X et Y : $\text{card}(X \wedge Y) = \text{card}(X) \times \text{card}(Y)$.

Probabilité conditionnelle

La probabilité conditionnelle d'une variable X sachant Y est définie par l'équation (1.1) :

$$\mathbb{P}(X|Y) = \frac{\mathbb{P}(X \wedge Y)}{\mathbb{P}(Y)} \tag{1.1}$$

Cette probabilité conditionnelle évalue le degré de vraisemblance de la variable X au regard des informations données par l'état de la variable Y.

Indépendance

Deux variables X et Y sont dites indépendantes si $\mathbb{P}(X|Y) = \mathbb{P}(X)$ (et inversement), ou en d'autres termes si $\mathbb{P}(X \wedge Y) = \mathbb{P}(X) \times \mathbb{P}(Y)$. Ici, le fait de connaître l'état de Y n'apporte pas plus d'informations sur l'état de X. C'est le cas pour des jets consécutifs d'un dé équilibré, le fait de connaître le résultat du jet précédant n'apporte pas d'informations sur le résultat du jet actuel : les variables X_{t-1} et X_t sont donc indépendantes.

De la même manière, deux variables aléatoires X et Y sont dites conditionnellement indépendantes selon une troisième variable aléatoire Z si $\mathbb{P}(X|Y \wedge Z) = \mathbb{P}(X|Z)$, ou en d'autres termes si $\mathbb{P}(X \wedge Y|Z) = \mathbb{P}(X|Z) \times \mathbb{P}(Y|Z)$. Ce cas est fréquent lorsque la variable Z est la cause des variations de X et Y, comme dans le cas de la fusion de capteurs, où Z serait la source et X et Y des capteurs.

1.2.2 Théorème de Bayes

Soit I une variable aléatoire d'intérêt et O une variable aléatoire d'observation. À partir de la formule 1.1, et puisque la conjonction est symétrique, le théorème de Bayes s'écrit comme suit :

$$\mathbb{P}(I|O) = \frac{\mathbb{P}(I) \times \mathbb{P}(O|I)}{\mathbb{P}(O)} \tag{1.2}$$

$\mathbb{P}(I)$ est appelé le *prior*, il correspond à la connaissance *a priori* de la variable I.

$\mathbb{P}(O|I = i)$ est appelé une vraisemblance, et représente la probabilité de réalisation de la variable O en émettant l'hypothèse que l'évènement $I = i$ est réalisé. Elle est connue car donnée par le modèle de O qui décrit son comportement.

$\mathbb{P}(I|O)$ est appelé le *posterior*, il correspond au degré de croyance que l'on peut attribuer à la réalisation de la variable recherchée I à partir de l'observation réalisée O.

1.2.3 Fusion Bayésienne

Appliqué à un problème de fusion de données, le théorème de Bayes permet de déterminer la distribution d'une variable d'intérêt I en fonction d'un ensemble d'observations issues de n capteurs $\{O_k\}_{1\leq k\leq n}$. L'équation 1.2 devient :

$$\mathbb{P}(I|\wedge_{k=1}^{n} O_k) = \frac{1}{Z} \times \mathbb{P}(I) \times \mathbb{P}(\wedge_{k=1}^{n} O_k|I) \tag{1.3}$$

où Z est le facteur de normalisation et vaut $\mathbb{P}(\wedge_{k=1}^{n} O_k)$ soit la probabilité de la conjonction des observations. Il s'agit d'une constante qu'il n'est pas nécessaire de calculer en pratique, puisque pour obtenir une distribution de probabilité en normalisant, il suffit de diviser chaque terme par la somme des termes non normalisés. On effectue donc en général le calcul "à une constante multiplicative près", ce qui s'exprime en utilisant le symbole $\propto$ qui signifie "est proportionnel à".

En considérant I de cardinal m, et que les variables $\{O_k\}_{1\leq k\leq n}$ sont conditionnellement indépendantes selon I et donnent les n observations $\{o_k\}_{1\leq k\leq n}$, alors l'équation 1.3 devient :

$$\forall\ j \in [1,m],\ \ \mathbb{P}(I = i_j|\wedge_{k=1}^{n} O_k = o_k) \propto \mathbb{P}(I = i_j) \prod_{k=1}^{n} \mathbb{P}(O_k = o_k|I = i_j) \tag{1.4}$$

On remarque que la probabilité conditionnelle inférée, à savoir $\mathbb{P}(I|\wedge_{k=1}^{n} O_k)$ consiste uniquement en un ensemble de multiplications de distributions de probabilité conditionnelle connues $\mathbb{P}(O_k|I)$ et d'un prior $\mathbb{P}(I)$. Cela se traduit dans un circuit dédié par une organisation spatiale du calcul selon une matrice de multiplications dont le nombre de lignes vaut $N_{\mathrm{L}} = m$ et le nombre de colonnes vaut $N_{\mathrm{C}} = n+1$ (+1 pour le *prior*). Rappelons que $m = Card(I)$ est le nombre des valeurs possibles de la variable d'intérêt I, et n le nombre de variables aléatoires connues. Cette architecture dédiée à la fusion Bayésienne de capteurs est développée en section 2.1.

Exemple du bateau :
Afin de rendre la compréhension du problème de la fusion de données plus intuitive, nous proposons de l'appliquer au cas concret de la localisation d'un bateau, schématisé en figure 1.2.
Un marin en expédition cherche à se localiser. Il dispose dans son bateau de capteurs de faible précision mesurant des informations relatives à trois phares (ou amers) situés sur la côte. Ces capteurs mesurent d'une part leur distance au phare, et d'autre part, l'angle formé par le phare et l'horizontale du capteur (parallèle à l'axe des abscisses). La localisation du bateau peut ainsi être obtenue en fusionnant les informations de 6 capteurs, 3 capteurs de distance D_k et 3 capteurs d'angle Θ_k.

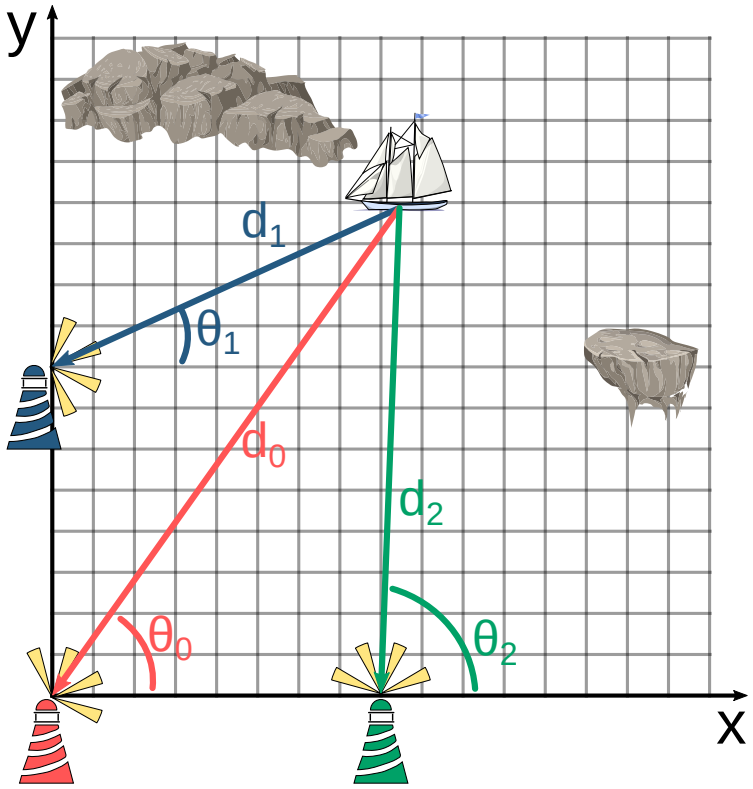

FIGURE 1.2 – *Schéma du problème de localisation d'un bateau par fusion de capteurs de distance et d'angle.*

La variable d'intérêt I correspond aux coordonnées (X, Y) du bateau, que l'on discrétise en une grille de 16 par 16. Elle peut donc prendre 256 valeurs.
Faire le calcul d'inférence $\mathbb{P}(I = i_j | \wedge_{k=1}^{n} O_k = o_k)$ revient à calculer la probabilité de présence du bateau pour chaque cellule $i_j = (x,y)_j$ de la grille en fonction des données des capteurs de distance et d'angle.
Le prior $\mathbb{P}(I)$ correspond à une connaissance *a priori* de la localisation du bateau. Par exemple dans la figure 1.2, connaissant la topologie de la côte, on peut estimer que la probabilité que le bateau se trouve sur un rocher est très faible voire nulle (à moins qu'il se soit échoué).
Les vraisemblances $\mathbb{P}(O_k|I)$ sont données à partir des valeurs attendues de chaque capteur pour chaque cellule de la grille ; du bruit de mesure des capteurs (potentiellement différent selon la nature du capteur : distance ou angle) ; et des mesures d_k et θ_k réalisées par ces capteurs. La méthode de calcul de telles vraisemblances est plus amplement détaillée en section 2.1.3.

Ce calcul exhaustif de la distribution du *posterior*, ou calcul d'inférence exacte, est possible lorsque le nombre de valeurs que peut prendre la variable d'intérêt I (son cardinal m) est faible, sans quoi le problème devient rapidement intractable. Lorsque la dimension du problème est plus grande, il est nécessaire de procéder autrement par un calcul d'inférence approchée. Dans ce cas, les méthodes par échantillonnage de type MCMC (Monte Carlo Markov Chain) sont plus adaptées. Leur

utilisation sort du cadre de cette thèse, mais le lecteur curieux pourra en trouver une description dans ce livre [47] qui leur est dédié.

Le fait que ce calcul d'inférence fasse intervenir exclusivement l'opérateur de multiplication entre distributions de probabilité (voir équation 1.4) motive la recherche de solutions pour implémenter efficacement cet opérateur dans un circuit. Comme nous allons le voir, c'est exactement ce que permet le calcul stochastique.

1.3 Calcul Stochastique

Le calcul stochastique a été introduit par von Neumann dès 1956 [48], et popularisé par les travaux de Gaines [49] dans les années 1970. Mais ce n'est que récemment que l'arithmétique stochastique a connu un regain d'intérêt, dans des domaines variés comme le traitement d'images [50,51], le traitement du signal [52,53], les réseaux de neurones [54,55], ou encore le calcul d'inférences Bayésiennes [31,32,56], notamment pour sa tolérance aux fautes [57] et son efficacité matérielle. En effet, cette représentation des nombres non standard permet des gains en surface et en puissance grâce à une logique simplifiée, en contrepartie d'une précision moindre et d'un temps de calcul plus long.

1.3.1 Principe

Les nombres stochastiques sont représentés sur une chaîne de bits ou *bitstream* d'une certaine longueur, et leur valeur est codée par la fréquence f_1 d'apparition du bit '1' dans ce *bitstream*. En représentation stochastique bipolaire, la valeur codée vaut $2 \times f_1 - 1$, elle est donc comprise dans le segment $[-1,1]$. En représentation unipolaire, la valeur codée correspond directement à cette fréquence f_1, elle est donc comprise dans le segment $[0,1]$. Le tableau 1.1 donne quelques exemples de *bitstreams* et leur valeur codée en représentation bipolaire et unipolaire.

Bitstream	Valeur bipolaire	Valeur unipolaire
01100000	$-4/8$	$2/8$
01101001	0	$4/8$
10111111	$6/8$	$7/8$

TABLEAU 1.1 – *Représentation bipolaire et unipolaire de différents bitstreams*

La représentation unipolaire est la plus répandue, elle est celle que nous utilisons puisque nous souhaitons représenter des valeurs de probabilités (qui sont donc toutes comprises entre 0 et 1).

Remarquons que plus long est le *bitstream*, plus on pourra coder de valeurs, et donc plus la précision de ce codage sera importante.

Les *bitstreams* (en français, flux de bits) sont généralement produits en série, c'est-à-dire que les différents bits qui le composent correspondent à la variation de l'état d'un unique bit matériel (fil) dans le temps. De ce fait, augmenter la précision du calcul revient à augmenter le temps de calcul. Le calcul stochastique peut ainsi être considéré comme une méthode structurelle de calcul approché (ou *Approximate Computing* [58]).

1.3.2 Génération de nombres stochastiques

Ces *bitstreams* sont générés grâce à des générateurs de nombres stochastiques, ou Stochastic Number Generators (SNGs), qui sont généralement constitués :

— d'un générateur de nombres aléatoires (RNG - Random Number Generator),

— et d'un convertisseur binaire / stochastique, permettant la génération d'un *bitstream* à partir d'une valeur de biais binaire et d'un nombre aléatoire.

Générateur de nombres aléatoires

Il existe deux types de RNGs :

— les générateurs de nombres pseudo aléatoires, ou Pseudo Random Number Generators (PRNGs), qui génèrent une séquence de nombres en apparence aléatoires de manière algorithmique, et qui est en ce sens totalement déterministe ;

— les générateurs de nombres réellement aléatoires, ou True Random Number Generators (TRNGs), dont la source d'entropie provient de phénomènes physiques comme le bruit thermique d'un circuit.

Le RNG le plus communément utilisé pour le calcul stochastique est le Linear Feedback Shift Register (LFSR) [59] pour son efficacité matérielle. Il s'agit en effet d'un PRNG très simple ne nécessitant que n bascules et quelques portes XORs pour générer un nombre aléatoire de n bits. L'architecture et le fonctionnement de ce LFSR, ainsi que ceux de plusieurs autres RNGs sont décrits en section 2.3.

Conversion binaire / stochastique

La conversion d'un nombre binaire (consigne de biais) en *bitstream* à partir d'un nombre aléatoire tiré de manière uniforme est réalisée grâce à ce que l'on appelle un convertisseur binaire / stochastique. Celui-ci correspond généralement à un simple comparateur, la conversion étant faite par tirage de Bernoulli. Si le nombre aléatoire est inférieur à la consigne, alors le comparateur donne un '1', sinon il donne un '0'. De cette manière, la sortie du comparateur donnera un '1' avec une probabilité $\mathbb{P} = biais/2^{res_p}$, res_p étant la taille en bits de la donnée de biais, comme montré en figure 1.3 où res_p = 4 bits. Cette méthode est la plus largement utilisée pour sa simplicité, la rendant très efficace en termes de ressources logiques utilisées.

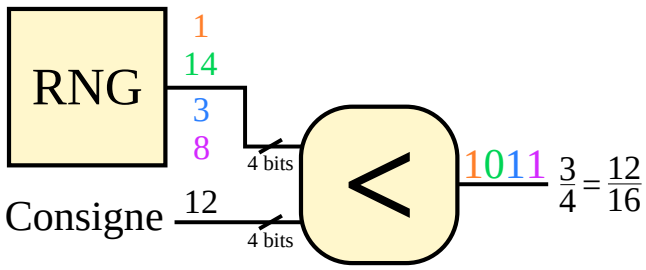

FIGURE 1.3 – *Conversion binaire / stochastique par tirage de Bernoulli.*

Une autre manière de réaliser cette conversion binaire / stochastique utilisant le Weighted Binary Generator est décrite en section 2.2.2.

Conversion stochastique / binaire

La conversion inverse est réalisée grâce à des compteurs, l'un comptant le nombre nb_1 de bits à '1' dans le *bitstream* concerné, et l'autre comptant le nombre total de cycles écoulés (la longueur totale L du *bitstream*). On retrouve alors la valeur du *bitstream* en binaire par la division : $\frac{nb_1}{L}$. Cependant, pour certaines applications comme la fusion Bayésienne de capteurs, il n'est pas nécessairement requis de connaître la valeur exacte des *bitstreams* de sortie, car le calcul de la distribution de probabilité recherchée peut se faire à une constante près. À titre d'exemple, il n'est effectivement pas nécessaire de normaliser la distribution pour identifier la valeur de probabilité maximale, ni pour en tirer un échantillon. Ainsi, l'implémentation de la division, très coûteuse en matériel, n'est pas requise.

1.3.3 Intérêts et spécificités du calcul stochastique

Réduction des ressources logiques

La représentation stochastique des nombres permet principalement de simplifier grandement des opérations qui peuvent s'avérer complexes à implémenter en matériel, comme la multiplication. En effet, en représentation unipolaire, la multiplication de deux *bitstreams* indépendants est réduite à une simple porte logique ET. La figure 1.4 montre ce principe sur un *bitstream* de longueur 8.

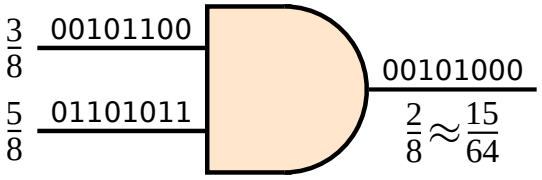

FIGURE 1.4 – *Implémentation de la multiplication en logique stochastique.*

Cette simplification des ressources logiques utilisées se traduit par une réduction de la surface silicium, et donc des coûts de production et de la consommation statique, mais aussi de la puissance instantanée consommée.

Aléa, corrélation et précision

Les nombres stochastiques sont communément générés grâce à des RNGs afin de garantir l'indépendance (ou du moins la faible corrélation) requise entre les opérandes. La corrélation entre deux *bitstreams* est mesurée par la *Stochastic Computing Correlation* [60] décrite en section 1.4.3. Différentes initialisations de RNGs (graines) peuvent donner des *bitstreams* plus ou moins corrélés, le cas extrême étant des graines de même valeur. Une étude préalable de la corrélation des *bitstreams* selon les graines des RNGs peut alors être effectuée [61] pour identifier l'initialisation introduisant la corrélation minimale.

Si la génération de nombres aléatoires est une condition suffisante pour réduire la corrélation entre *bitstreams*, elle n'est cependant pas nécessaire. Il existe en effet des générateurs de nombres s'appuyant sur le fin contrôle de tous les nombres du

circuit de manière totalement déterministe pour converger plus rapidement vers la consigne de biais : ce sont les *Low Discrepancy* SNGs décrits en section 2.2.3.

Enfin, bien que la faible corrélation semble être une condition nécessaire et suffisante pour obtenir une bonne précision de calcul dans le cas de la multiplication stochastique [62], cela n'est pas nécessairement le cas pour d'autres fonctions logiques. Ainsi, la corrélation entre *bitstreams* peut être exploitée [60,63] pour réaliser des opérations logiques de manière tout autant précise qu'en utilisation classique (avec des *bitstreams* décorrélés). Une porte logique peut alors remplir plusieurs fonctions suivant la corrélation de ses *bitstreams* d'entrée. Par exemple, nous avons vu qu'une porte logique ET remplit la fonction de multiplication lorsque les *bitstreams* d'entrée sont indépendants (voir figure 1.4). Mais lorsque les nombres sont fortement corrélés, la porte ET remplit la fonction du minimum. À titre d'exemple, soit $S = 01100100$ et $S' = 01110101$ deux *bitstreams* fortement corrélés de biais respectifs $P(S) = \frac{3}{8}$ et $P(S') = \frac{5}{8}$, alors le *bitstream* de sortie de la porte ET est $S \wedge S' = 01100100$ de biais $P(S \wedge S') = \frac{3}{8} = P(S) = min(P(S), P(S'))$.

La figure 1.5 montre les relations logiques entre aléa, corrélation et précision (la flèche est à interpréter comme une implication).

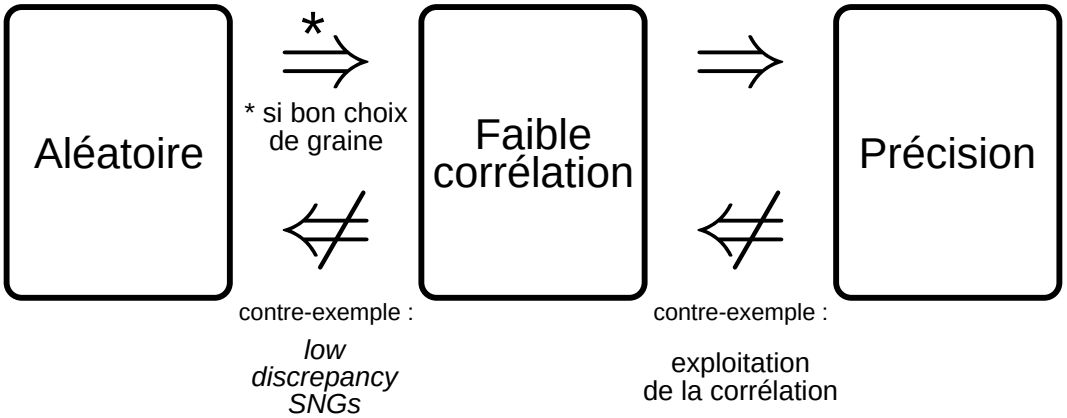

FIGURE 1.5 – *Relations logiques entre aléa, corrélation et précision.*

Robustesse

Un autre avantage de la représentation stochastique est sa robustesse aux fautes transitoires et aux effets de l'environnement comme les variations PVT (*process, voltage and temperature*) [57]. En effet, l'inversion d'un bit sur un *bitstream* de taille N aura très peu d'impact sur la valeur représentée, l'erreur produite sera de $1/N$. Or N peut être très grand. A contrario, dans une représentation classique des nombres, l'inversion d'un bit peut générer des erreurs non négligeables sur la valeur représentée, en particulier s'il s'agit d'un bit de poids fort. Cette robustesse de calcul peut permettre de relâcher les contraintes sur l'alimentation du système ainsi que sur les contraintes temporelles jusqu'alors nécessaires pour s'assurer de la validité des calculs. De cette manière, il pourrait être possible de sortir des marges prévues par les outils de conception en utilisation standard sans que cela n'impacte de manière significative la précision du calcul.

Précision progressive

De plus, le calcul stochastique permet ce qu'on appelle la précision progressive. En effet, contrairement à la représentation binaire standard, ici les résultats en sortie s'affinent dans le temps, au fur et à mesure que sont générés les bits du *bitstream*, ce qui peut être intéressant pour une application où l'on recherche d'abord rapidement une valeur approchée, avant d'affiner ce résultat. Le calcul stochastique offre donc un compromis entre précision et temps de calcul, chose impossible en représentation standard, la précision étant fixée.

Dilution temporelle

Ce dernier point fait aussi cependant les limites de l'arithmétique stochastique car, à précision égale, son temps de calcul est en moyenne beaucoup plus long que pour la représentation classique. Dans le cas de la multiplication stochastique, cela est d'autant plus vrai que le nombre d'opérations à réaliser grandit : c'est ce que l'on appelle la *dilution temporelle*. En effet, à l'instar d'une représentation en virgule fixe, enchaîner les multiplications de nombres entre 0 et 1 demande une grande précision afin de représenter les résultats de manière correcte. Cette précision élevée requise se traduit en logique stochastique par un temps de calcul plus long.

Grande consommation énergétique

Ce temps de calcul élevé de la logique stochastique peut contrebalancer ses bonnes performances en termes de puissance, puisque l'on demande certes moins de puissance que l'arithmétique classique, mais on la demande sur un intervalle de temps plus long. L'énergie consommée par le calcul peut donc s'en retrouver affectée. Cependant, cela tend à s'atténuer pour des précisions moindres, et le calcul stochastique est visiblement avantageux sur le plan énergétique lorsque l'on souhaite effectuer du calcul à faible précision.

Résumé des avantages et inconvénients

Ainsi, en comparaison à une représentation classique des nombres, le calcul stochastique peu paraître soit trop peu précis, soit trop long. Mais pour des applications où la rapidité de calcul et la précision ne sont pas une priorité, il peut s'avérer très intéressant car il permet deux choses. D'une part, une réduction conséquente de la surface silicium, ce qui se traduit par un coût de production plus faible. D'autre part, il donne lieu à une réduction de la consommation électrique par la simplification des multiplicateurs et le gain en robustesse.

Dans des solutions d'*approximate computing* comme le calcul stochastique, l'énergie consommée est étroitement liée à la précision car toutes deux dépendent du temps de calcul. Pour pouvoir faire des comparaisons de performances, il est alors essentiel de définir et d'utiliser des jeux de données précis (voir section 3.2.2) et des métriques de précision adaptées qui sont définies dans la section suivante.

1.4 Métriques pour la mesure de précision

Nous faisons état ici des différentes métriques de précision utilisées pour comparer les résultats de calculs obtenus avec différentes architectures. D'une manière générale dans cette section, on note $\mathbb{P}$ la distribution approximée expérimentale, et $\mathbb{Q}$ correspond à la distribution réelle de référence.

1.4.1 Racine de l'Erreur Quadratique Moyenne

Le calcul de la racine de l'erreur quadratique moyenne (RMSE - *Root-Mean-Squared Error*) est une mesure d'erreur très utilisée pour sa simplicité et son interprétabilité. Pour la comparaison de distributions $\mathbb{P}$ par rapport à la référence $\mathbb{Q}$, elle est définie par :

$$RMSE(\mathbb{P}||\mathbb{Q}) = \sqrt{\frac{1}{Card(\mathbb{P})} \times \sum_{j=1}^{Card(\mathbb{P})} (\mathbb{P}(I = i_j) - \mathbb{Q}(I = i_j))^2}$$

Contrairement à l'erreur absolue moyenne, cette mesure n'est pas linéaire et donne un poids plus élevé aux grandes erreurs (on considère qu'un écart à la référence de 6 est plus de 2 fois pire qu'un écart de 3).

La RMSE est dépendante du jeu de données et des unités utilisées. En ce sens, elle ne peut être utilisée que de manière relative, pour comparer des cas de test similaires. Pour s'en servir comme d'une mesure absolue, il est nécessaire de la normaliser par l'étendue des données ($v_{max} - v_{min}$), on parle alors de *Normalized Root-Mean-Squared Error* (NRMSE). Cependant, comme nous manipulons des probabilités, nous sommes ici dans le cas particulier où $v_{max} - v_{min} = 1$ et donc où la RMSE correspond aussi à la NRMSE.

1.4.2 Divergence de Kullback-Leibler

Une autre métrique, basée sur la théorie de l'information, et spécifique à la comparaison de distributions de probabilité, est la divergence de Kullback-Leibler (KLD). Elle mesure la différence entre une distribution étudiée $\mathbb{P}$ et une distribution de référence $\mathbb{Q}$ par la quantité d'information perdue si on utilisait $\mathbb{Q}$ pour approximer $\mathbb{P}$. Elle est définie par :

$$KLD(\mathbb{P}||\mathbb{Q}) = \sum_{j=1}^{Card(\mathbb{P})} \mathbb{P}(I = i_j) \times \texttt{log2}\left(\frac{\mathbb{P}(I = i_j)}{\mathbb{Q}(I = i_j)}\right)$$

La KLD est toujours positive. Plus elle est faible, plus les distributions sont proches, et donc meilleure est la précision. La KLD est aussi liée à la discrimination des données : à RMSE constante, une distribution proche de la loi uniforme aura une KLD plus élevée qu'une distribution "piquée".

1.4.3 Stochastic Computing Correlation

Dans [60], Alaghi *et al.* ont introduit une mesure de la corrélation entre deux *bitstreams* X et Y : la *Stochastic Computing Correlation* (SCC). Elle est définie par :

$$SCC(X,Y) = \begin{cases} 0 & \text{si } \delta(X,\,Y) = 0 \\ \frac{\delta(X,\,Y)}{\min(p_X,\,p_Y) - p_X \times p_Y} & \text{si } \delta(X,\,Y) > 0 \\ \frac{\delta(X,\,Y)}{p_X \times p_Y - \max(p_X + p_Y - 1,\,0)} & \text{si } \delta(X,\,Y) < 0 \end{cases}$$

où p_X et p_Y sont respectivement les valeurs codées par les *bitstreams* X et Y, $p_{X \wedge Y}$ est la valeur codée par le *bitstream* en sortie d'une porte ET avec X et Y comme entrée, et $\delta(X,Y) = p_{X \wedge Y} - p_X \times p_Y$.

La mesure de SCC varie dans le segment $[-1;\,1]$. SCC $= 0$ signifie que les deux *bitstreams* sont non corrélés, SCC $= 1$ signifie qu'ils sont égaux, et SCC $= -1$ qu'ils sont opposés, c'est-à-dire que chaque bit de X à '0' correspond à un '1' pour Y et vice versa.

Pour calculer la corrélation entre deux générateurs de nombres stochastiques (SNGs) S et S', nous calculons la SCC moyenne sur toutes les valeurs possibles de X et Y, comme défini dans [64] :

$$SCC_{\text{paire}}(S,S') = \sum_{i=0}^{2^{\text{res}_\text{p}}-1} \sum_{j=0}^{2^{\text{res}_\text{p}}-1} \frac{|SCC(S_i,S'_j)|}{(2^{\text{res}_\text{p}} \times 2^{\text{res}_\text{p}})}$$

où res_p est la résolution des données (en bits), et S_i et S'_j sont les *bitstreams* générés avec le SNG S (respectivement S') et avec la valeur codée $p_{S_i} = i/2^{\text{res}_\text{p}}$ (respectivement $p_{S'_j} = j/2^{\text{res}_\text{p}}$). Ainsi, SCC_{paire} est toujours positive, et plus elle est faible, moins les SNGs sont corrélés.

Pour mesurer la qualité stochastique d'un ensemble de n SNGs, nous calculons SCC_{moy}, soit la moyenne sur les $\binom{n}{2}$ SCC_{paire} deux à deux. En pratique, ce calcul devient rapidement intractable à mesure que le nombre de SNGs augmente, c'est pourquoi il est souvent effectué avec une étude de Monte Carlo sur un grand nombre de données aléatoires.

1.4.4 Taux de Reconnaissance du Maximum de distribution

Dans certaines applications, la bonne fidélité de représentation de la distribution du *posterior* n'est pas nécessaire car seule la recherche du maximum de distribution compte. C'est le cas d'applications de classification ou bien de localisation comme dans l'exemple du bateau. En effet, dans ce cas, l'inférence Bayésienne calcule une distribution de probabilité sur toutes les positions possibles, et pour en retenir une seule, celle qui intéresse le marin, on choisit la plus probable : le maximum de distribution.

Nous introduisons donc dans cette thèse la métrique du Taux de Reconnaissance du Maximum de distribution (TRM) qui est liée au cas de test de recherche du maximum, et dont un jeu de données est décrit en section 3.2.2. Cette métrique consiste

à répéter n_{rep} fois un calcul d'inférence dont la variable d'intérêt a une valeur fixée i_{true} : c'est la vérité-terrain (ou *ground-truth* en anglais). On compte alors le nombre de fois où le maximum de la distribution du *posterior* correspond effectivement à la vérité-terrain : $n_{\text{reco}} = \sum_{exp}^{n_{\text{rep}}} (i_{\max_{exp}} == i_{\text{true}})$. Le TRM correspond alors au ratio du nombre de reconnaissances du maximum par le nombre de répétitions :

$$TRM = \frac{n_{\text{reco}}}{n_{\text{rep}}}$$

Dans l'exemple du bateau, cette métrique permet de calculer la fréquence à laquelle la fusion des 6 capteurs aboutit au bon résultat i_{true} de la vérité terrain : la position réelle du bateau pour un observateur omniscient.

2
État de l'art

Ce chapitre présente l'état de l'art des circuits stochastiques dédiés à la fusion Bayésienne de capteurs, ainsi que des architectures de calcul stochastique et des générateurs de nombres aléatoires (RNGs).

Sommaire

Nous avons introduit dans le chapitre précédent les principaux concepts théoriques manipulés dans le cadre de cette thèse, qui s'intéresse à la conception de circuits numériques dédiés aux calculs d'inférence Bayésienne nécessaires à la résolution des problèmes de fusion de capteurs.

Dans ce chapitre, nous effectuons une brève revue de la littérature pour faire part :

— des architectures existantes de calcul stochastique pour l'inférence Bayésienne,

— des solutions pour réduire la surface et la consommation énergétique des circuits stochastiques,

— de différents RNGs possiblement utilisables avec le calcul stochastique.

2.1 Architecture stochastique pour la fusion Bayésienne de capteurs

De nombreux travaux se sont intéressés à développer des circuits pour l'inférence Bayésienne (voir [65] pour une revue), que ce soit de manière totalement numériques sur des circuits FPGA [66, 67] ou via des circuits dédiés [68], avec des méthodes de calcul analogique [69, 70] ou bien même de manière quantique [71]. D'autres encore ont orienté leur recherches sur des architectures stochastiques [56, 72–75], tirant parti de la nature probabiliste de cette représentation et de ses gains importants en termes de ressources logiques. [31] et [32] introduisent une proposition particulière d'architecture stochastique pour la fusion Bayésienne de capteurs qui a servi de point de départ pour les travaux de cette thèse, et qui est développée dans cette section.

2.1.1 Rappels théoriques

Un circuit dédié à la résolution de la fusion Bayésienne de capteurs peut s'implémenter efficacement sous-forme d'une matrice de multiplications de vraisemblances. Ces vraisemblances sont obtenues à partir du modèle des capteurs. Les nombreuses multiplications peuvent être efficacement effectuées en représentation stochastique permettant des gains en surface et consommation de puissance.

De telles architectures sont composées, d'une part, d'un bloc permettant la lecture en mémoire des vraisemblances associées aux données capteurs (que l'on nomme par la suite "génération des vraisemblances") et d'autre part du coeur de calcul stochastique.

2.1.2 Coeur de calcul stochastique

Matrice de multiplications

Selon l'équation de fusion Bayésienne :

$$\forall\, j \in [1, N_\mathrm{L}],\ \ \mathbb{P}(I = i_j \,|\, \wedge_{k=1}^{N_\mathrm{O}} O_k = o_k) \propto \mathbb{P}(I = i_j) \prod_{k=1}^{N_\mathrm{O}} \mathbb{P}(O_k = o_k | I = i_j)$$

pour un problème dont la variable d'intérêt prend N_L valeurs et qui fusionne N_O capteurs, il est nécessaire de réaliser N_L fois en parallèle la multiplication d'un *prior* et de N_O vraisemblances. Ainsi, on obtiendrait en sortie une distribution de N_L valeurs de probabilité *a posteriori* de la variable d'intérêt sachant les N_O observations o_k, comme montré en figure 2.1.

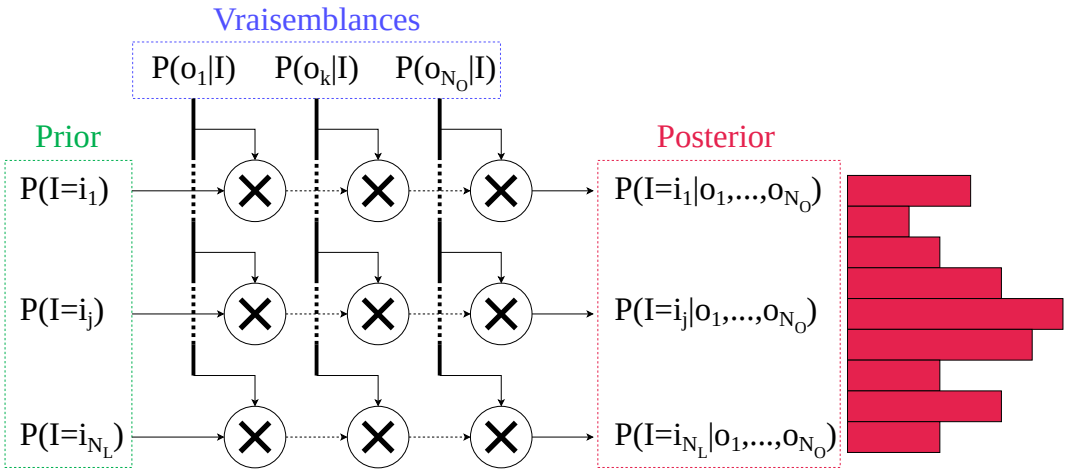

FIGURE 2.1 – *Matrice de multiplications pour la résolution de fusion Bayésienne de capteurs.*

Matrice stochastique

Pour représenter ces valeurs de probabilités en stochastique, il faudrait implémenter *a priori* un RNG différent pour le *prior* et pour chaque vraisemblance. Cependant, les lignes de la matrice sont indépendantes du point de vue du calcul. Cette indépendance des calculs permet de réutiliser les mêmes nombres aléatoires pour les différentes lignes, ce qui se fait en partageant chaque RNG sur toute une colonne.

L'architecture du coeur de calcul stochastique est montrée en figure 2.2.

La matrice est divisée en $N_\mathrm{L} \times N_\mathrm{C}$ briques élémentaires appelées cellules Bayésiennes ($N_\mathrm{C} = N_\mathrm{O} + 1$ pour le *prior*).

Chaque cellule Bayésienne intègre :

— un registre stockant le biais (le *prior* ou la vraisemblance correspondante),

— un convertisseur binaire / stochastique (*e.g.* un comparateur) prenant en entrée le biais de la cellule et le nombre aléatoire de la colonne,

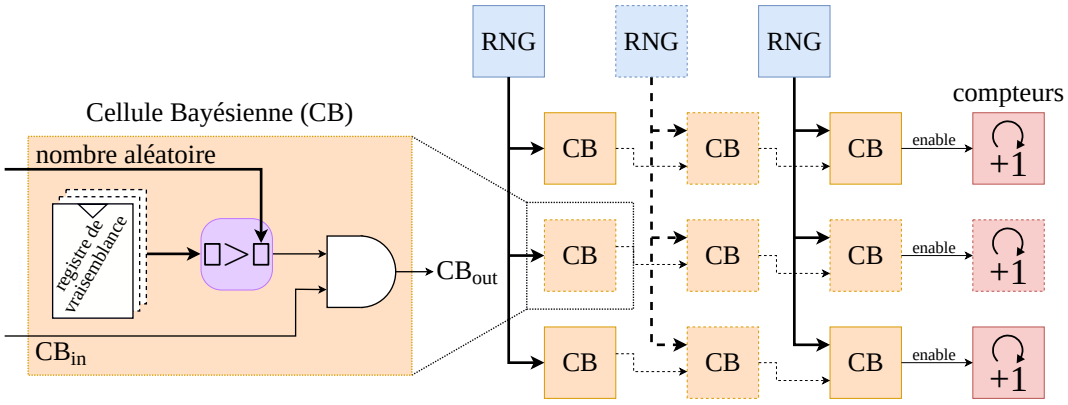

FIGURE 2.2 – *matrice de multiplications pour la résolution de fusion Bayésienne.*

— une porte ET [1] réalisant la multiplication entre ce *bitstream* généré et le *bitstream* en sortie de la cellule précédente.

Au bout de chaque ligne, un compteur s'incrémente à chaque '1' dans le *bitstream* de sortie, convertissant la probabilité inférée de la représentation stochastique vers la représentation binaire.

Normalisation par le maximum [2]

En réalité, la distribution obtenue ne correspond pas à la réelle distribution, de part l'approximation de l'équation 1.4, mais elle lui est proportionnelle. Pour retrouver la vraie valeur de probabilité, il faut diviser la valeur du compteur de sortie par la somme de tous les compteurs : $\mathbb{P}_{real\ j} = \frac{count_j}{\sum_i^{N_L} count_i}$. Ce calcul est néanmoins rarement effectué car bien souvent seul le rapport de proportionnalité entre les probabilités de la distribution est utile.

Ce principe, aussi développé en section 2.2.3, offre la possibilité de normaliser par le maximum, c'est-à-dire d'appliquer une constante multiplicative aux valeurs de *priors* et de vraisemblances par colonne avant le calcul d'inférence, permettant des valeurs plus élevées et donc des temps de calcul plus courts à précision égale. Ainsi, pour une distribution de *prior* uniforme (on ne sait rien *a priori* de l'état de la variable d'intérêt I), au lieu de générer des *bitstreams* de biais $1/card(I)$, nous générons des *bitstreams* de biais maximaux $\frac{2^{res_P}-1}{2^{res_P}}$, réduisant considérablement le temps de calcul stochastique, tout en gardant le rapport de proportionnalité entre les lignes.

1. En réalité, les multiplications de chaque ligne sont effectuées par une grande porte ET à plusieurs entrées en bout de ligne plutôt qu'une porte ET à 2 entrées par cellule.
2. Notons qu'ici, le terme de normalisation ne fait pas référence au fait que la somme des probabilités doit sommer à 1, mais plutôt à l'application d'une constante multiplicative par colonne.

Exemple du bateau :
Pour la localisation du bateau, la matrice de multiplications comporte $N_C = N_O + 1 = 7$ colonnes, avec 1 colonne de *prior*, 3 colonnes pour les capteurs de distance et 3 autres pour les capteurs d'angle. Elle sera aussi composée de $N_L = 256$ lignes, correspondant aux 16×16 coordonnées possible des variables d'intérêt $I = (X, Y)$.
Dans chaque cellule de la matrice, on stocke le *prior* (si on se trouve en première colonne), ou la vraisemblance correspondante normalisés.
Les 256 compteurs au bout des lignes s'incrémentent au fur et à mesure, et rendent une distribution de la plausibilité de la position du bateau à partir des données de distance et d'angle. La ligne du compteur donnant la plus grande valeur correspond aux coordonnées les plus crédibles de la position du bateau au regard des données observées.

2.1.3 Génération de vraisemblances

Afin de tenir compte de l'imprécision des capteurs, considérons que chaque distribution de vraisemblance $\mathbb{P}(O_k|I = i_j)$ suit une loi normale de moyenne μ_{jk}, et d'écart-type σ_k : $\mathbb{P}(O_k|I = i_j) \sim \mathcal{N}(\mu_{jk}, \sigma_k^2)$. La moyenne μ_{jk} correspond à la valeur attendue par le modèle physique du capteur O_k, considérant que la variable d'intérêt vaut i_j. L'écart-type σ_k dépend uniquement du capteur O_k et correspond à l'imprécision du capteur et de la correspondance du modèle physique à la réalité.

Les variables capteurs O_k sont en réalité des variables aléatoires discrètes, il faut donc discrétiser les distributions de vraisemblance $\mathbb{P}(O_k|I = i_j)$ sur 2^{res_o} valeurs, res_o étant la résolution des capteurs en bits. Étant donné que ces distributions sont connues et constantes pour un problème donné, il est possible de les tabuler pour chaque ligne j et colonne k de la matrice. Chaque vraisemblance peut alors être déterminée en adressant ces tables par la valeur de l'observation o_k.

Mieux encore, en tirant partie de la symétrie de la loi normale, et puisque leur écart-type σ_k ne dépend que de la colonne k, il est possible, pour chaque colonne, de ne tabuler que la moitié supérieure d'une Gaussienne centrée (loi demi-normale) et de l'adresser par la valeur $|o_k - \mu_{jk}|$. On comprend que plus l'échantillon est proche de la valeur attendue, plus la vraisemblance sera élevée, et vice versa. Ainsi, au lieu de stocker $N_L \times N_O$ Gaussiennes de 2^{res_o} valeurs, nous stockons N_O demi-Gaussiennes de 2^{res_o} valeurs et $N_L \times N_O$ valeurs moyennes μ_{jk} de res_o bits. Ce principe est montré en figure 2.3.

Comme mentionné en section 2.1.2, les distributions obtenues en sortie ne correspondent pas aux réelles distributions car elles ne sont pas normalisées, ce qui n'est pas essentiel dans des applications recherchant le maximum de distribution par exemple. Ainsi, nous pouvons normaliser les tables demi-Gaussiennes, c'est-à-dire qu'à l'adresse 0 se trouve le maximum de probabilité possible, $\frac{255}{256}$ pour des vraisemblances sur 8 bits par exemple. Cela permet de maximiser le nombre de '1' dans les *bitstreams*, permettant d'optimiser la précision du calcul stochastique toute chose égale par ailleurs.

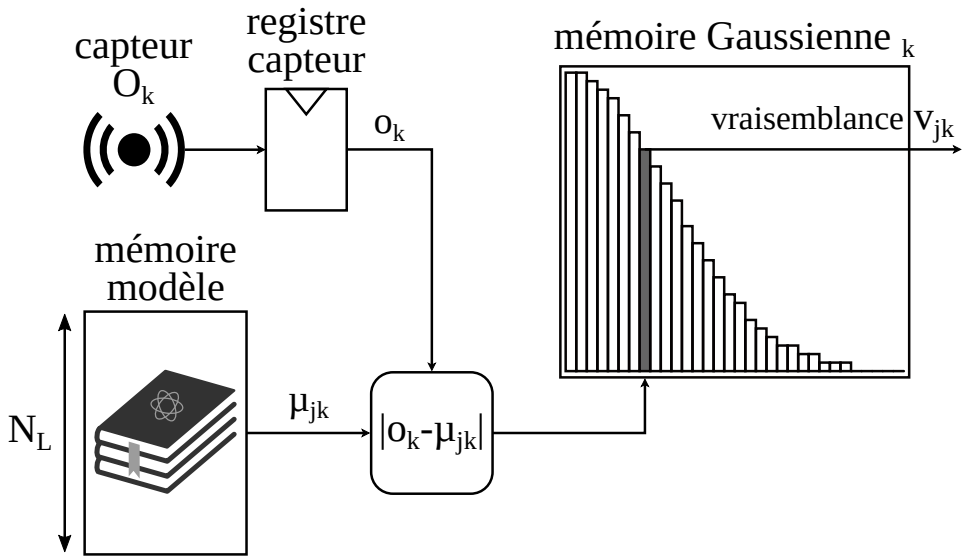

FIGURE 2.3 – *Génération de vraisemblance pour une colonne de la matrice.*

Exemple du bateau :
Pour la localisation du bateau, les distributions de vraisemblances suivent des lois normales sensiblement différentes suivant la nature du capteur.
Pour les capteurs de distance D_k, on s'attend à ce qu'ils donnent la distance cartésienne entre le capteur de coordonnées (x_k, y_k) et la position potentielle du bateau de coordonnées (x_j, y_j) : $\mu_{jk\_D} = \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}$. Leur écart-type σ_D est dû au bruit de mesure des capteurs de distance.
Pour les capteurs de phase, Θ_k, on modélise l'angle que forme la position potentielle du bateau par rapport à l'horizontale du capteur : $\mu_{jk\_\Theta} = arctan(\frac{y_j - y_k}{x_j - x_k})$. Leur écart-type σ_Θ est dû à l'incertitude de ces capteurs d'angle, *a priori* différent de celui des capteurs de distance.
Ainsi, on tabule 3 Gaussiennes d'écart-type σ_D et 3 autres d'écart-type σ_Θ. Les moyennes μ_{jk_D} et μ_{jk_Θ} sont aussi stockée et ne sont donc pas à calculer en matériel. Les vraisemblances de distances sont générées en adressant les mémoires Gaussiennes correspondantes par $|d_k - \mu_{jk\_D}|$, d_k étant la distance mesurée par le capteur D_k. De la même manière, les vraisemblances de phase sont générées en adressant les mémoires Gaussiennes d'écart-type σ_Θ par $|\theta_k - \mu_{jk\_\Theta}|$, θ_k étant l'angle mesuré par le capteur Θ_k.
Prenons l'exemple où chaque case de la grille fait 10m par 10m, que l'incertitude des capteurs de distance est de $\sigma_D = 30m$, est celle des angles est de $\sigma_\theta = 10°$. Si le capteur de distance D_0 mesure une distance au bateau de 100m, alors la vraisemblance que $D_0 = 100m$ en émettant l'hypothèse que le bateau est au centre de la première case de coordonnées (5m, 5m) est très faible. En effet, dans ce cas, le bateau serait à $5\sqrt{2} \approx 7$m (c'est la mesure attendue μ_{I0D0}) ce qui est très loin de la mesure que nous donne le capteur (distance de plus de 3 écarts-types). En prenant en compte l'incertitude Gaussienne du capteur, la vraisemblance vaut :
$P(D_0 = 100m | I = (X = 5m, Y = 5m)) \approx \frac{1}{2\pi \times 30} \times exp(\frac{(100-7)^2}{2 \times 30^2}) \approx 1.1e-4.$

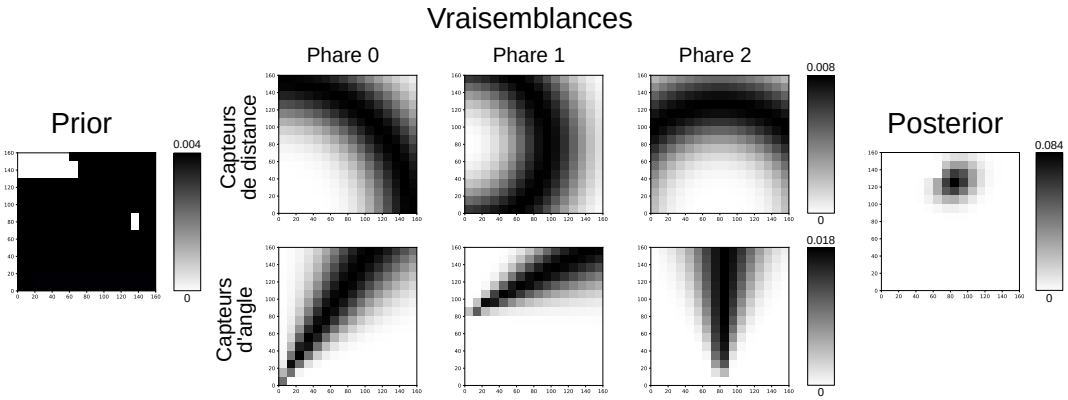

FIGURE 2.4 – *Distribution des différents priors, vraisemblances et le résultat d'inférence associé pour l'exemple du bateau aux coordonnées (85m, 125m).*

En procédant ainsi avec tous les capteurs et pour toutes les cellules de la grille, nous obtenons les distributions de la figure 2.4. On peut y voir l'impact de l'imprécision des capteurs sur les distributions de vraisemblances, mais aussi l'influence du *prior* sur le résultat final. Il y a en effet une rupture du dégradé en haut à gauche de la distribution du *posterior*, car nous avons établit qu'il était a priori impossible que le bateau se trouve sur le rocher.

Cette architecture stochastique pour la fusion Bayésienne de capteurs constitue la référence de ces travaux de thèse, dont le but est son amélioration en termes de surface et d'énergie consommée en développant des optimisations nouvelles. L'implémentation physique de cette architecture de référence ainsi que sa caractérisation sont décrites dans le chapitre 3.

2.2 État de l'art pour l'efficacité énergétique et matérielle du calcul stochastique

2.2.1 Le calcul stochastique au delà du monde numérique

Arguant qu'il est paradoxal d'utiliser la représentation numérique binaire, intrinsèquement déterministe, pour générer des *bitstreams* aléatoires, le calcul stochastique connaît une certaine popularité dans la communauté analogique et des nouvelles technologies. Le calcul stochastique y est connu sous le terme de *p-bit* [76], par analogie au q-bit du calcul quantique [77]. Le p-bit est introduit comme une représentation intermédiaire entre les deux extrêmes que sont :

— la représentation binaire, simple et facilement implémentable mais dont l'état d'un bit ne peut prendre que deux valeurs '0' et '1' ;

— la représentation quantique, dont l'état d'un q-bit est une superposition quantique linéaire de ses deux états de base $|0\rangle$ et $|1\rangle$.

Le p-bit, quant à lui, est en pratique tout aussi facilement intégrable que le bit binaire, mais il permet de représenter des valeurs sur tout un continuum entre 0 et 1. Dans [76], il est décrit comme "le q-bit du pauvre", permettant de résoudre à température ambiante des problèmes habituellement adressés par le calcul quantique, comme le "problème des Hamiltoniens stoquastiques" [78, 79] (notons la différence d'orthographe faisant la liaison entre quantique et stochastique).

De plus, des composants issus de technologies émergentes peuvent produire naturellement des nombres stochastiques (*bitstreams*) comme les mémoires résistives (RRAMs) [80] ou bien les jonctions magnétiques à effet tunnel (MTJs) [81–83].

En effet, lorsque les RRAMs sont alimentées en-dessous de leur tension nominale, de fortes variations temporelles apparaissent lors de leur mise en marche [84]. Ces variations temporelles, qui correspondent au temps requis pour qu'une RRAM soit "allumée", suivent une loi de probabilité de Poisson. En mesurant le courant à travers la RRAM, il est possible de connaître l'état (allumé ou éteint) de la mémoire. Ainsi, en fixant un certain délai d'attente et une tension d'alimentation, un tel dispositif permet de générer des *bitstreams* d'un biais souhaité.

De son côté, le MTJ est composé de deux couches de matériau ferromagnétique séparées par une fine couche isolante jouant le rôle de barrière à effet tunnel. Les deux couches possèdent un certain *spin*, et le MTJ ne possède que deux états : parallèle (les deux couches ont le même *spin*) ou anti-parallèle (*spins* opposés). Il est possible de changer le *spin* d'une des deux couches (la couche libre), et donc l'état du MTJ, en lui appliquant un certain courant : c'est le mécanisme de *Spin Torque Transfert* [85]. Le temps de transfert est fonction du courant suivant un processus (vraiment) aléatoire, ce qui signifie qu'une impulsion de courant donnée ne change l'état du MTJ qu'avec une certaine probabilité P. Ainsi, en configurant l'amplitude et la largeur de l'impulsion, il est possible de générer des *bitstreams* avec le biais souhaité.

Le MTJ est un des composants majeurs de la spintronique [86], domaine dans lequel le p-bit connaît un large succès [87]. Des travaux ont notamment montré qu'il

était possible de construire des portes logiques inversibles [88, 89]. De telles portes possèdent deux modes de fonctionnement :

— un mode classique : les entrées sont fixées et la sortie est déduite de leurs états,

— un mode inversé : la sortie est fixée et les entrées fluctuent de telle sorte que leurs valeurs soient toujours cohérentes avec la valeur de la sortie.

Ces implémentations de rupture promettent une génération de *bitstreams* à très faible coût énergétique avec ces composants de technologies émergentes. Néanmoins, elles ne sont pour le moment pas aussi matures, robustes et scalables que la technologie CMOS ce qui les rend difficilement implémentables. De plus, ce sont technologies non commerciales, auxquelles nous n'avons pas eu accès, ni au laboratoire TIMA, ni à l'entreprise HawAI.tech. Par la suite, nous nous intéressons principalement au calcul stochastique purement numérique qui est la cible principale de nos contributions.

2.2.2 Partage de ressources

Bien que l'arithmétique stochastique permette des gains en termes de ressources logiques, la génération des *bitstreams* représente une part non négligeable de la surface et de la consommation du coeur de calcul stochastique. Certains travaux proposent de réduire cet impact par des méthodes de partage de ressources.

Isolation stochastique

La notion d'"isolation stochastique", déjà présente dans les travaux de Gaines en 1967 [49], et plus récemment développée par Chen et Hayes [62], permet de partager le même RNG pour générer plusieurs *bitstreams*.

Réutiliser la même source d'aléa pour générer différents *bitstreams* induit une corrélation entre ceux-ci. L'isolation stochastique consiste à utiliser des *isolateurs* pour réduire cette corrélation. Il s'agit ni plus ni moins d'une bascule insérée pour décaler le *bitstream* d'un cycle dans le temps, comme illustré en figure 2.5. Ainsi, en considérant l'autocorrélation du RNG comme nulle (hypothèse à vérifier au cas par cas selon le RNG utilisé), les *bitstreams* sont indépendants bit à bit.

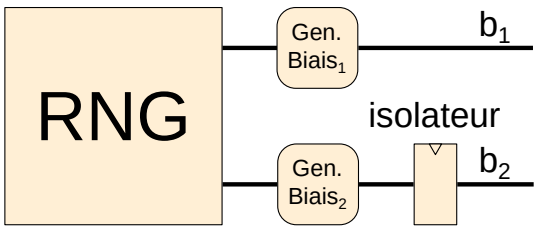

FIGURE 2.5 – *Implémentation d'un isolateur stochastique.*

Cette méthode permet d'économiser l'implémentation de différents RNGs pour générer plusieurs *bitstreams*, mais induit tout de même une potentielle corrélation entre les *bitstreams* ainsi générés [90] suivant le type de RNG partagé. Une étude

comparative de différents RNGs est réalisée en section 5.2 afin de déterminer quels RNGs sont compatibles avec l'isolation stochastique et quels sont les gains attendus.

Permutation de bits aléatoires

De manière analogue, certains travaux se sont intéressés au partage d'un même RNG pour générer plusieurs *bitstreams* via une décorrélation par permutation des bits du nombre aléatoire généré. Dans [64], Ichihara *et al.* introduisent l'idée d'une permutation circulaire des bits aléatoirement générés pour décorréler les *bitstreams*. Mais c'est Salehi [91] qui propose un algorithme capable de déterminer les meilleures permutations, celles qui minimisent la corrélation entre plusieurs *bitstreams* ainsi générés. Il s'avère que pour générer deux *bitstreams*, la permutation qui minimise leur corrélation est celle d'un ordre des bits totalement inversé : les bits de poids fort deviennent les bits de poids faibles et inversement. Cette permutation inverse ne provoque presque pas de perte en précision comparée à une implémentation avec deux RNGs différents, mais au delà de deux *bitstreams* générés par RNG, la précision des calculs s'en retrouve impactée, quand bien même les permutations sélectionnées sont optimales.

Weighted Binary Generator

Dans [92], Gupta et Kumaresan ont introduit un convertisseur binaire / stochastique alternatif au comparateur utilisé de manière standard : le *Weighted Binaray Generator* (WBG). La figure 2.6 montre son fonctionnement pour des nombres de 4 bits. Comme on peut le voir, le WBG est composé d'une partie qui ne dépend que du nombre aléatoire d'entrée r_{3-0}, le *Weight Generator* (WG), et d'une autre qui dépend de ce nombre aléatoire ainsi que de la consigne de biais b_{3-0}, le *Probability Encoder* (PE).

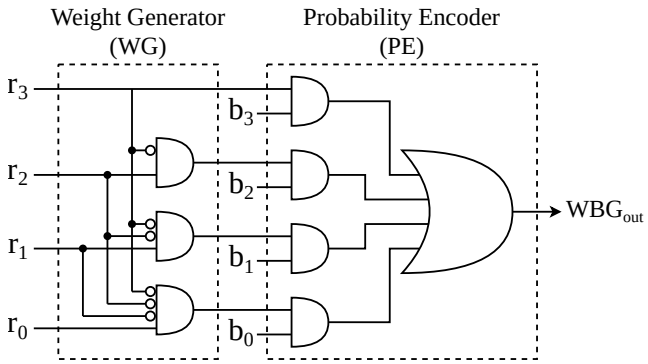

FIGURE 2.6 – *Décomposition d'un Weighted Binary Generator à 4 bits.*

Le WG génère des *bitstreams* w_i tels que :

$$
\begin{aligned}
w_3 &= r_3 \\
w_2 &= \bar{r_3} \wedge r_2 \\
w_1 &= \bar{r_3} \wedge \bar{r_2} \wedge r_1 \\
w_0 &= \bar{r_3} \wedge \bar{r_2} \wedge \bar{r_1} \wedge r_0
\end{aligned}
$$

ce qui donne les biais suivants :

$$\mathbb{P}(w_3) = \frac{1}{2} \quad \mathbb{P}(w_2) = \frac{1}{4} \quad \mathbb{P}(w_1) = \frac{1}{8} \quad \mathbb{P}(w_0) = \frac{1}{16}$$

Le PE renvoie la sortie :

$$WBG_{out} = w_3 \wedge b_3 \ \vee \ w_2 \wedge b_2 \ \vee \ w_1 \wedge b_1 \ \vee \ w_0 \wedge b_0$$

Puisque les *bitstreams* w_i sont mutuellement exclusifs (il est impossible que deux w_i soient tout les deux à 1), la probabilité de la sortie donne :

$$\begin{aligned}
\mathbb{P}(WBG_{out}) &= \mathbb{P}(w_3).b_3 + \mathbb{P}(w_2).b_2 + \mathbb{P}(w_1).b_1 + \mathbb{P}(w_0).b_0 \\
&= \frac{b_3}{2} + \frac{b_2}{4} + \frac{b_1}{8} + \frac{b_0}{16} \\
&= \frac{2^3 b_3}{16} + \frac{2^2 b_2}{16} + \frac{2^1 b_1}{16} + \frac{2^0 b_0}{16} \\
&= \frac{biais}{16}
\end{aligned}$$

Le WBG se comporte donc bel et bien comme un convertisseur binaire / stochastique, au même titre que le comparateur.

De prime abord, ce convertisseur semble moins intéressant qu'un comparateur car il demande plus de ressources logiques à mettre en place. Cependant, dans [93], Yang *et al.* ont proposé une manière de factoriser le WG, permettant des gains en surface et consommation de puissance par rapport à une implémentation standard avec comparateurs. En effet, comme le WG ne dépend pas du biais à générer, il peut être partagé pour générer des nombres stochastiques indépendants du point de vue du calcul, à l'instar des RNGs. Nous utiliserons ce principe dans la section 5.1.3.

Ainsi, seul le PE, près de deux fois plus compact qu'un comparateur, doit être implémenté pour chaque *bitstream* indépendant à générer. Cette factorisation des ressources permet donc une économie en puissance et surface consommées. Elle est d'autant plus grande que le nombre de *bitstreams* indépendants dans le calcul est élevé.

2.2.3 Réduction du temps de calcul pour l'efficacité énergétique

Même si le calcul stochastique permet une implémentation de circuit de très faible puissance, son temps de calcul élevé implique une consommation énergétique conséquente. Pour certaines applications dites d'*energy harvesting* (ou récolte d'énergie) [94], réduire la puissance consommée est le plus important, car elles disposent d'une source d'énergie continue dont la puissance électrique est très faible. Cependant, pour la plupart des applications fonctionnant sur batteries, réduire l'énergie consommée pour un calcul donné est fondamental. Ainsi, d'autres recherches ont portées sur la réduction du temps de calcul stochastique afin de contrebalancer ses mauvaises performances en termes de consommation énergétique.

Méthode du *slicing*

Dans [32], Frisch *et al.* ont proposé une méthode dite de *slicing* pour contrer le problème de la *dilution temporelle*. Ce problème est relatif à la multiplication de nombreux *bitstreams* entre eux. Les opérandes étant compris entre 0 et 1, le résultat de ces multiplications devient de plus en plus petit, et il devient difficile de représenter ces nombres de manière précise, à l'instar de la représentation en virgule fixe. Pour garder une précision satisfaisante, il faut donc augmenter le temps de calcul à chaque nouvelle multiplication stochastique effectuée. La méthode de *slicing* consiste à diviser les n multiplications à réaliser en t tranches de taille s : $n = t \times s$. Au bout de chaque tranche, les résultats parallèles sont renormalisés par le maximum de la colonne par conversion / regénération, c'est-à-dire qu'il sont convertis en représentation binaire, normalisés puis regénérés en *bitstream*. Les valeurs sont donc plus élevées pour la tranche d'après, mais la proportionnalité est respectée. La normalisation est en réalité effectuée par des compteurs qui s'incrémentent à chaque bit à '1' dans les *bitstreams* (convertisseur stochastique / binaire), et en stoppant le compte dès lors qu'un des compteurs a atteint sa valeur maximale $2^{\text{res}_\text{p}} - 1$, ce qui donne la valeur de probabilité maximale $\frac{2^{\text{res}_\text{p}}-1}{2^{\text{res}_\text{p}}}$, les probabilités étant codées sur res_p bits. Ce principe est montré en figure 2.7.

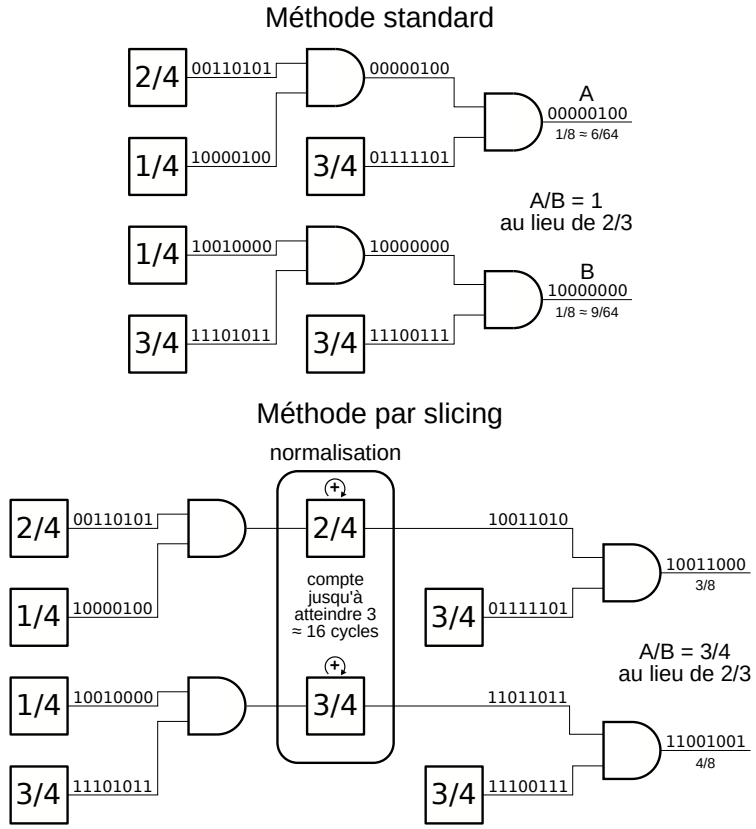

FIGURE 2.7 – *Principe de renormalisation par slicing pour réduire le temps de calcul stochastique.*

Dans cet exemple, il faut à la méthode de *slicing* environ 24 cycles (16 pour la normalisation et 8 pour les nouveaux *bitstreams*) pour avoir un rapport de propor-

tion de $A/B = 3/4$ (au lieu de $2/3$ pour le calcul exact). Cela prendrait en moyenne 32 cycles à la méthode standard ($A = 6/64 = 3/32$ et $B = 9/64 \approx 4/32$). Mieux encore, pour obtenir en moyenne le bon rapport $A/B = 2/3$, il faut 32 cycles à la méthode par *slicing* au lieu de 64 pour la méthode standard. Notons qu'un calcul exact [3] prendrait $2^{\text{res}_\text{p} \times n}$ cycles pour la méthode standard, n étant le nombre de multiplications, comparé à $t \times 2^{\text{res}_\text{p} \times s}$ pour la méthode par *slicing*. D'une manière générale, si L est une longueur de *bitstream* permettant d'atteindre une précision donnée, et en considérant la normalisation comme parfaite, il faudrait en moyenne L^n cycles pour retrouver cette précision après n multiplications en méthode standard, et $t \times L^s$ cycles pour la méthode de *slicing*. Cela réduit donc considérablement le temps de calcul pour une précision donnée. Cependant, comme mentionné dans la thèse [30], en raison des approximations faites lors de la normalisation, l'utilisation de cette méthode induit une réduction de la précision à chaque tranche. Il y a donc un compromis à trouver entre réduction du temps de calcul et donc de la consommation d'énergie, et précision du calcul réalisé.

Low Discrepancy SNGs

Plusieurs travaux [95–97] se sont tournés vers une manière totalement déterministe de générer des nombres stochastiques, balayant l'idée commune selon laquelle de tels nombres doivent être générés par une source d'aléa. À la place, ces Low Discrepancy SNGs (LDSNGs), utilisent des séquences déterministes (séquence de Halton [95] ou de Sobol [96]) pour converger plus rapidement vers la consigne de biais. En effet, l'erreur de ces LDSNGs évolue en $O(\frac{1}{n})$ lorsque la longueur du *bitstream* n tend vers l'infini, pour une évolution en $O(\frac{1}{\sqrt{n}})$ avec l'implémentation conventionnelle (par le théorème central limite). Cependant, l'implémentation de tels LDSNGs est d'une part assez conséquente, si bien que [98] propose qu'à partir de 3 *bitstreams* indépendants à générer, il est préférable d'utiliser un SNG aléatoire conventionnel. Et, d'autre part, ces générateurs déterministes ne sont pas ou très peu compatibles avec les méthodes de partage de ressources mentionnées dans la sous-section précédente, puisqu'ils reposent sur le fin contrôle des nombres générés à chaque instant.

Calcul stochastique parallèle

Enfin, certaines contributions ont proposé l'implémentation de calcul stochastique parallèle, le plus souvent en utilisant des LDSNGs [99–101] afin de réduire le temps de calcul stochastique. Cependant, ces architectures parallèles souffrent des mêmes inconvénients des LDSNGs et d'un grand surcoût en surface. En effet, leur but étant plus orienté sur l'augmentation des performances de calcul, elles sont des solutions massivement parallèles capable de résoudre un calcul stochastique de faible longueur de *bitstream* en 1 seul cycle d'horloge, mais ne sont à ce titre pas efficientes en termes de surface d'implémentation et sous optimales en termes d'énergie.

3. Le calcul peut être exact si tous les nombres dans $[0,\ 2^{\text{res}_\text{p}} - 1]$ n'apparaissent qu'une fois dans la séquence aléatoire, comme dans un LFSR.

2.3 Générateurs de nombres aléatoires pour le calcul stochastique

Une part importante de la surface et de la consommation des circuits stochastiques est liée à la génération des nombres aléatoires. Nous avons donc cherché à identifier dans la littérature différentes solutions existantes pour voir quelle serait la plus appropriée à nos besoins.

2.3.1 *Linear Feedback Shift Register*

Le RNG le plus communément utilisé pour le calcul stochastique est le *Linear Feedback Shift Register* (LFSR) [59] pour son extrême simplicité.

Le LFSR est un générateur de nombres pseudo-aléatoires (PRNG) très simple constitué d'un registre à décalage et de portes XORs, comme illustré en figure 2.8.

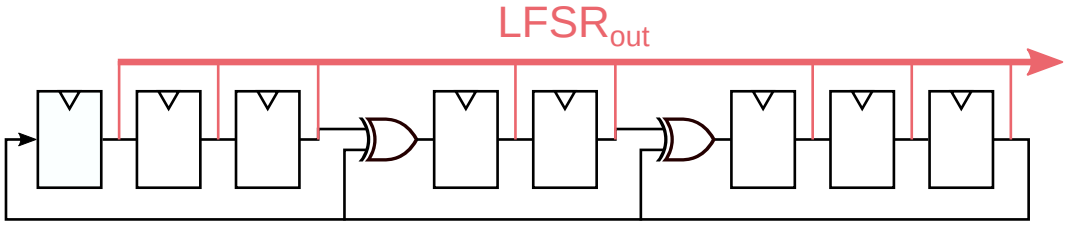

FIGURE 2.8 – *Implémentation d'un LFSR Galois à 8 bits.*

Le LFSR génère une séquence périodique de nombres pseudo-aléatoires. La période d'aléa du LFSR dépend de son polynôme associé, c'est-à-dire du placement des portes XORs entre les bascules, et vaut au maximum $2^n - 1$ pour un LFSR de profondeur n (i.e. dont le nombre de bascules vaut n). Le nombre de portes XORs nécessaires pour construire un LFSR de période d'aléa maximale ne semble pas dépendre de la profondeur du LFSR. L'étude [102] montre que pour des LFSRs de profondeur entre 3 et 168 bits, il existe pour chacun des polynômes ne nécessitant que 2 à 4 portes XORs. Ainsi, pour générer n bits aléatoires, le LFSR ne requiert l'implémentation que de n bascules et de 2 à 4 portes XORs, le rendant très efficace en ressources.

Notons qu'un tel RNG possède des qualités statistiques médiocres, avec une forte corrélation entre les nombres générés à deux pas de temps consécutifs (autocorrélation) à cause du registre à décalage. Cependant, cela n'est pas un facteur limitant pour la logique stochastique [4] qui nécessite une faible corrélation entre différents opérandes (*bitstream* indépendants), mais pas nécessairement entre les bits successifs d'un même *bitstream* à des pas de temps consécutifs. Une métrique pour quantifier cette corrélation entre différents *bitstreams* est appelée *Stochastic Computing Correlation* (SCC) [60] et est détaillée en section 1.4.3. De plus, le choix des graines (états initiaux des registres) des différents LFSRs utilisés pour générer différents nombres stochastiques est crucial pour limiter la corrélation entre ces SNGs et donc garantir la précision des calculs [61].

4. tant que l'isolation stochastique n'est pas utilisée

Il est vrai que dans le cas général, la piètre qualité statistique des LFSRs n'est pas un problème pour le calcul stochastique. Cependant, avec l'isolation stochastique décrite en section 2.2.2 ils peuvent engendrer des pertes en précision [60, 90]. D'autres RNGs plus sophistiqués sont alors à considérer.

2.3.2 *S-Box Number Generator*

Le *S-Box Number Generator* (SBoNG) [90] est un PRNG conçu par Neugebauer *et al.* spécifiquement pour la génération de nombres stochastiques. Il est basé sur une implémentation à petite échelle de l'étape de substitution de l'algorithme de chiffrement AES (*Advanced Encryption Standard*) introduit par [103]. Il utilise notamment des LFSRs et des *S-Box*, des tables de substitutions permettant de casser la linéarité des chiffrements cryptographiques. La *S-Box* utilisée ici exploite des mots de 4 bits, elle est décrite dans le tableau 2.1.

Entrée	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Sortie	6	B	5	4	2	E	7	A	9	D	F	C	3	1	0	8

TABLEAU 2.1 – *S-Box 4 bits utilisé en codage hexadécimal.*

En cryptographie, les *S-Box* sont généralement stockées dans des tables. Cependant, cela augmenterait significativement la surface du circuit dans une utilisation pour le calcul stochastique. Celles-ci sont donc implémentées sous la forme de logique combinatoire suivante, avec la sortie $S = s_3...s_0$ et l'entrée $E = e_3...e_0$:

$$
\begin{aligned}
s_3 &= \bar{e_2}e_3 + e_0\bar{e_1}\bar{e_3} + e_0e_1e_2 \\
s_2 &= e_0\bar{e_1}e_2\bar{e_3} + e_1\bar{e_2} + \bar{e_0}\bar{e_2}\bar{e_3} + e_0\bar{e_2}e_3 + \bar{e_0}e_1\bar{e_3} \\
s_1 &= \bar{e_0}e_1\bar{e_2}e_3 + \bar{e_1}\bar{e_3} + e_2\bar{e_3} + e_0\bar{e_1}e_2 \\
s_0 &= e_0\bar{e_1}\bar{e_2} + \bar{e_0}e_1\bar{e_3} + \bar{e_1}e_3 + \bar{e_0}\bar{e_2}e_3
\end{aligned}
$$

Cela correspond naïvement à 29 portes ET et 12 portes OU, mais des simplifications peuvent peut-être être effectuées lors de la synthèse logique, ou bien lors du *technology mapping* pour raccorder ces équations logiques avec des cellules standards appropriées.

Le fonctionnement du SBoNG est défini par l'algorithme 2.1.

```
class SBONG(seed0, seed1, polynome, size):
    r0 = seed0
    r1 = seed1
    LFSR_MASK = polynome
    RNG_size = size

    def next_random():
        # LFSR
        lsb = r0[0]
        r0 >>= 1
        if (lsb): r0 ^= LFSR_MASK

        r1 ^= r0
        for i in range(0, RNG_size/4):
            r1[4*i : 4*(i+1)] = sbox(r1[4*i : 4*(i+1)])
        r1 = ror(r1, 1)
        result = r1
        r1 ^= r0
        return result
```

Algorithme 2.1 – Classe décrivant le fonctionnement du SBoNG.

Ici, $r0$ et $r1$ correspondent à des registres stockant les valeurs internes de l'algorithme. $r0$ est plus spécifiquement le registre d'un LFSR. La fonction *ror* correspond à une rotation vers la droite des bits du nombre en entrée. Pour un nombre de n bits et une rotation à droite de x, cela signifie que les x bits de poids forts du nombre deviennent ses x bits de poids faibles (dans le même ordre), et ses $n-x$ bits de poids faibles deviennent ses $n-x$ bits de poids forts. Cela revient à réaliser une opération OU entre un décalage à droite de x et un décalage à gauche de $n-x$. Par soucis de clarté, les registres $r0$ et $r1$ sont parfois représentés comme des listes de bits, en effectuant des appels par indice ($r[0]$) ou bien par tranches ($r1[4\times i : 4\times(i+1)]$). Ils restent cependant des variables scalaires, et sont à interpréter comme telles. Ces registres doivent être de la même taille que le nombre de bits aléatoires *RNG_size* à générer.

Un tel RNG intégrant notamment un LFSR, il est de fait plus volumineux. Pour générer n bits aléatoires, le SBoNG requiert $2\times n$ bascules, $2\times n$ portes XORs (plus 2 à 4 pour le LFSR), $\frac{n}{4}\times 29$ portes ET et $\frac{n}{4}\times 12$ portes OU.

Cela peut paraître imposant, mais Neugebauer *et al.* l'utilise avec l'isolation stochastique décrite en section 2.2.2 en le partageant pour générer des *bitstreams* de manière efficace. Ils sont ainsi capables de réduire jusqu'à 20% les ressources logiques utilisées en le partageant sur 100 SNGs.

2.3.3 Xoroshiro

Le Xoroshiro [104] (nommé d'après les opérations qu'il réalise : *XOR - rotation - shift - rotation*), est un PRNG ayant une bonne qualité statistique. Il passe notamment l'ensemble des 106 tests statistiques de la collection BigCrush de la bibliothèque TestU01 [105], incluant notamment les jeux de tests DIEHARD [106], FIPS 140-2 [107] et NIST SP 800-20 [108], considérés comme des références pour l'analyse statistique de RNGs.

Il existe plusieurs versions de ce RNG, l'une des plus économes en ressources logiques est le Xoroshiro++, son fonctionnement est décrit par l'algorithme 2.2.

```
class XOROSHIRO(seed0, seed1, params):
    r0 = seed0
    r1 = seed1
    A, B, C, D = params

    def next_random():
        r1 ^= r0
        r0 = rol(r0, A) ^ r1 ^ (r1 << B)
        r1 = rol(r1, C)
        return rol(r0 + r1, D) + r0
```

Algorithme 2.2 – Classe décrivant le fonctionnement du Xoroshiro++.

Dans cet algorithme, r0 et r1 correspondent à des registres stockant des variables internes à l'algorithme. Ils sont initialisés par les valeurs *seed*0 et *seed*1 en tout début de calcul. La fonction *rol* est la fonction de rotation des bits d'un nombre vers la gauche. Les valeurs A, B, C et D correspondent à des constantes de fonctionnement de l'algorithme. En modifiant ces paramètres, il est possible d'obtenir des séquences de nombres de plus ou moins bonne qualité statistique. Dans notre usage, nous utilisons les paramètres recommandés par l'auteur [104, 109].

Bien que les décalages et rotations ne représentent aucun coût matériel, ce RNG consomme beaucoup plus de ressources logiques que le LFSR. Pour générer n bits aléatoires il lui faut en effet $2 \times n$ bascules, $3 \times n$ portes XORs et 2 additions n bits, comparés aux n bascules et quelques portes XORs du LFSR.

Cependant, comme nous l'avons mentionné, ce PRNG bénéficie d'une excellente qualité statistique, si bien qu'il est utilisé comme RNG de base dans les systèmes d'exploitations IoT Zephyr et Mbed [109].

2.3.4 *Self-timed ring based True Random Number Generator*

Le *Self-timed ring based True Random Number Generator* (STRNG) [110] est un TRNG utilisant la fluctuation temporelle des signaux logiques, le *jitter*, comme source d'entropie. Il est basé sur un anneau auto-séquencé, constitué de portes de Muller (ou C-élément), une brique élémentaire des contrôleurs de circuits asynchrones dont la table de vérité est présentée dans le tableau 2.2.

e_0	e_1	s
0	0	0
0	1	s^{-1}
1	0	s^{-1}
1	1	1

TABLEAU 2.2 – *Table de vérité de la porte de Muller.*

Dans ce tableau, e_0 et e_1 sont les deux entrées de la porte de Muller, et s sa sortie. s^{-1} correspond à la valeur de la sortie au cycle précédant, cette porte possède donc un effet mémorisant. Ces portes logiques sont généralement implémentées avec la cellule standard d'une porte majorité à 3 entrées dont la sortie est rebouclée sur une des entrées.

Les anneaux oscillants à base de C-éléments, présentés en figure 2.9, permettent de propager plusieurs événements simultanément sans collision, grâce au protocole de requêtes et acquittements asynchrones. Lorsque N événements se propagent dans un anneau à L étages, et que N et L sont premiers entre eux, alors ces événement se répartissent naturellement dans une demi période d'oscillation selon un déphasage $\Delta\phi = \frac{T}{2L}$.

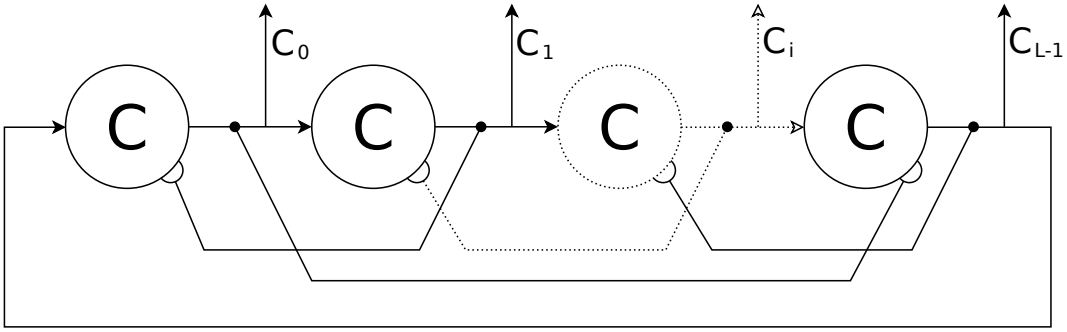

FIGURE 2.9 – *Mise en place d'un anneau auto-séquencé de portes de Muller.*

Ainsi, pour un grand nombre de portes de Muller, ce déphasage devient de plus en plus petit, jusqu'à être inférieur à l'écart-type du *jitter*. Dans ce cas, il est possible d'extraire l'entropie de ce bruit physique en échantillonnant les L signaux par autant de bascules synchronisées par une même horloge. Les sorties de ces bascules sont alors "brassées" dans une porte XOR à L entrées pour générer les bits aléatoires. Un post-traitement peut alors être mis en place pour corriger d'éventuels biais statistiques avec un filtre de parité d'un certain degré n. Ce filtre est constitué d'un registre à décalage de n étages dont l'entrée est la sortie aléatoire brute du bloc décrit précédemment, et dont les sorties de toutes les bascules passent par une porte XOR à n entrées, dont la sortie est stockée dans une bascule.

Le nombre de portes de Muller requis pour atteindre un déphasage inférieur au *jitter* dépend grandement du noeud technologique d'implémentation physique. Sur un circuit implémenté sur une cible FPGA Altera Cyclone III, un anneau de 63 portes avec un filtre de parité d'ordre 7 donne de bonnes propriétés statistiques en passant les tests FIPS 140-2 [111]. Ainsi, pour implémenter un tel TRNG, il faudrait $63+8=71$ bascules, 63 portes majorités, une porte XOR à 63 entrées et une autre à 7 entrées. Cela est beaucoup plus volumineux que les RNGs précédents, mais il possède une période d'aléa infinie et une bonne qualité statistique.

3

Implémentation physique et caractérisation d'un circuit stochastique de fusion Bayésienne de capteurs

Ce chapitre traite de l'implémentation physique d'architectures stochastiques dédiées à la fusion Bayésienne de capteurs développée en section 2.1. Sa consommation énergétique est comparée à celle d'un microcontrôleur ultra basse consommation, le Renesas RE01. Nos mesures ont permis de mettre en évidence l'intérêt du calcul stochastique pour la fusion Bayésienne de capteurs, notamment pour de faibles précisions, avec une consommation énergétique jusqu'à 6000 fois inférieure à celle du microprocesseur.

Sommaire

Afin de rendre compte de l'intérêt de la logique stochastique pour résoudre efficacement des problèmes d'inférence Bayésienne, un prototype a été conçu et implémenté physiquement dans le contexte du projet Microbayes [28]. Celui-ci reprend la même architecture décrite dans l'état de l'art en section 2.1. Notons qu'à cette époque, en tant qu'ingénieur d'étude, mon implication dans ce prototype a été principalement axée sur la vérification fonctionnelle du circuit avant fabrication. Ma contribution durant cette thèse a été son test et sa caractérisation en vue de comparer ses performances à celles d'un microcontrôleur ultra basse consommation, le Renesas RE01. Ce prototype représente à notre connaissance l'une des premières implémentations physique de circuit stochastique dédiée à la résolution d'inférence Bayésienne avec [75].

3.1 Implémentation physique du prototype

3.1.1 Dimensions et caractéristiques du circuit

Les dimensions générales du circuit stochastique de fusion de capteurs sont définies comme suit :

- les données capteurs sont codés sur $\text{res}_\text{o} = 8$ bits signés,
- les vraisemblances (valeurs de probabilités) sont codées sur $\text{res}_\text{p} = 8$ bits non signées,
- les RNGs implémentés sont des LFSRs 32 bits, afin d'avoir une période d'aléa élevée et donc d'atteindre une plus grande précision sans être limité par le RNG,
- le convertisseur binaire / stochastique utilisé est un comparateur de 8 bits (res_p),
- les nombres aléatoires correspondent aux 8 bits (res_p) de poids faibles des sorties des LFSRs,
- les compteurs sont de taille 32 bits, afin de pouvoir convertir de longs *bitstreams* pour les précisions élevées.

Il existe deux manières d'arrêter le calcul stochastique : soit un des compteurs atteint une valeur seuil `MAX_COUNTERS` ; soit le temps de calcul (longueur des *bitstreams*), mesuré par un compteur indépendant, atteint une valeur seuil `TIMEOUT`. Ces deux signaux sont configurés par le système hôte et sont codés sur 32 bits.

Quatre Architectures Stochastiques pour la Fusion Bayésienne de capteurs (ASFB) de différentes dimensions sont intégrées dans la puce : ASFB-2x11, -16x11, -32x5 et -64x9_gen, où le premier chiffre correspond au nombre de lignes (N_L) et le second au nombre de colonnes ($N_\text{C} = N_\text{O} + 1$). La dernière architecture, 64x9_gen, est configurable. Elle a un maximum de 64 lignes et 9 colonnes, mais elle peut effectuer le calcul avec des dimensions inférieures, en activant seulement la quantité correspondante de lignes et de colonnes.

À chaque architecture sont associés :

— N_O mémoires de modèle stockant N_L valeurs μ de données signées sur 8 bits,

— N_O registres capteurs stockant les observations signées sur 8 bits,

— N_O mémoires demi-Gaussiennes stockant 256 valeurs (2^{res_o}) de 8 bits (res_p).

Une stratégie de mise en veille des mémoires est employée afin de réduire leur consommation lorsqu'elles ne sont pas utilisées. Le coût énergétique de la sortie de veille pouvant être conséquent, cette stratégie n'est rentable que pour de longues périodes d'inactivité : c'est le cas lors du calcul stochastique d'inférence. Les mémoires ne sont mises en veille que dans ce cas précis.

Une seule architecture est activée à la fois, et l'horloge des autres architectures est coupée (*clock gating*) afin de mesurer séparément la consommation électrique de chaque configuration. Le système est conçu pour effectuer des calculs à la fréquence maximale de 50 MHz. Un diviseur d'horloge est implémenté pour diminuer cette fréquence lorsque l'on diminue la tension en dessous de sa valeur nominale de 1.1V. Cela permet d'étudier la tenue en tension en évaluant la fréquence maximale de fonctionnement dans des conditions dégradées.

3.1.2 Interfaces, communication et contrôle

La puce communique avec le système hôte par l'intermédiaire d'un bus SPI (*Serial Peripheral Interface*), comme présenté dans la figure 3.1.

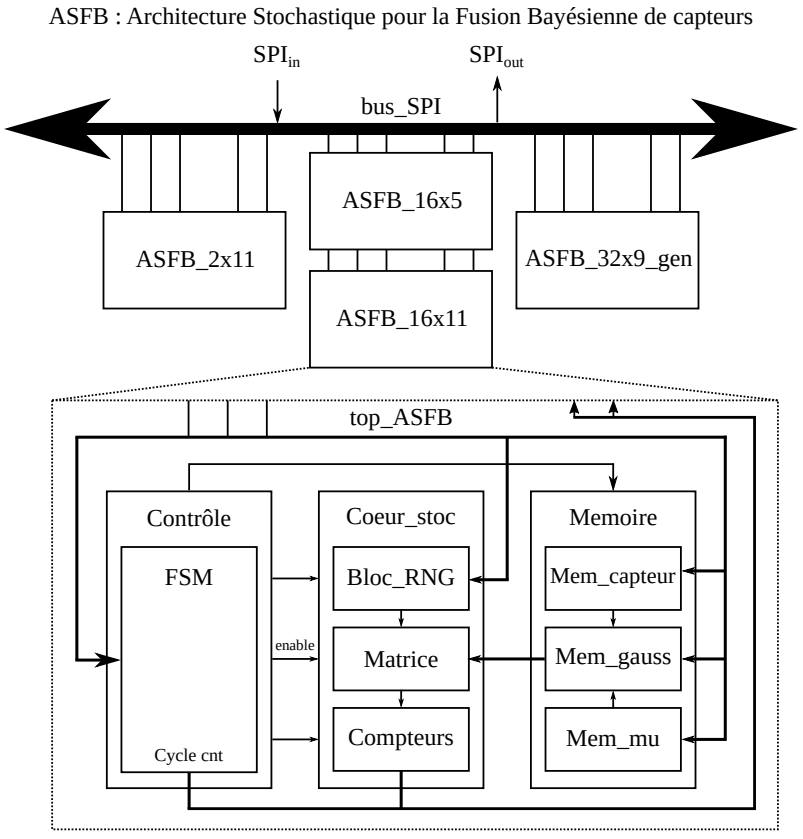

FIGURE 3.1 – *Architecture top-level de la puce.*

Le bus SPI permet au système hôte de :

— configurer la puce en écrivant dans les registres de configuration,

— charger les données d'entrée en mémoire et récupérer les données de sortie.

Connecté au bus SPI, un bus dédié envoie les données aux blocs respectifs.

Une machine à états finis (FSM) envoie des signaux de contrôle aux différents blocs en fonction des registres configurés afin de remplir plusieurs fonctions :

— générer des vraisemblances comme développé dans la section 2.1.3 ou envoyer les vraisemblances directement depuis le système hôte,

— effectuer ou non le calcul d'inférence, ceci est utile pour évaluer séparément la part de consommation du calcul stochastique et celle du chargement des données,

— traiter ou non les opérations en boucle, ceci est utile pour mesurer la moyenne de puissance électrique sur de longs intervalles de temps.

3.1.3 Implémentation physique

La figure 3.2 montre le plan du circuit prototype après placement / routage.

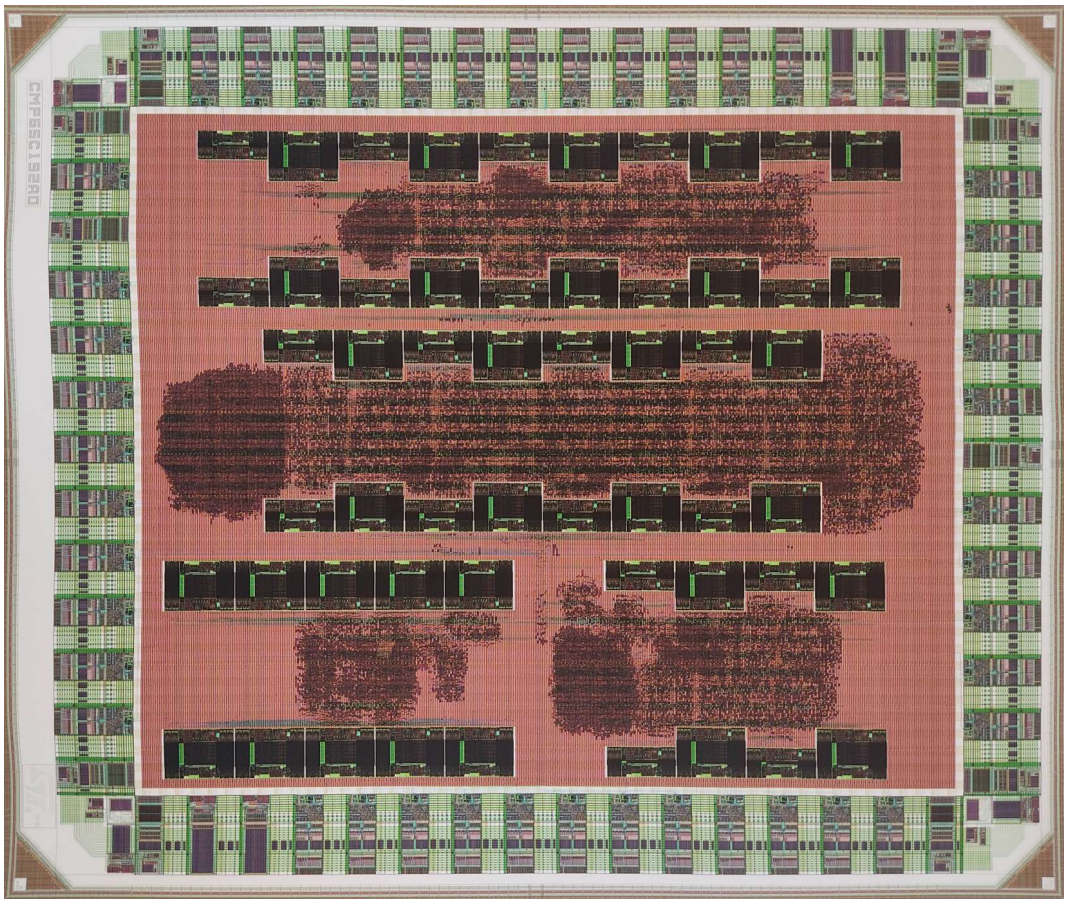

FIGURE 3.2 – *Plan du circuit intégrant les 4 architectures, placé et routé.*

On observe que la surface est principalement tributaire des nombreux ports d'entrées / sorties, la plupart remplissant des fonction de *debug* : on dit que la puce est

pad limited. Ainsi, beaucoup d'espace est inutilisé (fond rouge). Cependant, sur l'espace utilisé (fond sombre), les mémoires DRAMs (rectangles noirs) sont responsables d'une grande partie de la surface. En terme de surface utile des cellules logiques, nos mesures montrent qu'elle est occupée jusqu'à 83% par les blocs mémoires pour l'architecture 2x11.

3.2 Caractérisation

3.2.1 Environnement de test

Pour effectuer les mesures de ce circuit, on utilise l'équipement suivant :

- la puce du prototype à tester, gravée dans la technologie CMOS 65nm de ST Microelectronics, encapsulée dans un boîtier BGA (*Ball Grid Array*) avec 64 broches ;
- une alimentation électrique délivrant les tensions requises. Les tensions nominales sont : 1.1 V pour la puce et 1.8 V pour les plots d'entrées / sorties (IOs) ;
- une carte PCB reliant la puce au circuit d'alimentation et au microcontrôleur, et qui intègre :
  - des LEDs d'affichage ;
  - une interface périphérique GPIO pour la communication avec le microcontrôleur ;
  - des broches de sortie pour les mesures de tension et de courant du circuit ;
- un microcontrôleur programmé par le système hôte via le port USB, qui envoie et reçoit des données de la puce via un protocole SPI ;
- une carte Arduino assurant l'interface entre le système hôte et le microcontrôleur ;
- un système hôte qui permet de configurer les données et les paramètres de configuration, de les envoyer à la puce via la carte Arduino et le microcontrôleur, puis de recevoir et d'analyser les résultats.

La figure 3.3 présente le dispositif de mesure.

3.2.2 Jeux de données

Nous définissons différents jeux de données pour la vérification fonctionnelle et les mesures de puissance et de précision. Les jeux de données nul, certain et randomisé (`RAND`) sont utilisés pour la vérification fonctionnelle et les mesures de puissance dans un premier temps. Pour les mesures de précision, ce sont les jeux de données randomisé (`RAND`), normalisé (`NORM`) et recherche du maximum de distribution (`RMAX`) qui sont utilisés. Dans ce cas, les deux jeux de données `RAND` et `NORM`

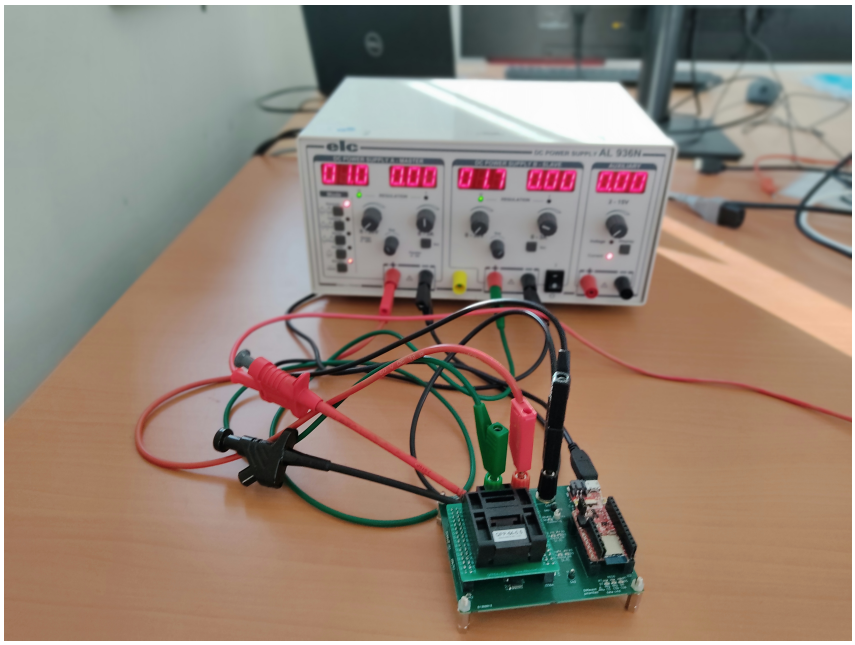

FIGURE 3.3 – *Environnement de test.*

évaluent la qualité de représentation de l'entièreté de la distribution en utilisant les métriques de RMSE et / ou de KLD, alors que `RMAX` s'intéresse uniquement à la bonne correspondance du maximum de distribution, évaluée par le Taux de Reconnaissance du Maximum (TRM).

Jeux nul et certain

Le jeu nul (respectivement certain) consiste simplement à effectuer le calcul stochastique avec des *priors* et des vraisemblances tous à 0 (respectivement tous à 1). Cela permet un premier niveau de vérification fonctionnelle ainsi que de déduire approximativement la part des compteurs et des SNGs dans la consommation globale du circuit.

Vérité-terrain pour la mesure de précision

Les mesures de précision prennent comme référence la vérité-terrain (*ground-truth en anglais)*, c'est par exemple la position réelle du bateau pour un observateur omniscient. Le calcul de la "vraie" distribution de *posterior* est ensuite effectué en virgule flottante de bout en bout, et sans ajout de bruit potentiel. À noter que tous les calculs réalisés dans le matériel montreront toujours un certain écart par rapport à ce vrai posterior, y compris le calcul de produits en virgule flottante car ils souffrent d'approximations de la virgule fixe pour la représentation des mesures capteurs o_k, des valeurs modèles μ_{jk} ainsi que pour les tables Gaussiennes.

Jeu randomisé (RAND)

Le jeu de données randomisé, RAND, consiste simplement au calcul d'inférence avec des *priors* et vraisemblances aléatoires tirés de manière informe entre 0 et 1. Il est utilisé aussi bien pour la vérification fonctionnelle que pour l'étude de précision. Ce choix aléatoire des valeurs de probabilités du jeu de données a pour effet de causer une certaine incohérence dans les informations fournies par les vraisemblances : le premier capteur peut par exemple associer une forte probabilité à la position i_1, alors que dans le même temps un second capteur lui associe une probabilité très faible. Ce jeu de données correspond donc en quelque sorte au pire cas pour le calcul stochastique, car il faut une grande précision pour représenter efficacement de telles distributions d'informations contradictoires. Cette forte précision requise implique une grande dépense d'énergie en logique stochastique, là où la multiplication flottante ne perd pas nécessairement plus de précision, ni ne consomme plus que pour une distribution plus "piquée".

Jeu normalisé (NORM)

Le jeu de données normalisé, NORM, consiste à calculer une inférence idéale, où le *prior* et les vraisemblances de la vérité-terrain ont une valeur maximale de 1. Ainsi, en représentation stochastique, le *bitstream* de sortie de la ligne correspondante i_{max} ne comporte que des '1', donnant très rapidement une distribution contrastée. Les vraisemblances des autres lignes sont déterminées de telle sorte qu'on obtienne comme *posterior* une distribution normale centrée sur i_{max} et d'écart-type $Card(I)/3$. Les résultats sont les plus discriminés possible, il s'agit du meilleur cas pour le calcul stochastique.

Jeu de recherche du maximum (RMAX)

Le jeu de données RMAX, consiste uniquement en la recherche du maximum de la distribution calculée, sans nécessairement évaluer la précision de représentation de l'ensemble de la distribution. Il peut correspondre à des applications de classification, ou encore de localisation, comme celle du bateau décrite en section 1.2.3. Il se trouve que l'identification du maximum de la distribution ne requiert pas forcément le calcul de toutes les valeurs de probabilités de manière très précise, et peut donc se faire avec le calcul stochastique en beaucoup moins de pas de temps.

Pour construire ce jeu de données, nous considérons le *prior* comme uniforme, donc chaque biais à la valeur de probabilité maximale $\frac{2^{\text{resp}}-1}{2^{\text{resp}}}$ comme mentionné en section 2.1.2. On considère que les vraisemblances suivent des loi normales centrées en une certaine valeur attendue μ_{jk} et du même écart-type $\sigma_{\texttt{RMAX}}$:

$$\mathbb{P}(O_k|I=i_j) \sim \mathcal{N}(\mu_{jk},\ \sigma^2_{\texttt{RMAX}})$$

Les valeurs attendues sont définies par :

$$\mu_{jk} = (2^{\text{res}_\text{o}}-1) \times \left[\frac{j \times 2^{\text{res}_\text{o}} + rand_k}{N_\text{L}} \pmod{2^{\text{res}_\text{o}}} \right]$$

res_o étant la résolution des capteurs (en bits), j l'indice de la ligne, k l'indice de la colonne et N_L le nombre de lignes de la matrice. De ce fait, pour une colonne k donnée, elles sont régulièrement réparties sur les 2^{res_o} valeurs possibles, c'est le cas d'une discrimination parfaite des caractéristiques des données. En d'autres termes, les valeurs modèles étant les plus espacées possible, les risques de confusion sont minimisés.

La composante selon les colonnes n'ayant pas d'importance dans ce jeu de données, elle est choisie aléatoirement ($rand_k$).

L'écart-type unique $\sigma_{\texttt{RMAX}}$ est souhaité conservateur, c'est-à-dire assez large afin d'obtenir des vraisemblances élevées permettant un calcul d'inférence stochastique rapide. Cet écart-type, n'est pas nécessairement le même que celui du bruit de mesure défini plus bas, puisqu'ici, le but n'est pas de rendre compte de la bonne représentation de l'entièreté de la distribution, mais uniquement du maximum.

Les échantillons capteurs o_k sont donnés en fixant une ligne $j_{\texttt{MAX}}$ comme étant celle du maximum de distribution, c'est celle de la vérité-terrain. Chaque échantillon est alors défini comme la valeur attendue de la ligne $j_{\texttt{MAX}}$ à laquelle on ajoute un bruit Gaussien d'écart-type $\sigma_{\texttt{noise}}$:

$$o_k = \mu_{j_{\texttt{MAX}}k} + \texttt{noise}, \quad \texttt{noise} \sim \mathcal{N}(0,\, \sigma^2_{\texttt{noise}})$$

L'écart-type $\sigma_{\texttt{noise}}$ est défini de telle sorte qu'en calcul flottant, le TRM atteigne la valeur de 90%. Cela traduit le fait que, même avec une précision idéale, 10% des maximum de distribution ne correspondent pas à la vérité-terrain dû au bruit et aux approximations du modèle.

3.2.3 Vérification fonctionnelle

Pour garantir la fonctionnalité de la puce à la tension nominale, plusieurs vérifications sont effectuées pour chaque architecture, avec :

— les jeux de données nul, certain et randomisé ;

— une longueur de *bitstream*, `TIMEOUT`, de 1000, 10000 et 100000 cycles ;

— une valeur maximale de compteurs, `MAX_COUNTERS`, de 2^8, 2^{12} et 2^{16}.

Il en résulte que pour un jeu de données aléatoires répété, chaque distribution de sortie est identique à celle obtenue avec un simulateur du circuit en langage C++ exact au bit et au cycle près. Ce simulateur est équivalent à celui décrit dans [112]. Par conséquent, nous pouvons considérer que chaque bloc de chaque architecture est entièrement fonctionnel.

3.2.4 Tenue en tension

Afin de tester la tenue en tension du circuit, nous reproduisons exactement le même protocole de vérification fonctionnelle qu'en section précédente, mais avec des plages de tension différentes. Plus précisément, le calcul est effectué en boucle et la tension est réduite jusqu'à obtenir des distributions différentes de la simulation, reflétant un dysfonctionnement du circuit.

D'après nos mesures, il est possible de réduire la tension d'alimentation principale VDD à 0.71 V sans aucune erreur pour chaque architecture. En dessous de cette tension, des inversions de bits (*bit-flips*) apparaissent dans les compteurs. De même, il est possible de réduire la tension d'alimentation des IOs, VDDE, jusqu'à 0.83 V, en dessous de laquelle la communication avec la liaison SPI est perdue.

Les mêmes résultats sont obtenus avec la fréquence maximale de 50 MHz, ou des fréquences inférieures : 25 MHz, 12.5 MHz et 6.25 MHz. Ainsi, le chemin critique ne semble pas être le facteur limitant, ce qui signifie que la fréquence pourrait certainement être fixée plus haut que 50 MHz.

3.2.5 Mesures de puissance

Protocole

La puissance instantanée du circuit est obtenue en mesurant la tension U_R aux bornes d'une résistance présente sur le PCB entre le potentiel d'alimentation VDD et la puce. La valeur de la résistance est choisie la plus faible possible pour limiter la chute de tension sur l'alimentation de la puce, tout en restant suffisamment élevée pour obtenir une mesure de tension précise. Dans notre expérience, elle est fixée à $R = 499\,\text{m}\Omega$. La mesure est réalisée grâce à un multimètre permettant une précision jusqu'au µV. La puissance consommée par la puce est donnée par la loi d'Ohm : $P_{cons} = \frac{U_R \times V_{DD}}{R}$.

Les mesures sont effectuées en exécutant les calculs en boucle afin de moyenner les mesures sur une plus longue période de temps et de les rendre plus précises. Pour analyser plus en détail la contribution des modules individuels à la consommation d'énergie, les tests individuels des blocs suivants sont effectués :

— consommation statiques des mémoires seules (`MEM`), sans génération de vraisemblances ou calcul d'inférence, il n'y a alors ni lecture ni écriture, les mémoires sont simplement "allumées",

— génération et chargement des vraisemblances (`LKH`), sans calcul,

— calcul stochastique seul (`CMP`), sans génération de vraisemblances,

— consommation du système entier (`ALL`), avec génération de vraisemblances et calcul stochastique d'inférence de 10000 cycles.

De plus, nous comparons les différences de consommation électrique pour les jeux de données nul, certain et randomisé, un jeu de données avec des vraisemblances plus élevées ayant tendance à activer les compteurs plus souvent et donc à consommer plus de puissance. Enfin, l'alimentation varie entre la tension nominale (1.1 V) et la tension minimale sans erreur (0.71 V) telle que déterminée dans la section 3.2.4.

Résultats

Les résultats de la consommation de puissance sont rassemblés dans le tableau 3.1. La figure 3.4 montre la consommation de puissance du système entier (`ALL`) pour toutes les architectures et avec différents jeux de données.

	VDD	1.1 V				0.71 V			
Jeu de données	config.	MEM	LKH	CMP	ALL	MEM	LKH	CMP	ALL
randomisé (RAND)	2x11	3.40	3.47	1.11	1.11	1.36	1.38	0.39	0.39
	16x11	6.29	6.61	1.75	1.75	2.55	2.67	0.63	0.63
	32x5	2.73	2.88	1.01	1.01	1.08	1.14	0.36	0.36
	64x9_gen	5.34	5.69	2.17	2.19	2.15	2.29	0.83	0.85
nul (tout à 0)	2x11	3.41	3.47	1.12	1.10	1.35	1.38	0.40	0.39
	16x11	6.29	6.37	1.67	1.67	2.53	2.56	0.60	0.60
	32x5	2.74	2.77	0.95	0.96	1.08	1.09	0.33	0.33
	64x9_gen	5.36	5.46	1.98	1.98	2.15	2.18	1.13	0.72
certain (tout à 1)	2x11	3.41	3.47	1.11	1.11	1.36	1.39	0.40	0.39
	16x11	6.29	6.40	1.82	1.84	2.53	2.58	0.66	0.67
	32x5	2.73	2.78	1.28	1.29	1.08	1.10	0.47	0.47
	64x9_gen	5.36	5.46	2.60	2.65	2.15	2.19	0.99	1.00

TABLEAU 3.1 – *Consommation de puissance (en mW) des différentes architectures avec plusieurs jeux de données et alimentations.* `MEM` *: mémoires uniquement,* `LKHD` *: calcul des vraisemblances,* `CMP` *: calcul stochastique uniquement,* `ALL` *: calcul des vraisemblances puis calcul stochastique de 10000 cycles.*

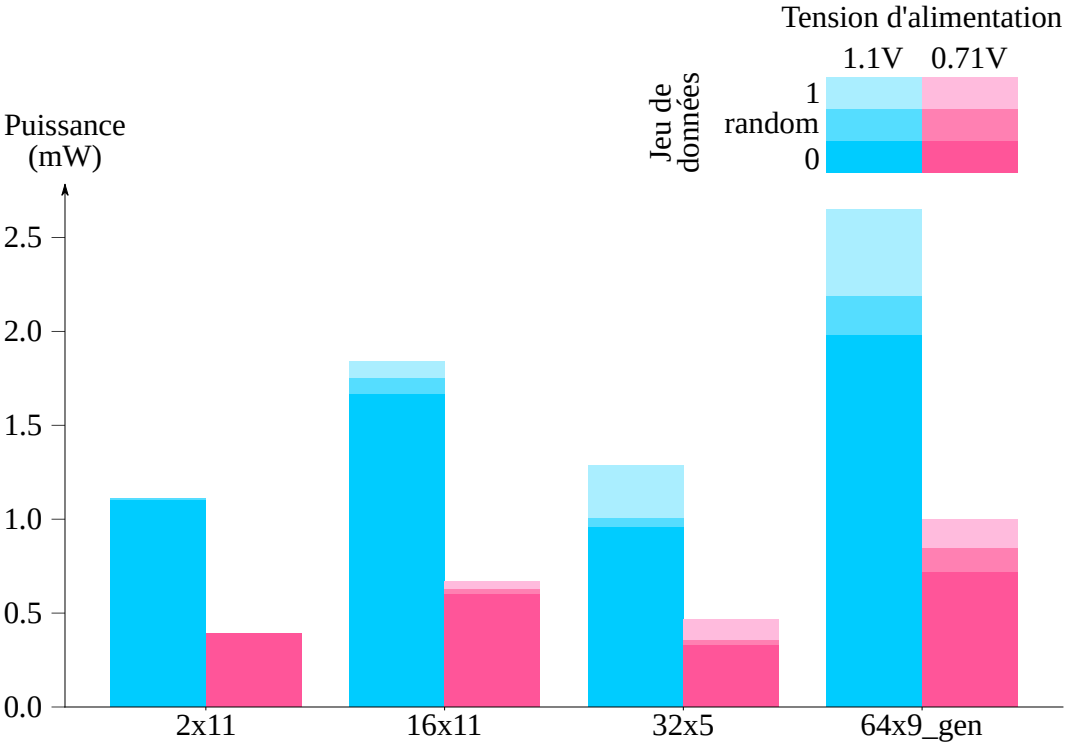

FIGURE 3.4 – *Consommation de puissance pour l'ensemble du calcul Bayésien en fonction de l'architecture, du jeu de données et de l'alimentation électrique.*

On observe que la consommation de puissance n'est pas directement proportionnelle au nombre de lignes, car multiplier le nombre de lignes par 8 de l'architecture 2x11 à 16x11 n'augmente la consommation que d'un facteur 1.6. D'un autre côté, le comportement de la consommation en fonction du nombre de colonnes semble beaucoup plus linéaire, avec une augmentation d'un facteur 2.2 lorsque l'on double le nombre de colonnes [1] de l'architecture 4x32 à 8x64. Cela est principalement dû aux RNGs, responsables d'une grande partie de la consommation, et qui sont implémentés à chaque colonne.

De plus, les configurations utilisant les mémoires (`MEM` et `LKH`) consomment le double ou triple de la puissance de la configuration `CMP` où elles sont éteintes. Cela montre l'impact des mémoires sur la consommation du circuit, et l'efficacité de la solution de mise en veille de celles-ci.

On remarque aussi que le jeu de données influe sur la consommation, avec la plus grande puissance dépensée pour le cas où les vraisemblances sont toutes à une probabilité de 1, et la plus faible pour le cas où ces vraisemblances sont de probabilités nulles. Cela est dû à la consommation des compteurs, qui s'incrémentent à chaque cycle d'horloge pour le premier cas, et ne sont jamais activés pour le second. Dans le jeu de donnée 0, les compteurs, les portes ET, le bloc de contrôle et les mémoires ne consommant *a priori* que très peu, la consommation qui en résulte peut être en grande partie imputée à la génération des nombres stochastiques. Dans nos simulations du circuit, celle-ci est en effet estimée responsable jusqu'à 70% de la consommation totale du circuit, avec 40% pour les RNGs et 30% pour les comparateurs. Le jeu de donnée aléatoire, quant à lui, donne une consommation de puissance entre ces deux cas extrêmes.

Enfin, réduire la tension de 1.1 à 0.71V, soit d'un facteur 1.6, permet de diviser par plus de 2.5 la consommation de puissance, confirmant le comportement quadratique de la consommation électrique en fonction de la tension d'alimentation.

3.2.6 Mesures de précision

Protocole

Comme mentionné dans la section 1.3.3, l'énergie consommée par un circuit stochastique est très liée à la précision souhaitée car toutes deux dépendent du temps de calcul. Avant toute comparaison de performances énergétiques avec d'autres circuits, ils est donc essentiel de rendre compte de la précision que de telles architectures stochastiques peuvent atteindre pour un temps de calcul donné. Ces mesures sont effectuées en utilisant les jeux de données `RAND`, `NORM` et `RMAX` définis en section 3.2.2.

Nous procédons au préalable à une étude (semblable à celle de [61]) afin de déterminer les graines optimales minimisant la corrélation entre SNGs (SCC_{moy}).

1. Remarquons que nous doublons aussi le nombre de lignes.

Résultats

Les résultats de précision en termes de fidélité de représentation des distributions par la KLD (par opposition à la recherche du maximum) sont détaillés dans le tableau 3.2.

Jeu de données	RAND				NORM			
Longueur de *bitstream* (en cycles)	2^{N_C+1}	2^{N_C+5}	2^{N_C+9}	2^{N_C+13}	2^4	2^8	2^{12}	2^{16}
KLD	2.7e-1	3.5e-2	5.5e-3	8.3e-4	2.9e-2	5.8e-3	3.4e-4	6.1e-5

TABLEAU 3.2 – *Précision atteinte par toutes les architectures stochastiques en fonction du temps de calcul et de leur nombre de colonnes pour les jeux de données* `RAND` *et* `NORM`.

On observe que, dans le jeu de données `RAND`, la précision dépend bien sûr du temps de calcul, mais aussi du nombre de colonnes N_C. En effet, pour chaque ligne il y a N_C opérations ET à réaliser. Or pour obtenir un '1' en fin de ligne afin d'incrémenter le compteur, il faut que les N_C *bitstreams* de biais aléatoires donnent tous un '1' en même temps, soit en moyenne une chance sur 2^{N_C}. C'est le phénomène de la dilution temporelle décrit en section 1.3.3.

En revanche, dans le jeu de données `NORM`, la précision ne dépend que du temps de calcul. En effet, les données sont normalisées, il n'y a donc pas de perte en précision due au nombre de colonnes car tous les compteurs s'incrémentent de manière idéale (le compteur du maximum de distribution s'incrémente par exemple à chaque cycle). La seule chose qui pourrait affecter cette précision est la corrélation des RNGs, mais l'étude des graines montre que pour 10 colonnes ou moins, cela n'est pas significatif.

A titre d'exemple, la figure 3.5 montre les distributions de probabilité obtenues pour le jeu de données `NORM` (construit pour converger vers une distribution Gaussienne), sur 512 lignes, et pour des valeurs de KLD de 10^{-1}, 10^{-2}, 10^{-3} et 10^{-4}.

Notons que la précision du calcul en virgule flottante simple précision est d'environ KLD $= 6 \times 10^{-4}$ pour le cas `RAND`, et d'environ 3×10^{-5} pour `NORM`, et elle correspond approximativement à la précision maximale possible due à la discrétisation des échantillons du capteur.

Concernant le jeu de données `RMAX`, la figure 3.6 montre le taux de reconnaissance du maximum (TRM) simulé pour différentes longueurs de *bitstreams*.

On remarque qu'après $2^8 = 256$ cycles, tous les circuits stochastiques étudiés présentent la même précision que la multiplication en virgule flottante (90 %), il n'est donc pas nécessaire d'augmenter la taille du *bitstream* au delà. De plus, la courbe atteint un point d'inflexion pour une longueur de *bitstream* de $2^5 = 32$ cycles, où le TRM se trouve entre 85 % et 90 %.

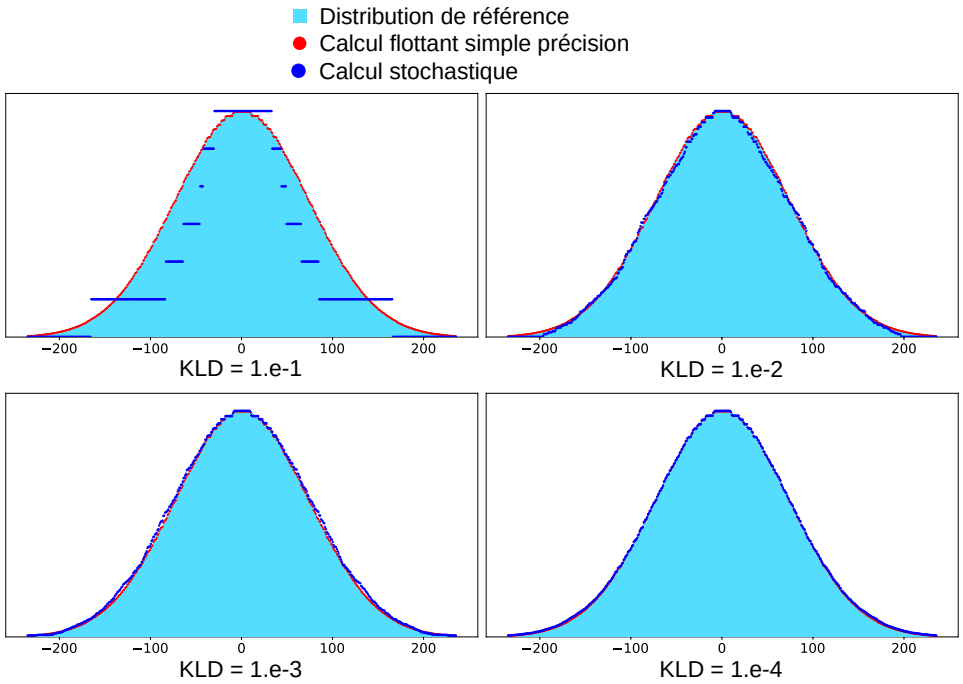

FIGURE 3.5 – *Distributions de probabilités obtenues en calcul stochastique avec les cas tests `NORM` en fonction de leur valeur de KLD, en comparaison avec les résultats en calcul flottant.*

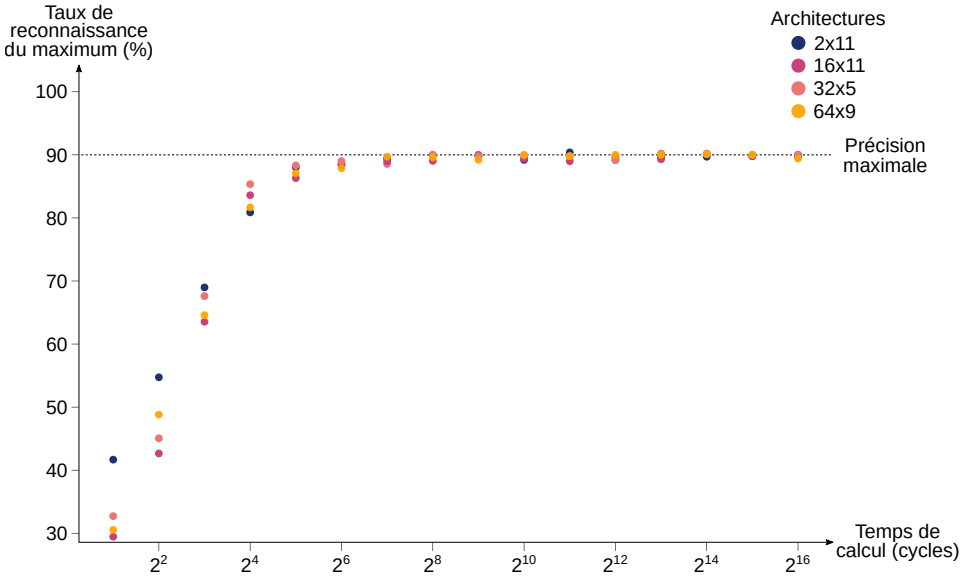

FIGURE 3.6 – *Taux de reconnaissance du maximum de distribution des différentes architectures en fonction de la longueur de leur bitstreams.*

3.3 Comparaison avec un microcontrôleur ultra basse consommation

Nous comparons la consommation énergétique de la puce avec celle du microcontrôleur Renesas RE01.

3.3.1 Caractéristiques du microcontrôleur Renesas RE01

Le Renesas RE01 [113] [2] est un microcontrôleur conçu pour l'IoT, basé sur la technologie *Silicon on Thin Buried Oxyde* (SOTB) [114] permettant une très faible consommation avec un courant d'activité de 35 µA / MHz, pour une tension d'alimentation minimale de 1.65V. Il possède même une interface pour l'*energy harvesting* pour permettre son alimentation grâce à de faibles sources de puissance comme des cellules photovoltaïques. Sa fréquence d'horloge peut atteindre 64 MHz à tension nominale de 3.3V, mais elle est réduite à 32 MHz pour la tensions d'alimentation de 1.65V. Ce microcontrôleur réalise des opérations d'entiers sur 8 bits, mais des opérations en virgule flottante peuvent être émulées via un compilateur associé. Cette classe de microcontrôleurs fait partie des principaux concurrents aux circuits dédiés à la fusion de capteurs.

3.3.2 Consommation d'énergie du microcontrôleur

En utilisant des multiplications en virgule flottante 32 bits à 32 MHz et une tension d'entrée nominale de 1.65 V, le microcontrôleur consomme environ 2 mW pour traiter les inférences. Cependant, une seule multiplication à virgule flottante de 32 bits nécessite environ 100 cycles d'horloge pour être émulée.

La consommation d'énergie du pour le calcul d'inférence est la même pour tous les cas de test étudiés, elle est donnée dans le tableau 3.3.

Architecture	Puissance consommée(mW)	Temps de calcul (cycles)	Energie consommée (nJ)
2x11	2.030	64.81	131.5
16x11	1.960	491.6	963.6
32x5	1.969	402.6	792.8
64x9	1.996	1588	3169

TABLEAU 3.3 – *Consommation énergétique du microcontrôleur Renesas RE01 en fonction de différentes dimensions de matrice.*

2. Le site de Renesas ne semble plus intégré la description ni le manuel du RE01, nous citons ici le site de vente Avnet.com qui en fait la description.

3.3.3 Comparaison énergétique

En comparaison, la consommation d'énergie du calcul stochastique de notre circuit à 0.71 V, avec les jeux de données `RAND`, `NORM` et `RMAX` selon les différents niveaux de précision est présentée dans le tableau 3.4.

	Jeu de données					
Précision :	Randomisé		Normalisé		Recherche du max	
Niveau	Faible	Haute	Faible	Haute	Faible	Haute
Metrique	KLD		KLD		TRM	
Valeur	2.7e-1	8.3e-4	2.9e-2	6.1e-5	85%	90%
Architecture	Consommation d'énergie en nJ (changement relatif à RE01)					
2x11	31.9	131000	0.125	511	0.250	3.99
	(÷4.12)	(×996)	(÷1050)	(×3.88)	(÷527)	(÷32.9)
16x11	51.6	211000	0.202	826	0.403	6.45
	(÷18.7)	(×219)	(÷4770)	(÷1.17)	(÷2390)	(÷149)
32x5	0.461	1890	0.115	472	0.230	3.69
	(÷1720)	(×2.38)	(÷6890)	(÷1.68)	(÷3440)	(÷215)
64x9	17.0	69600	0.266	1080	0.519	8.3
	(÷187)	(×22)	(÷11900)	(÷2.92)	(÷6110)	(÷382)

TABLEAU 3.4 – *Consommation d'énergie pour le calcul d'inférence du circuit stochastique, et comparaison relative au Renesas RE01, pour différentes dimensions de matrice, jeux de données et précisions.*

Ces résultats confirment l'efficacité énergétique du calcul stochastique pour une faible précision, même pour le cas particulièrement défavorable du jeu de données `RAND`. En effet, pour une KLD $= 0.27$, il permet de diviser la consommation énergétique par 4 pour la pire configuration (2 lignes et 11 colonnes) et jusqu'à 1700 pour la meilleure configuration (32 lignes et 5 colonnes) par rapport à la consommation du microcontrôleur. Cependant, le temps de calcul augmente exponentiellement avec l'inverse de la KLD, de sorte qu'avec la meilleure précision (KLD $= 8.3 \times 10^{-4}$), aucune des architectures stochastiques n'est meilleure que le microcontrôleur, la meilleure étant plus de 2 fois plus consommatrice. De plus, comme le temps de calcul pour une précision constante dépend exponentiellement du nombre de colonnes dans le cas de test randomisé et que la consommation électrique instantanée dépend linéairement de ce nombre de colonnes, alors la consommation d'énergie est approximativement proportionnelle à $N_C \times 2^{N_C}$. Par conséquent, les architectures à 9 ou 11 colonnes deviennent rapidement moins intéressantes avec une grande précision pour ce pire cas de test. Une manière de contrebalancer ce phénomène serait d'utiliser la méthode de *slicing* de [32] et décrite en section 2.2.3. Cela permettrait de réduire le facteur exponentiel de la consommation

d'énergie à $t \times 2^s$ au lieu de 2^{N_C}, t étant le nombre de tranches et s le nombre de multiplications par tranche. Cependant le *slicing* induit des erreurs lors de la renormalisation, il serait donc intéressant d'étudier son impact effectif sur la consommation.

En utilisant le jeu de données `NORM`, la précision atteinte ne dépend que du temps de calcul, c'est donc l'architecture avec la plus grande dimension (64x9) qui est la plus économe en énergie par rapport au microcontrôleur. Il est donc possible de diviser la consommation d'énergie par près de 12000 pour une précision faible (KLD $= 2.9 \times 10^{-2}$), et par 2.85 pour une précision élevée (KLD $= 6.10 \times 10^{-5}$) équivalente à celle de la virgule flottante. De manière générale, pour une grande précision, les mesures de notre circuit sont du même ordre de grandeur que celles du microcontrôleur. Seule la plus petite architecture de 2x11 dimensions donne de moins bons résultats que le Renesas, multipliant l'énergie consommée par presque quatre.

Comme dans le cas précédant, la précision du jeu de données `RMAX` ne dépend pas des dimensions de la matrice de multiplications, mais uniquement de la longueur du *bitstream*. Par conséquent, l'architecture 64x9 est également la plus économe en énergie par rapport au Renesas RE01. Il lui faut 32 cycles pour atteindre un TRM de 85%, ce qui correspond à une consommation énergétique de 0.519 nJ, soit 6110 fois moins que le microcontrôleur. Pour obtenir un taux de reconnaissance similaire au calcul en virgule flottante, autour de 90%, le temps de calcul stochastique doit atteindre environ 256 cycles, consommant 8.3 nJ, soit plus de 380 fois moins d'énergie que le Renesas.

3.3.4 Analyse générale de l'évolution énergétique selon la précision

La figure 3.7 montre d'autres résultats de la consommation d'énergie de l'architecture avec 5 colonnes et 32 lignes en fonction de la précision obtenue.

Nous pouvons voir que l'énergie augmente linéairement avec la valeur de KLD (l'échelle est logarithmique sur les deux axes) pour les jeux de données `RAND` et `NORM`, jusqu'à atteindre un plateau vertical proche de la valeur KLD pour la virgule flottante, qui peut être considéré comme la précision maximale possible. La mesure de l'énergie pour la virgule flottante semble être le point d'inflexion de la courbe du jeu de données normalisé (en bleu), de sorte que l'énergie du circuit stochastique converge linéairement vers ce point. Pour le jeu de données aléatoires, cependant, ce point d'inflexion semble être un peu plus élevé que la mesure en virgule flottante.

La courbe du jeu de données normalisé a une pente légèrement plus douce (augmenter la précision consomme moins d'énergie), et elle est décalée vers la droite de plus d'une décade, même pour la mesure en virgule flottante. Ceci peut être attribué à la métrique utilisée, la KLD évaluant la divergence de la distribution entière en elle-même par rapport à celle attendue. Ainsi, même si les erreurs absolues sont les mêmes, une distribution obtenue avec le cas `NORM`, avec des données plus discriminées, est considérée comme plus informative qu'une distribution avec des données plutôt similaires obtenues avec le cas `RAND`.

Pour d'autres dimensions de matrice, le comportement des jeux de données

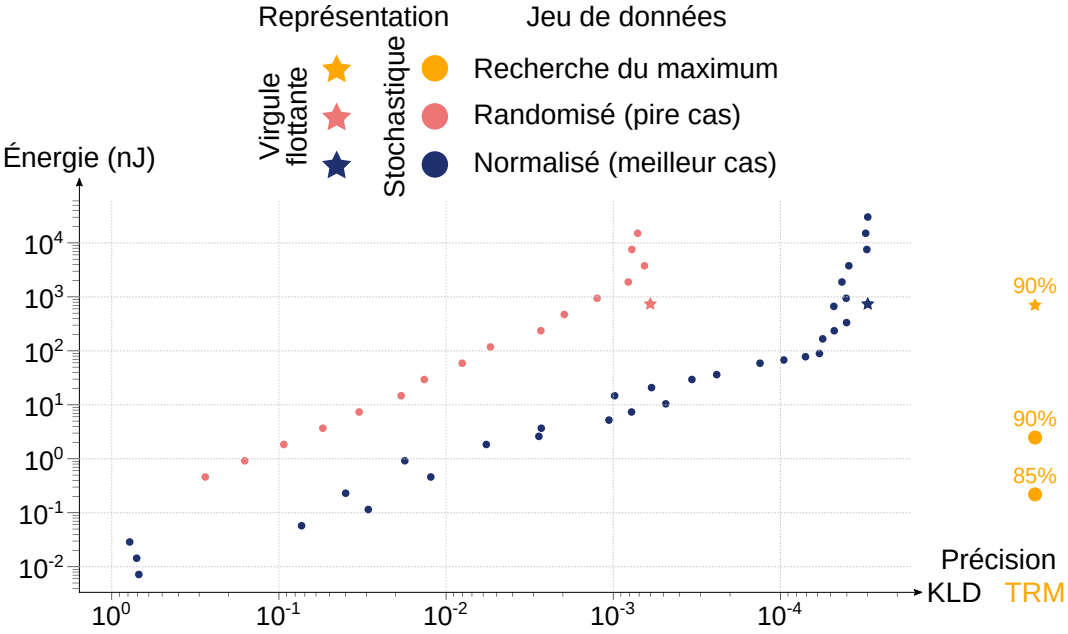

FIGURE 3.7 – *Consommation d'énergie de l'architecture 32x5 en fonction de sa précision de sortie pour les jeux données aléatoires (`RAND`), normalisées (`NORM`) et de recherche du maximum (`RMAX`).*

`NORM` et `RMAX` serait presque le même, avec seulement un léger décalage vertical proportionnel à leur différence de consommation de puissance par rapport à l'architecture 32x5, montré dans le tableau 3.1. De son côté, le comportement du jeu de données `RAND` devrait montrer un décalage vertical plus important, proportionnel à la différence de puissance multipliée par la différence de colonnes à la puissance 2 : $\Delta P \times 2^{\Delta N_C}$.

Enfin, comme il y a une grande partie des ressources qui ne dépend pas du nombre de lignes (mémoires, bloc de contrôle, RNGs), la consommation d'énergie augmente linéairement avec le nombre de lignes mais avec un *offset* important, de sorte que multiplier par 8 les nombres de lignes, de l'architecture 2x11 à 16x11, n'augmente que de 60% la consommation d'énergie.

En d'autres termes, notre circuit est particulièrement adapté aux applications avec un faible nombre de capteurs (colonnes) ou avec l'utilisation de la méthode de *slicing* de [32] ; pour un nombre élevé de lignes ; et une faible précision requise,des mesures capteurs cohérentes entre elles ou bien un cas de test favorable comme celui de la recherche du maximum de distribution.

3.3.5 Discussion

Comme dans toute comparaison, ces résultats doivent être contextualisés et tempérés car plusieurs paramètres sont différents dans ces implémentations.

Tout d'abord, la tension d'alimentation n'est pas la même, passant de 1.65 V pour le microcontrôleur à 0.71 V pour notre circuit. Cependant, nous avons remarqué que la consommation énergétique du microcontrôleur augmente linéairement avec la tension d'alimentation, contrairement à l'évolution quadratique at-

tendue. Ceci est probablement dû à la présence de régulateurs pour assurer un courant minimum dans le circuit. Ainsi, s'il était possible de diminuer l'alimentation du microcontrôleur à 0.71 V, sa consommation énergétique serait divisée par $1.65/0.71 \approx 2.3$. Cela ne disqualifierait pas notre circuit qui pourrait toujours présenter des performances énergétiques de l'ordre de 1000 fois supérieures à celles du microcontrôleur.

D'autre part, la fréquence de calcul est également différente, passant de 32 MHz pour le microcontrôleur à 50 MHz pour notre circuit, et peut certainement être fixée plus haut comme mentionné dans la section 3.2.4. Ainsi, pour une comparaison à fréquence égale, la tension d'alimentation du microcontrôleur devrait augmenter pour atteindre notre fréquence de 50 MHz, et sa consommation d'énergie avec.

Enfin, le circuit stochastique présenté ici ne bénéficie pas de plusieurs optimisations présentées dans l'état de l'art, telles que la permutation des bits aléatoires pour partager un RNG sur deux colonnes ou bien l'utilisation du WBG pour factoriser les ressources dans une colonne entière. Ces avancées permettraient un gain certain en termes de consommation énergétique, qui est développé en section 5.1.

3.4 Conclusion

L'implémentation physique d'architectures stochastiques pour la fusion Bayésienne de capteurs et leur caractérisation ont permis, d'une part, de montrer la faisabilité de telles architectures. D'autre part, elles ont révélé l'intérêt de ces architectures en termes de consommation énergétique lorsque la précision requise est faible, ou lorsque le jeu de données est favorable, avec des performances jusqu'à 6000 fois supérieures à celles d'un microcontrôleur ultra basse consommation Renesas RE01.

Cependant, cela a aussi permis de mettre en exergue trois goulots d'étranglements pour le passage à l'échelle de tels circuits vers des matrices de grandes dimensions :

— les mémoires, notamment celles de distributions Gaussiennes, qui peuvent représenter la majorité de la consommation lorsqu'elle sont allumées, et une grande partie de la surface utile du circuit,

— la génération des nombres stochastiques qui peut être responsable de plus de la moitié de la consommation de puissance du circuit, et représenter près d'un quart de sa surface,

— la consommation d'énergie qui peut s'avérer élevée pour des applications demandant une grande précision, parfois même supérieure à la multiplication flottante.

4

Optimisation des mémoires de distributions Gaussiennes

Ce chapitre propose différentes manières de réduire l'impact des mémoires de distributions demi-Gaussiennes sur la surface totale du circuit. De premières méthodes consistent en la réorganisation des mémoires par séquentialisation de la génération des vraisemblances, ou bien par implémentation d'une unique mémoire demi-Gaussienne lorsque le problème le permet. Cela rend possible de diviser par plus de deux la surface utile d'un circuit de 32 lignes et 9 colonnes. Nous proposons aussi deux algorithmes de compression mémoire, la compression par dérivation et la compression par quantification, toutes deux capables de diviser la profondeur de mémoire demi-Gaussienne requise par 4. Elles permettent ainsi une réduction de 36% de la surface totale d'un circuit de mémoires séquentialisées de 32 lignes et 9 colonnes. La surface de l'architecture originale est quant à elle divisée par 2.

Sommaire

La conception et la caractérisation du prototype dans le chapitre 3 a montré que les mémoires de distributions demi-Gaussiennes utilisées pour générer les vraisemblances représentent une partie non négligeable de la consommation et jusqu'à plus des trois quarts de la surface totale d'un circuit de fusion Bayésienne de capteurs. Nous proposons d'optimiser ces mémoires d'une part, en les réorganisant au moyen d'une séquentialisation de la génération des vraisemblances ; et d'autre part, en proposant deux méthodes de compression mémoire sans perte. Notons que dans ce chapitre, nous nous permettrons l'abus de langage de qualifier ces mémoires de "Gaussiennes" alors qu'il s'agit en réalité de mémoires de distributions demi-normales (ou demi-Gaussiennes).

4.1 Réorganisation des mémoires

4.1.1 Séquentialisation de la génération de vraisemblances

L'évolution de la surface des mémoires DRAMs en fonction de leur profondeur est affine avec un large *offset* dû à la logique d'adressage. Cet *offset* représente la majorité de la surface des mémoires dont la profondeur est inférieure ou égale à 256 mots dans la technologie 65 nm de ST Microelectronics.

Afin de s'affranchir du surcoût de ces nombreux *offsets*, nous proposons l'implémentation de mémoires uniques de $N_O \times N_L$ valeurs modèles μ (respectivement $N_O \times 2^{res_o}$ valeurs Gaussiennes) au lieu de N_O mémoires de N_L valeurs modèles (respectivement N_O mémoires de 2^{res_o} valeurs Gaussiennes). Cela permet de n'implémenter qu'une fois la logique d'adressage au lieu de N_O fois originellement, conduisant à un gain en surface. Cependant, cela nécessite aussi un traitement séquentiel de la génération des vraisemblances, au lieu de la parallélisation par colonne de la méthode originale. En effet, une seule valeur Gaussienne ne peut être adressée à chaque cycle d'horloge. Cette séquentialisation multiplie par N_O le temps de génération des vraisemblances.

Les mesures de surface et de consommation dans ce chapitre sont effectuées dans la technologie CMOS 65 nm de ST Microelectronics. Les *cuts* mémoires DRAM à notre disposition sont de 16, 32, 64 et 256 mots de 8 bits. Par régression linéaire, un schéma précis se dégage : l'*offset* dû à la logique d'adressage vaut 3976 μm² et chaque mots représente une surface de 12.425 μm². Les 4 mesures de surface passent exactement par cette droite affine. En considérant que ce comportement continue, on peut alors extrapoler les surfaces de mémoires plus volumineuses.

Pour une matrice de 32 lignes et 9 colonnes (soit $N_O = 8$), et avec une résolution des capteurs $res_o = 8$ bits, implémenter une seule mémoire modèle de 256 mots, et une seule mémoire Gaussienne de 2048 mots permet de diviser par 2.5 la surface du module de génération de vraisemblances, passant de 93800 à 38100 μm² (valeur obtenue par régression linéaire). Cela correspond à une économie de 40% de la surface totale du circuit (de 136700 à 81100 μm²). En contrepartie, le temps de génération des vraisemblances est $N_O = 8$ fois plus long, passant de 32 à 256 cycles. Il s'agit d'un surcoût modeste, qui est peu impactant dans la mesure où cette phase d'initialisation durant maintenant $N_L \times N_C$ cycles peut être largement dominé par le

nombre de cycles requis pour effectuer le calcul d'inférence comme montré dans le tableau 3.2 dans le chapitre précédant.

4.1.2 Mémoire de distribution Gaussienne unique

Lorsque les capteurs à fusionner sont du même type, les tables de distributions Gaussiennes possèdent le même écart-type. Dans ce cas, il est possible d'implémenter une unique mémoire Gaussienne pour tout le circuit de fusion de capteurs au lieu d'une mémoire par colonne. Pour ce faire, chaque vraisemblance doit être générée séquentiellement une à une, comme mentionné dans la section précédente. Cela permet de diviser la surface des mémoires Gaussiennes par un facteur N_O, mais réduit le domaine d'application de tels circuits et rallonge le temps de génération des vraisemblances de ce même facteur.

Pour une matrice de 32 lignes et 9 colonnes, n'implémenter qu'une seule mémoire Gaussienne de 256 mots permet de diviser par 6 la surface du module de génération de vraisemblances, passant de 93800 à 15600 µm², soit une division par 2.3 de la surface totale du circuit (de 136700 à 58400 µm²).

4.2 Compression de mémoire Gaussienne

Afin de réduire l'impact des mémoires Gaussiennes dans le circuit, nous proposons deux méthodes de compression qui réduisent considérablement la taille de la mémoire en contrepartie de l'implémentation d'un algorithme de décompression. L'objectif est de réduire la redondance des données dans la mémoire en profitant de la monotonie de la loi demi-normale. Son stockage original est décrit en figure 2.3.

4.2.1 Compression par dérivation

Une première approche pour compresser la mémoire Gaussienne consiste à stocker pour chaque abscisse la différence en ordonnée avec l'abscisse précédente. Ces valeurs, proches de la dérivée, peuvent être représentées en moins de bits que les valeurs brutes de la mémoire Gaussienne, notamment dans la queue de la distribution, ce qui réduit fortement la taille mémoire nécessaire. Cette méthode de compression est appelée compression par dérivation (CD).

Pour rétablir la vraisemblance correspondante, il faut soustraire les différences d'ordonnées consécutives (stockées dans der_mem) au maximum de la Gaussienne ($2^{res_p} - 1$, en considérant qu'elle est normalisée) pour chaque indice idx entre 0 et l'adresse de la Gaussienne (la différence absolue $|o - \mu|$). Le fonctionnement de cette décompression naïve est décrit dans l'algorithme 4.1.

```
def decompr_deriv(gauss_addr):
    lkhd = 2^res_p-1
    for addr in range(gauss_addr):
        lkhd -= der_mem[addr]
    return lkhd
```

Algorithme 4.1 – Algorithme naïf de décompression par dérivation.

Pour une application avec res_p et res_o sur 8 bits, il est possible de stocker les 255 dérivées correspondantes D_{0-254} dans 56 mots de 8 bits organisés comme indiqué dans le Tableau 4.1. Une étude montre que cette organisation des données permet de représenter des Gaussiennes avec des écarts types de 60 à 255 ; en dehors de ces bornes, les données doivent s'exprimer sur plus de bits, et il est impossible d'obtenir une mémoire de moins de 64 mots.

Indice de la dérivée	Taille de la dérivée (en bits)	Nombre de dérivées par mot de 8 bits	Indice du mot
0 -187	2	4	0 - 46
188 - 254	1	8	47 - 55

TABLEAU 4.1 – *Répartition des données pour la compression mémoire par dérivation lorsque* $\text{res}_\text{p} = \text{res}_\text{o} = 8$ *bits.*

Pour les applications nécessitant des Gaussiennes dont l'écart-type est inférieur à 60, nous proposons une autre méthode de compression : la compression par quantification.

4.2.2 Compression par quantification

La compression par quantification (CQ) proposée tire parti de la quantification des données de probabilité. En effet, avec l'erreur de quantification, une même ordonnée peut correspondre à plusieurs abscisses. Nous proposons donc de stocker les occurrences des abscisses que les 2^{res_p} ordonnées peuvent avoir. En d'autres termes, les axes des abscisses et des ordonnées sont en quelque sorte permutés, et nous stockons le nombre de points partageant les mêmes nouvelles abscisses (anciennes ordonnées), comme le montre la figure 4.1.

Pour rétablir les données originales, il faut accumuler les occurrences jusqu'à ce que le résultat soit supérieur à l'adresse Gaussienne (la différence absolue $|o-\mu|$). La vraisemblance correspondante est donc donnée en soustrayant le nombre d'itérations écoulées à la valeur de vraisemblance maximale ($2^{l_{size}}-1$). Le fonctionnement correspondant est développé dans l'algorithme 4.2.

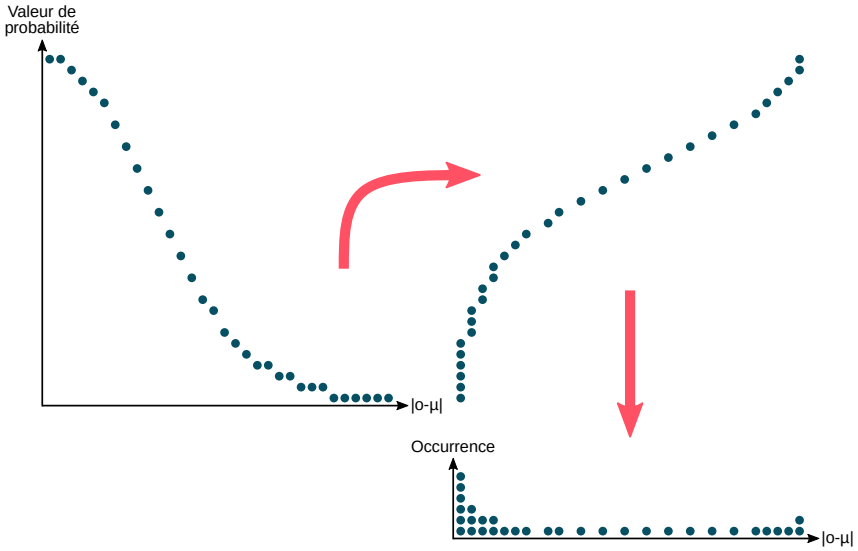

FIGURE 4.1 – *Principe de la compression par quantification (CQ).*

```
def decompr_quant(gauss_addr):
    accu_occ = 0
    for lkhd in range(2^lsize-1, 0, -1):
        accu_occ += occ_mem[lkhd]
        if accu_occ > gauss_addr: break
    return lkhd
```

Algorithme 4.2 – Algorithme naïf de décompression par quantification.

Pour une application avec res_p et res_o de 8 bits, il est possible de stocker les 255 occurrences correspondantes O_{0-254} dans 54 données de 8 bits organisées comme indiqué dans le tableau 4.2.

Indice de l'occurrence	Taille de l'occurrence (en bits)	Nombre d'occurrences par mot de 8 bits	Indice du mot
0 - 2	8	1	0 - 2
3 - 14	4	2	3 - 8
15 - 86	2	4	9 - 26
87 - 214	1	8	27 - 42
215 - 250	2	4	43 - 51
251 - 254	4	2	52 - 53

TABLEAU 4.2 – *Répartition des données de mémoire (CQ).*

D'un point de vue algorithmique, on peut voir la mémoire compressée comme une liste de 54 listes W_i, contenant $n_{sub\_data} \in \{1,2,4,8\}$ valeurs Occ_j. Cette organisation des données permet de représenter des distributions Gaussiennes avec des écarts types de 1 à 130, l'abscisse maximale étant 255.

En raison de la forte dépendance entre les valeurs de données compressées dans ces algorithmes, il est nécessaire de balayer toutes les boucles jusqu'à atteindre l'adresse d'entrée Gaussienne. Ainsi, pour une application avec $\text{res}_\text{p} = \text{res}_\text{o} = 8$ bits, il faut en moyenne 128 appels mémoire (donc 128 cycles d'horloge) par génération de vraisemblance pour l'algorithme CD et environ 160 cycles d'horloge pour le CQ.

4.2.3 Optimisation

Même si ces algorithmes sont simples et facilement implémentables en matériel, leur temps de calcul élevé représente un inconvénient majeur qui peut impacter l'efficacité énergétique du circuit.

Par conséquent, nous proposons d'optimiser les algorithmes de décompression naïfs 4.1 et 4.2 en introduisant des *checkpoints* d'accumulation afin de ne pas scanner toute la boucle, mais de la commencer avec la valeur de *checkpoint* la plus proche. Cela nécessite une implémentation de logique et mémoire supplémentaire au profit d'un temps de décompression beaucoup plus faible.

De plus, l'organisation des données du tableau 4.1 et 4.2 permet de prédire, dans la plupart des cas, la prochaine donnée sans appel mémoire. Dans l'exemple de l'algorithme CQ, cela permet de vérifier, dans le même cycle d'horloge, si l'accumulation des occurrences ajoutée aux n_{sub_data} occurrences d'un mot donné W_i est toujours inférieure à l'adresse Gaussienne ($accu\_occ + \sum_j^n W_i[j] < gauss\_addr$). Dans ce cas, nous pouvons accéder directement au mot suivant W_{i+1} sans analyser les occurrences de n_{sub_data}.

```
def decompr_quant_opt(gauss_addr):
    # move to the right checkpoint
    chkpt_index = 0
    while gauss_addr > accu_occ_chkpts[chkpt_index+1]:
        chkpt_index += 1

    accu_occ = accu_occ_chkpts[chkpt_index]
    lkhd = lkhd_chkpts[chkpt_index]
    data_index = data_index_chkpts[chkpt_index] # i
    Wi = mem_compr[data_index]
    sum_sub_data = sum(Wi)

    # move to the right word Wi
    while gauss_addr > accu_occ + sum_sub_data:
        n_sub_data = len(Wi)
        sum_sub_data = sum(Wi)
        lkhd -= n_sub_data
        accu_occ += sum_sub_data
        data_index -= 1
        Wi = mem_compr[data_index]

    # move to the right occurrence
    occ_index = 0
    while gauss_addr > accu_occ:
        lkhd -= 1
        occ_index += 1
        accu_occ += Wi[occ_index]
    return lkhd
```

Algorithme 4.3 – Algorithme de décompression par quantification optimisé.

Le fonctionnement de décompression optimisé est présenté dans l'algorithme 4.3.

Grâce à cette méthode, en utilisant 8 *checkpoints*, il est possible de réduire le temps moyen pour générer une vraisemblance à 6 cycles pour l'algorithme CD ou 7 pour le CQ, le divisant par plus de 20 fois par rapport aux implémentations naïves. Notez que cela reste supérieur à la méthode d'adressage direct décrite dans la section 2.1.3 qui prend un cycle par vraisemblance.

4.2.4 Comparaison matérielle

Implémentation

Nous implémentons ces méthodes de compression dans une technologie CMOS 65nm de ST Microelectronics et la comparons à une architecture avec séquentialisation de la génération des vraisemblances.

Nous considérons un problème avec les dimensions suivantes :

— $N_{\mathrm{O}} = 8$ capteurs à fusionner,

— une variable d'intérêt qui peut prendre $N_{\mathrm{L}} = 32$ valeurs,

— les vraisemblances codées sur $\mathrm{res_p} = 8$ bits,

— les valeurs des capteurs sur $\mathrm{res_o} = 8$ bits également,

— une fréquence d'horloge de 50 MHz.

Les *cuts* mémoires à notre disposition sont de profondeurs 16, 32, 64 et 256 mots de 8 bits.

Protocole

Afin de mesurer l'impact de la compression sur la consommation énergétique, nous simulons un calcul d'inférence avec un *bitstream* stochastique de taille 10000.

La génération des vraisemblances est traitée séquentiellement dans les architectures compressée et à adressage direct. Comme mentionné précédemment, nos solutions permettent de diviser par 4 la taille de la mémoire Gaussienne, de 256 mots de 8 bits à 64 mots, puisqu'il faut 54 ou 56 mots, la profondeur de la mémoire devant être une puissance de 2.

Nous avons besoin d'une valeur modèle pour chaque cellule de la matrice, donc 8 fois 32 valeurs modèle de 8 bits, stockées dans une mémoire de profondeur 256 pour les deux architectures. Dans le cas général, les capteurs n'étant pas *a priori* de même nature, chacun doit être associé à une distribution Gaussienne. Celles-ci sont stockées dans 8 tables de 256 mots pour l'architecture à adressage directe (une mémoire par colonne), et les distributions compressées sont stockées dans 2 mémoires de 256 mots (une mémoire pour 4 colonnes).

Résultats matériels

La comparaison en termes de surface et d'énergie des architectures à adressage direct et compressée est présentée dans le tableau 4.3.

On constate que l'implémentation de l'algorithme de décompression induit un surcoût dans le bloc de contrôle, passant de 1274 µm² dans l'architecture originale à

		Architecture		
		Séquentielle	Dérivée	Quantifiée
Surface (µm²)	Top	108900	69110	69530
	Coeur_stoch.	39750	39750	39750
	Contrôle	1274	3650	4045
	Mémoire Gauss.	57250	14310	14310
Puissance (µW)	Top	742	730	735
	Coeur_stoch.	425	437	430
	Contrôle	57.9	92.8	98.8
	Mémoire Gauss.	221	153	159
Temps de calcul (cycles)	Memories init.	2362	826	826
	likelihood gen.	256	1680	1854
	SC	10000	10000	10000
	total	12618	12506	12680
	total (µs)	126.18	125.06	126.8
Énergie (nJ)	Top	93.7	91.3	93.2
	Coeur_stoch.	53.6	54.7	54.5
	Contrôle	7.31	11.6	12.5
	Mémoire Gauss.	27.9	19.1	20.2

TABLEAU 4.3 – *Comparaison en surface et consommation énergétique des architectures à mémoire par adressage direct et compressée.*

3650 µm² dans la méthode CD, ou 4045 µm² dans la méthode CQ. Ce surcoût est cependant compensé par le gain d'un facteur 4 en compression mémoire, conduisant à une surface de mémoire Gaussienne de 57250 µm² à 14310 µm² pour les méthodes de compression. Pour l'ensemble du circuit, ces méthodes de compression permettent une réduction de la surface de 36%.

Dans le même temps, la consommation de puissance reste pratiquement la même, avec même une légère réduction pour nos propositions, le surcoût des implémentations de décompression étant compensé par la réduction de la consommation des mémoires compressées. Cette réduction de la consommation des mémoires inclue la consommation statique, un enjeu crucial pour les dispositifs de l'IoT qui sont régulièrement en veille. Ces analyses pour la consommation de puissance restent valables également pour la consommation d'énergie, le temps de calcul étant très comparable lorsque le calcul stochastique, durant 10000 cycles, est prédominant.

Comme l'initialisation des mémoires prend plus de temps pour l'architecture originale (il y a 8 mémoires de 256 mots à instancier au lieu de 2), les temps de génération des vraisemblances plus longs pour les méthodes de compression sont

compensés, totalement (pour la CD) ou presque (pour la CQ). Cela conduit à une consommation d'énergie équivalente, avec même une légère réduction de 2% pour la CD, et de 0,5% pour la CQ.

Notons enfin que comparée à l'architecture originale, sans séquentialisation, les méthodes de compression permettent de diviser par 2 la surface du circuit total, passant de 136700 à 69110 (ou 69530) µm².

Discussion

La disposition des mémoires décrite dans la section 4.2.4 est due au choix limité des *cuts* mémoire à notre disposition. Mais il aurait été plus efficace de stocker les 8 mémoires Gaussiennes non compressées dans une seule mémoire de 2048 mots, et les 2 compressées dans une mémoire de 512 mots, afin de s'affranchir l'*offset* du système d'adressage de plusieurs blocs mémoire. Pour donner une idée des résultats de l'implémentation optimisée, nous proposons des estimations de régression de surface de telles architectures similaires à celles effectuées en section 4.1. Nous pensons ces résultats proches de la réalité tant la linéarité des surfaces des différents *cuts* mémoire à notre disposition est grande. Les résultats attendus en termes de surface sont présentés dans le tableau 4.4.

		Architecture		
		Séquentielle	Dérivée	Quantifiée
Surface (µm²)	Top	81060	65130	65560
	Coeur_stoch.	39750	39750	39750
	Contrôle	1274	3650	4045
	Mémoire Gauss.	29420	10340	10340

TABLEAU 4.4 – *Comparaison attendue en termes de surface des mémoire par adressage direct et compressées obtenue via régression linéaire.*

On constate que même avec ces implémentations séquentielles optimisées, nos propositions permettent toujours de réduire la surface d'environ 19% pour l'ensemble du circuit, en divisant par 2.8 la surface de la mémoire de distribution.

4.3 Conclusion

Le chapitre 3 a montré le surcoût que représentent les mémoires de distributions sur la surface totale du circuit. Dans ce chapitre, diverses méthodes de réduction de la surface des mémoires de distributions Gaussiennes ont été proposées.

Dans un premier temps, nous effectuons une réorganisation séquentielle des mémoires de distribution Gaussienne et du modèle des capteur, jusqu'alors implémen-

tées en parallèle. Cela permet d'économiser 40% de la surface totale d'un circuit de 32 lignes et 9 colonnes, et jusqu'à 57% de cette surface lorsqu'une seule mémoire Gaussienne est utilisée. Cette dernière implémentation n'est possible que dans le cas où tous les capteurs sont considérés comme ayant le même niveau de bruit de mesure.

Dans un second temps, deux méthodes de compression des mémoires de distributions Gaussiennes sont proposées : les méthodes de compression par dérivation et par quantification. La première méthode permet une représentation de distributions Gaussiennes d'un écart-type dans un intervalle de 60 à 255 alors que cet intervalle est de 1 à 130 pour la seconde. Les deux méthodes permettent de diviser par 4 le nombre de mots de 8 bits utilisés dans les mémoires. Elles permettent toutes deux une réduction de 36% de la surface totale d'un circuit de 32 lignes et 9 colonnes par rapport à une implémentation séquentielle tout en gardant une consommation énergétique équivalente. Enfin, par rapport à la méthode originale de génération de vraisemblances, la surface peut être divisée par 2.

5

Optimisation de la génération de nombres stochastiques

Ce chapitre développe différentes méthodes pour réduire la consommation électrique liée à la génération de nombres stochastiques. Premièrement, une adaptation de l'architecture de fusion Bayésienne de capteurs originale à diverses méthodes de partage de ressources de l'état de l'art stochastique est effectuée. De plus, une étude comparative de différents RNGs a permis de mettre en lumière la qualité d'aléa suffisante pour le calcul stochastique, soulignant notamment le besoin de RNGs de faible autocorrélation pour l'utilisation de l'isolation stochastique. Enfin nous proposons le SRI, une méthode d'isolation stochastique à base de registres à décalage isolant non pas les bitstreams, mais les nombres aléatoires eux-mêmes. Il économise ainsi jusqu'à près de la moitié de la puissance consommée du coeur de calcul par rapport à une architecture optimisée, tout en gardant une surface équivalente pour une matrice de 8 lignes et 16 colonnes.

Sommaire

5.1 Adaptation de l'architecture originale à l'état de l'art du calcul stochastique

Différentes optimisations sur la génération des nombres stochastiques, inspirées de l'état de l'art de la section 2.2.2, ont été effectuées sur l'architecture originale décrite en section 2.1. Nous les décrivons successivement dans les sections suivantes.

5.1.1 Utilisation pleine des nombres aléatoires

Il était mentionné en section 3.1 que les RNGs utilisés pour la génération de nombres stochastiques dans le circuit prototype étaient des LFSRs de résolution $res_{LFSR} = 32$ bits afin d'obtenir une période d'aléa conséquente pour potentiellement représenter des *bitstreams* avec une grande précision. Sur ces 32 bits, seuls les 8 bits de poids faibles en sortie des LFSRs étaient gardés, la résolution des valeurs de probabilité étant de 8 bits également. Cette approche a l'inconvénient de consommer des ressources pour générer 32 bits aléatoires à chaque cycle d'horloge, mais n'en utilise que 8 bits, ce qui n'est pas très efficace. Or, il est possible d'utiliser la totalité des bits aléatoires générés en les partageant sur plusieurs colonnes. Pour ce faire, il suffit de diviser les 32 bits de sortie d'un LFSR en 4 ensembles de 8 bits qui constitueront les nombres aléatoires pour 4 colonnes. Ou, d'une manière plus générale, on divise les res_{RNG} bits de sortie d'un RNG en $res_{RNG}//res_p$ nombres aléatoires de res_p bits. Cela permet de diviser la surface et la consommation des SNGs par un facteur $res_{RNG}//res_p$ (=4 pour des LFSRs de 32 bits et une résolution des probabilités de 8 bits). Notons qu'ici l'approche de [61] consistant à choisir des graines optimales de LFSRs pour minimiser la corrélation entre LFSRs distincts n'a pas ou peu d'effet sur les *bitstreams* générés en répartissant les bits aléatoires d'un même LFSR. En conséquence, un inconvénient théorique de cette approche serait une plus grande corrélation entre les SNGs. En pratique, cela a relativement peu d'incidence car nous avons pu vérifier expérimentalement que pour 8 colonnes et une longueur de *bitstream* de 2^{12} cycles, la SCC est très comparable passant de 0.054 à 0.055.

5.1.2 Réutilisation de nombres aléatoires par permutation

Comme proposé par Salehi dans [91], il est possible de partager à moindre frais des nombres aléatoires sur plusieurs SNGs en appliquant des permutations de bits sur ces nombres. Salehi montre que tout comme il existe des graines de RNG optimales minimisant la corrélation entre *bitstreams*, il existe aussi des permutations optimales. Il en conclut que mettre en place un grand nombre de permutations, mêmes optimales, pour un seul RNG entraîne une forte augmentation de la corrélation. Cependant, cette perte en précision reste faible pour une seule permutation, soit un RNG partagé sur 2 SNGs. Cette permutation optimale unique correspond alors à une inversion totale de l'ordre des bits aléatoires, les bits de poids forts devenant les bits de poids faibles et vice versa. Un SNG aura donc pour nombre aléatoire celui directement généré par le RNG, et l'autre aura ce même nombre dont l'ordre des bits est inversé. Cela permet de diviser encore par 2 la surface et la consommation

des RNGs, pour une légère augmentation de la SCC moyenne des SNGs passant de 0.054 à 0.068 pour 8 colonnes et une longueur de *bitstream* de 2^{12} cycles.

5.1.3 WBG partagé

Dans [93], Yang *et al.* ont proposé une manière de factoriser une partie du WBG sur un ensemble de SNGs indépendants du point de vue du calcul. En effet, comme développé en section 2.2.2, le *Weight Generator* (WG) est indépendant de la probabilité à encoder, il peut donc, à l'instar d'un RNG, être partagé sur toute une colonne. Ce concept est illustré en figure 5.1.

Puisque le WG consiste en un grand nombre de portes logiques ET, il génère beaucoup de *glitches*, ce qui entraîne une forte surconsommation en puissance. Pour pallier ce problème, nous introduisons des registres tampons entre la sortie du WG et le reste du circuit.

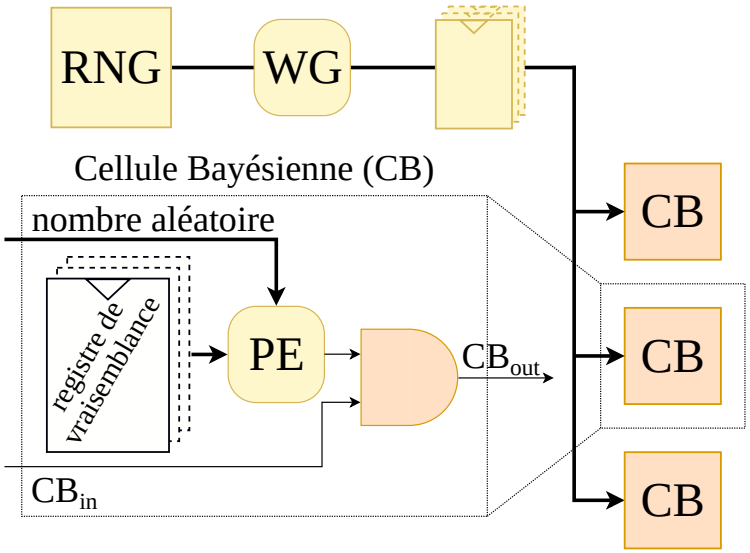

FIGURE 5.1 – *Partage d'un WBG sur une colonne.*

Comme le *Probability Encoder* (PE), la seule partie du WBG à être intégrée dans chaque cellule, est plus petite qu'un comparateur, cela permet un gain en surface et consommation des SNGs par rapport à l'architecture originale. En effet, dans la technologie 65nm de ST Microelectronics, un comparateur 8 bits fait environ 50 µm², alors qu'un PE en fait environ 23. Le WG avec les registres tampons occupent quand à eux une surface de 115 µm² environ. Ainsi, pour une matrice de 8 colonnes et 64 lignes, cette méthode permet une économie de près de 15% sur la totalité du coeur de calcul, avec une surface passant de 89556 à 76299 µm². Cela implique surtout une réduction de 32% de la consommation en puissance, passant de 1.286 à 0.871 mW. Enfin, l'utilisation du WBG a aussi pour effet la réduction de la corrélation entre SNGs, avec une SCC moyenne passant de 0.054 à 0.048.

5.1.4 Résultats de l'ensemble optimisé

L'architecture intégrant l'ensemble des optimisations est illustrée en figure 5.2.

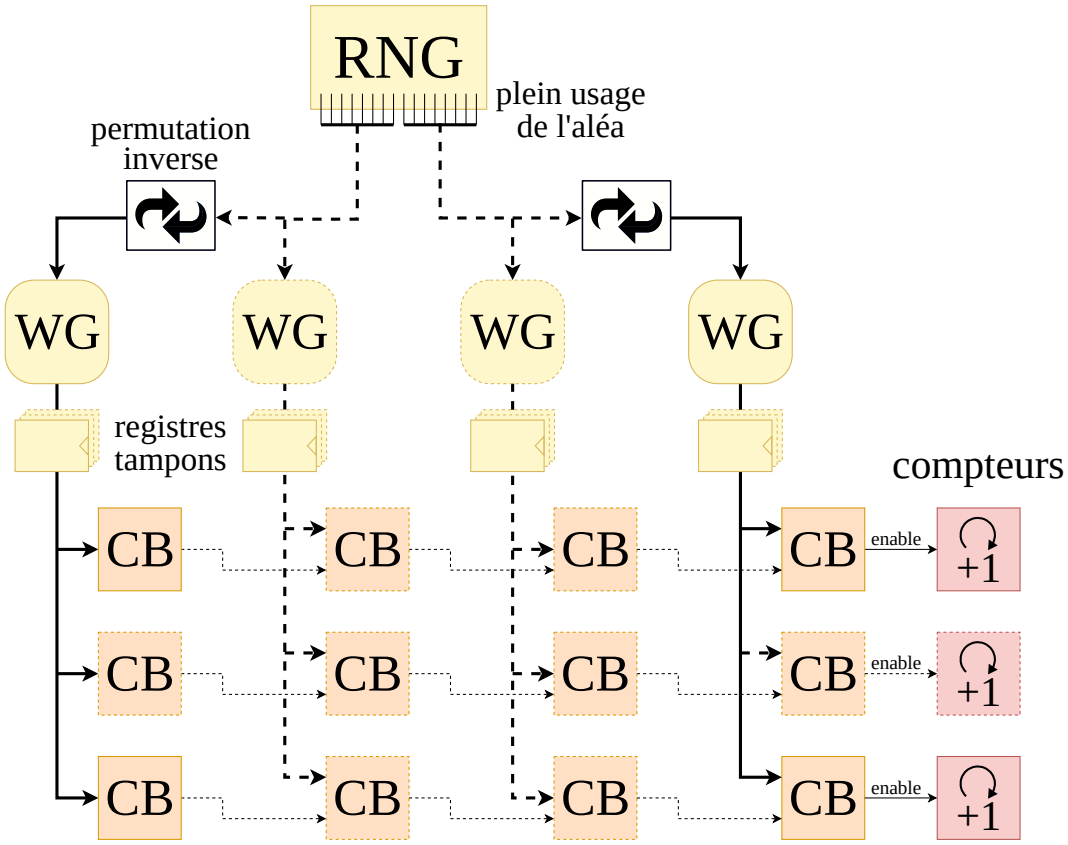

FIGURE 5.2 – *Architecture optimisée intégrant le plein usage de l'aléa, la permutation inverse des bits, et le WBG.*

Pour cette même matrice de 64 lignes et 8 colonnes, toutes ces optimisations permettent ensemble un gain de surface du coeur de calcul de plus de 21%, passant de 89556 à 70157 µm², en divisant la surface du bloc de RNGs par 3.9, de 7037 à 1815 µm². La puissance du coeur de calcul, quant à elle, est divisée par 3.4 passant de 1.286 à 0.376 mW. La précision reste sensiblement la même, avec une SCC passant de 0.054 à 0.064. Notons que pour toutes les différences de SCC mesurées dans cette section 5.1, la mesure de KLD ne donne pas de différence significative entre toutes ces architectures, avec des divergences mesurées autour de 0.04 pour une longueur de *bitstream* de 2^{12} cycles.

Cette architecture intégrant l'ensemble des optimisations de ces 3 dernières sous-sections, et illustrée par la figure 5.2 constitue une nouvelle référence pour la comparaison avec nos contributions. Elle est qualifiée par la suite d'architecture "optimisée".

5.1.5 Isolation stochastique

Une méthode dite d'isolation stochastique [49, 62] permet de partager un seul RNG pour générer plusieurs nombres stochastiques. Pour ce faire, on introduit une bascule (isolateur) dans les *bitstreams* afin de décaler le *bitstream* d'un cycle dans le temps. Appliquée à notre matrice de multiplications stochastiques, cette méthode

donne l'architecture illustrée dans la figure 5.3.

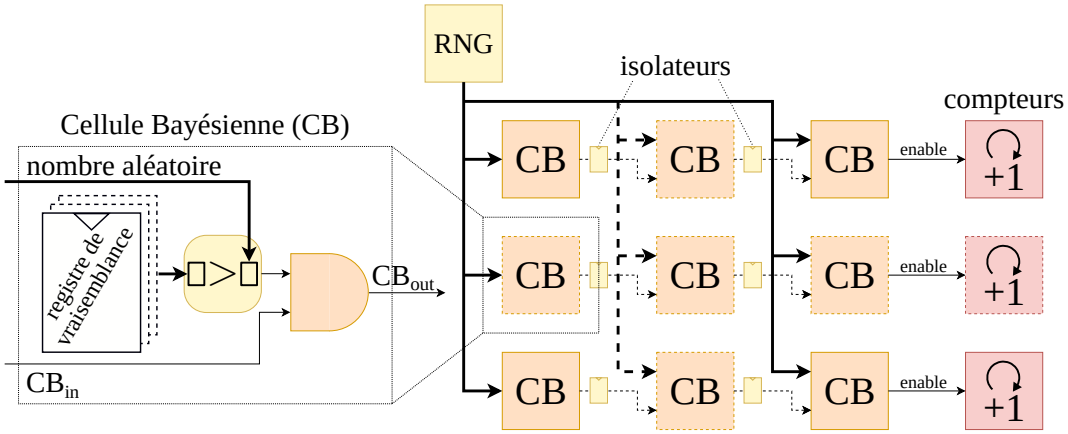

FIGURE 5.3 – *Implémentation de l'isolation stochastique dans la matrice de multiplications.*

En réalité, les optimisations des sous-sections précédentes peuvent être mises en place dans cette architecture. Au lieu de réduire le nombre de RNGs (on en utilise un seul dans ce cas) ces optimisations sont plutôt utilisées pour arrêter régulièrement le RNG afin qu'il consomme moins. Par soucis de clarté, cette méthode n'est pas développée ici, mais un système similaire est introduit plus tard dans la section 5.3.2.

Cependant, cette méthode souffre de plusieurs inconvénients.

D'abord, la grande quantité de bascules à insérer dans la matrice induit des surcoûts qui annulent parfois le gain dû au fait de n'implémenter qu'un seul RNG. C'est le cas d'une matrice de 64 lignes et 8 colonnes en utilisant un LFSR de 32 bits. En effet, avec une architecture optimisée, pour 8 colonnes et res_{RNG} = 32 bits, on utilise déjà un unique RNG. La mise en place des bascules pour chaque cellule de la matrice (ou presque) implique donc un surcoût matériel. Il représente une augmentation de 9% de la surface de la matrice (de 57170 à 62390 µm²) soit plus de 6% de la surface totale (de 70410 à 75110 µm²). Pire, la puissance totale est doublée (de 376 à 752 µW), du fait de la consommation de la matrice qui, elle, a plus que triplé (de 218 à 692 µW).

Cependant, dans le cas plus favorable d'une matrice de 8 lignes et 16 colonnes, la tendance s'inverse. En effet, pour 16 colonnes et res_{RNG} = 32 bits, il est nécessaire d'implémenter 2 LFSRs 32 bits pour l'architecture optimisée, là où on n'en utilise qu'un seul pour la méthode d'isolation stochastique. Cela a pour effet une légère réduction de la surface totale de 3% (de 18980 à 18480 µm²), et une baisse plus importante de la consommation, de 30% (de 328 à 227 µW).

Enfin, et surtout, cette méthode mène à une sévère perte en précision. En effet, en utilisant un LFSR de 32 bits comme unique RNG, la SCC moyenne passe de 0.054 à 0.34. De la même manière, la KLD atteint une valeur de 0.55, au lieu des 0.04 initiaux, ce qui est très significatif et non acceptable. Ce résultat est dû au fait que le LFSR est un RNG de faible qualité d'aléa, dont les nombres aléatoires sont fortement corrélés d'un cycle à l'autre (autocorrélés). Or, la méthode d'isolation stochastique utilise justement cette dimension temporelle pour réutiliser les *bits-*

treams, ce qui n'est efficace sans perte en précision que pour des RNGs avec une faible autocorrélation. Ce phénomène avait déjà été souligné dans [62] et dans [90], Neugebauer *et. al* proposent le SBoNG, un RNG de faible autocorrélation spécifiquement conçu pour l'utilisation de l'isolation stochastique.

Cela motive la mise en place d'une étude comparative de différents RNGs.

5.2 Étude comparative de RNGs

La plupart des circuits stochastiques intègrent comme RNG des LFSRs pour leur extrême simplicité et leur efficacité en termes de surface et de consommation. Cependant, de tels RNGs sont de piètre qualité statistique, et cela peut avoir un impact sur la précision stochastique. C'est pourquoi nous réalisons une étude comparative de différents RNGs en termes de précision, surface et consommation afin de déterminer quels RNGs peuvent être utilisés de manière optimale dans notre architecture stochastique de fusion Bayésienne de capteurs. En effet, nous cherchons à économiser le plus d'énergie possible sans pour autant sacrifier la précision des résultats.

5.2.1 Protocole

Cette étude est divisée en deux parties. Premièrement, une comparaison en termes de corrélation et de précision est réalisée au niveau logiciel, en simulant les comportements des architectures avec les différents RNGs en langage C++. Deuxièmement, une comparaison en termes de surface et de puissance est réalisée au niveau matériel en implémentant ces architectures dans une technologie CMOS 65 nm de STMicroelectronics et en utilisant des simulations rétro-annotées.

Nous comparons les RNGs suivants :

— des LFSRs [59] donnant des nombres aléatoires de 8, 16 et 32 bits, notés respectivement LFSR8, LFSR16 et LFSR32,

— des Xoroshiros [104] donnant des nombres aléatoires de 16 et 32 bits, notés respectivement XORO16 et XORO32,

— des SBoNGs [90] donnant des nombres aléatoires de 8, 16 et 32 bits, notés respectivement SBONG8, SBONG16 et SBONG32,

— un générateur de nombres réellement aléatoires, le STRNG de [110] de 63 étages.

Ces RNGs qui ont été identifiés comme les principaux candidats dans notre revue de la littérature de la section 2.3 sont implémentés comparativement avec ou sans les optimisations de la section 5.1.

Dans le cadre de cette étude, le STRNG n'a pas pour but d'être implémenté en matériel car il nécessiterait une grande quantité de ressources logiques comparé aux RNGs déterministes introduits ici, ce qui induirait une grande surface et puissance consommée. Ce dernier est plutôt utilisé en tant que référence, car en tant

que TRNG, il peut être considéré comme idéal. Son comportement est défini à partir d'enregistrements de millions de bits aléatoires effectués sur un circuit fabriqué dans la technologie AMS 350 nm dans le cadre de la thèse de K. Cherkaoui [111].

Nous définissons les dimensions du problème comme suit :

— les vraisemblances et nombres aléatoires sont codés sur $\text{res}_\text{p} = 8$ bits,

— les compteurs ont une taille de 8 bits,

— le nombre de colonnes (nombre de capteurs + 1) est fixé à 8,

— le nombre de lignes est fixé à 64 pour la comparaison matérielle, et est choisi aussi grand que possible pour la comparaison logicielle afin d'obtenir des résultats stables dans un temps de calcul raisonnable,

— la longueur des *bitstreams* varie exponentiellement de 2^3 à 2^{23} cycles.

Pour mettre ces chiffres en perspective, notons que l'application de localisation de sources sonores présentée dans [32] met en œuvre la fusion Bayésienne de capteurs avec 4096 lignes et 104 colonnes (regroupées en tranches de 10 colonnes) et considère des longueurs de *bitstreams* typiques variant entre 2^{12} et 2^{23}.

Pour une comparaison équitable, nous effectuons au préalable une étude sur les graines des RNGs, similaire à celle développée dans [61], en recherchant celles minimisant la SCC_{moy} (définie en section 1.4.3) entre chaque paire possible de SNGs. Cette étude est réalisée de manière exhaustive pour le LFSR 8 bits, mais le problème devient rapidement intractable avec la taille croissante des RNGs. Par conséquent, nous utilisons des échantillons de Monte Carlo pour calculer la SCC_{moy} par paires avec un ensemble de graines générées aléatoirement $N_{\text{essais}} \geq 1000$ fois, et enregistrons celles qui ont la plus faible SCC_{moy} par paires. Notre étude montre que le choix optimal des graines réduit la KLD jusqu'à 30% par rapport à des graines choisies aléatoirement, ce qui est significatif.

De même, le calcul de SCC_{moy} est difficile sur les 8 colonnes, il est donc également effectué avec des valeurs de vraisemblances générées aléatoirement sur un grand nombre de lignes (environ 10000) et non pas de manière exhaustive sur $2^{N_\text{C} \times \text{res}_\text{p}} = 2^{64}$ lignes.

Les mesures de KLD sont effectuées en même temps avec ce même ensemble de données aléatoires. Notons que ce cas de figure correspond au jeu de données `RAND` "pire cas" décrit en section 3.2.2. La KLD est mesurée en utilisant comme référence le résultat d'un calcul en virgule flottante 64 bits. Ici il n'est pas question de se référer aux résultats d'une vérité-terrain car nous ne comparons que des architectures stochastiques entre elles.

Les mesures de surface et de puissance étant indépendantes du temps de calcul, les résultats matériels ne dépendent pas de la longueur du *bitstream*. Pour les simulations matérielles, cette longueur est arbitrairement choisie à 10000 cycles, un nombre suffisamment grand pour obtenir des résultats représentatifs. Les résultats présentés par la suite restent donc valables quelque soit la valeur de longueur de *bitstream*.

5.2.2 Comparaison en précision

Architecture originale et optimisée

La figure 5.4 montre l'impact des RNGs sur la KLD dans une architecture sans les optimisations de la section 5.1.

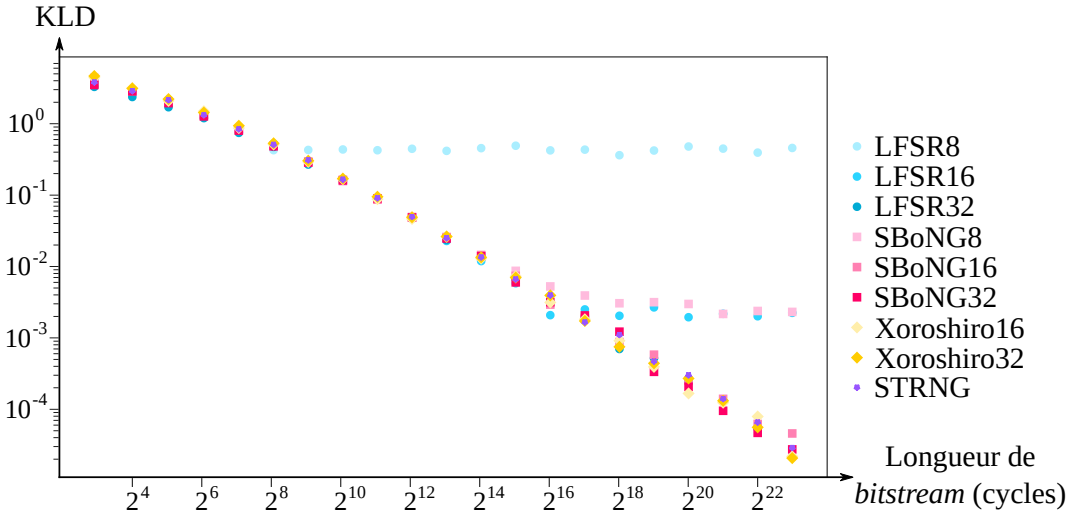

FIGURE 5.4 – *Comparaison en précision (KLD) d'architectures stochastiques utilisant différents RNGs en fonction de la longueur du bitstream.*

On en déduit qu'indépendamment du RNG choisi, si la longueur du *bitstream* est plus courte que la période d'aléa du RNG [1], tous les RNGs étudiés conduisent à des KLDs similaires, même le TRNG. En d'autres termes, en ce qui concerne la précision de la multiplication stochastique, même les RNGs à forte autocorrélation comme les LFSRs ont tendance à se comporter comme des RNGs idéaux tant que le *bitstream* est plus court que leur période d'aléa. Au-delà de cette période, la courbe de la KLD atteint un plateau et la précision ne peut plus être améliorée par une plus grande longueur de *bitstream*.

En réalité, comme les optimisations de la section 5.1 n'impactent pas ou très peu la précision du calcul, on n'observe pas une différence significative de mesure de KLD avec les résultats précédents. Les courbes de la figure 5.4 peuvent donc aussi bien correspondre à celles d'une architecture optimisée avec l'implémentation du WBG, d'une permutation inverse ou avec l'utilisation pleine des bits aléatoires.

Ainsi, bien que cela paraisse peu intuitif et à condition qu'on ne dépasse pas leur période, l'utilisation de simples LFSRs montrant de très faibles qualités d'aléa, est suffisante et n'engendre pas de perte en précision pour des architectures n'utilisant pas l'isolation stochastique. Pour de telles architectures, il n'est donc pas nécessaire d'utiliser des RNGs plus sophistiqués comme le SBoNG, le Xoroshiro ou bien le STRNG car cela induirait un surcoût superflu en termes de surface et de consommation.

1. $2^{res_{RNG}} - 1$ pour le LFSR et $2^{2\times res_{RNG}}$ pour le SBoNG et le Xoroshiro, res_{RNG} étant la précision de sortie du RNG en bits

Isolation stochastique

La figure 5.5 montre l'impact de l'isolation stochastique sur la KLD.

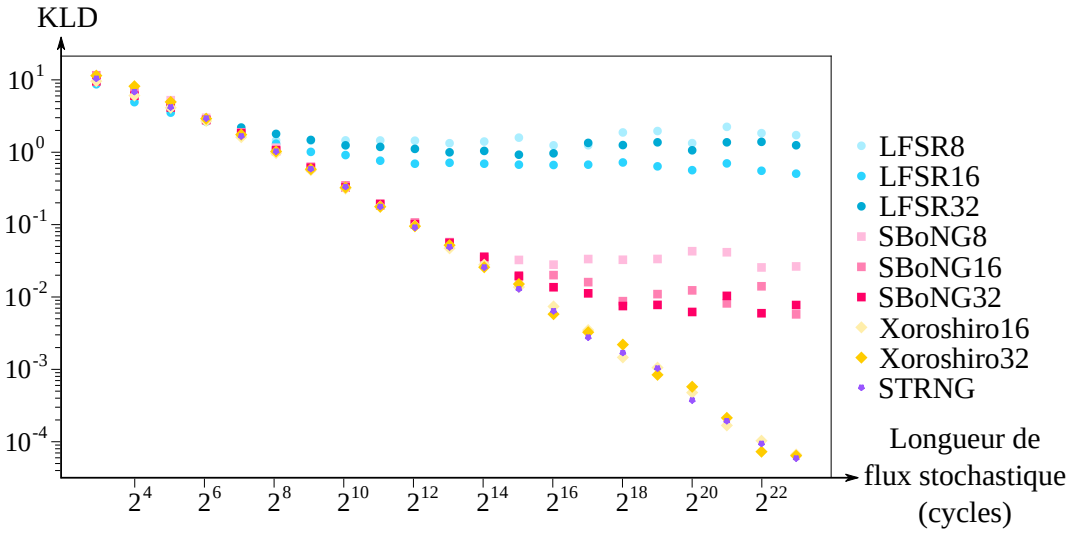

FIGURE 5.5 – *Comparaison en précision (KLD) d'architectures stochastiques utilisant l'isolation stochastique avec différents RNGs en fonction de la longueur du bitstream.*

Les RNGs simples et fortement autocorrélés, tels que les LFSRs, ne sont pas compatibles avec l'isolation stochastique car ils entraînent une perte de précision significative. La précision est également affectée lors de l'utilisation de SBoNGs, mais si la longueur du *bitstream* ne dépasse pas 2^{13} cycles, la perte de KLD reste négligeable. Pour les RNGs à faible autocorrélation, comme le Xoroshiro ou le STRNG, l'isolation stochastique n'affecte pas la KLD, du moins pas pour les 2^{23} premiers cycles.

Le tableau 5.1 fournit des mesures plus détaillées de la KLD et de la SCC pour différentes configurations de SNG, y compris avec l'isolation stochastique, pour une longueur de *bitstream* de 2^{13}.

On remarque que toutes les architectures atteignent sensiblement la même précision idéale, la même que celle atteinte par le STRNG. Jusqu'à ce point de rupture de 2^{13}, l'architecture isolée atteint également une précision idéale équivalente pour les RNGs faiblement autocorrélés tels que le SBoNG et le Xoroshiro.

Conclusion

En conclusion, lors de l'utilisation de RNGs pseudo-aléatoires, la précision du calcul augmente avec la longueur du *bitstream* tant qu'elle est inférieure à la période du RNG. Ce résultat tend à soutenir l'utilisation exclusive de RNGs simples tels que les LFSRs pour le calcul stochastique puisque les RNGs plus sophistiqués et volumineux tels que Xoroshiro ou le STRNG n'améliorent pas significativement la précision. Cependant, pour permettre l'utilisation de l'isolation stochastique, les RNGs doivent montrer une faible autocorrélation.

Architecture	RNG	Mesure de précision après 2^{13} cycles	
		SCC	KLD
Originale	LFSR-16	0.037	0.018
	LFSR-32	0.037	0.016
	SBoNG-8	0.036	0.014
	Xoroshiro-16	0.034	0.014
	STRNG	0.034	0.014
Optimisée	LFSR-16	0.041	0.014
	LFSR-32	0.048	0.016
	SBoNG-8	0.042	0.017
	Xoroshiro-16	0.043	0.015
	STRNG	0.042	0.015
Isolation stochastique	SBoNG-8	0.048	0.026
	Xoroshiro-16	0.029	0.015
	STRNG	0.029	0.013

TABLEAU 5.1 – *Comparaison en termes de précision d'architectures état de l'art, optimisée et avec isolation stochastique en utilisant différents RNGs.*

5.2.3 Comparaison matérielle

RNGs seuls

La figure 5.6 montre les mesures en termes de surface et de consommation des différents RNGs uniquement.

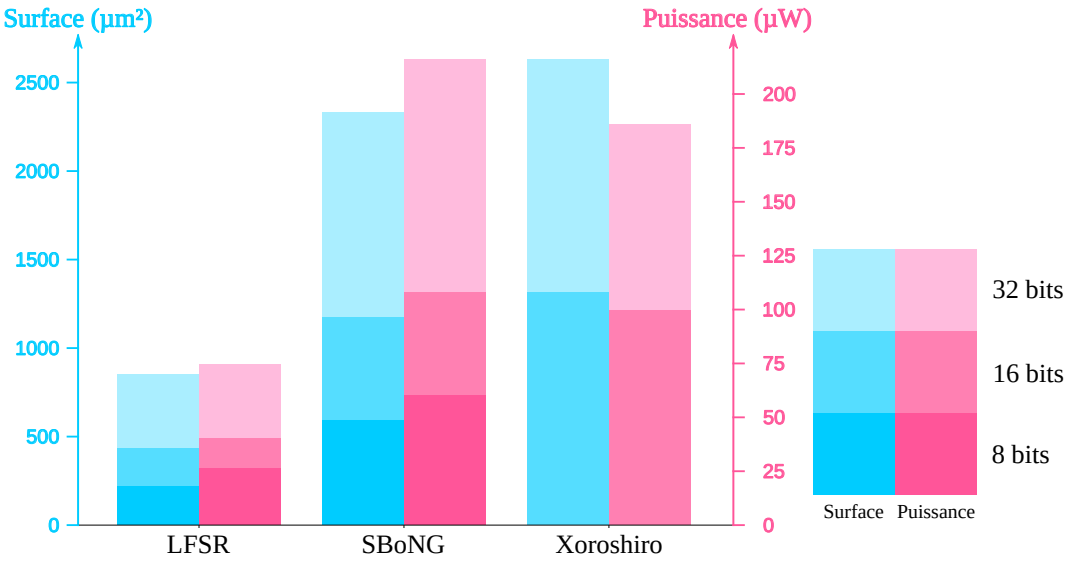

FIGURE 5.6 – *Comparaison en surface et consommation de différents RNGs.*

On observe que les LFSRs sont de loin les RNGs les plus compacts et les moins consommant. Un LFSR 32 bits est en effet 3 fois plus petit (de 850 à 2630 µm²), et 2.5 fois moins consommant (de 75 à 186 µW) qu'un Xoroshiro 32 bits. Les rapports sont du même ordre de grandeur pour le SBoNG. L'utilisation de ces RNGs sophistiqués est donc à exclure pour une architecture n'utilisant pas l'isolation stochastique.

Architecture optimisée

Le tableau 5.2 compile ces mêmes mesures matérielles des différents RNGs dans une architecture stochastique optimisée de 64 lignes et 8 colonnes.

RNG	LFSR			SboNG			Xoroshiro	
Résolution (bits)	8	16	32	8	16	32	16	32
	Surface (µm²)							
Coeur_stoch.	70490	70440	70410	71970	71950	71930	72230	72230
> bloc_RNG	1893	1842	1815	3369	3350	3327	3635	3629
> > RNGs	870.5	860.1	849.7	2357	2340	2329	2625	2630
> Matrice	57170	57170	57170	57170	57170	57170	57170	57170
> Compteurs	11430	11430	11430	11430	11430	11430	11430	11430
	Puissance consommée (µW)							
Coeur_stoch.	419	388	376	521	494	495	487	474
> bloc_RNG	172	147	135	298	272	272	264	250
> > RNGs	106	80.6	74.6	241	216	216	199	186
> Matrice	223	220	218	202	202	201	202	202
> Compteurs	22.9	20.4	22.3	20.6	20.7	22.1	21.3	21.3

TABLEAU 5.2 – *Comparaison en surface et consommation d'une matrice de multiplications stochastique optimisée de 64 lignes et 8 colonnes avec différents RNGs.*

On remarque sans surprise que ce sont les architectures implémentant les LFSRs qui sont les plus compactes et les moins consommantes. En effet, une architecture optimisée avec un SBoNG 32 bits est 2% plus volumineuse et 32% plus consommante que celle avec le LFSR 32 bits. L'architecture du Xoroshiro 32 bits atteint quand à elle une augmentation de 3% de la surface et 26% de la puissance.

De plus, avec l'utilisation pleine des bits aléatoires générés, utiliser peu de RNGs de résolution élevée ou bien plusieurs RNGs de faible résolution donnent sensiblement les mêmes résultats de surface et de consommation. Il semblerait même que mettre en place moins de RNGs mais avec une plus grande résolution entraîne une réduction de la surface et de la consommation. Cela est sans doute dû

au fait que tout n'est pas linéaire dans ces implémentations, avec des ressources incompressibles pour la mise en place d'un RNG donné, si bien qu'implémenter un RNG de résolution $n \times r$ coûte moins de n fois plus cher qu'un RNG de résolution r. Utiliser des RNGs de plus grande résolution permet, de plus, d'élargir la plage d'utilisation de ces architectures sans perte en précision, car ils ont de fait une plus grande période d'aléa. Cela se fait peut-être au détriment d'une faible perte en précision car les RNGs dont les bits aléatoires sont partagés sur plusieurs colonnes ne profitent pas ou peu de l'optimisation des graines de [61].

Le RNG idéal pour une architecture optimisée est donc un LFSR de grande résolution, mais qui ne dépasse pas $\text{res}_\text{p}/n_\text{perm} = 4$ fois le nombre de colonnes, sans quoi on ne profiterait plus de certains bits aléatoires qu'il génère. D'une manière générale, en considérant des matrices d'au moins 8 colonnes, le LFSR 32 bits paraît être un bon candidat.

Isolation stochastique

Les RNGs permettant l'utilisation de l'isolation stochastique sans perte en précision sont les SBoNGs (jusqu'à une longueur de *bitstream* de 2^{13}), et les Xoroshiro (potentiellement jusqu'à $2^{2\text{res}_\text{RNG}}$).

Comme les optimisations précédentes sont mises en place dans l'architecture utilisée pour évaluer cette méthode, les RNGs du même type consomment approximativement la même énergie, peu importe leur résolution res_RNG. Ainsi, pour une matrice de 64 lignes et 8 colonnes, les architectures utilisant des SBoNGs de 8, 16 et 32 bits consomment respectivement 758, 753 et 761 µW. Notons que cela est principalement le fruit de l'utilisation pleine des nombres aléatoires.

Cependant, leur surface augmente avec res_RNG. En effet, pour une petite matrice de 8 lignes et 8 colonnes, utiliser un Xoroshiro 32 bits au lieu d'un Xoroshiro 16 bits induit une augmentation de la surface de près de 14% (de 10820 à 12300 µm²), alors que leur consommation reste inchangée à 140 µW. Cette différence de comportement en termes de surface vis à vis d'une architecture optimisée est due au fait qu'ici, augmenter la résolution du RNG ne réduit pas le nombre de RNGs à mettre en place, on en implémente toujours un seul, peu importe sa taille.

Ainsi, pour la mise en oeuvre de l'isolation stochastique, il est préférable de garder des RNGs de plus faible résolution possible, tout en gardant une précision idéale dans une certaine plage d'utilisation en termes de longueur de *bitstream*. Cependant, ici le SBoNG n'est utilisable que jusqu'à 2^{13} cycles alors qu'un Xoroshiro 16 bits est utilisable jusqu'à 2^{32} cycles potentiellement. Le gain en surface de l'implémentation d'un SBoNG 8 bits face à celle d'un Xoroshiro 16 bits (800 µm² tout au plus) n'est pas suffisant pour justifier d'un tel choix. Le RNG idéal pour l'utilisation de l'isolation stochastique est donc un Xoroshiro 16 bits.

Comparaison optimisation / isolation

Nous comparons donc une architecture avec l'isolation stochastique utilisant un Xoroshiro 16 bits, appelée `ISO_XORO16`, et une architecture optimisée utilisant un LFSR 32 bits, appelée `OPT_LFSR32`, et ce pour des dimensions de matrices entre 8 et 64 lignes et 8 et 16 colonnes.

La figure 5.7 illustre cette comparaison en termes de puissance.

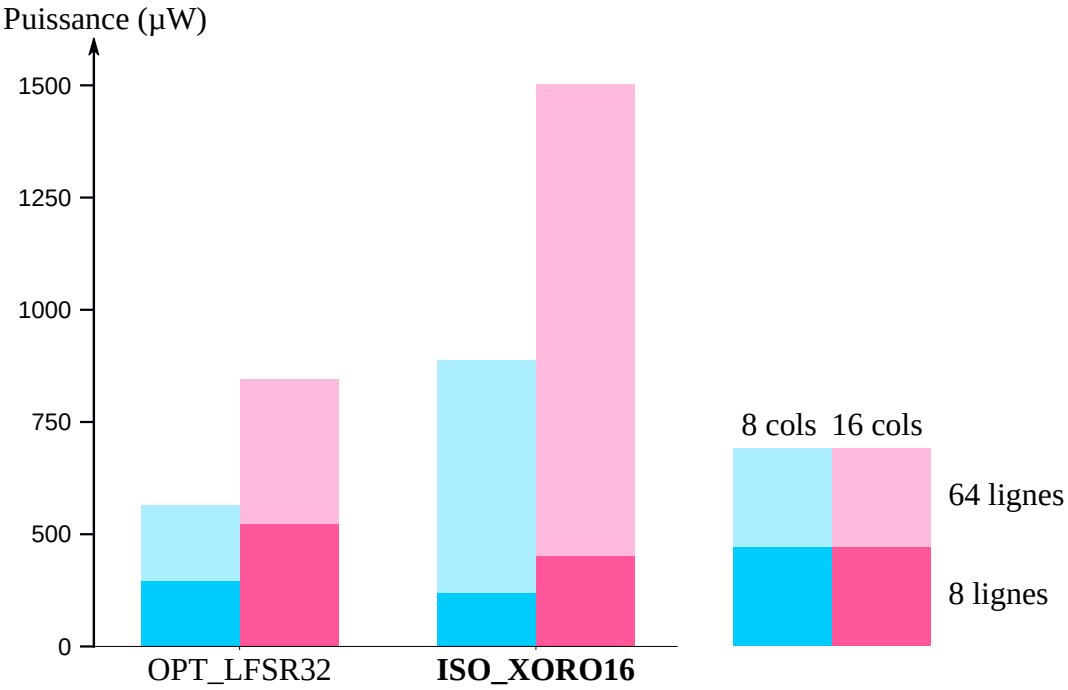

FIGURE 5.7 – *Comparaison en consommation des architectures optimisée et isolée pour différentes dimensions de matrice.*

On observe que pour un faible nombre de lignes, la méthode d'isolation est plus efficace en termes de consommation, avec une réduction jusqu'à -27% (de 328 à 238 µW) pour 8 lignes et 16 colonnes. En effet, c'est le nombres d'isolateurs à insérer entre chaque cellule de la matrice qui est responsable de la majorité de la consommation, or avec seulement 8 lignes, ce nombre est limité. Le nombre de colonnes, en revanche, est en faveur de l'isolation qui n'utilise qu'un seul RNG, là où une architecture optimisée nécessite davantage de RNGs.

Cependant, lorsque le nombre de lignes augmente, la tendance s'inverse, et c'est l'architecture optimisée qui est plus efficace avec jusqu'à plus de 2 fois moins de puissance consommée pour 64 lignes et 16 colonnes (de 711 à 1500 µW).

L'isolation stochastique apparaît donc comme une méthode intéressante, mais qui peine à passer à l'échelle pour des matrices de plus grandes dimensions, notamment en termes de lignes, à cause de ses nombreux isolateurs (bascules) à insérer entre chaque cellule de la matrice.

5.3 Shift Register Isolator

Nous introduisons une nouvelle architecture d'isolation stochastique permettant de partager un RNG sur l'entièreté du coeur de calcul stochastique tout en réduisant l'impact matériel de l'isolation, le *Shift Register Isolator* (SRI).

5.3.1 Principe

Le principe du SRI est de placer un registre à décalage, de profondeur N_C, en sortie d'un RNG. Ce RNG représente la seule source d'aléa du circuit, il est partagé sur toute la matrice de multiplications. Chaque étage du registre à décalage joue alors le rôle du RNG de la colonne dans l'architecture d'origine, donnant à toutes les cellules de la colonne un nombre aléatoire différent de ceux des autres colonnes, avec espérons-nous une faible corrélation.

L'architecture du SRI est illustrée par la figure 5.8.

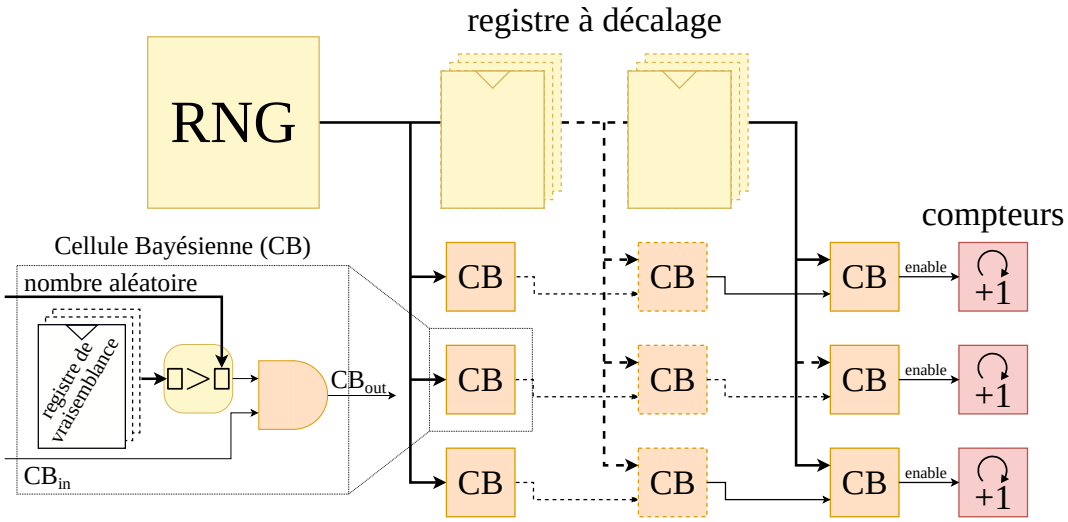

FIGURE 5.8 – *Principe du Shift Register Isolator proposé.*

Du point de vue de la logique stochastique, le SRI peut être considéré comme une méthode d'isolation stochastique (voir section 5.1.5). En effet, le registre à décalage joue le rôle de ligne d'isolateurs stochastiques, en permettant la réutilisation au temps t_{n+i} pour la colonne C_i du même nombre aléatoire généré au temps t_n pour la colonne C_0. Comme pour la méthode d'isolation stochastique, l'utilisation d'isolateurs stochastiques introduit une latence correspondant au nombre de colonnes. Cependant, cette latence est négligeable par rapport au temps de calcul associé à la longueur du *bitstream*. Et comme pour la méthode d'isolation stochastique, un unique RNG peut être partagé sur l'ensemble des SNGs.

Comme montré en section 5.2.2, en tant que méthode d'isolation stochastique, cette solution nécessite l'utilisation de RNGs possédant une faible autocorrélation, ce qui n'est pas le cas des LFSR utilisés classiquement en logique stochastique, car elle réutilise des nombres aléatoires du même RNG générés à des cycles consécutifs.

La différence avec les isolateurs de [49] et [62] est qu'ici, l'isolation stochastique est réalisée non pas au niveau des *bitstreams* avec une bascule tampon, mais au

niveau des RNGs avec un système de registres à décalage. De ce fait, pour effectuer l'isolation stochastique, il n'est plus nécessaire d'insérer une bascule après chaque cellule de la matrice de multiplications, mais uniquement un registre de res_p bits par colonne. Ainsi, pour une colonne donnée, le nombres de bascules à insérer ne correspond plus au nombre de lignes de la matrice, mais est fixé à res_p. L'architecture SRI devient alors plus intéressante que la méthode d'isolation standard de [49, 62] dès lors que le nombre de lignes est supérieur à res_p. En outre, contrairement à la méthode d'isolation stochastique standard, dans notre proposition, les portes ET ne sont pas séparées par des bascules. Elles sont par conséquent optimisées pendant la synthèse en une plus grande porte ET par ligne.

5.3.2 Optimisations

Il est possible d'intégrer au SRI les optimisations réalisées en section 5.1.

En effet, l'utilisation pleine des bits aléatoires et les permutations de ceux-ci permettent de faire fonctionner le RNG de manière alternée, un cycle d'horloge sur $N_\text{perm} \times (\text{res}_{RNG}//\text{res}_\text{p})$ [2], grâce à un système de multiplexage montré en figure 5.9. Cela permet une économie de puissance du même facteur. D'une manière générale, la résolution du RNG n'a de ce fait plus ou peu d'impact sur la puissance consommée, puisque augmenter cette résolution aura pour effet de réduire la fréquence d'utilisation dudit RNG.

De plus, la mise en place du WBG permet de n'utiliser qu'un seul WG pour tout le circuit, connecté à la sortie du RNG, juste avant le registre à décalage. Chaque cellule n'intègre alors qu'un PE, plus petit qu'un comparateur. Les étages du registre à décalage ne stockent donc plus en réalité des nombres aléatoires, mais des nombres "pondérés" (les sorties de WG). Ces étages prennent aussi la fonction des registres tampons utilisés pour couper les *glitches* en sortie des WGs introduits dans la section 5.1. L'implémentation du registre à décalage est donc en quelque sorte "gratuite" vis à vis de l'architecture optimisée de cette section.

Enfin, nous proposons une compression des données en sortie du WG. En effet, le WG introduit une forte redondance dans ses nombres de sortie, avec toujours uniquement 1 bit à '1', les autres étant nuls. Ce bit à '1' se trouve à la position p dans le nombres avec une probabilité de $\frac{1}{2^{p+1}}$. Il est alors aisé de compresser ces données en ne stockant dans le registre à décalage que la position du bit à '1' dans la sortie du WG. Cela permet de n'implémenter que $\log_2(\text{res}_\text{p})$ bascules dans chaque étage du registre à décalage plutôt que res_p, soit 3 bascules au lieu de 8 dans le cas où res_p = 8 bits. Le SRI devient alors plus avantageux que l'isolation stochastique standard, non plus si le nombre de lignes est supérieur à res_p, mais s'il est supérieur à $\log_2(\text{res}_\text{p})$. La décompression est réalisée en sortie du registre à décalage, juste avant les PEs.

En réalité, dans le cas où le nombre aléatoire vaut 0, la sortie du WG donne 0 aussi, il n'y a donc pas de bit à '1' dans ce nombre. Nous faisons le choix de confondre ce cas à celui du nombre aléatoire valant 1. Cela évite de mettre en place une bascule supplémentaire dans chaque étage du registre à décalage uniquement

2. $2 \times (32//8) = 4$ par exemple pour un RNG de 32 bits

pour le cas du nombre aléatoire nul, en contrepartie d'une moins bonne représentation des nombres stochastiques. Néanmoins, les mesures de KLD d'un tel système ne montrent pas une perte significative de la précision, cette inexactitude semblant être masquée par l'incertitude du calcul stochastique.

La figure 5.9 montre l'architecture optimisée du SRI.

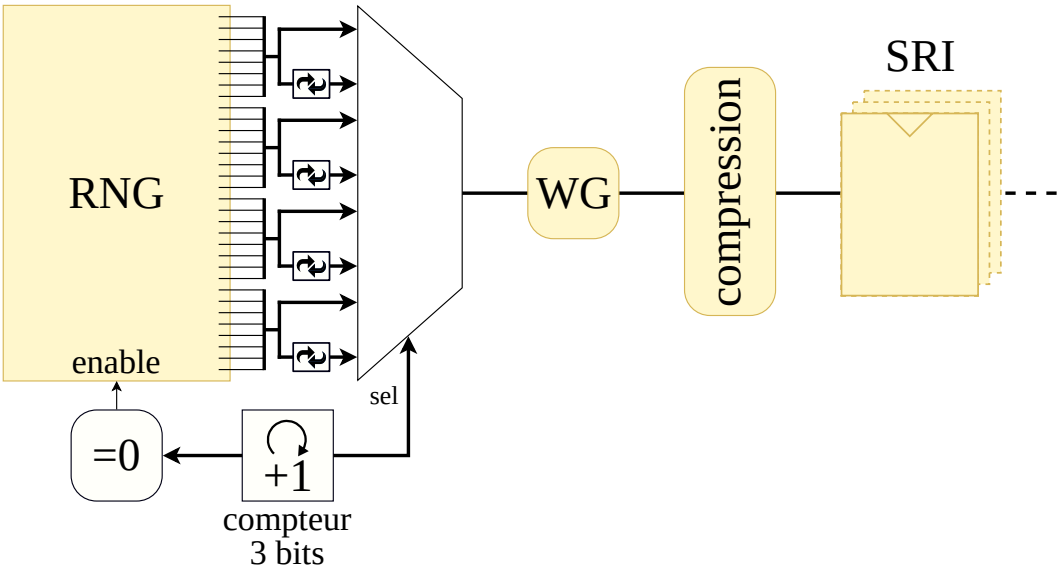

FIGURE 5.9 – *Implémentation optimisée du Shift Register Isolator proposé pour un RNG 32 bits.*

L'utilisation du SRI nécessite l'implémentation d'un RNG de bonne qualité statistique, c'est-à-dire possédant une faible autocorrélation. Afin de déterminer quel RNG il est possible d'utiliser sans dégradation de la précision, nous réalisons la même étude comparative qu'en section 5.2 avec cette fois la mise en place du SRI. Le protocole de mesure étant identique qu'en section 5.2.1, nous ne le développons pas ici.

5.3.3 Comparaison en précision

Pour l'étude de la précision, le SRI se comporte exactement comme l'isolation stochastique standard. En effet, d'un point de vue des *bitstreams*, l'architecture avec isolateurs se comporte comme si l'ordre d'attribution des colonnes pour les étages du registre à décalage étaient totalement inversé : le nombre aléatoire le plus récent est utilisé pour générer le *bitstream* de la dernière colonne et inversement. Cette inversion n'a pas d'effet notable sur la précision, elle dépend toujours autant de l'autocorrélation des RNGs.

L'étude donne donc des résultats tout à fait similaires à ceux de la sous-section 5.2.2. Nous en tirons les mêmes conclusions : l'architecture garantissant une perte en précision négligeable et la plus efficace en termes de surface et de consommation est celle utilisant comme RNG un Xoroshiro 16 bits.

Il reste à montrer si cette proposition est plus efficace que l'isolation stochastique standard lors du passage à l'échelle. L'implémentation d'un seul RNG plus

sophistiqué, et donc plus volumineux, est-elle plus compacte que plusieurs LFSRs très simples mais très autocorrélés ? La qualité d'une seule source d'aléa l'emporte-t-elle sur la quantité de petites sources ?

5.3.4 Comparaison matérielle

Nous comparons, pour des dimensions de matrices entre 8 et 64 lignes et 8 et 16 colonnes, les implémentations suivantes :

— architecture originale sans aucune optimisation appelée `ORI_LFSR32` développée en section 2.1,

— architecture optimisée utilisant un LFSR 32 bits, appelée `OPT_LFSR32`,

— architecture utilisant l'isolation stochastique avec un Xoroshiro 16 bits, appelée `ISO_XORO16`,

— notre proposition utilisant le SRI avec un Xoroshiro 16 bits appelée `SRI_XORO16`.

La figure 5.10 présente les résultats de comparaison matérielle de ces différentes architectures.

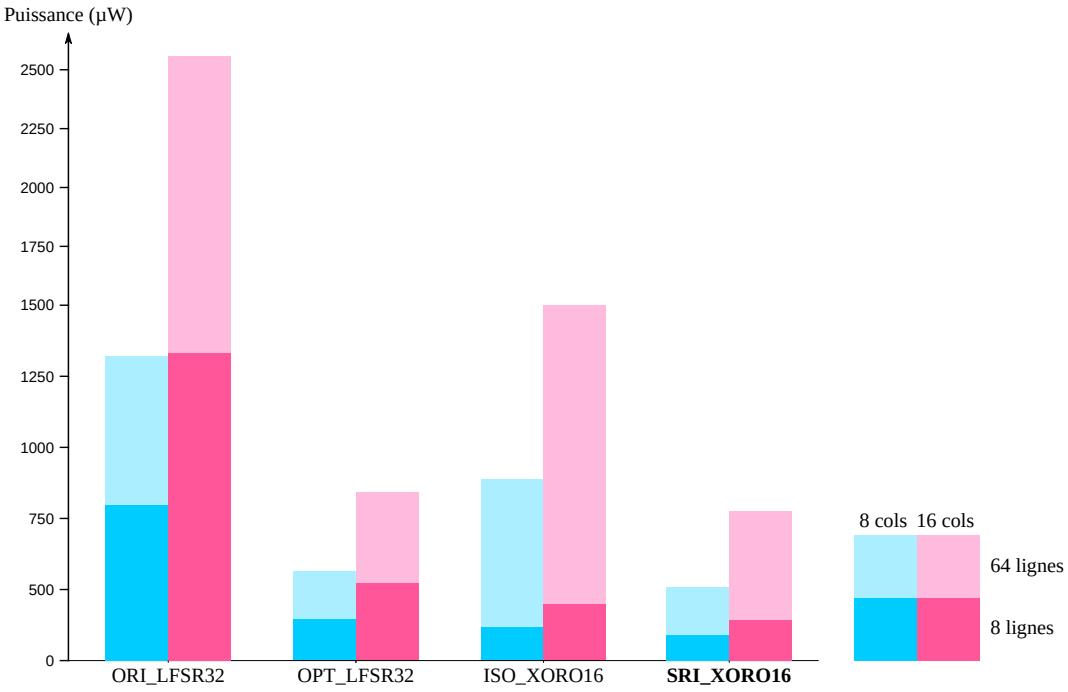

FIGURE 5.10 – *Comparaison en consommation des architectures originale, optimisée, isolée et utilisant le Shift Register Isolator pour différentes dimensions de matrice.*

On remarque que, pour toutes les dimensions de la matrice, notre proposition `SRI_XORO16` consomme moins que les deux architectures `ISO_XORO16` et `OPT_LFSR32` considérées comme références. Au maximum, avec une matrice de 8 lignes et 16 colonnes, le SRI permet une économie de 48% de la puissance du coeur de calcul par rapport à `OPT_LFSR32`, soit 28% par rapport à `ISO_XORO16`. Comparée à l'architecture originale de fusion Bayésienne de capteurs, cela représente une division par presque 8 de la puissance.

En termes de surface, le SRI donne approximativement les mêmes résultats que les deux autres architectures de référence, en étant cependant toujours meilleur d'environ 5% que `ISO_XORO16`. Comparé à l'architecture originale, la surface est presque divisée par 2.

5.4 Conclusion

Dans un premier temps, différentes optimisations inspirées de l'état de l'art (le plein usage de l'aléa, les permutations de bits aléatoires et le WBG) ont été mises en place dans l'architecture stochastique de fusion Bayésienne de capteurs. Elles ont permis des gains matériels conséquents en réduisant la surface de plus de 21% et en divisant la puissance consommée par 3.4 pour une matrice de 64 lignes et 8 colonnes, et ce sans perte en précision notable. La mise en place de l'isolation stochastique quand à elle, bien qu'intéressante car ne nécessitant l'utilisation que d'un seul RNG pour tout le circuit, entraîne cependant une forte perte en précision lorsque utilisée avec des RNGs fortement autocorrélés comme les LFSRs. De plus, elle n'est efficace que pour des matrices de petite dimension, car la mise en place de bascules entre chaque cellule de la matrice induit de forts coûts matériels.

Cela a motivé d'une part, la mise en oeuvre d'une étude comparative de plusieurs RNGs afin de déterminer quelles sont les implémentations occupant le moins de surface et consommant le moins d'énergie tout en gardant une précision optimale pour les architectures optimisées et isolées ; et, d'autre part, la recherche d'une solution d'isolation stochastique qui réussit le passage à l'échelle.

L'étude comparative a établi que pour des architectures sans isolation stochastique, pour tous les RNGs étudiés, la précision du calcul augmente de manière idéale avec la longueur du *bitstream* tant qu'elle est inférieure à la période du RNG. Le RNG le plus efficace pour une architecture optimisée est donc le LFSR, car il obtient la même précision que des RNGs plus sophistiqués, tout en étant 3 fois plus compact et 2.5 fois moins consommant. Pour l'isolation stochastique en revanche, seuls les RNGs de bonne qualité statistique, comme le Xoroshiro, le SBoNG et le STRNG, atteignent une précision idéale. Le Xoroshiro 16 bits apparaît donc comme le meilleur candidat pour sa plage d'utilisabilité sans perte en précision (jusqu'à 2^{23} cycles au moins), et sa frugalité en ressources logiques.

Dans un second temps, nous avons proposé une nouvelle méthode d'isolation stochastique basée sur l'exploitation d'un registre à décalage permettant de n'utiliser qu'un seul RNG pour tout le circuit : le SRI. Cette méthode permet de n'implémenter qu'un étage de registre à décalage par colonne au lieu de N_L bascules, conduisant à un gain en ressources utilisées par rapport à la méthode d'isolation standard. Les différentes optimisations de la section 5.1 ont pu être intégrées dans notre architecture. Comme c'est le cas pour la méthode d'isolation stochastique standard, les RNGs simples et fortement corrélés comme les LFSRs ne sont pas compatibles avec notre proposition car ils induisent de graves pertes en précision. De telles architectures requièrent alors la mise en place de RNGs de bonne qualité

statistique, comme les SBoNGs ou les Xoroshiros. Le SRI permet une économie jusqu'à 48% de la puissance par rapport à l'architecture optimisée de référence tout en gardant une surface comparable voire légèrement plus faible. Comparé à l'architecture originale sans optimisations, la puissance est divisée par près de 8, et la surface par près de 2. Ainsi, nous avons montré que des RNGs plus sophistiqués, et donc de fait plus volumineux, peuvent paradoxalement permettre une conception plus frugale en surface et en consommation, et ce sans perte en précision.

6

Amélioration de l'efficacité énergétique par codage multirail

Ce chapitre traite de l'amélioration de l'efficacité énergétique de la matrice de multiplications stochastiques par l'implémentation d'une architecture stochastique parallèle : le calcul multirail. Cette solution consiste à démultiplier chaque ligne de la matrice en un certain nombre de rails afin de réduire la latence du circuit, et donc son énergie. Elle se base sur le SRI ainsi que sur la permutation de ses sorties sur les colonnes de la matrice afin de générer les nombres stochastiques nécessaires à moindre coût. Le calcul multirail permet ainsi de diviser jusqu'à 5 la consommation énergétique du circuit total pour une augmentation de la surface de seulement 10%.

Sommaire

Bien que le calcul stochastique permette la simplification de ressources logiques et donc la réduction de surface occupée et de puissance, son temps de calcul élevée implique une consommation énergétique conséquente. Or, pour la plupart des applications fonctionnant sur batteries, réduire l'énergie consommée pour un calcul donné est un besoin fondamental.

Plusieurs solutions de parallélisation du calcul stochastique ont été décrites en section 2.2.3. Ces architectures permettent de réduire le temps de calcul stochastique et ainsi d'obtenir un gain en énergie par rapport à une implémentation totalement séquentielle. Cependant, leur but étant plus orienté sur l'augmentation des performances en temps de calcul, elles offrent des solutions massivement parallèles capables de réaliser un calcul stochastique de faible longueur de *bitstream* en très peu de cycles d'horloge, mais ne sont à ce titre pas efficientes en termes de surface d'implémentation et sous optimales en termes d'énergie.

Dans ce chapitre, nous proposons une architecture hybride, séquentielle et multirail, permettant de paralléliser le calcul stochastique, et donc de réduire la consommation énergétique, et ce sans augmenter significativement la surface du circuit stochastique. Une telle solution offre la possibilité d'un compromis entre la surface et la consommation d'énergie, qualité souvent absente des propositions de parallélisation de l'état de l'art.

6.1 Principe

Notre proposition, illustrée en figure 6.1, consiste à démultiplier chaque ligne de la matrice de multiplications en N_R rails stochastiques, permettant théoriquement de diviser par N_R le temps de calcul. Cette parallélisation en N_R rails multiplie d'autant le nombre de portes ET et de convertisseurs binaire / stochastique par rapport à une architecture totalement séquentielle standard dite monorail. Les compteurs sont aussi modifiés pour incrémenter plus d'un bit par cycle d'horloge, ce qui les complexifient sensiblement. En revanche, le nombre de registres de vraisemblance reste inchangé, ils constituent une ressource partagée entre tous les rails.

De plus, cette architecture multirail s'appuie sur la mise en place du *Shift Register Isolator* (SRI) dont l'architecture est présentée en section 5.3. Cela rend possible une génération de nombres stochastiques plus efficace que l'état de l'art en termes de puissance consommée. Aussi, l'implémentation du WBG permet de ne mettre en place pour chaque cellule de la matrice qu'un *Probability Encoder* (PE), plus de 2 fois plus compact que le comparateur utilisé classiquement. Notons que les optimisations du SRI de la figure 5.9 sont aussi mises en place dans notre proposition, mais, par soucis de clarté, elles ne sont pas illustrées en figure 6.1 afin de ne montrer que les éléments clés, à savoir le SRI et les permutations de colonnes.

Enfin, il est nécessaire d'alimenter tous les rails en nombres aléatoires différents et idéalement décorrélés, sans quoi arrêter le calcul N_R fois plus tôt n'aurait pour effet que de réduire la précision du calcul stochastique. Pour ce faire, les différents nombres aléatoires requis pour chaque rail sont générés sans coût supplémentaire en

permutant simplement les sorties du SRI entre les colonnes. Cette permutation peut se voir, dans l'architecture originale, comme un changement de l'ordre des RNGs sur les colonnes. Ainsi, chaque rail d'une ligne produit un *bitstream* indépendant puisqu'il reçoit un nombre aléatoire différent en entrée, provenant d'une source faiblement autocorrélée. Cela se fait avec une perte de précision acceptable jusqu'à un certain nombre de rails. Cette question est discutée en section 6.2.

L'architecture stochastique multirail proposée est décrite en figure 6.1.

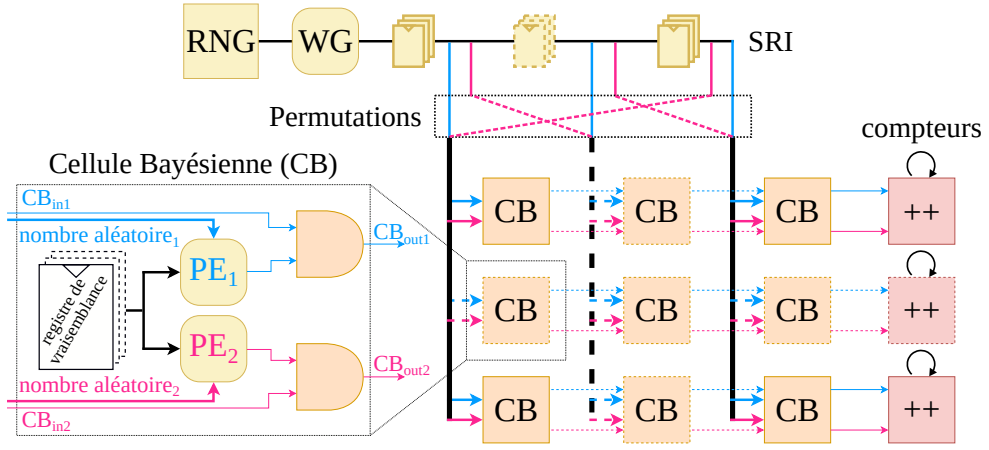

FIGURE 6.1 – *Architecture stochastique multirail proposée pour la fusion Bayésienne de capteurs. Ici le circuit comporte 2 rails, l'un en bleu et l'autre en rose.*

L'implémentation de rails supplémentaires conduit à une augmentation de la surface et de la consommation de puissance. Son réel avantage est énergétique. En effet, de ce point de vue, la consommation est liée à l'activité du circuit, c'est à dire le nombre de transitions binaires de '0' à '1' ou de '1' à '0' qui ont lieu dans le circuit dans un temps donné. À première vue, paralléliser ne permet pas en soi de réduire l'énergie consommée du calcul stochastique, car son activité reste la même, on ne fait que translater un partie du calcul de la dimension temporelle à la dimension spatiale. Cependant, comme de nombreuses ressources sont partagées entre les rails (RNG, WG, SRI, registres de vraisemblance, et même bloc de contrôle et mémoires de distributions), elle ne sont à ce titre pas concernées par la parallélisation. Ainsi, diviser le temps de calcul par N_R conduirait à réduire d'autant leur activité (sans compter la consommation statique de tout le circuit), permettant un gain significatif en énergie. C'est tout l'avantage d'utiliser le SRI et les permutations de colonnes : ils maximisent les ressources partagées non concernées par la parallélisation, permettant une plus grande réduction de l'activité du circuit et ainsi un meilleur gain en énergie consommée.

Par soucis de clarté, nous nous référons à cette proposition originale de calcul parallèle intégrant le SRI ainsi que les permutations de nombre aléatoire en tant que calcul multirail. Notons que nous n'avons pas la prétention d'avoir inventé le calcul stochastique parallèle dont il est fait état en section 2.2.3.

La consommation énergétique étant intimement liée à la précision pour des circuits stochastiques, nous étudions l'impact de notre proposition, notamment celui des permutations des colonnes, sur la précision du calcul avant de procéder aux mesures matérielles.

6.2 Étude de la précision

6.2.1 Sélection des permutations

De même qu'il existe des graines de RNGs [61], ou des permutations de bits au sein d'un nombre aléatoire [91] optimales minimisant la corrélation des séquences stochastiques obtenues, il existe des permutations de colonnes optimales. Une étude analogue à celle de [91] pour les permutations de bits dans un nombre aléatoire est donc réalisée, mais cette fois-ci pour des permutations de nombres aléatoires entiers sur des colonnes de la matrice. Son objectif est de déterminer les meilleurs ensembles de permutations de colonnes minimisant les pertes de précision. Pour cela, une simulation logicielle du circuit est réalisée en langage C++.

L'étude exhaustive de la corrélation entre plusieurs SNGs par SCC_{moy} devient rapidement intractable à mesure que le nombre de SNGs grandit. Celle-ci est donc effectuée par méthode de Monte-Carlo, sur un grand nombre de données aléatoires par SNG, qui correspond au nombre de lignes N_{L}. La mesure de KLD est également effectuée avec ces mêmes jeux de données en comparant la distribution résultante de notre circuit stochastique avec celle calculée en virgule flottante (référence).

Ce nombre de lignes doit donc être suffisamment grand pour obtenir une bonne mesure de la précision tout en permettant un temps de simulation acceptable. Il est ainsi fixé à 30000. Le nombre de colonnes est fixé à 8. La longueur de *bitstream* est variable dans le but d'atteindre trois niveaux de précision fixés à KLD = 0.021, 0.012 et 0.007. Ces valeurs de précision sont choisies de telle sorte que les longueurs de *bitstream* requises pour les atteindre, pour une architecture monorail, sont respectivement de 6000, 12000 et 24000, des valeurs facilement divisibles.

Nous mesurons d'abord cette précision avec 2 rails (donc une permutation de colonne) de manière exhaustive pour toutes les $N_{\text{C}}! == 8! = 40320$ permutations possibles, sur plusieurs jeux de données randomisés. Nous disposons donc d'un ensemble de mesures qui reflètent l'impact par paire des permutations de colonnes sur la précision, indépendamment des données. En utilisant ces mesures, nous avons construit un algorithme capable de sélectionner les permutations minimisant leur corrélation par paire pour un nombre donné de rails. Ensuite, la précision (KLD) est mesurée pour s'assurer qu'elle n'est pas trop dégradée par rapport à une implémentation sans permutation, la permutation est par exemple rejetée si la mesure de KLD a doublé. Enfin, l'ensemble des permutations sélectionnées est directement codé en dur dans le circuit. Cette méthode représente un gain conséquent en temps de simulation, la méthode exhaustive devenant rapidement intractable avec un temps d'exécution proportionnel à $8!^{N_{\text{R}}}$.

6.2.2 Longueur de *bitstream*

Quand bien même cet algorithme permet une sélection optimale des permutations de colonnes, cette méthode n'a pas un impact nul sur la précision. Or, pour une comparaison équitable, par la suite, nous souhaitons obtenir la même KLD que l'architecture monorail, afin de pouvoir comparer la consommation énergétique à précision équivalente. Nous devons donc augmenter légèrement les longueurs des

bitstreams multirails au lieu de simplement diviser celle d'une implémentation monorail par le nombre de rails. En conséquence, cette correction de précision augmente légèrement la consommation d'énergie.

Le tableau 6.1 montre les différentes longueurs de *bitstream* nécessaires pour atteindre une KLD de 0.021, 0.012 et 0.007 en fonction du nombre de rails.

	Nombre de rails N_R							
	1	2	3	4	5	6	7	8
KLD	Longueur de *bitstream* (en cycles)							
0.021	6000	3000	2050	1580	1330	1100	930	880
0.012	12000	6050	4150	3050	2500	2100	1800	1700
0.007	24000	12900	8900	6900	5900	5400	4600	4500

TABLEAU 6.1 – *Longueur du bitstream nécessaire pour atteindre différentes valeurs de KLD en fonction du nombre de rails.*

La figure 6.2 montre l'augmentation de longueur de *bitstream* nécessaire par rapport à celle théorique de $\frac{L_{\text{monorail}}}{N_R}$ (si les permutations n'engendraient pas de perte en précision) pour atteindre une précision donnée en fonction du nombre de rails.

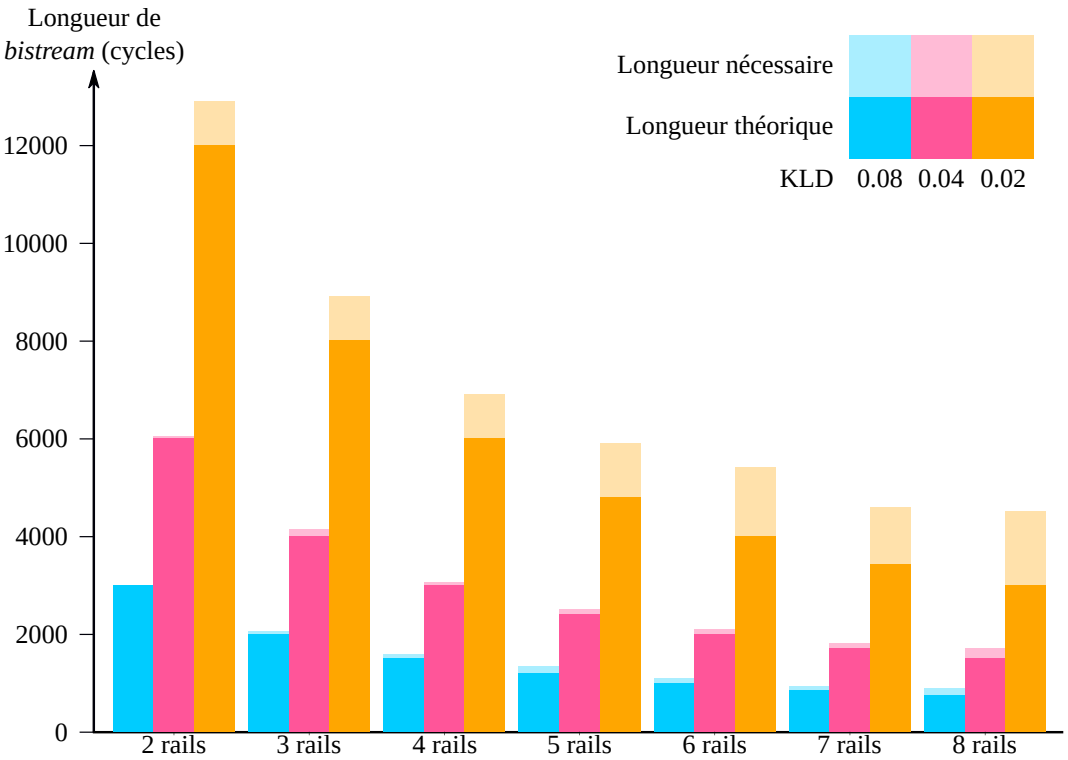

FIGURE 6.2 – *Comparaison de longueurs de bitstream théoriques et nécessaires pour atteindre une certaine précision en fonction du nombre de rails.*

On constate que l'augmentation du nombre de permutations semble limiter la capacité du circuit à atteindre de hautes précisions, comme le montre la dernière ligne du tableau 6.1. En effet, pour atteindre une KLD de 0.007, le nombre de cycles supplémentaires requis devient très important, plus de 1000 cycles lorsqu'on utilise plus de 4 rails. Cet effet reste cependant limité pour les précisions plus faibles de KLD 0.012 et 0.021. De plus, plus le nombre de rails est important, plus il faut ajouter de cycles à la longueur de *bitstream* théorique pour atteindre la KLD de référence. Ces points montrent, malgré nos efforts pour le limiter, l'impact négatif des permutations sur la précision, qui doit être compensé par une longueur de *bitstream* plus élevé et donc une consommation énergétique plus importante.

6.3 Mesure de la surface et de la consommation d'énergie

Les résultats matériels suivants en termes de surface et de consommation d'énergie sont obtenus avec une technologie CMOS 65 nm de STMicroelectronics et en utilisant des simulations post-synthèse rétro-annotées. Les dimensions du circuit sont les suivantes :

— Les vraisemblances sont codées sur 8 bits, ainsi que les entrées aléatoires des PEs, et grâce à la compression, le SRI est codé sur 3 bits ;

— Le RNG est un Xoroshiro de 16 bits, possédant la faible autocorrélation nécessaire pour utiliser le SRI ;

— Le nombre de colonnes est fixé à 8 et le nombre de lignes varie de 8 à 64 ;

— Les compteurs ont des sorties de 8 bits ;

— La précision de référence (KLD) varie dans la plage {0,007, 0,012, 0,021} ;

— La fréquence d'horloge est fixée à 50 MHz ;

— Le nombre de rails varie de 1 à 8.

Les résultats sont mesurés sur l'ensemble du circuit de fusion Bayésienne de capteurs, comprenant le coeur de calcul stochastique décrit dans la figure 6.1, mais aussi un bloc de contrôle (Finite State Machine) et les mémoires modèles et Gaussiennes (non compressées).

Le tableau 6.2 montre la comparaison absolue en consommation d'énergie et de surface lorsque le nombre de rails et la précision souhaitée varient.

La figure 6.3 montre les changements relatifs de ces mesures par rapport à l'architecture monorail pour une KLD de 0.012. Notons que les comparaisons relatives ne sont pas linéaires puisque, par exemple, le doublement de la surface entraînerait une augmentation de +100%, tandis que la division par deux de l'énergie entraînerait une réduction de -50%.

			Nombre de rails							
	KLD	Rows	1	2	3	4	5	6	7	8
Consommation d'énergie (nJ)	0.021	8	30.7	18.2	13.0	10.3	8.78	7.30	6.44	6.35
		64	79.2	49.4	37.6	31.7	29.5	28.1	25.4	24.7
	0.012	8	61.4	36.7	26.3	19.9	16.5	13.9	12.5	12.3
		64	158	99.6	76.1	61.2	55.5	53.6	49.2	47.8
	0.007	8	123	78.2	56.4	45.1	38.9	35.9	31.8	32.5
		64	317	212	163	139	131	138	126	127
Surface (10^3µm²)		8	72.3	74.8	76.1	76.9	78.0	79.0	79.9	81.1
		64	160	181	192	199	208	216	223	231

TABLEAU 6.2 – *Comparaison de l'énergie et de la surface en fonction du nombre de rails.*

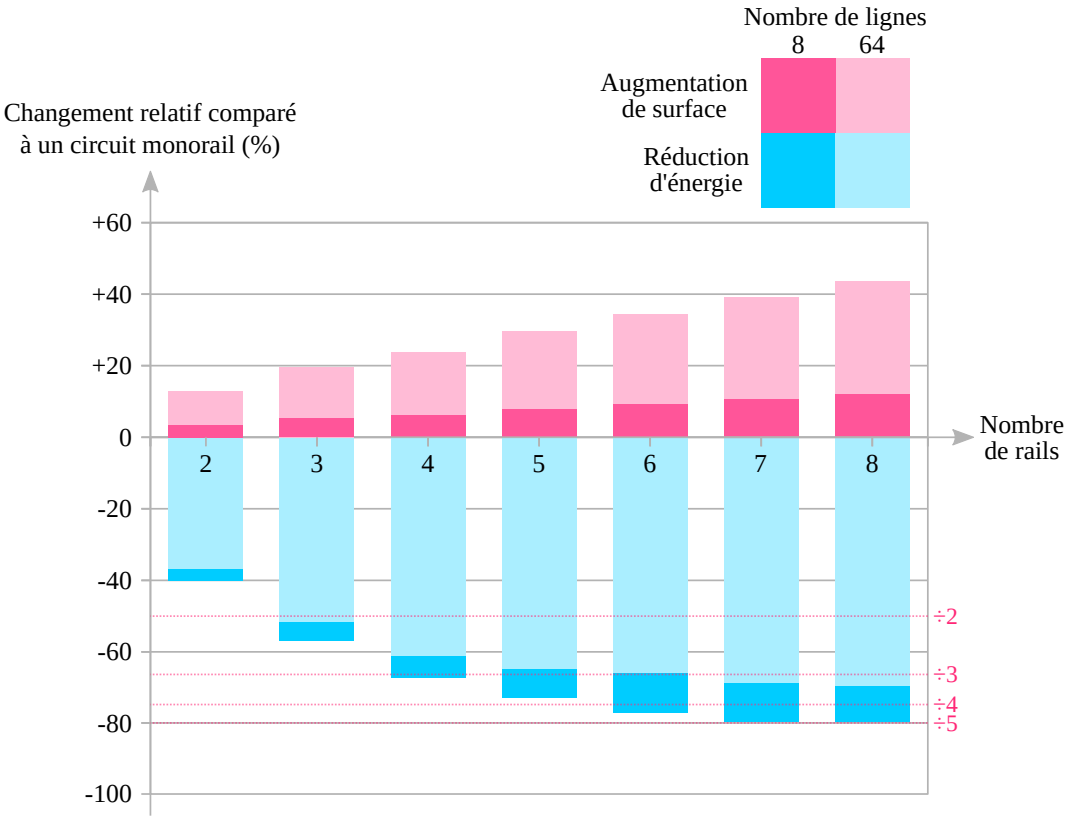

FIGURE 6.3 – *Variation relative de la surface et de la consommation d'énergie par rapport à une architecture à 1 rail avec un KLD=0.012.*

Nous pouvons constater que plus le nombre de rails est important, plus l'énergie est économisée, jusqu'à atteindre un plateau avec 8 rails. De plus, en comparaison relative, une implémentation avec plus de lignes est moins efficace pour l'énergie consommée car la part des ressources partagées dans le circuit est moins importante. Ainsi, avec un KLD de 0,04 et 8 rails, il est possible d'économiser jusqu'à 80% d'énergie (diviser par 5) avec 8 lignes et jusqu'à près de 70% (diviser par 3.3) avec 64 lignes.

D'autre part, la surface du circuit augmente linéairement avec l'augmentation du nombre de rails, avec une pente plus douce pour 8 lignes que pour 64. Cela s'explique aussi par le fait que la part des ressources partagées dans le circuit total est plus élevée pour un plus petit nombre de lignes, et donc dédoubler chaque lignes de la matrice en N_R rails n'a pas le même impact sur la surface totale pour 8 lignes que pour 64. Avec 8 rails, le surcoût total de la surface du circuit est de 12% avec 8 lignes et de 44% avec 64 lignes. Notons que pour le bloc spécifique du coeur stochastique décrit en figure 6.1, ces surcoûts atteignent 85% avec 8 lignes (de 10300 à 19100 µm²) et jusqu'à 100% avec 64 lignes (de 70000 à 140000 µm²).

Enfin, par la définition du nombre de rails à implémenter, l'architecture multirail offre au concepteur la possibilité d'un compromis, de choisir où placer le curseur entre l'efficacité énergétique et la compacité du circuit. Cependant, il semblerait qu'utiliser 8 rails ne soit pas un choix judicieux puisque nous obtenons les mêmes performances énergétiques avec 7 rails (-80%) tout en réduisant le surcoût de surface (+10% au lieu de +12%). Nous n'avons donc pas étudié les solutions au-delà de 8 rails, le plateau étant atteint.

6.4 Conclusion

Nous avons introduit une nouvelle façon de paralléliser les circuits stochastiques afin de réduire la consommation d'énergie : l'architecture stochastique multirail. Le gain en énergie consommée est dû à la réduction de l'activité des ressources partagées n'étant pas concernées par la parallélisation (RNGs, registres de vraisemblances, mémoires ou encore bloc de contrôle). Afin de maximiser ces ressources partagées, cette proposition est basée sur le SRI présenté en section 5.3 et sur les permutations de ses nombres aléatoires de sortie pour alimenter les différents rails à moindre coût matériel.

Une simulation logicielle du circuit a été réalisée afin, d'une part, de déterminer les permutations optimales minimisant leur corrélation, et d'autre part, de mesurer l'impact de ces permutations de colonnes sur la précision globale du circuit. Il en résulte que ces permutations, même optimales, entraînent une perte en précision grandissante à mesure que le nombre de rails augmente. Cependant, cette perte peut être compensée par une plus grande longueur de *bitstream* que celle théorique de $\frac{L_{\text{monorail}}}{N_R}$. Cela entraîne une légère surconsommation en énergie, mais permet une comparaison juste, à précision équivalente avec l'architecture monorail. Des comparaisons en termes de surface et d'énergie ont été réalisées et notre proposition est

capable d'économiser jusqu'à 80% d'énergie du circuit total avec 7 rails, 8 lignes, 8 colonnes et une KLD de 0.012, en contrepartie d'une augmentation de 10% de la surface du circuit total.

Finalement, bien que cette proposition ait été présentée pour une étude de cas de fusion Bayésienne de capteurs, elle pourrait bénéficier à la communauté du calcul stochastique car elle devrait se généraliser à d'autres applications. En effet, la configuration matricielle du calcul n'est pas nécessaire à l'utilisation du calcul multirail, qui pourrait tout à fait s'adapter et garder son efficacité énergétique pour des problèmes à une seule ligne par exemple. Les résultats de l'étude montrent même que l'architecture multirail est particulièrement adaptée pour une configuration avec un faible nombre de lignes.

7

Bilan et comparaisons

Ce chapitre effectue le bilan des contributions de la thèse en les comparant à l'architecture stochastique de fusion Bayésienne de capteurs état de l'art originale, au microcontrôleur Renesas RE01 et à un circuit dédié réalisant le calcul d'inférence avec une multiplication en virgule flottante non standard. Il en résulte que nos contributions permettent de diviser la surface totale du circuit jusqu'à plus de 2 et la consommation énergétique jusqu'à près de 40 par rapport à l'architecture état de l'art. En comparaison au microcontrôleur, elles permettent de diviser la consommation énergétique jusqu'à près de 9000 pour une précision équivalente à une multiplication flottante émulée. Enfin, comparée à un circuit dédié utilisant une multiplication flottante non standard, nos contributions divisent par près de 30 la consommation énergétique.

Sommaire

Afin de donner une vision d'ensemble des contributions de cette thèse, des mesures en termes de surface et d'énergie consommées sont effectuées sur une architecture totalement optimisée intégrant l'ensemble des propositions de ces trois derniers chapitres. On nommera cette architecture `CONTRIB`. Ces résultats sont comparés avec :

— l'architecture état de l'art de la section 2.1 en simulation mais aussi avec les résultats de la caractérisation de la puce ;

— les résultats du microcontrôleur ultra basse consommation Renesas RE01 ;

— une architecture dédiée réalisant le calcul de fusion Bayésienne de capteurs en multiplication flottante avec des précisions non standard.

7.1 Ensemble des contributions

7.1.1 Dimensionnement

L'architecture `CONTRIB` en question intègre :

— la compression mémoire avec séquentialisation de la génération des vraisemblances. Ici, l'algorithme mis en place est celui de la compression par dérivation, mais il a été montré en section 4.2.4 que cela n'avait que peu d'effet sur les mesures de surface et de consommation comparé à la compression par quantification ;

— la mise en place du SRI avec les optimisations associées (plein usage des bits aléatoires, permutations de bits aléatoires et WBG) décrit en figure 5.9, permettant une génération efficace de nombres stochastiques en termes de surface et de puissance ;

— l'implémentation de la solution multirail, avec permutations des sorties du SRI sur les colonnes afin de réduire la consommation énergétique.

Les mesures de surface et de consommation du circuit sont effectuées dans la technologie 65 nm de ST Microelectronics par simulations post-synthèse rétro-annotées.

Les paramètres du circuit sont les suivants :

— Les vraisemblances sont codées sur 8 bits, et les étages du SRI sont codés sur 3 bits grâce à la compression ;

— Le RNG est un Xoroshiro de 16 bits ;

— Les dimensions de la matrice de multiplications sont celles de la puce caractérisée dans le chapitre 3, variant donc dans l'ensemble 2x11, 16x11, 32x5, 64x9 (où le premier nombre est le nombre de lignes, et le deuxième nombre celui des colonnes) ;

— Les compteurs ont des sorties de 16 bits ;

— Le nombre de rails varie de 1 à 8 ;

— La fréquence d'horloge est fixée à 50 MHz ;

— Les conditions de la simulation sont à tension nominale de 1.1V et une température de 25°C.

Les permutations des nombres aléatoires sur les colonnes sont sélectionnées afin de minimiser leur impact sur la précision selon le même protocole que celui de la section 6.2.1.

7.1.2 Étude de précision

Une étude équivalente à celle de la section 6.2 est réalisée pour les différentes dimensions mentionnées. Elle a pour but de déterminer la longueur de *bitstream* nécessaire pour atteindre une valeur de précision de référence en fonction du nombre de rails implémentés et du jeu de données, selon les protocoles décrits en section 3.2.2.

Les précisions de référence sont choisies identiques à celles utilisées pour la comparaison de la puce au microcontrôleur Renesas RE01 dans le tableau 3.4. On distingue donc deux niveaux de précision. Les précisions dites de niveau faible (respectivement élevé) sont les précision de KLD = 2.7e-1 (respectivement 8.3e-4) pour le jeu de données `RAND`, KLD = 2.9e-2 (respectivement 6.1e-5) pour `NORM` et TRM = 85% (respectivement 90%) pour `RMAX`.

Pour rappel, le jeu de données `RMAX` possède un niveau de bruit tel que le calcul d'inférence Bayésienne en multiplication flottante simple précision atteint un TRM de 90% par rapport à la vérité-terrain. Atteindre un TRM de 90% en calcul stochastique correspond donc à atteindre un niveau de précision équivalent au calcul flottant. D'une manière générale, les précisions dites élevées correspondent toutes à des précisions équivalentes au calcul flottant simple précision.

Ces résultats de longueur de *bitstream* nécessaire pour atteindre ces précisions de référence sont compilés dans le tableau 7.1.

On remarque que la solution multirail semble très efficace pour réduire le temps de calcul du jeu de données aléatoire avec les architectures ayant un grand nombre de colonnes. En effet, pour les architectures 2x11, 16x11 et 64x9, la longueur de *bitstream* nécessaire est très proche de la longueur de *bitstream* théorique $L_{th} = \frac{L_{\text{monorail}}}{Nb_{\text{rails}}}$, même pour un grand nombre de rails. Cependant, pour l'architecture 32x5 qui possède peu de colonnes, la réduction du temps de calcul atteint un palier après 5 rails. Ce phénomène est sans doute dû au fait que les possibilités de permutations des nombres aléatoires sur les colonnes sont plus rares quand le nombre de colonnes est moindre. En effet, le nombre de permutations possibles correspond à $N_C! = 120$ pour 5 colonnes, mais vaut 362880 pour 9 colonnes, et 39916800 pour 11 colonnes. Ainsi, la sélection des permutations se faisant dans un champs des possibles plus restreint, ces permutations, même optimales, entraînent de fait une plus grande corrélation entre les différents rails. Une des solutions consisterait alors à mettre en place plus d'étages de SRI que d'opérandes, et d'utiliser une partie des nombres aléatoires pour chaque rail. Cela impliquerait certes une faible augmentation de la surface et puissance consommée, mais permettrait de réduire la corrélation entre les rails, et donc l'énergie consommée.

			Nombre de rails							
			1	2	3	4	5	6	7	8
Jeu de données	Architecture	Niveau de précision	Longueur de *bitstream* nécessaire pour atteindre la précision de référence							
Aléatoire (RAND)	2x11 et 16x11	Faible	4100	2050	1400	1000	840	700	590	540
		Élevé	1.6e7	8.3e6	5.7e6	4.2e6	3.5e6	2.9e6	2.4e6	2.1e6
	32x5	Faible	64	22	18	18	18	18	18	18
		Élevé	2.6e5	1.4e5	1.1e5	8.5e4	7.9e4	7.9e4	7.9e4	7.9e4
	64x9	Faible	1050	580	420	350	250	230	200	170
		Élevé	4.2e6	2.1e6	1.3e6	1.0e6	8.3e5	7.1e5	6.0e5	5.2e5
Normalisé (NORM)	Toutes	Faible	16							
		Élevé	65536							
Recherche du maximum (RMAX)	2x11	Faible	32	16	16	16	16	16	8	8
		Élevé	256	128	64	32	32	32	32	32
	16x11	Faible	32	16	16	16	16	16	16	16
		Élevé	256	256	128	128	128	128	64	32
	32x5	Faible	32	16	16	16	16	16	16	16
		Élevé	128	128	128	64	32	64	64	64
	64x9	Faible	32	16	16	16	16	16	16	16
		Élevé	256	256	128	64	32	32	64	256

TABLEAU 7.1 – *Longueur du bitstream nécessaire pour atteindre la valeur de précision de référence en fonction des jeux de données, des dimensions de la matrice et du nombre de rails.*

Concernant le cas de test normalisé NORM, les longueurs de *bitstream* nécessaires pour atteindre les précisions de référence de KLD respectives $2.9e-2$ et $6.1e-5$ sont donc les mêmes qu'en section 3.2.6, soit respectivement 2^4 et 2^{16} cycles, et ce pour toutes les architectures. Le calcul multirail ne semble donc pas permettre une meilleure précision avec un plus grand nombre de rails. Il ne peut ainsi pas être utilisé pour réduire le temps de calcul et donc la consommation énergétique. La configuration optimale pour ce cas de test correspond alors à une architecture optimisée intégrant la compression mémoire, le SRI et une structure monorail.

Enfin, pour le jeu de données RMAX, la mise en place de plus de rails permet de converger plus rapidement vers la précision maximale de TRM=90%, jusqu'à atteindre un certain palier. Si bien que pour 5 rails, les 90% sont atteints pour 128 cycles de calcul pour l'architecture 16x11, et 32 cycles pour les autres au lieu des 256 cycles nécessaires pour des architectures monorail. La figure 7.1 montre la convergence du taux de reconnaissance du maximum en fonction de la longueur de *bitstream* pour un rail (équivalent aux résultats de la figure 3.6) en comparaison avec 5 rails.

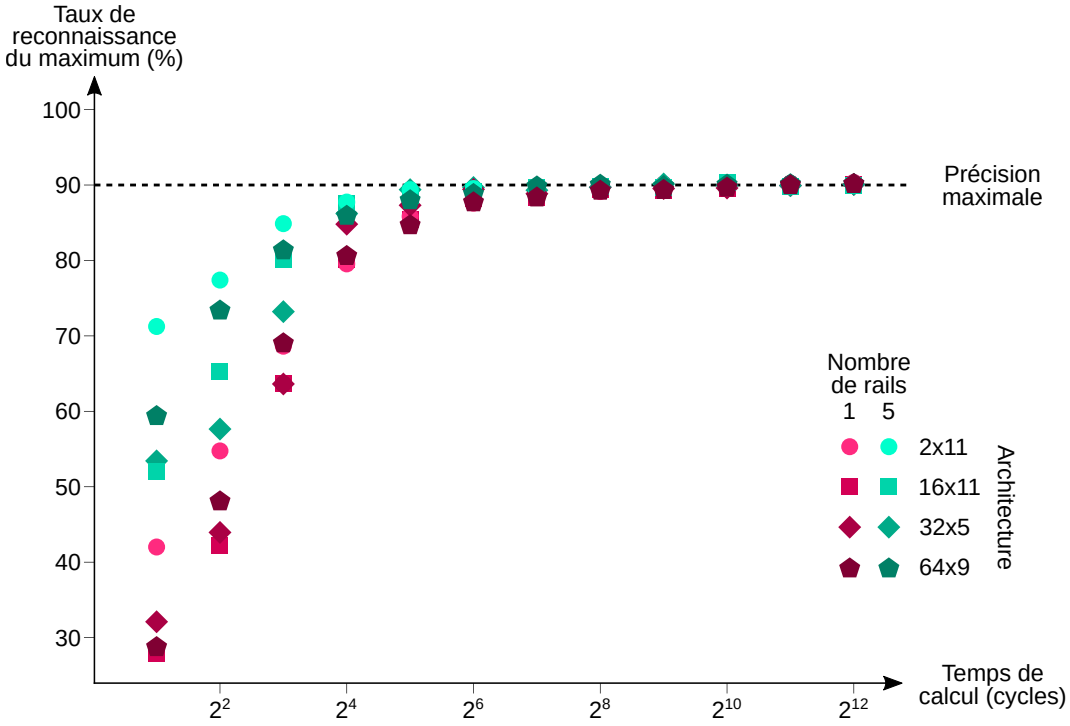

FIGURE 7.1 – *Évolution du taux de reconnaissance du maximum en fonction de la longueur de bitstream pour toutes les architectures et pour 1 et 5 rails.*

7.1.3 Résultats matériels

Le tableau 7.2 rassemble les résultats de surface et de puissance de cette architecture `CONTRIB` en fonction du nombre de rails.

	Nombre de rails							
	1	2	3	4	5	6	7	8
Architecture	Surface (µm²)							
2x11	41410	42390	42800	43200	43430	43940	44110	44190
16x11	67170	75560	79010	82460	83980	87740	89120	90220
32x5	56470	69680	72130	74690	77940	78430	79540	82500
64x9	128700	158100	169800	178800	190500	199500	200900	211400
Architecture	Puissance consommée (µW)							
2x11	194	202	207	217	224	224	237	244
16x11	285	328	380	446	470	531	566	582
32x5	223	282	308	340	375	395	432	480
64x9	631	765	898	1000	1110	1230	1270	1420

TABLEAU 7.2 – *Surface et puissance consommée de l'architecture `CONTRIB` en fonction du nombre de rails.*

Ainsi, en y associant les résultats du tableau 7.1, on en déduit les configurations avec les nombres de rails optimaux minimisant l'énergie consommée en fonction du jeu de données. Ces résultats sont montrés dans le tableau 7.3.

		Jeu de données					
	Précision :	Aléatoire (RAND)		Normalisé (NORM)		Recherche du maximum (RMAX)	
	Niveau	Faible	Élevé	Faible	Élevé	Faible	Élevé
	Métrique	KLD		KLD		TRM	
Architecture	Valeur	2.7e-1	8.3e-4	2.9e-2	6.1e-5	85%	90%
2x11	rails	8	8	1	1	7	4
	Énergie (nJ)	2.64	10300	0.062	254	0.038	0.139
16x11	rails	8	8	1	1	2	8
	Énergie (nJ)	6.3	24500	0.091	373	0.105	0.373
32x5	rails	3	4	1	1	2	5
	Énergie (nJ)	0.111	578	0.071	292	0.09	0.24
64x9	rails	8	8	1	1	2	5
	Énergie (nJ)	4.82	14800	0.2	827	0.245	0.71

TABLEAU 7.3 – *Énergie consommée de l'architecture `CONTRIB` avec le nombre de rails optimal en fonction du jeu de données et des dimensions des matrices.*

7.2 Architecture état de l'art en simulation

Nous comparons les résultats de surface et de consommation énergétique de l'architecture `CONTRIB` à l'architecture stochastique de fusion Bayésienne de capteurs de l'état de l'art 2.1. Cette dernière possède un implémentation parallèle des mémoires, avec une mémoire modèle et une mémoire de distribution Gaussienne par colonne. Son coeur de calcul stochastique intègre un LFSR 32 bits pour chaque colonne, et des comparateurs comme convertisseur binaire / stochastique.

Ces résultats matériels sont recueillis par simulation post-synthèse, dans des conditions nominales de tension (1.1V) et de température (25°C). Nous les comparons dans un second temps avec les résultats de l'implémentation physique de la section 3.3.3, afin d'extrapoler la consommation énergétique d'une implémentation physique de nos contributions.

7.2.1 Puissance et surface consommées

Le tableau 7.4 montre ces résultats de surface et consommation en puissance pour la simulation et la caractérisation physique.

Architecture	Surface (µm²) (réduction de CONTRIB)	Puissance (µW) (réduction de CONTRIB)
2x11	94000 (÷2.3)	1050 (÷5.4)
16x11	161000 (÷2.4)	1440 (÷5.0)
32x5	91900 (÷1.6)	700 (÷3.2)
64x9	215000 (÷1.7)	1690 (÷2.7)

TABLEAU 7.4 – *Résultats de surface et de puissance consommée pour l'architecture stochastique de fusion Bayésienne état de l'art en simulation.*

On constate que l'architecture CONTRIB est jusqu'à 2.4 fois plus efficace en termes de surface pour l'architecture 16x11 avec 1 rail, principalement dû à la compression mémoire. Notons que par soucis de justesse de comparaison, le cas de l'implémentation de Gaussienne unique n'est pas comparé ici, car il n'est utilisable que lorsque les vraisemblances des modèles capteurs ont un bruit du même écart-type. Cependant, il pourrait montrer des gains de surface encore plus importants. Les contributions consomment également jusqu'à 5.4 fois moins de puissance pour cette même configuration de matrice, en grande partie grâce à la mise en place du SRI et des optimisations de SNG associées.

7.2.2 Comparaison en énergie

En reprenant les longueurs de *bitstream* associées aux différentes précisions pour les jeux de données RAND, NORM et RMAX de l'architecture état de l'art, en section 3.2.6, on en déduit l'énergie de tels circuits pour les précisions de référence. La comparaison à CONTRIB est montrée dans le tableau 7.5.

	Jeu de données					
Précision :	Aléatoire (RAND)		Normalisé ((NORM)		Recherche du maximum (RMAX)	
Niveau	Faible	Élevé	Faible	Élevé	Faible	Élevé
Métrique	KLD		KLD		TRM	
Valeur	2.70E-01	8.30E-04	2.90E-02	6.10E-05	85%	90%
Architecture	Énergie consommée (nJ) (réduction de CONTRIB)					
2x11	85.7 (÷33)	351000 (÷34)	0.335 (÷5)	1370 (÷5)	0.67 (÷18)	5.36 (÷39)
16x11	118 (÷19)	482000 (÷20)	0.46 (÷5)	1880 (÷5)	0.92 (÷9)	7.36 (÷20)
32x5	0.899 (÷8)	3680 (÷6)	0.225 (÷3)	921 (÷3)	0.45 (÷5)	3.6 (÷15)
64x9	34.7 (÷7)	142000 (÷10)	0.542 (÷3)	2220 (÷3)	1.08 (÷4)	8.67 (÷12)

TABLEAU 7.5 – *Résultats de l'énergie consommée pour l'architecture stochastique de fusion Bayésienne de capteurs état de l'art en simulation et comparaison avec les contributions de la thèse.*

7.3 Circuit physique et extrapolation

Selon le tableau 3.1, la consommation de puissance du circuit physique pour l'ensemble de la génération des vraisemblances et du calcul stochastique d'inférence est, sous tension nominale de 1.1V, de 1.11 mW pour l'architecture 2x11, 1.75 mW pour 16x11, 1.01 mW pour 32x5 et 2.19 mW pour 64x9. On remarque une augmentation d'environ 30% entre la puissance électrique simulée et celle du circuit caractérisé. Cela peut être en partie expliqué par le fait que, pour la caractérisation, toutes les architectures sont présentes sur la même puce. Et bien que les architectures qui ne sont pas testées ne calculent rien, elles ajoutent tout de même une consommation statique au circuit testé.

Notons que les simulations sont effectuées sous les mêmes conditions de tension (1.1V) et de température (25°C) que la première partie de la caractérisation.

Ainsi, en considérant que le rapport de proportionnalité entre l'architecture état de l'art et CONTRIB reste équivalent, on peut extrapoler la consommation de puissance que donnerait une puce caractérisée de nos contributions à la tension 0.71V (deuxième caractérisation). Ces données extrapolées sont utilisées par la suite pour comparer les résultats en énergie de nos contributions avec le microcontrôleur.

7.4 Comparaison avec le Renesas RE01

La consommation énergétique de l'architecture `CONTRIB` extrapolée, sous une tension d'alimentation de 0.71V, ainsi que sa comparaison avec le microcontrôleur Renesas RE01 sont montrées dans le tableau 7.6.

	Jeu de données					
Précision :	Aléatoire (`RAND`)		Normalisé (`NORM`)		Recherche du maximum (`RMAX`)	
Niveau	Faible	Élevé	Faible	Élevé	Faible	Élevé
Métrique	KLD		KLD		TRM	
Valeur	2.7e-1	8.3e-4	2.9e-2	6.1e-5	85%	90%
Architecture	Énergie consommée (nJ) (changement relatif au RE01)					
2x11	0.983	3820	0.023	94.5	0.014	0.052
	÷133	×29	÷5700	÷1.39	÷9300	÷2540
16x11	2.76	10700	0.04	164	0.046	0.163
	÷350	×11.1	÷24100	÷5.89	÷20900	÷5900
32x5	0.089	389	0.037	150	0.046	0.123
	÷8950	÷2.04	÷21700	÷5.29	÷17100	÷6450
64x9	2.42	7410	0.101	415	0.123	0.356
	÷1310	×2.34	÷31300	÷7.63	÷25800	÷8900

TABLEAU 7.6 – *Consommation énergétique des contributions extrapolées à la tension d'alimentation de 0.71V et comparées avec la consommation du microcontrôleur RE01.*

On remarque qu'ici, seuls 3 cas sont défavorables à notre architecture dédiée, ce qui est à comparer aux 5 du tableau 3.4, et avec un facteur ×29 pour le pire cas. De plus, notre architecture est capable de diviser jusqu'à un facteur 30000 la consommation énergétique comparée au microcontrôleur pour une faible précision de calcul, et jusqu'à presque 9000 pour une précision équivalente au calcul flottant.

7.5 Multiplication en virgule flottante non standard

Une partie des gains de la comparaison avec le Renesas est due au fait que le circuit est dédié au calcul d'inférence Bayésienne, et donc optimisé pour cela, alors que le microcontrôleur a pour vocation d'être programmable.

Pour mesurer le gain de notre contribution sur le calcul flottant, nous implémentons le même circuit dédié, mais en remplaçant la multiplication stochastique par une multiplication flottante non standard.

Les nombres en virgule flottante sont représenté sur 16 bits, avec 8 bits de mantisse (*man*) et 8 bits d'exposant (*exp*). Le bit de signe n'est pas nécessaire puisque nous manipulons des valeurs de probabilités. De même, les nombres représentés variant dans le segment $[0,\ 1]$, l'exposant est toujours négatif ou nul. De plus, si on interprète la mantisse comme un nombre entier variant entre 0 et 255, l'exposant doit être biaisé de -8. Le nombre flottant représenté vaut donc : $x = man \times 2^{e-8}$.

Réaliser une multiplication flottante entre deux nombres revient alors à :

— multiplier en virgule fixe leurs mantisses de 8 bits, donnant un nombre sur 16 bits,

— additionner leurs exposants de 8 bits,

— décaler la mantisse résultante vers la gauche tant que son bit de poids fort est un 0, et décrémenter l'exposant résultant d'autant,

— garder les 8 bits de poids forts de la mantisse résultat.

Réaliser une multiplication flottante de cette manière nécessite 2 cycles de calcul à 50 MHz dans la technologie 65 nm de ST Microelectronics.

Une seule multiplication flottante est mise en place pour toute la matrice, qui réalise le calcul d'inférence de manière séquentielle.

Les résultats de puissance et la comparaison avec nos contributions en termes d'énergie pour le cas de recherche du maximum sont montrés dans le tableau 7.7.

Architecture	Puissance consommée (µW)	Énergie (nJ) (réduction de `CONTRIB`)	
		Faible précision	Haute précision
2x11	85.5	0.075 (÷5.3)	(÷1.5)
16x11	153	1.08 (÷23)	(÷6.6)
32x5	237	1.52 (÷33)	(÷12)
64x9	444	10.2 (÷83)	(÷29)

TABLEAU 7.7 – *Consommation d'un circuit dédié au calcul d'inférence en multiplication flottante non standard et comparaison avec l'architecture `CONTRIB` pour le cas de la recherche du maximum de distribution. Faible précision : taux de reconnaissance de 85%, haute précision : taux de reconnaissance maximal de 90%.*

On observe que notre circuit permet jusqu'à une économie d'énergie d'un facteur 83 par rapport au calcul flottant pour une faible précision, et jusqu'à 29 pour une précision équivalente. D'une manière générale, notre architecture stochastique devient plus efficace à mesure que les dimensions de la matrice augmentent, en particulier le nombre de lignes.

7.6 Conclusion

Dans ce chapitre, nous avons dressé le bilan chiffré des contributions de la thèse en les comparant à l'architecture état de l'art de fusion Bayésienne de capteurs, au microcontrôleur Renesas RE01, et une multiplication flottante non standard. Il en résulte que, dans les meilleurs cas, nos contributions permettent de diviser par plus de 2 la surface et par près de 40 la consommation énergétique par rapport à l'architecture état de l'art. En comparaison au microcontrôleur, elles permettent de diviser la consommation énergétique jusqu'à près de 9000 fois pour une précision équivalente à une multiplication flottante émulée. Enfin, comparées à un circuit dédié utilisant une multiplication flottante non standard, nos contributions divisent jusqu'à près de 30 fois la consommation énergétique pour le jeu de données de la recherche du maximum.

Conclusion

À l'heure de l'internet des objets (IoT), rapprocher l'intelligence de calcul au plus près des capteurs est devenu un enjeu crucial afin de réduire les communications non nécessaires avec le *cloud*, et ainsi économiser de l'énergie : c'est le but de la fusion de capteurs. Dans ce cadre, l'implémentation de circuits stochastiques dédiés à la résolution de fusion Bayésienne de capteurs permet d'une part d'obtenir des résultats de manière explicable, avec peu de données et en tenant compte de leur incertitude, de par son approche Bayésienne. D'autre part, du fait de la représentation stochastique du calcul, de tels circuits possèdent une grande efficacité en termes de surface et de puissance.

Cependant, ces architectures montrent plusieurs limites lorsque la taille du problème augmente. Premièrement, les distributions de probabilité nécessaires au calcul des vraisemblances requièrent une grande quantité de mémoire, qui peut représenter une partie importante de la surface totale du circuit, et donc de sa consommation statique. De plus, le coût de la génération de nombres stochastiques en surface et en puissance représente un frein pour adresser des problèmes de plus grande dimension. Enfin, le temps de calcul nécessaire pour atteindre une précision donnée peut s'avérer élevé. Ce phénomène inhérent à l'arithmétique stochastique induit un coût énergétique élevé, parfois supérieur au calcul flottant, et limite l'intérêt de ces circuits aux problèmes qui ne requièrent que de faibles précisions.

Contributions et bilan

Après une étude bibliographique décrite dans les chapitres 1 et 2, puis la caractérisation d'un prototype stochastique de fusion Bayésienne de capteurs dans le chapitre 3 qui confirme les limites de telles architectures, cette thèse adresse ces trois goulots d'étranglement dans les chapitres respectifs 4, 5 et 6, alors que le chapitre 7 dresse le bilan chiffré des différentes contributions de la thèse. Nous développons ici plus en détail les conclusions de chaque contribution.

Dans le chapitre 3, la caractérisation du circuit stochastique de fusion Bayésienne de capteurs confirme les avantages du calcul stochastique lorsque la précision demandée est faible. Dans les cas les plus favorables en dimensions et jeux de données, il présente des gains en énergie allant jusqu'à un facteur 6000 pour une faible

précision et près d'un facteur 400 pour une précision équivalente comparée à la multiplication flottante émulée par le microcontrôleur Renesas RE01. Cependant, pour le cas le plus défavorable avec des données aléatoires et à précision équivalente, il consomme jusqu'à 1000 fois plus que le microcontrôleur. Cette étude met alors en exergue la grande consommation énergétique des circuits stochastiques due à un temps de calcul qui peut être long pour une grande précision requise. De plus, elle souligne la part conséquente de la surface des mémoires dans la surface totale du circuit et l'impact de la génération des nombres stochastiques sur sa consommation globale. Cela motive grandement les travaux des chapitres 4, 5 et 6.

Dans le chapitre 4, nous proposons différentes manières de réduire l'impact des mémoires de distributions Gaussiennes sur la surface totale du circuit. Premièrement, en séquentialisant la génération des vraisemblances, il est possible de regrouper toutes les mémoires de distributions Gaussiennes (respectivement les mémoires modèles) de chaque colonne en une seule grosse mémoire. Cela permet de s'affranchir de l'*offset* dû à la logique d'adressage, conduisant à une économie de 40% de la surface totale du circuit pour une matrice de 32 lignes et 9 colonnes. De plus, lorsque les capteurs sont de même nature et qu'ils possèdent un bruit du même écart-type, il est possible de n'utiliser qu'une seule mémoire Gaussienne pour l'ensemble du circuit. Cela divise par 2.3 la surface totale du circuit pour cette même matrice de 32 lignes et 9 colonnes. Enfin, la contribution originale de ce chapitre consiste en 2 algorithmes de compression mémoire, la compression par dérivation et la compression par quantification, tout deux capables de diviser la profondeur de mémoire Gaussienne requise par 4. Ces méthodes permettent une réduction de la surface totale du circuit de 36% tout en gardant une consommation énergétique équivalente voire légèrement inférieure à l'implémentation séquentialisée. En ce qui concerne l'architecture originale, sa surface peut être divisée par 2.

Dans le chapitre 5, nous développons différentes méthodes pour réduire la consommation électrique liée à la génération de nombres stochastiques. Premièrement, nous adaptons diverses méthodes de partage de ressources de l'état de l'art stochastique à l'architecture de fusion Bayésienne de capteurs, comme le plein usage des bits aléatoires, les permutations inverses ou la factorisation du WBG. Ces optimisations combinées permettent de réduire la surface du coeur de calcul stochastique de 21% et de diviser la puissance par 3.4 pour une matrice de 64 lignes et 8 colonnes, et cela sans perte en précision notable. Cette architecture optimisée à l'état de l'art nous sert de référence de comparaison pour évaluer les contributions originales de nos travaux. De plus, une étude comparative de différents RNGs a permis de mettre en lumière la qualité d'aléa suffisante pour le calcul stochastique. Ainsi, pour des circuits standards n'utilisant pas l'isolation stochastique, de simples LFSRs se comportent comme des RNGs idéaux et atteignent la même précision que des RNGs plus sophistiqués. Cependant, ceux-ci ne sont plus compatibles avec l'isolation stochastique car elle requiert des RNGs avec une faible autocorrélation comme le STRNG, le Xoroshiro ou le SBoNG. Nous introduisons enfin le SRI, une méthode d'isolation stochastique à base de registres à décalage, isolant non pas les *bitstreams* dans le temps, mais les nombres aléatoires eux-mêmes. Cette nouvelle méthode permet ainsi d'économiser jusqu'à près de la moitié de la puissance du coeur de calcul par rapport à une architecture optimisée, tout en gardant une surface équivalente pour

une matrice de 8 lignes et 16 colonnes. Comparée à l'architecture originale sans optimisation, la puissance est divisée par près de 8 et la surface par près de 2.

Dans le chapitre 6, nous introduisons une méthode de parallélisation du calcul stochastique rendant possible une réduction de la latence de calcul et donc de l'énergie consommée par les ressources partagées : le calcul multirail. Il consiste à démultiplier chaque ligne de la matrice en différents rails, translatant une partie de la dimension temporelle des *bitstreams* dans la dimension spatiale. Il s'appuie sur l'implémentation du SRI ainsi que sur la permutation des nombres aléatoires sur les colonnes afin de générer à moindre coût ceux requis pour les nouveaux rails. Les permutations induisent une perte en précision qui est étudiée, minimisée par le choix de la permutation, puis compensée par une faible augmentation de la longueur de *bitstream*. Cette méthode permet de diviser jusqu'à 5 fois l'énergie totale du circuit avec 7 rails, 8 lignes et 8 colonnes, pour une augmentation de la surface de seulement 10%.

Le chapitre 7 effectue le bilan chiffré des contributions de la thèse, et une comparaison avec l'architecture état de l'art, le microcontrôleur Renesas RE01, et une multiplication flottante non standard. Il en résulte que nos contributions permettent de diviser la surface totale du circuit par un facteur allant jusqu'à plus de 2 et la consommation énergétique par un facteur allant jusqu'à près de 40 par rapport à l'architecture originale. En comparaison au microcontrôleur, elles permettent de diviser la consommation énergétique jusqu'à près de 9000 fois pour une précision équivalente à une multiplication flottante émulée. Enfin, comparées à un circuit dédié utilisant une multiplication flottante non standard, nos contributions divisent jusqu'à près de 30 fois la consommation énergétique.

Discussion

Dans ces travaux de thèse, nous nous sommes principalement intéressés à des problèmes simples de fusion de capteurs par inférence Bayésienne exacte. Cela permet d'obtenir les distributions les plus justes au vu des données observées. Mais lorsque le cardinal de la variable d'intérêt augmente, le calcul d'inférence exacte devient rapidement intractable. Dans ce cas, des méthodes par échantillonnage de type MCMC (Markov Chain Monte Carlo) sont préférables.

De plus, dans ces travaux, de nombreux paramètres de dimensionnement ont été utilisés que ce soit pour le codage des données avec la résolution des probabilités $\mathrm{res_p}$, la résolution des capteurs $\mathrm{res_o}$ ou la taille des compteurs ; ou pour des dimensions de l'architectures comme le nombre de lignes N_L, le nombre de colonnes N_C ou bien le nombre de rails N_R. Cela rend les analyses difficiles et le choix des paramètres optimaux peu aisé. Même si nous pensons que nos contributions peuvent s'adapter à d'autres dimensions, une étude plus approfondie en ce sens serait intéressante afin d'observer dans quelles mesures ces propositions sont efficaces.

Enfin, bien que nos contributions apportent des gains significatifs par rapport à l'architecture originale et à une multiplication flottante émulée sur le microcontrô-

leur Renesas RE01, en comparaison à un circuit dédié intégrant une multiplication flottante, les résultats peuvent être plus mitigés. Les méthodes de cette thèse ne sont en effet pas compétitives avec le calcul en virgule flottante dans les très hautes précisions. Cependant, dans certains cas d'usage pratiques comme la recherche du maximum de distribution pour la fusion de capteurs bruités, la très haute précision n'est souvent pas nécessaire, et nos architectures montrent un gain significatif en énergie.

Perspectives de recherche

Dans le chapitre 3, nous nous sommes principalement intéressés à des méthodes permettant de restituer de manière exacte les valeurs des vraisemblances, soit par réorganisation des mémoires, soit par compression sans perte. Cependant, cela semble en désaccord avec le principe même de la logique stochastique qui est d'effectuer des calculs de manière approchée afin de réduire les coûts d'implémentation et de consommation. Ainsi il serait intéressant de mettre en place une méthode de compression moins précise mais plus simple comme par exemple en découpant la courbe Gaussienne en segments et en ne stockant que leurs pentes. Il faudrait alors étudier la précision des distributions en sortie de ces circuits afin de s'assurer qu'elle reste équivalente à celle utilisant une compression sans perte.

De plus, il serait judicieux d'étudier plus en profondeur l'impact de la méthode de *slicing* de [32] sur la précision, en déterminant le nombre optimal de colonnes en fonction du nombre de capteurs à fusionner et de la précision demandée. Cette étude permettrait de mettre en évidence le gain en consommation énergétique que peut promettre le *slicing*. Il serait d'autant plus intéressant de mesurer les gains de cette méthode combinée avec notre proposition de calcul multirail qui semble plus performante avec un plus grand nombre de colonnes.

Dans ces travaux, nous nous sommes efforcé de réduire la consommation énergétique des circuits stochastiques. Cependant, le plus gros avantage du calcul stochastique réside peut-être dans sa très faible consommation de puissance instantanée. Il pourrait alors tout à fait trouver sa place pour des applications d'*energy harvesting* disposant d'un apport en énergie continu mais avec une très faible puissance.

Aussi, la nature robuste du circuit stochastique aux fautes transitoires peut être intéressante à caractériser et à exploiter. La conception et la fabrication d'une puce dont la partie stochastique possède son propre rail d'alimentation permettrait de rendre compte de son intérêt. On pourrait alors réduire la tension du coeur de calcul stochastique jusqu'à obtenir des inversions de bits. Si elles sont peu fréquentes, ces inversions de bits ne risquent pas d'endommager grandement la précision du calcul, et on pourrait alors utiliser le circuit stochastique à des tensions inaccessibles pour un circuit en représentation standard.

Enfin, de nombreux travaux utilisent des composants de technologies émergentes comme les RRAMs [84] ou bien les jonctions magnétiques à effet tunnel

(MTJs) [82] pour générer efficacement des *bitstreams*. Combiner nos contributions à ces composants prometteurs pourrait permettre des gains en énergie encore plus intéressants, en particulier avec le calcul multirail.

D'une manière générale, les résultats de ces travaux laissent à penser que nos contributions peuvent profiter pleinement à la communauté scientifique, notamment pour les propositions du SRI et du calcul multirail qui peuvent s'adapter parfaitement à tout type d'architecture de calcul stochastique, et pas nécessairement celles avec une structure matricielle, promettant de forts gains en temps de calcul et en énergie consommée.

L'intelligence artificielle nourrit bien des fantasmes, et elle est souvent représentée dans la science-fiction comme une intelligence centralisée, semblable au cerveau humain mais plus performante en tout point. Et si l'IA de demain était plurielle, dispersée, continue ? Si au lieu de questionner un expert surentraîné nous faisions confiance à la *sagesse de la foule* [115] ? Nous pourrions espérer de cette IA qu'elle soit plus informée, ouverte et flexible face aux incertitudes. En réduisant la consommation énergétique et les coûts de productions, déverrouillant deux des principaux freins à son développement, nous espérons avoir contribué à notre échelle à l'avènement de cette IA plurielle.

Publications

Ces travaux de thèse ont donné lieu à deux publications scientifiques, l'une dans un journal et l'autre dans une conférence à comité de lecture :

- Belot, J., Cherkaoui, A., Laurent, R., & Fesquet, L. (2021). An Area-and Power-Efficient Stochastic Number Generator for Bayesian Sensor Fusion Circuits. IEEE Design & Test, 38(6), 69-77.
Cet article de journal présente la méthode du SRI ainsi que l'étude comparative de différents RNGs pour l'isolation stochastique développés dans le chapitre 5.

- Belot, J., Cherkaoui, A., Laurent, R., & Fesquet, L. (2021, November). An Energy Efficient Multi-Rail Architecture for Stochastic Computing : A Bayesian Sensor Fusion Case Study. In 2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS) (pp. 1-5). IEEE.
Cet article présente le calcul multirail, une méthode de parallélisation du calcul stochastique utilisant le SRI ainsi que la permutation de ses sorties, développée dans le chapitre 6.

Nous prévoyons également de soumettre prochainement deux autres publications :

- Belot, J., Cherkaoui, A., Laurent, R., Simatic, J., & Fesquet, L. (2022). Characterization of a Prototype Dedicated to Stochastic Bayesian Sensor Fusion.
Cet article prévu pour un journal présentera la caractérisation du prototype d'architecture stochastique de fusion Bayésienne de capteurs et sa comparaison avec le microcontrôleur Renesas RE01 développées dans le chapitre 3.

- Belot, J., Cherkaoui, A., Laurent, R., & Fesquet, L. (2022). Gaussian Memory Compression Methods for Bayesian Sensor Fusion Circuits.
Cet article présentera les méthodes de compression de mémoires Gaussiennes développées dans le chapitre 4.

Une communication interne au laboratoire TIMA a aussi été présentée :

- Belot, J. (2021). Stochastic architectures for Bayesian sensor fusion. TIMA Scientific Days.
Cette présentation introduisait l'architecture stochastique originale pour la fusion Bayésienne de capteurs décrite en section 2.1.

Bibliographie

[1] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[2] M. Newborn, *Kasparov versus Deep Blue : Computer chess comes of age*. Springer Science & Business Media, 2012.

[3] "A boy and his atom : The world's smallest movie." https://www.youtube.com/watch?v=oSCX78-8-q0, 2013. Accédé le : 10-07-2022.

[4] "IBM unveils world's first 2 nanometer chip technology, opening a new frontier for semiconductors." https://newsroom.ibm.com/2021-05-06-IBM-Unveils-Worlds-First-2-Nanometer-Chip-Technology,-Opening-a-New-Frontier-for-Semiconductors, 2021. Accédé le : 10-07-2022.

[5] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, *et al.*, "Building watson : An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv :2107.03374*, 2021.

[9] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*, pp. 8821–8831, PMLR, 2021.

[10] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv :2204.06125*, 2022.

[11] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, *Deep learning techniques for music generation*, vol. 1. Springer, 2020.

[12] "Better language models and their implications." https://openai.com/blog/better-language-models, 02 2019. Accédé le : 10-07-2022.

[13] A. M. Turing, "Computing machinery and intelligence," in *Parsing the turing test*, pp. 23–65, Springer, 2009.

[14] R. A. Brooks, "Elephants don't play chess," *Robotics and autonomous systems*, vol. 6, no. 1-2, pp. 3–15, 1990.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] R. K. Mobley, *An introduction to predictive maintenance*. Elsevier, 2002.

[18] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "Xai - explainable artificial intelligence," *Science Robotics*, vol. 4, no. 37, p. eaay7120, 2019.

[19] E. T. Jaynes, *Probability theory : The logic of science*. Cambridge university press, 2003.

[20] P. Bessiere, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha, *Bayesian programming*. CRC press, 2013.

[21] "Evolution of AI hardware." https://hawai.tech/ia-waves-and-evolution-of-hardware, 2022. Accédé le : 10-07-2022.

[22] D. Kirk, "Nvidia Cuda software and GPU parallel computing architecture," in *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, (New York, NY, USA), p. 103–104, Association for Computing Machinery, 2007.

[23] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the graphcore IPU architecture via microbenchmarking," *arXiv preprint arXiv :1912.03413*, 2019.

[24] "Vision de l'entreprise HawAI.tech." https://hawai.tech/vision. Accédé le : 10-07-2022.

[25] S. Li, L. D. Xu, and S. Zhao, "The Internet of Things : a survey," *Information systems frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[26] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing : A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[27] "Bottom-up Approaches to Machines dedicated to Bayesian Inference." https://cordis.europa.eu/project/id/618024. Accédé le : 10-07-2022.

[28] "Probabilistic machines for low-level sensor interpretation." https://persyval-lab.org/en/sites/content/microbayes. Accédé le : 10-07-2022.

[29] M. Faix, *Conception de machines probabilistes dédiées aux inférences bayésiennes*. PhD thesis, Université Grenoble Alpes, 2016. Thèse de doctorat

dirigée par Mazer, Emmanuel et Fesquet, Laurent Mathématiques et Informatique Université Grenoble Alpes (ComUE) 2016.

[30] R. Frisch, *Machines stochastiques dédiées à l'inférence Bayésienne pour la localisation et séparation de sources*. PhD thesis, Université Grenoble Alpes, 2019. Thèse de doctorat dirigée par Fesquet, Laurent et Mazer, Emmanuel Informatique Université Grenoble Alpes (ComUE) 2019.

[31] A. Coninx, P. Bessière, E. Mazer, J. Droulez, R. Laurent, M. A. Aslam, and J. Lobo, "Bayesian sensor fusion with fast and low power stochastic circuits," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, IEEE, 2016.

[32] R. Frisch, R. Laurent, M. Faix, L. Girin, L. Fesquet, A. Lux, J. Droulez, P. Bessière, and E. Mazer, "A Bayesian stochastic machine for sound source localization," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, IEEE, 2017.

[33] J. Rayleigh, "On our perception of sound direction," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 13, no. 74, pp. 214–232, 1907.

[34] R. Krishnamurthi, A. Kumar, D. Gopinathan, A. Nayyar, and B. Qureshi, "An overview of IoT sensor data processing, fusion, and analysis techniques," *Sensors*, vol. 20, no. 21, p. 6076, 2020.

[35] T. Adali, Y. Levin-Schwartz, and V. D. Calhoun, "Multimodal data fusion using source separation : Application to medical imaging," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1494–1506, 2015.

[36] H. Durrant-Whyte and T. C. Henderson, "Multisensor data fusion," *Springer handbook of robotics*, pp. 867–896, 2016.

[37] A. Daniel, K. Subburathinam, B. Anand Muthu, N. Rajkumar, S. Kadry, R. Kumar Mahendran, and S. Pandian, "Procuring cooperative intelligence in autonomous vehicles for object detection through data fusion approach," *IET Intelligent Transport Systems*, vol. 14, no. 11, pp. 1410–1417, 2020.

[38] Y.-L. Hsu, P.-H. Chou, H.-C. Chang, S.-L. Lin, S.-C. Yang, H.-Y. Su, C.-C. Chang, Y.-S. Cheng, and Y.-C. Kuo, "Design and implementation of a smart home system using multisensor data fusion technology," *Sensors*, vol. 17, no. 7, p. 1631, 2017.

[39] B. P. L. Lau, S. H. Marakkalage, Y. Zhou, N. U. Hassan, C. Yuen, M. Zhang, and U.-X. Tan, "A survey of data fusion in smart city applications," *Information Fusion*, vol. 52, pp. 357–374, 2019.

[40] F. Castanedo, "A review of data fusion techniques," *The scientific world journal*, vol. 2013, 2013.

[41] J. Sasiadek, "Sensor fusion," *Annual Reviews in Control*, vol. 26, no. 2, pp. 203–228, 2002.

[42] D. P. Bertsekas, "Incremental least squares methods and the extended Kalman filter," *SIAM Journal on Optimization*, vol. 6, no. 3, pp. 807–822, 1996.

[43] M. H. DeGroot, “A conversation with David Blackwell,” *Stat. Sci.*, vol. 1, pp. 40–53, 1986.

[44] D. Heckerman, “Bayesian networks for data mining,” *Data mining and knowledge discovery*, vol. 1, no. 1, pp. 79–119, 1997.

[45] B. Cai, L. Huang, and M. Xie, “Bayesian networks in fault diagnosis,” *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2227–2240, 2017.

[46] Y. Zhang and J. Koren, “Efficient Bayesian hierarchical user modeling for recommendation system,” in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 47–54, 2007.

[47] R. M. Neal, *Probabilistic inference using Markov Chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada, 1993.

[48] V. Neumann, “Probabilistic logics and the synthesis of reliable organisms from unreliable components,” in *Automata Studies*, (Princeton Press), pp. 46–98, 1956.

[49] B. R. Gaines, “Stochastic computing,” in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS ’67 (Spring), p. 149–156, Association for Computing Machinery, 1967.

[50] C. Ma, S. Zhong, and H. Dang, “High fault tolerant image processing system based on stochastic computing,” in *2012 International Conference on Computer Science and Service System*, pp. 1587–1590, 2012.

[51] A. Alaghi, C. Li, and J. P. Hayes, “Stochastic circuits for real-time image-processing applications,” in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2013.

[52] Y.-N. Chang and K. K. Parhi, “Architectures for digital filters using stochastic computing,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2697–2701, 2013.

[53] J. Chen and J. Hu, “A novel FIR filter based on stochastic logic,” in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2050–2053, 2013.

[54] B. Brown and H. Card, “Stochastic neural computation. i. computational elements,” *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[55] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, “A survey of stochastic computing neural networks for machine learning applications,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 2809–2824, 2021.

[56] J. S. Friedman, L. E. Calvet, P. Bessière, J. Droulez, and D. Querlioz, “Bayesian inference with Muller C-elements,” *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 63, no. 6, pp. 895–904, 2016.

[57] A. Coelho, R. Laurent, M. Solinas, J. Fraire, E. Mazer, N.-E. Zergainoh, S. Karaoui, and R. Velazco, “On the robustness of stochastic Bayesian machines,” *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2276–2283, 2017.

[58] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, 2016.

[59] S. Golomb and L. Welch, *Shift Register Sequences*. Holden-Day series in information systems, Holden-Day, 1967.

[60] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 39–46, 2013.

[61] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1550–1555, 2016.

[62] T. Chen and J. P. Hayes, "Analyzing and controlling accuracy in stochastic circuits," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pp. 367–373, 2014.

[63] R. K. Budhwani, R. Ragavan, and O. Sentieys, "Taking advantage of correlation in stochastic computing," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[64] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pp. 361–366, 2014.

[65] L. Bagheriye and J. K. Kwisthout, "Brain-inspired hardware solutions for inference in Bayesian networks," *Frontiers in Neuroscience*, p. 1462, 2021.

[66] Z. Kulesza and W. Tylman, "Implementation of bayesian network in fpga circuit," in *Proceedings of the International Conference Mixed Design of Integrated Circuits and System, 2006. MIXDES 2006.*, pp. 711–715, 2006.

[67] M. Lin, I. Lebedev, and J. Wawrzynek, "High-throughput bayesian computing machine with reconfigurable hardware," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, (New York, NY, USA), p. 73–82, Association for Computing Machinery, 2010.

[68] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn : Hardware acceleration of Bayesian neural networks," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 476–488, 2018.

[69] B. Vigoda, "Analog logic : Continuous-time analog circuits for statistical signal processing," *Online] Sep*, 2003.

[70] P. Mroszczyk and P. Dudek, "The accuracy and scalability of continuous-time bayesian inference in analogue cmos circuits," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1576–1579, IEEE, 2014.

[71] S. E. Borujeni, S. Nannapaneni, N. H. Nguyen, E. C. Behrman, and J. E. Steck, "Quantum circuit representation of Bayesian networks," *Expert Systems with Applications*, vol. 176, p. 114768, 2021.

[72] E. M. Jonas, *Stochastic architectures for probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2014.

[73] M. Faix, E. Mazer, R. Laurent, M. O. Abdallah, R. Le Hy, and J. Lobo, "Cognitive computation : a Bayesian machine case study," in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pp. 67–75, IEEE, 2015.

[74] D. H. K. Hoe and C. Pajardo, "Implementing stochastic Bayesian inference : Design of the stochastic number generators," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1105–1109, 2019.

[75] R. Faria, J. Kaiser, K. Y. Camsari, and S. Datta, "Hardware design for autonomous Bayesian networks," *Frontiers in Computational Neuroscience*, vol. 15, 2021.

[76] K. Y. Camsari, B. M. Sutton, and S. Datta, "P-bits for probabilistic spin logic," *Applied Physics Reviews*, vol. 6, no. 1, p. 011305, 2019.

[77] B. Schumacher, "Quantum coding," *Physical Review A*, vol. 51, no. 4, p. 2738, 1995.

[78] S. Bravyi, D. P. Divincenzo, R. I. Oliveira, and B. M. Terhal, "The complexity of stoquastic local hamiltonian problems," *arXiv preprint quant-ph/0606140*, 2006.

[79] K. Y. Camsari, S. Chowdhury, and S. Datta, "Scaled quantum circuits emulated with room temperature p-bits," *arXiv preprint arXiv :1810.07144*, 2018.

[80] P. Knag, S. Gaba, W. Lu, and Z. Zhang, "Rram solutions for stochastic computing," in *Stochastic Computing : Techniques and Applications*, pp. 153–164, Springer, 2019.

[81] L. A. de Barros Naviner, H. Cai, Y. Wang, W. Zhao, and A. Ben Dhia, "Stochastic computation with spin torque transfer magnetic tunnel junction," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, 2015.

[82] K. Y. Camsari, S. Salahuddin, and S. Datta, "Implementing p-bits with embedded mtj," *IEEE Electron Device Letters*, vol. 38, no. 12, pp. 1767–1770, 2017.

[83] D. Vodenicarevic, N. Locatelli, A. Mizrahi, J. S. Friedman, A. F. Vincent, M. Romera, A. Fukushima, K. Yakushiji, H. Kubota, S. Yuasa, *et al.*, "Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing," *Physical Review Applied*, vol. 8, no. 5, p. 054045, 2017.

[84] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–5878, 2013.

[85] D. C. Ralph and M. D. Stiles, "Spin transfer torques," *Journal of Magnetism and Magnetic Materials*, vol. 320, no. 7, pp. 1190–1216, 2008.

[86] I. Žutić, J. Fabian, and S. D. Sarma, "Spintronics : Fundamentals and applications," *Reviews of modern physics*, vol. 76, no. 2, p. 323, 2004.

[87] K. Y. Camsari, B. M. Sutton, and S. Datta, "P-bits for probabilistic spin logic," *Applied Physics Reviews*, vol. 6, no. 1, p. 011305, 2019.

[88] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Physical Review X*, vol. 7, no. 3, p. 031014, 2017.

[89] A. Z. Pervaiz, L. A. Ghantasala, K. Y. Camsari, and S. Datta, "Hardware emulation of stochastic p-bits for invertible logic," *Scientific reports*, vol. 7, no. 1, pp. 1–13, 2017.

[90] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *2017 Euromicro Conference on Digital System Design (DSD)*, pp. 1–8, 2017.

[91] S. A. Salehi, "Low-cost stochastic number generators for stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 992–1001, 2020.

[92] P. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[93] M. Yang, B. Li, D. J. Lilja, B. Yuan, and W. Qian, "Towards theoretical cost limit of stochastic number generators for stochastic computing," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 154–159, IEEE, 2018.

[94] S. Priya and D. J. Inman, *Energy harvesting technologies*, vol. 21. Springer, 2009.

[95] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–4, 2014.

[96] M. H. Najafi, D. J. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2018.

[97] S. Asadi, M. H. Najafi, and M. Imani, "A low-cost FSM-based bit-stream generator for low-discrepancy stochastic computing," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 908–913, 2021.

[98] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2018.

[99] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel sobol sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1326–1339, 2018.

[100] Z. Wang, S. Mohajer, and K. Bazargan, "Low latency parallel implementation of traditionally-called stochastic circuits using deterministic shuffling networks," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 337–342, 2018.

[101] Y. Zhang, R. Wang, X. Zhang, Z. Zhang, J. Song, Z. Zhang, Y. Wang, and R. Huang, "A parallel bitstream generator for stochastic computing," in *2019 Silicon Nanoelectronics Workshop (SNW)*, pp. 1–2, 2019.

[102] P. Alfke, "Efficient shift registers, LFSR counters, and long pseudo-random sequence counters," *Xilinx Application Note XAPP 052*, pp. 1–6, 1996.

[103] M. Gay, J. Burchard, J. Horácek, A.-S. Messeng Ekossono, T. Schubert, B. Becker, M. Kreuzer, and I. Polian, "Small scale AES toolbox : algebraic and propositional formulas, circuit-implementations and fault equations," *TRUEDEVICE 2016*, 2016.

[104] D. Blackman and S. Vigna, "Scrambled linear pseudorandom number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 47, no. 4, pp. 1–32, 2021.

[105] P. L'ecuyer and R. Simard, "Testu01 : A C library for empirical testing of random number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 4, pp. 1–40, 2007.

[106] G. Marsaglia, "Diehard : a battery of tests of randomness," https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/, 1996.

[107] N. F. PUB, "140-2 : Security requirements for cryptographic modules," *Information Technology Laboratory, National Institute of Standards and Technology*, 2001.

[108] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, *et al.*, *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010.

[109] S. Vigna, "xoshiro / xoroshiro generators and the prng shootout." https://prng.di.unimi.it. Accédé le : 20-07-2022.

[110] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, "A self-timed ring based true random number generator," in *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pp. 99–106, 2013.

[111] A. Cherkaoui, *Générateurs de nombres véritablement aléatoires à base d'anneaux asynchrones : conception, caractérisation et sécurisation*. Theses, Université Jean Monnet - Saint-Etienne, June 2014.

[112] R. Frisch, M. Faix, E. Mazer, L. Fesquet, and A. Lux, "A cognitive stochastic machine based on Bayesian inference : A behavioral analysis," in *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, pp. 124–131, 2018.

[113] "Renesas re01 1500kb." https://www.avnet.com/wps/portal/silica/products/new-products/npi/2020/renesas-re01-1500kb. Accédé le : 10-07-2022.

[114] T. Ishigaki, R. Tsuchiya, Y. Morita, H. Yoshimoto, N. Sugii, T. Iwamatsu, H. Oda, Y. Inoue, T. Ohtou, T. Hiramoto, *et al.*, "Silicon on thin box (sotb)

cmos for ultralow standby power with forward-biasing performance booster," *Solid-state electronics*, vol. 53, no. 7, pp. 717–722, 2009.

[115] J. Surowiecki, *The wisdom of crowds*. Anchor, 2005.

Table des figures

Liste des tableaux

Liste d'algorithmes

Glossaire

Autocorrélation	Corrélation entre plusieurs nombres aléatoires générés consécutivement par un même RNG.
Bistream	Chaîne de bits représentant un nombre stochastique.
CD	Compression par dérivation. Une des deux méthodes de compression mémoire proposée, liée à la différence entre deux ordonnées consécutives.
CQ	Compression par quantification. Une des deux méthodes de compression mémoire proposée, liée au nombre d'abscisse que partage une ordonnée.
Graine	Etat initial d'un RNG.
KLD	Divergence de Kullback-Leibler. Métrique utilisée pour mesurée la distance entre deux distributions.
IA	Intelligence Artificielle.
LFSR	Linear Feedback Shift Register. Un des PRNGs les plus simples, n'utilisant qu'un registre à décalage et des portes XORs pour générer des nombres aléatoires.
MTJ	Jonction Magnétique à effet Tunnel. Composé de deux matériaux ferromagnétiques séparés par une barrière à effet tunnel, il est utilisé par la communauté du calcul stochastique pour générer des *bitstreams* à faible coût.

N_C	Nombre de colonnes (= N_O + 1 pour le *prior*) de la matrice d'inférence Bayésienne.
N_L	Nombre de lignes de la matrice d'inférence Bayésienne.
N_O	Nombre d'observations / capteurs (= N_C- 1) de la matrice d'inférence Bayésienne.
N_R	Nombre de rails utilisés lors de la parallélisation du calcul stochastique.
Posterior	Probabilité de réalisation de la variable d'intérêt connaissant les observations.
Prior	Connaissance *a priori* de la variable d'intérêt avant tout calcul d'inférence.
PRNG	Générateur de nombres pseudo-aléatoires. Il génère les nombres via un algorithme et est en ce sens totalement déterministe.
res_o	Résolution des données des observations (capteurs) en bits.
res_p	Résolution des données de probabilités en bits.
res_{RNG}	Résolution des RNGs en bits.
RNG	Générateur de nombres aléatoires. Il en existe de deux sortes : les PRNGs et les TRNGs.
SBoNG	RNG spécialement conçu pour l'isolation stochastique. Il est basé sur l'implémentation d'un LFSR et d'une *S-Box* en logique combinatoire.
SCC	*Stochastic Computing Correlation*. Métrique de précision utilisée pour mesurer la corrélation entre deux *bitstreams*. Elle varie entre -1 et 1, et vaut 0 lorsque les *bitstreams* sont décorrélés.
SNG	Un générateur de nombres stochastiques, ou *Stochastic Number Generator*, génère un *bitstream* en utilisant généralement un RNG et un convertisseur binaire / stochastique.
SRI	*Shift Register Isolator*. Méthode d'isolation stochastique isolant les nombres aléatoires eux-mêmes et non les *bitstreams*. Il est basé sur l'implémentation d'un registre à décalage en sortie d'un RNG.
STRNG	TRNG à base d'anneau asynchrone auto-séquencé. Sa source d'entropie est l'incertitude temporelle des signaux électriques : le *jitter*.

TRM	Taux de Reconnaissance du Maximum. Métrique que nous proposons afin d'évaluer la précision de nos architectures. Il mesure la fréquence à laquelle l'architecture reconnaît une vérité-terrain.
TRNG	Générateur de nombres vraiment aléatoires. Les TRNGs tirent leur entropie de phénomènes physiques comme le bruit thermique des circuits.
Vraisemblance	Probabilité de réalisation d'une observation sachant l'état de la variable d'intérêt.
WBG	Weighted Binary Generator. Convertisseur binaire / stochastique composé du Weight Generator (WG) indépendant du biais à généré, et du Probability Encoder (PE).
Xoroshiro	PRNG d'une très bonne qualité statistique et d'une taille raisonnable. Il est basé sur des opérations d'additions, de rotation et de décalage des bits aléatoires.

Vers des calculateurs Bayésiens compacts, à faible énergie et à précision ajustable

Towards compact, low power and with adjustable accuracy Bayesian computers

Résumé

L'intelligence artificielle (IA) révolutionne déjà nos habitudes à travers des technologies permettant l'analyse, le filtrage et la classification d'une grande quantité de données qui ne peuvent être traitées par des algorithmes classiques de par leur complexité et leur incomplétude. Ainsi, il existe aujourd'hui une forte demande pour le matériel dédié à l'IA alors que les objets capables de capter, traiter et distribuer l'information sont de plus en plus contraints en énergie et que la puissance de calcul requise ne fait qu'augmenter. Dans ce contexte, l'approche Bayésienne semble une piste particulièrement intéressante pour réaliser des tâches de fusion d'information de manière explicable, avec peu de données et à faible coût énergétique. De plus, en réduisant les multiplications à une simple porte ET logique, l'arithmétique stochastique permet de réduire significativement la taille des opérateurs (et donc les coûts de fabrication) ainsi que leur consommation ce qui la rend particulièrement adaptée aux calculs d'inférence Bayésienne. De premiers résultats montrent l'efficacité énergétique de cette approche stochastique lorsque la taille du problème à traiter est réduite et lorsque la précision requise est faible. Dans cette thèse, nous proposons des solutions permettant d'adresser les principaux goulots d'étranglement lorsque la taille du problème augmente : l'espace mémoire nécessaire pour le stockage des distributions de probabilité, le coût en surface et en puissance consommée pour générer les nombres aléatoires et le temps de calcul élevé dû à la logique stochastique réduisant les performances énergétiques du système. Par rapport à l'état de l'art, ces contributions permettent de diviser jusqu'à 2 fois la surface et jusqu'à 30 fois la consommation énergétique, tout en divisant jusqu'à 8 fois le temps de calcul.

Mots-clés : Inférence Bayésienne, Calcul stochastique, Fusion de capteurs, IA, Faible consommation, RNG

Abstract

Artificial intelligence (AI) is already revolutionizing our habits through technologies allowing the analysis, filtering and classification of large amounts of data that cannot be processed by classical algorithms due to their complexity and incompleteness. Thus, there is today a strong demand for hardware dedicated to AI while the objects capable of capturing, processing and distributing information are more and more constrained in energy and the required computing power is increasing. In this context, the Bayesian approach seems to be a particularly interesting way to perform information fusion tasks in an explainable way, with little data and low energy cost. Moreover, by reducing multiplications to a simple logical AND gate, stochastic arithmetic allows to significantly reduce the size of the operators (and thus the manufacturing costs) as well as their power consumption, which makes it particularly adapted to Bayesian inference computations. First results show the energy efficiency of this stochastic approach when the size of the problem to be treated is reduced and when the required accuracy is low. In this thesis we propose solutions to address the main bottlenecks when the problem size increases : the memory space required to store the probability distributions, the cost in area and power consumed to generate the stochastic numbers and the high computation time due to the stochastic logic, reducing the energy performance of the system. Compared to the state of the art, our contributions allow to divide up to 2 times the area and up to 30 times the power consumption, while dividing up to 8 times the computation time.

Keywords : Bayesian inference, stochastic computing, sensors fusion, AI, low power, RNG