Architectures de circuits nanoélectroniques neuro-inspirée

Djaafar Chabi

UNIVERSITÉ PARIS SUD

Comprendre le monde, construire l'avenir®

THÈSE DE DOCTORAT

SPÉCIALITÉ : PHYSIQUE

Ecole Doctorale : « Science et Technologies de l'Information des Télécommunication et des Systèmes »

Présenté par

Djaafar CHABI

Sujet :

« Architecture de circuits nanoélectroniques neuro-inspirée »

Soutenue le 09 mars 2012 devant le jury composé de :

M. Ian O'connor	LIRMM, Montpellier	*Professeur*
M. Gilles Sassatelli	INL, Lyon	*Directeur de recherche*
M. Pandavoine Michel	LEAD, Bourgogne	*Professeur*
Mme. Dezan Catherine	Univ. Bretagne occi	*Maître de conférences*
M. Leveugle Régis	TIMA, Grenoble	*Professeur*
Jacques-Olivier KLEIN	IEF, Paris sud	*Directeur de thèse.*

"We can't solve problems by using the same kind
of thinking we used when we created them."
"Aucun problème ne peut être résolu sans
changer l'état d'esprit qui l'a engendré. "
Albert EINSTEIN
(Prix Nobel de Physique de 1921, 1879 - 1955)

Remerciement

*Au terme de ces travaux effectués au sein de l'Institut d'Electronique Fondamental (**IEF**) du groupe NANOARCHI, à l'université Paris sud, je tien à remercier :*

*Le directeur du laboratoire **Claude Chappert** et les deux responsables de département Nanoélectronique, **Dafiné Ravelosona** et **Philippe Dollfus** pour la confiance qu'ils m'ont témoignée en m'accueillant dans leur service ;*

*Mon directeur de thèse **Jacques-Olivier Klein** pour m'avoir encadré durant ces trois années. Il a su guider efficacement mes travaux de recherche tout en me laissant une liberté très appréciable. Merci également à Weisheng zhao et Damien Querlioz pour l'intérêt qu'ils ont porté à mes travaux et les conseils qu'ils m'ont dispensés ;*

Les membres de mon jury, Messieurs Gilles Sassatelli et Ian O'connor, respectivement Directeur de recherche à l'université de Montpellier (LIRMM) et Professeur à l'INL (Lyon), pour avoir accepté d'être rapporteur de ce travail et pour leurs commentaires pertinents. Je voudrais remercier aussi Monsieur Michel Paindavoine de m'avoir fait l'honneur de présider le jury, et Monsieur Régis Leveugle et Madame Catherine Dezan, pour avoir accepté de faire partie du jury de ma thèse.

A tous les partenaires du projet « PANINI ». Notamment, du CEA-LEM : Vincent Derycke, Guillaume Agnus et Karim Gassem ; de Le2i (université de Bourgogne) : Michel Paindavoine et Olivier Brousse ; à Cristell Maneux maître de conférence à l'université de Bordeaux et son doctorant Si-Yu.

A tous mes collègues de l'équipe NST, aux doctorants et post-doctorants que j'ai rencontré pendant la préparation de ma thèse. Merci à Thibaut et Joo-Von pour l'animation des débats pendant la pause café, à Nicolas pour ses anecdotes, à Sébastien pour son humour (qui fait déraper parfois les débats scientifiques ou politiques vers des histoires de kiné de Bures, :), à Sylvain notre ingénieur au même temps animateur de l'équipe (merci pour les sorties et les dîners), à Capucine et Ivanca (merci d'avoir choisi le 2ème étage), à Ruben, Jean-Marie, Arnaud, Pierrick, Laurence, Annerose, Minh, Sumanta, Yahya, Jean-Paul, Cyril, Yue, Na, Weiwei, Daniel, Charlotte, Olga, Paulo, Vincent, Fulvio, Salim, Hani, Katia, Mariem, Fatma, Houssien, et particulièrement je remercie mon ami Simon d'avoir lu et corrigé les fautes d'orthographe de mon manuscrit.

A tous mes amies et compagnons, à Olga (merci d'avoir emprunté le fer à repasser, ☺), Hamada et sa femme, à Rochdi et Hafid erachetti, Hichem, à mon cousin Hafid, à Yasmina et tous qui m'ont soutenu de près ou de loin.

Je tiens enfin à exprimer ma reconnaissance à toute ma famille et mes proches pour leur soutien, merci à mes chères sœurs et à mon frère et mille mercis à mes parents pour leur indéfectible soutient, sans eux je ne serais pas là où j'en suis aujourd'hui.

Table des matières

Introduction générale

L'industrie du semi-conducteur s'est mobilisée pour respecter la loi empirique annoncée par *G. Moore* (cofondateur d'Intel) en 1965, qui prévoyait le doublement du nombre de transistors dans un circuit intégré (CI) chaque dix-huit mois à deux ans. Pour atteindre cet objectif, l'effort a été principalement concentré sur l'amélioration des technologies qui permet d'avoir une meilleure finesse de gravure et par conséquent un nombre de transistors intégrés par unité de surface en augmentation dans les CI. Cette course à la miniaturisation a eu pour effet une augmentation exponentielle de la puissance de calcul de sorte que la loi de Moore a « tué » l'innovation concernant les nouvelles architectures matérielles fondées sur d'autres composants que le CMOS, notamment celles qui exploitent de nouveaux paradigmes de calcul. Aujourd'hui les industriels se heurtent aux propriétés des transistors qui atteignent des échelles nanométriques où ils deviennent régis par les lois quantiques. De l'aveu même de son inventeur, la limite de la loi de Moore approche et aucune solution alternative n'est clairement identifiée pour l'instant bien que de nombreuses technologies concurrentes émergent.

Les technologies émergentes désignent tous les systèmes et dispositifs en voie de développement dans le but de compléter voire remplacer la technologie basée sur le CMOS afin de préserver la loi de Moore. Les nouvelles technologies permettant de combler les lacunes technologiques du CMOS se divisent en deux grands groupes. Tout d'abord, on peut citer les technologies utilisant une physique similaire qui devraient permettre de pousser la technologie CMOS jusqu'à ses limites ultimes. La majorité des composants contenant un canal en silicium fonctionne de la même façon que le CMOS, et cela indépendamment de la nature de substrat, *Si* ou silicium sur isolant (SOI), et de l'architecture du composant (simple ou double grille, grille planaire ou 3D). De nouveaux nanomatériaux permettent de remplacer le canal de CMOS par des Nano Tube de Carbone comme le CNTFET, ou par du graphème pour les GFET, ou des nano fils de silicium (SiNW) qui semblent très prometteurs [Cui01, Cui03]. On trouve ensuite les technologies émergentes qui sont basées sur des phénomènes physiques différents afin d'éviter les limitations intrinsèques de la technologie CMOS. Au lieu d'utiliser la charge de l'électron comme variable d'état ces nouvelles technologies font appel au spin, à l'état moléculaire ou au bit quantique (q-bit). Parmi ces nouveaux nanocomposants, on peut citer les commutateurs moléculaires basés sur des molécules comme le rotaxane, les memristors, le SpinFET, les automates cellulaires quantiques (QCA) [Lent1] et les transistors à un seul électron (SET) [Chen96]. L'apparition de ce type de composants a ouvert la voie vers de nouvelles familles d'architectures à nanocomposants à caractère reconfigurable

comme par exemple les automates cellulaires quantiques (QCA), l'électronique moléculaire, la spintronique ou les architectures neuromorphiques (figure I.1).

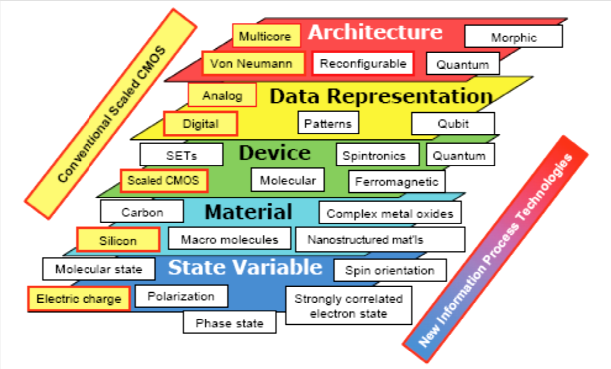


Figure I.1 : Taxonomie de domaine de recherche des technologies émergentes [ITRS 2007].

L'un des avantages majeurs de ces technologies réside dans la possibilité d'obtenir des circuits à haute densité d'intégration (10^{12} composant/cm^2) [Jo09] grâce à des techniques d'auto-assemblage [Gree01, Gold01, Brown07] qui utilisent des procédés chimiques pour l'assemblage et pour l'alignement des nanocomposants, dont les coûts de fabrication sont beaucoup plus faibles que ceux des circuits CMOS. Cependant le manque de fiabilité et la présence de défauts inhérents à cette technique de fabrication peut causer un sérieux frein à leur adoption. Le défi actuel est de diminuer le taux élevé de défauts pour obtenir des composants compétitifs. Ainsi, sauf rupture technologique majeure, la tolérance aux fautes reste le seul moyen qui permettra de surmonter ces défauts et de concevoir d'une façon économique des systèmes de haute densité. Les techniques de tolérance conventionnelles qui considèrent un seul composant de circuit défaillant, déjà utilisées dans le cas des architectures CMOS, ne sont plus valables dans le cas des nanoarchitectures où le taux de défauts est très grand. Il faut donc développer de nouvelles techniques de tolérance aux fautes et de nouveaux paradigmes de calcul, qui vont permettre d'intégrer ces nouveaux nanocomposants de manière fondamentalement différente de celle introduite par *Von Neumann* [Neu37]. Plusieurs auteurs proposent d'adapter les techniques de tolérance classiques aux nanoarchitectures, comme la TMR (Triple Modular Redundancy) [Niko02, Kuek05, Bhad05] ou le multiplexage de *Von Neumann* [Han04, Sad04, Niko01]. L'inconvénient majeur de ces techniques est leur mise en œuvre, nécessitant des surfaces importantes ce qui fait perdre l'intérêt des nanoarchitectures. D'autres auteurs proposent des techniques basées sur la reconfiguration et l'auto-test pour tolérer les défauts des nanoarchitectures. Ces techniques ont montré des taux de tolérance aux défauts plus important que les deux premières. Cependant, pour des circuits à haute densité d'intégration, le temps nécessaire pour l'étape de test et la mémoire nécessaire pour stocker la carte des défauts rend cette approche de reconfiguration inapplicable.

En effet, la haute densité d'intégration de ce type d'architectures rend difficile le contrôle de leur taux de défauts élevé. La solution la plus prometteuse pour tolérer ce

taux de défauts élevé sans perdre l'intérêt des nanoarchitectures est de développer une nouvelle approche de reconfiguration qui soit intrinsèquement tolérante aux défauts et aux dispersions technologiques. En se basant sur plusieurs études montrant l'efficacité des réseaux de neurones artificiels (RNA) en termes de fiabilité et de rapidité de traitement de l'information [Emm91, Ker94, Ass97, Mats94, Tana89, Bed88, Cun89], nous pensons qu'une technique de configuration inspirée de l'apprentissage neuronal peut constituer une solution à ces problèmes. Dans ce contexte, les travaux de cette thèse proposent une nanoarchitecture originale de type crossbar d'inspiration neuronale, développent des techniques d'apprentissage adaptées pour ce type d'architectures qui serviront à configurer les nanocomposants pour apprendre des fonctions logiques, étudient la tolérante aux défauts et aux variations technologiques pour montrer la robustesse de ce type d'architecture.

Les travaux antérieurs au sein de notre équipe de recherche menés par *J-O. KLEIN* et *Eric BELAIRE* et les travaux de thèse de *Michel He* ont permis de développer les premiers modèles d'architecture crossbar basée sur des nanocomposants reconfigurables à multi-niveaux (memristor, CBRAM, ReRAM, OGCNTFET...). La technique proposée pour configurer ce type d'architecture est basée sur l'apprentissage neuronal basé sur la règle de « *Delta Binaire* ». Ce type d'architectures sera nommé dans ce manuscrit « neuro-crossbar ». L'architecture crossbar a été choisie par rapport à la large utilisation ces dernières années pour implémenter des circuits nanoscopiques. L'intérêt d'une telle structure basée sur l'application d'une méthode d'apprentissage à une matrice de nanostructure est d'implémenter des fonctions d'une façon économique et compatible avec une grande densité de nanocomposants (10^{12} nanocomposant/cm^2). A l'image du cerveau humain, les réseaux de neurones artificiels (RNA) sont considérés comme des systèmes de calcul tolérants aux fautes à cause de la nature distribuée du codage de l'information et de leur redondance intrinsèque [Emm92, Ker94, Ass97].

Dans le cadre de ce travail de thèse, nous nous sommes appliqués à développer des structures de modèles plus complexes de « neuro-crossbar » pour implémenter par exemple des réseaux de neurones multicouches (MLP). Nous avons développé aussi des nouvelles méthodes d'apprentissage plus robustes. La majorité de nos travaux a été concentrée sur l'étude de la fiabilité de ce type d'architecture qui constitue le principal intérêt des méthodes de configuration d'inspiration neuronale.

Le présent manuscrit est composé de quatre chapitres. Dans le premier chapitre, nous présenterons l'état de l'art qui reprend d'une manière générale les problèmes actuels du CMOS et les différentes solutions possibles permettant de surmonter ces problèmes. Quelques travaux intéressants sur les nanoarchitectures sont présentés comme une solution à long terme pour l'amélioration ou le remplacement de la technologie CMOS. La troisième section de ce chapitre introduit les RNA et leurs différentes implémentation (analogique, ASIC, FPGA, nanométrique). La quatrième section présente les nouveaux nanocomposants de type memristif. L'intérêt de cette

section est de tirer partie des opportunités qu'offrent les nanocomposants pour réaliser des architectures plus complexes et plus flexibles que celles basées sur la technologie CMOS. Cependant, comme nous l'avons déjà mentionné cela ne peut pas être réalisé sans le développement de nouvelles techniques de tolérance aux fautes. Ceci nous amène à présenter dans la dernière section de ce chapitre les techniques de tolérance les plus utilisées.

Nous consacrons le deuxième chapitre à la description de l'architecture neuro-crossbar (NC) et les circuits qui permettent de l'implémenter. Nous présenterons les résultats de simulations d'apprentissage de fonctions logiques en utilisant des NC composés de memristors, ainsi que les applications possibles pour ce type d'architectures.

Le troisième chapitre, présente l'étude de la fiabilité des neuro-crossbars en présence de défauts et de variations technologiques des memristors qui composent le NC. Les simulations Monte-Carlo vont être utilisées pour mesurer le degré de tolérance de NC, en présence et en absence de redondance, pour chaque type de défaut et de dispersion. Des modèles probabilistes permettant de prévoir la probabilité de convergence d'un apprentissage de NC en présence de défauts ou de dispersions vont être proposés dans ce chapitre.

Le quatrième chapitre présente une démonstration expérimentale de l'apprentissage neuronal réalisé sur des vrais neuro-crossbars composés de transistors OG-CNTFET.

A la fin de ce manuscrit, je conclurai cette thèse en rappelant les principales contributions originales et je tracerai des perspectives.

CHAPITRE I

Etat de l'art

I. Introduction

L'augmentation de la densité d'intégration et la recherche de systèmes plus performants qui consomment de moins en moins d'énergie, constituent aujourd'hui un des axes majeurs de la recherche dans le domaine de la nanoélectronique. Dans ce contexte, une attention particulière est portée sur le développement de nouvelles nanostructures (nanotubes de carbones, nanofils, transistors moléculaires, diodes...) et leurs méthodes de fabrication. De nombreux défis se posent pour les architectures du futur :

1. La température doit être maitrisée (ce qui comprend également la distribution de l'alimentation, l'évacuation de la chaleur et le traitement des points chauds);
2. L'accroissement des dispersions doit être maitrisé et la fiabilité améliorée, en utilisant la redondance spatiale ou temporelle, ou les deux, avec un taux de redondance raisonnable ;
3. Les coûts associés au test doivent diminuer;
4. La connectivité doit être améliorée, en réduisant la longueur et le nombre de connexions;
5. La communication doit être améliorée, en optimisant les méthodes et les protocoles de communication ;
6. La logique binaire doit être optimisée afin de réduire le nombre de commutation, pour le calcul comme pour la communication;
7. Le coût de la lithographie extrême doit être maitrisé.

Face à la multitude des défis technologiques à surmonter, la question de l'abandon de la technologie CMOS au profit de ces technologies de pointe reste à ce jour sans réponse.

Nous présenterons dans la suite de ce chapitre l'évolution de la technologie CMOS et les limites menaçant l'arrêt de cette évolution avec une solution à court terme permettant de pousser cette évolution à l'extrême. Nous présenterons ensuite la solution à long terme qui consiste à proposer des nanoarchitectures pour l'amélioration ou le remplacement de la technologie CMOS. Nous nous intéressons par la suite à l'architecture neuromorphique représentée comme une approche tolérante aux taux élevé de défauts présents dans les nanoarchitectures. Nous détaillerons cette approche, en rappelant la définition des réseaux de neurones artificiels (RNA) et leurs différents algorithmes d'apprentissage, ainsi qu'un tour d'horizon de leurs nombreuses possibilités d'implémentation. Commençons par l'implémentation analogique basée sur le CMOS puis des implémentations plus évoluées basées sur les ASIC (Application Specific Integrated Circuit) et enfin l'implémentation basée sur les FPGA (Field Programmable Gate Array). Ces derniers peuvent être améliorés en se basant sur les dernières avancées de la nanotechnologie. Dans ce contexte, nous présenterons

quelques nouveaux composants et systèmes nanométriques et spécialement les systèmes memristifs que nous détaillerons, en présentant leurs différents types et leurs différentes applications (mémoire, logique, architecture neuromorphique). Nous clorons ce chapitre en présentant les architectures tolérantes aux fautes les plus connues.

II. Technologie CMOS

II.1 Pourquoi le CMOS ?

En général, le terme CMOS désigne le procédé de fabrication des puces sur une tranche de silicium. De plus il caractérise également l'architecture des cellules utilisées pour la conception de circuits numériques dans laquelle un bloc constitué de transistors PMOS créé un chemin entre la sortie et le rail d'alimentation, tandis qu'un bloc complémentaire constitué de transistors NMOS créé un chemin vers la masse pour les conditions complémentaires. L'absence du courants permanents entre les deux rails d'alimentation associée à l'isolation de l'électrode de commande dans un transistor MOS conduit à une puissance statique très limitée en comparaison des autres technologies de transistors. Cet avantage associé à la grande densité d'intégration de la technologie CMOS en ont fait la technologie dominante.

II.2 Evolution de CMOS

Au cours des 40 dernières années, l'industrie microélectronique a connu une croissance phénoménale, où la taille des transistors CMOS n'a pas cessée de diminuer validant ainsi la loi empirique définie en 1965 par *G. Moore*, co-fondateur de la société Intel, qui prévoyait la croissance exponentielle du nombre de transistors des circuits intégrés. Cette loi a évolué par la suite dans le sens d'un doublement du nombre de transistors tous les dix-huit mois à coût constant. De manière générale la diminution de la taille des transistors a conduit à des améliorations considérables des vitesses de circuits et des performances des ordinateurs. Par exemple le 4004, premier microprocesseur 4-bit lancé par Intel en 1971, contenait 2300 transistors en technologie 10µm et travaillait avec une fréquence de 108 kHz [Int1], alors qu'un microprocesseur Intel d'aujourd'hui, comme le i7-980X par exemple contient 1170 million de transistors fabriqués en technologie 32nm répartis en six cœurs fonctionnant chacun à une fréquence de 3.33 Ghz. L'ITRS (International Technology Roadmap for Semiconductors) annonce chaque année les futures générations technologiques, appelées «nœuds» (90nm, 65nm, 45nm, 22nm...). Selon ITRS-2010 le prochain nœud est le 22nm qui devrait être atteint entre 2010-2012, Intel a déjà sorti le premier processeur au monde gravé en 22nm sous le nom de Ivy Bridge. La production de la série de portable utilisant cette technologie est prévue pour la fin de l'année 2011. Le nœud 16nm est prévu pour 2013-2014. IBM a déjà démontré en 2010 la possibilité de graver à moins de 15nm avec la nouvelle technique NanoPattering

[Brug00]. Les laboratoires d'Intel ont aussi montré qu'ils pouvaient graver en 15 nm en utilisant le laser à exciplexe classique de 193 nm et prévoit de descendre jusqu'à 11 nm. Cela montre le degré de l'avancé technologique et les hautes performances que peuvent atteindre les systèmes électroniques d'aujourd'hui.

II.3 Limites de CMOS

Néanmoins, les différentes améliorations de performances que nous venons de citer sont suivies par une croissance exponentielle de la consommation statique des transistors et une forte variabilité de leurs caractéristiques (effet tunnel, effet de canal court) [Taur97b]. Il n'y a pas que l'augmentation de la densité d'intégration qui améliore la puissance de calcul dans les circuits intégrés mais le raccourcissement du temps nécessaire à chaque opération peut améliorer aussi la puissance de calcul. La figure I.1 présente les différentes contraintes physiques traduites par une relation entre la densité de composants et le temps moyen d'opération par composant. La miniaturisation des transistors CMOS s'approche de plus en plus des limites physiques (relativiste, quantique, dissipation thermique) qui commencent à apparaître comme de sérieux indicateurs. La plus connue de ces limitations est la vitesse de commutation, limitée selon le principe d'incertitude de Heisenberg, par la puissance minimale dissipée et le temps de commutation : $P_{mini}.t_{com}^{\ 2} > h$, h étant la constante de Planck. Pour éviter les erreurs quantiques le temps de commutation doit être plus long que le temps de transition quantique (10^{-14} s), ce temps apparait comme la ligne horizontale de la figure I.1 montrant la limite quantique [Fanet].

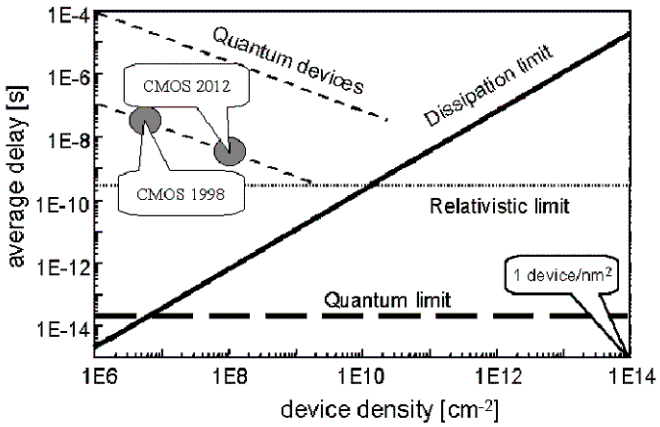


Figure I.1 : Délai moyen en fonction de la densité de composants, montrant la limite relativiste, la limite quantique et la limite en dissipation thermique, pour les circuits intégrés CMOS à température ambiante

II.4 Solution à court terme

Diverses questions se posent pour l'avenir : tout d'abord sur la possibilité de remplacer le transistor CMOS, et sur l'évolution des technologies émergentes. Il est important de trouver rapidement des réponses à ces questions car cela représente un

enjeu économique considérable pour l'industrie Microélectronique. Un changement de la technologie CMOS par un composant émergent, comme le transistor à un seul électron ou le Spin-FET, impliquerait des modifications à plusieurs niveaux (technologie, design, modèle et méthode de programmation) ce qui demande aussi un grand investissement pour les prochaines années voire au-delà en considérant le temps de mettre en œuvre toute l'architecture et le développement des nouveaux produits et logiciels. En attendant le développement d'une architecture à base de nanocomposants qui dépassera les performances des architectures CMOS, les fondeurs et fabricants continuent à trouver des solutions qui repoussent à l'extrême les limites physiques apparentes du CMOS (Nano-CMOS, CMOS ultime) [Taur97a]. Dans ce contexte, le CMOS a subi de nombreuses modifications. Les matériaux de la grille et le diélectrique (high k-dielectrics) ont été changés pour diminuer l'effet des courants de fuite [Gdwil]. Les matériaux d'interconnexions métalliques ont évolué de l'aluminium vers le cuivre pour diminuer la résistance. Les changements de la structure de base imposent des techniques de fabrications de plus en plus complexes ce qui fait augmenter le coût de fabrication. Néanmoins, l'apparition récente des transistors 3D (Tri-Gate transistors) à triple grille, fabriqués par le leader de l'industrie des semi conducteurs Intel, semble avoir stabilisé le coût de fabrication. Intel annonce une augmentation de coût de fabrication de quelques pourcents (2 à 3%) pour des transistors Tri-Gate fabriqués en 22nm avec une augmentation de 37% des performances en basse tension. L'expansion de la technologie des semi-conducteurs va donc poursuivre encore quelques années (figure I.2).

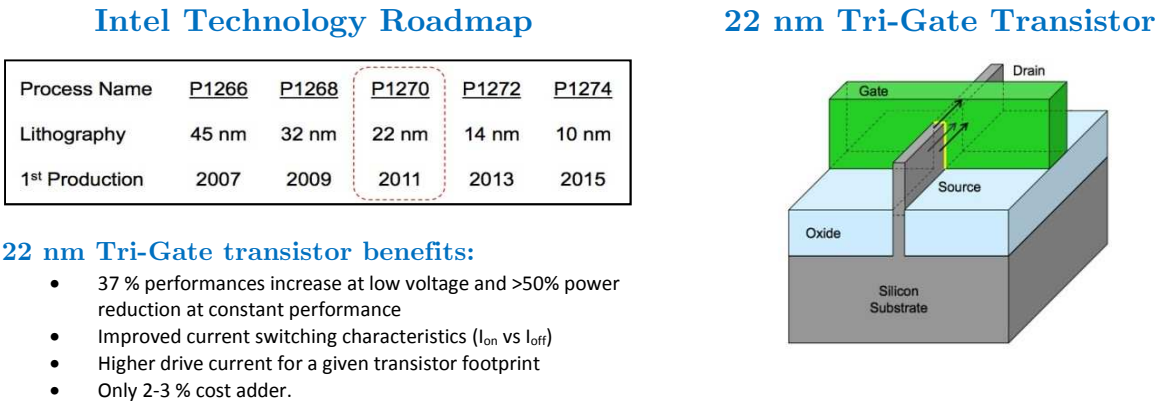


Figure I.2 : Dernière avancée technologique d'Intel sur les transistors CMOS.

III. Nanoarchitectures

Dans la section précédente nous avons présenté un aperçu de la situation actuelle de la technologie CMOS en soulignant les différents problèmes de variabilités et les limites physiques qui peuvent provoquer la fin de cette technologie. L'explosion du coût de fabrication des CMOS de dernière génération (ultime) et la dispersion intrinsèque des caractéristiques des nanocomposants nécessitent de développer de nouvelles architectures plus tolérantes aux fautes à moindre coût. Les dernières avancées du CMOS 3D présentées à la fin de la partie précédente démontrent la possibilité de maintenir la loi de Moore pendant encore quelques années au prix d'une complexification modérée des techniques de fabrication. Cela devrait permettre de gagner du temps pour bien exploiter les avancées de la nanoélectronique afin de préparer les nouvelles générations d'architectures et les nouveaux paradigmes de calcul qui prendront le relais.

Depuis plus de 50 ans, la plupart des architectures sont basées sur le modèle de *Von Neumann* [Neu37]. Les efforts de recherche importants menés par les acteurs majeurs de la microélectronique sont focalisés principalement sur la miniaturisation de dispositifs électroniques avec l'obsession de respecter la loi de Moore. Cependant, l'architecture de *Von Neumann* n'a subi que quelques améliorations, telles que l'amélioration du jeu d'instruction ou l'utilisation de la technique de *pipeline* pour diminuer les temps d'exécution dans les processeurs réalisés au début des années 80. Le grand changement qui a vraiment marqué l'architecture des microprocesseurs est l'invention du microprocesseur double cœur par IBM au début des années 2000. Cette invention a provoqué une rupture dans l'évolution des microprocesseurs : la course à la fréquence a cédé la place à la course aux nombres de cœurs intégrés dans le même circuit. Par exemple Intel a abandonné la course à la fréquence avec le Pentium V et s'est lancé dans la fabrication des microprocesseurs multicoeurs (i3, i5, i7, i9...). Toutefois, l'augmentation du nombre de cœurs peut poser beaucoup de difficultés au niveau de la gestion du parallélisme entre ces cœurs et provoquer aussi des problèmes liés au rendement énergétique et à la latence mémoire. Par conséquence Intel cherche à modifier l'architecture interne de ses microprocesseurs pour avoir une faible consommation (Clovertown et Tigerton). Tous ces efforts montrent que les architectures de calcul d'aujourd'hui sont en train d'atteindre leurs limites. Néanmoins, ces limitations constituent une opportunité pour explorer plus intensivement d'autres paradigmes de traitement de l'information comme les réseaux de neurones et le calcul quantique. Par ailleurs, on remarque une accumulation de nouvelles architectures à nanocomposants, comme la Nanocell [Tour03], la Nanofaric [Gold01], les architectures nano-FPGA détaillées dans [Deh04], l'architecture CMOL (CMOS/ nanowires /Molecular) [Likha04], les architectures crossbar moléculaires [Chen03, Adeh05, Gold01, Kuek01, Stan03], et neuromorphiques [Snid11, Gor10, Likha04, Gros01, Liao11].

Les différentes architectures et solutions proposées ne sont pas purement technologiques ou architecturales mais combinent généralement les deux approches. En résumé, il y'a deux approches possibles pour exploiter les nanocomposants :

- Un changement radical des architectures actuelles, en les remplaçant par exemple, par des systèmes auto-organisés comme le concept de la Nanocell ou les systèmes s'inspirant des réseaux de neurones ou des réseaux d'automates. Ces concepts sont très prometteurs pour réaliser des architectures avec des coûts très faibles et tolérantes aux défauts de fabrication.
- Les architectures hybrides qui permettent d'associer les dispositifs nanométriques aux architectures CMOS conventionnelles. Il peut s'agir par exemple d'ajouter des mémoires composées de nano-fils ou de nanotube de carbone à un circuit CMOS. Cette approche est connue sous le nom de Nano-Inside.

III.1 La Nanocell

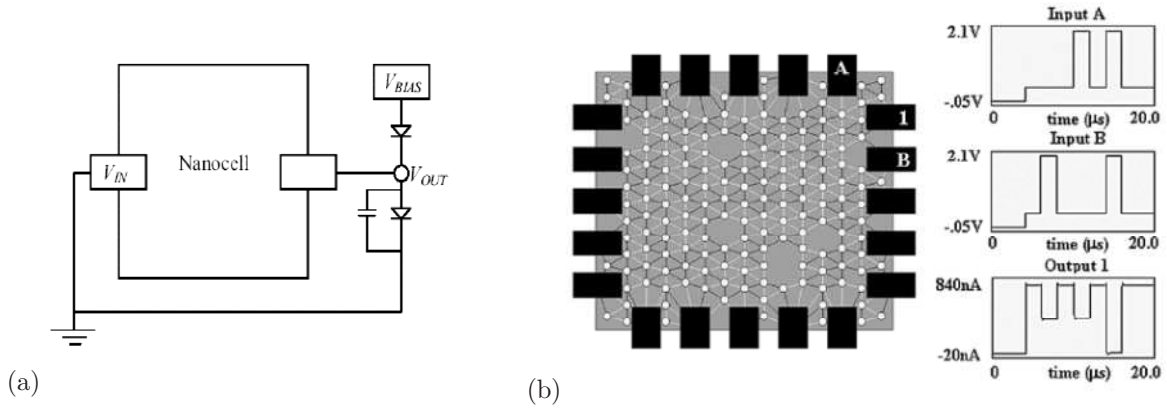
Proposé par l'équipe de *James Tour*, le concept de la Nanocell est une approche innovante et pragmatique permettant d'utiliser directement les molécules pour réaliser des dispositifs logiques de base avec des techniques de fabrication chimiques simples, permettant de dépasser les contraintes de fabrication de la lithographie extrême [Tour03, Chris03].

Comme le montre la figure I.3, La Nanocell est fabriquée à partir d'une cavité créée sur une tranche de silicium, composée d'un ensemble de molécules (nitroaniline) qui peuvent être en liaison chimique entre elles par l'intermédiaire de nanoparticules d'or. Cet ensemble est vu comme un réseau de dipôles non-linéaires interconnectés de manière aléatoire pour implémenter une fonction complexe. L'état de chaque molécule change selon la tension appliquée à ces bornes. Deux états sont distingués, un état conducteur *On* et un état non conducteur *Off*. Des plots électriques d'entrées/sorties sont disposés à la périphérie de la nanocell, permettant l'interconnexion avec d'autres nanocells ou avec d'autres circuits électroniques classiques. Le concept de la nanocell est fondé sur l'approche de fabrication bottom-up, dont l'une des conséquences est que le réseau de dipôles se trouve dans un état non défini après fabrication. La programmation de ce réseau peut s'effectuer en appliquant des tensions sur les plots périphériques pour changer les caractéristiques de ce réseau. En suivant un algorithme bien déterminé, il est possible d'établir une relation logique entre les tensions appliquées sur certains plots et les courants mesurés sur d'autres.

Tour et al ont réussit en simulation à programmer des fonctions logiques en utilisant la molécule de *nitroaniline* qui possède des caractéristiques d'une NDR (Negative Differential Resistance). La présence de cette caractéristique de résistance négative permet d'effectuer une fonction de type inverseur nécessaire pour un système logique complet. Le principe de la programmation est basé sur l'application des

tensions par les entrées A et B, et la restauration de niveau sur la sortie (1) est effectué grâce à un latch bistable, lui-même à base de deux molécules de type NDR.

Les nanocells peuvent être utilisées pour fabriquer des circuits logiques simples à consommation et coût très faibles. Cependant, la programmation de fonctions dans des vrais nanocells ne semble pas encore accessible et l'interconnexion de plusieurs nanocells dans un circuit reste un défi à surmonter.

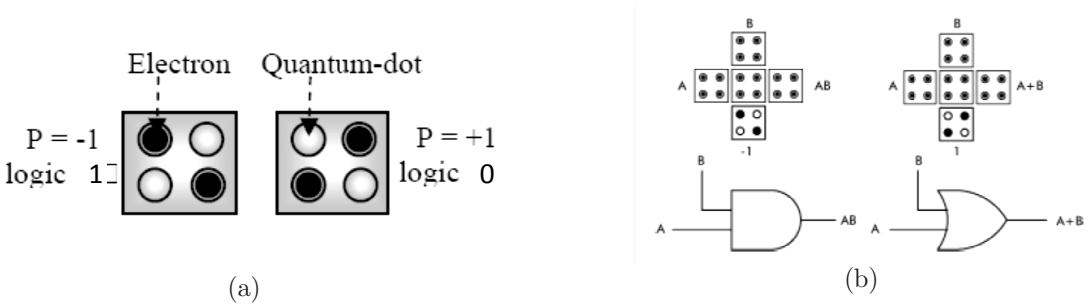


(a) (b)

Figure I.3 : (a) conversion courant-tension et restauration de niveau logique par utilisation d'un latch bistable à base de deux molécules présentant des caractéristiques de NDR Porte Nand à base de molécules, (b) Nanocell configurée en porte NAND [Chris03].

III.2 Les QCA (Quantum Cellular Automata)

Le concept d'automates cellulaires quantiques (QCA) est proposé pour la première fois par *Lent, Tourod, Prod et Bernstein* en 1993 [Lent93]. Le QCA est l'un des dispositifs nanotechnologiques capables de remplacer les technologies basées sur le silicium. Il s'agit d'une approche permettant de réaliser des calculs avec une représentation différente de l'information. Une cellule QCA est composée de quatre sites quantiques et de deux électrons (figure I.4 (a)). Sous l'effet de la force de répulsion Coulombienne, les deux électrons ne peuvent se placer que dans deux sites quantiques diamétralement opposés. Ainsi, l'effet interne de la cellule met en évidence deux configuration possibles, chacune va être utilisée pour représenter un état binaire « 0 » ou « 1 ». Une topologie de QCA est un pavage de cellules QCA. L'interaction entre les cellules permet de transmettre l'information ce qui donne la possibilité de remplacer l'interconnexion physique des dispositifs. Ainsi, des portes logiques sans fil peuvent être réalisées en mettant des cellules QCA côte à côte d'une façon adéquate (figure I.4 (b)). Plusieurs dispositifs QCA ont été démontrés. Certaines équipes ont implanté des cellules avec des points métalliques mais la meilleure densité serait atteinte avec une implantation moléculaire [Gar03, Lent03].



(a) (b)

Figure I.4 : (a) Schéma de cellule à quatre points quantiques, la force de la répulsion de coulomb impose aux électrons d'occuper les coins opposés de la cellule, deux états binaire se distinguent, l'état binaire « 1 » (P=+1) et l'état binaire « 0 » (P=-1). (b) En fixant la valeur d'une des entrées de la porte majorité à « 0 », on obtient une porte AND. Inversement, en fixant à « 1 » on a une porte OR.

La technologie QCA présente un certain nombre d'avantage qui la rend attractive. Une densité d'intégration qui peut atteindre les 50 Gbits/cm^2 [Fost02] avec des vitesses de calcul élevées. La fiabilité et la tolérance aux défauts des QCA ont été démontrées en utilisant la redondance [huan06], mais la sensibilité au bruit des QCA reste un problème à régler. D'autres difficultés liées à la conception et la fabrication de ces systèmes peuvent s'opposer à leur intégration, notamment des problèmes liés à la restauration de gain et la dissipation dans les cellules [Orlo02, Tim02] ainsi que des problèmes d'interfaçage avec les systèmes classiques.

III.3 Nanoarchitecture reprogrammable (FPGA, PLA,)

Dehon et al ont introduit des structures reprogrammables pour l'électronique moléculaire basée sur les nanotubes (NT) de carbone et les nanofils (NF) de silicium [Adeh02, Adeh05]. L'équipe de *Déhon* cherche à reproduire tout ce qui se fait en microélectronique à l'échelle nanométrique. Ils développent des architectures qui fournissent les fonctionnalités de la logique universelle avec la restauration de signal, des commutateurs électrostatiques comme les diodes et les FET (Field Effect Transistor) peuvent être réalisés en connectant des NT ou des NF (figure I.5 (a)). Des matrices basées sur ces éléments sont réalisés afin de construire des cellules de taille nanométrique de type PLA (Programmable Logic Array) [Adeh04]. L'idée de cette structure est d'avoir une collection de matrices interconnectées à l'échelle nanométrique pour réaliser des circuits reconfigurables de type FPGA (Field Programmable Gate Array). Les blocs composés de NF peuvent être programmés pour réaliser des portes logiques de type *OU,* or cette porte logique n'est pas un opérateur universel, il n'est donc pas possible de réaliser une porte de type inverseur à partir de cet opérateur. Un autre étage est implémenté à base de NF dopés, que l'en contrôle comme des FET pour l'inversion et la restauration de niveaux logiques(Figure I.5 (b)). Pour l'interfaçage entre les nanofils et les microfils (Figure I.5 (c)), *Dehon* s'est basé sur les travaux de *Lieber* qui consiste à utiliser des décodeurs composés de matrices de cNW-FET (Crossed semiconductor Nanowire Field-Effect Transistor) [Linco03, Zhon03].

Cette architecture est aussi conçue pour tolérer les défauts de fabrications avec le processus de « sparing and remapping » qui permet de localiser les défauts après un test pour pouvoir les éviter pendant le fonctionnement de circuit [Adeh96]. Dans la même optique, *Goldstein* propose des architectures reconfigurables composées de nano blocs moléculaires (Molecular Logic Array, MLA), fournissant ainsi un moyen de gérer les défauts par la programmation [Mbud01, Gold01].

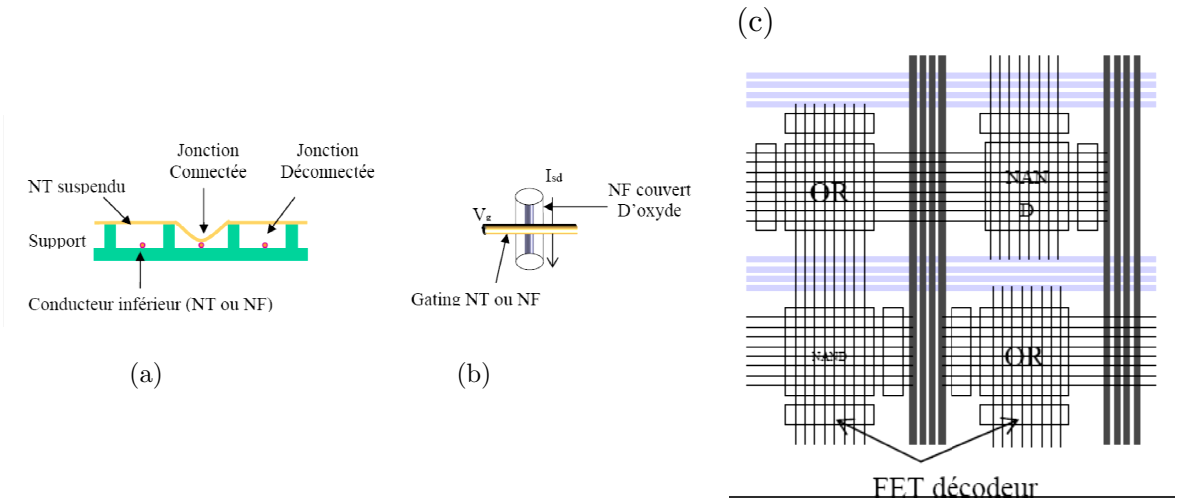


Figure I.5 : (a) Connexion commutée des nanotubes, (b) Jonction FET, (c) Interconnexion des nanoblocs fonctionnels [Adeh02].

Un autre type d'architecture reconfigurable de type FPGA a été proposée par l'équipe de *Likharev* [Likha05], il s'agit d'une architecture hybride appelée CMOL (CMOS/ nanowires /Molecular), elle est composée d'une couche de CMOS classique par-dessus laquelle est formée une couche de nanodispositifs. Selon *Likharev* ce type d'architecture représente la meilleure façon pour implémenter les réseaux de neurones. Dans ce contexte, son équipe a développé un circuit neuromorphique basé sur le CMOL et qu'il a nommé CrossNet (Distributed Crosspoint Networks) [Turel03, Turel04].

IV. Architecture neuromorphique

Les avantages majeurs des nanoarchitectures se situent au niveau de la miniaturisation, permettant de continuer à fabriquer des circuits de plus en plus petits, et au niveau de la consommation de puissance et du coût de fabrication qui sont beaucoup plus faibles que les circuits CMOS. Cependant, l'inconvénient des ces architectures réside dans le faible rendement lié au taux élevé de défauts dû aux techniques de fabrication qui s'approchent de leurs limites. En conséquence, il faudra trouver une solution pour augmenter la fiabilité de ces architectures. Ainsi, les techniques de tolérance aux fautes deviennent le seul moyen qui permettra de réaliser d'une façon économique des circuits fiables contenant des milliards de dispositifs. Les techniques de tolérance conventionnelles utilisées déjà pour l'architecture CMOS ne

sont plus valables ici, à cause de la grande densité de défauts. Il est nécessaire de développer des nouvelles techniques innovantes de tolérances aux fautes.

L'architecture basée sur les réseaux de neurones artificiels (RNA) ou neuromorphiques est parmi les approches prometteuses qui tolèrent un grand taux de défauts. La nature distribuée du codage de l'information et leur redondance intrinsèque, à l'image de cerveau humain, contribuent à la robustesse des réseaux de neurones. Un RNA est capable de fournir un résultat correct même en présence de défauts [Emm91, Ker94, Ass97].

Dans les sections suivantes nous développerons dans un premier temps tous les aspects permettant de comprendre les architectures neuromorphiques, en commençant par la définition des réseaux de neurones artificiels, et des techniques d'apprentissage (exemples d'algorithmes d'apprentissage). Dans un deuxième temps nous présenterons l'évolution de l'implémentation des architectures neuromorphiques, depuis l'implémentation analogique classique en passant par l'implémentation VLSI (ASIC, FPGA), jusqu'à l'implémentation nanométrique à base de nanocomposants memristifs.

IV.1 Réseau de neurones artificiels

Les Réseaux de Neurones Artificiels (RNA) constituent une approche fondamentale innovante du traitement de l'information. Il s'agit de systèmes parallèles, adaptatifs et distribués dont le fonctionnement imite celui du neurone biologique.

Les RNA offrent des solutions pour une large gamme de problèmes, dont certains sont difficiles à traiter par les approches classiques. Ils servent aujourd'hui à toutes sortes d'application dans divers domaines. Parmi ceux-ci nous pouvons citer par exemple, dans le domaine de l'automobile : un système de guidage pour les véhicules intelligents; dans le domaine de l'aviation : le pilotage automatique; dans le domaine du traitement de données : le traitement du signal et des images, la vision par ordinateur ; ou finalement en économie : les modèles de prédiction par ordinateur, pour faire des prévisions monétaires sur les marchés. Ils sont aussi utilisés pour résoudre des problèmes de modélisation et des problèmes d'optimisation.

Bien que les cellules nerveuses dans le cerveau meurent régulièrement, un réseau neuronal biologique fonctionne correctement sans interruptions. Ainsi, l'un des principaux avantages des RNA par rapport à d'autres types de systèmes de traitement de l'information est leur tolérance aux fautes. Cette caractéristique mesure l'aptitude des réseaux de neurones à exécuter la tâche qui leur est demandée en présence d'informations erronées et de maintenir leur capacité de calcul même si une partie du réseau est endommagée. En effet, le traitement réalisé par le RNA est distribué sur tout le réseau où chaque neurone contribue à l'établissement de la réponse finale. Par conséquent, en présence de dégradations (permanentes ou

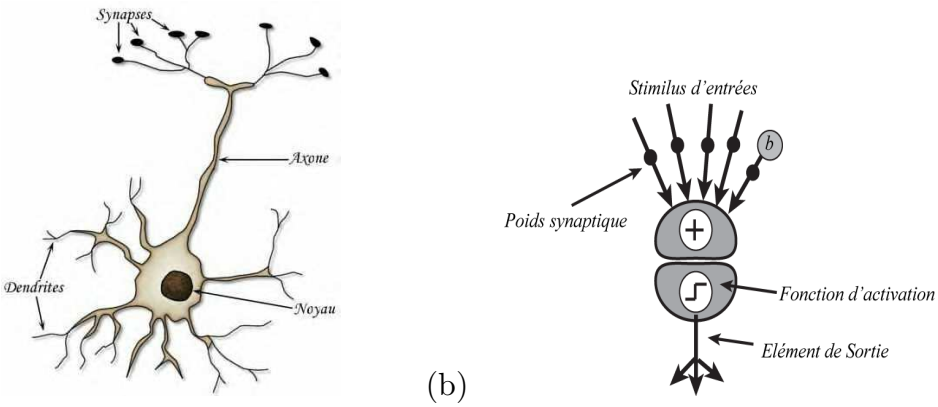
transitoires), les fautes sont supportées par tous les neurones du réseau au lieu de l'être par le neurone concerné, conduisant ainsi à une dégradation progressive plutôt qu'à une défaillance catastrophique. Cette robustesse intrinsèque est une caractéristique qui fait des RNA l'une des approche prometteuses pour remédier aux problèmes de défauts et de variations dans les architectures à nanocomposants. Le chapitre III est entièrement consacré à l'étude de la tolérance aux défauts et variations technologiques dans une architecture neuromorphique.

IV.2 Structure de base

Les neurones biologiques sont des cellules du système nerveux permettant la transmission et le traitement de l'information (influx nerveux). Par exemple pour une douleur transmise de la main vers le système nerveux, le traitement neuronal va permettre de donner une réponse (transmise en sens inverse) à la main pour bouger (le réflexe). Un cerveau humain est composé de plus de cent milliards de neurones qui communiquent entre eux par l'intermédiaire de centaine de milliers de connexions appelées **synapses** (figure I.6 (a)) (voir plus de détails section IV.5). Ces neurones s'organisent en réseaux pour traiter d'une manière adaptée les stimuli provenant de l'extérieur ou depuis les neurones voisins. Grâce à ces réseaux de neurones le cerveau s'adapte progressivement et mémorise des situations qui lui permettre de répondre à des situations de plus en plus complexes.

Comme le montre la figure I.6 (a) un neurone biologique est constitué d'un **corps cellulaire** qui contient un mécanisme de déclenchement dont la fonction est de sommer les excitations reçues, et de produire un influx nerveux dans le cas ou la somme dépasse un certain seuil (potentiel d'action). Dans le cas contraire aucun influx ne sera produit : la réponse est en tout ou rien. L'influx est transmis tout au long de l'**axone** jusqu'aux **dendrites** (ramifications terminales) pour exciter d'autres neurones à travers des synapses [Saig04].

En 1943 *McCulloch et Pitts* ont introduit le premier modèle de neurone artificiel : le **Perceptron** (figure I.6 (b)). Leur idée de base était de construire un modèle de calcul mathématique inspiré d'un réseau neuronal biologique simplifié. Le perceptron est composé d'un seul neurone qui représente l'unité élémentaire de traitement de l'information dans un RNA. Chaque neurone est composé d'une seule sortie Y_j et d'un nombre variable d'entrées X_i. Chacune des entrées est reliée à la sortie par un poids synaptique w_{ij} adaptatif qui représente la force de la connexion, et qui permet de stocker la « connaissance ». La valeur numérique du poids w_{ij} est ajustée dans une phase d'apprentissage que nous détaillons dans la partie suivante. Dans un réseau de neurone chaque neurone reçoit l'information en provenance de neurones amonts et calcule la sortie. Le résultat est diffusé aux neurones avals.



(a) (b)

Figure I.6 : (a) schéma d'un neurone biologique comparé au (b) schéma d'un neurone artificiel (perceptron).

Le modèle biologique du neurone fait intervenir une notion temporelle, dans le mécanisme de déclenchement, qui n'est pas prise en compte dans le calcul de la sortie du modèle du Perceptron. Cette notion est remplacée par une simple sommation pondérée des entrées reçues (Eq. I.1).

$$V_j = \sum_{i=1}^{N} X_i w_{ij} + b \qquad \text{(I.1)}$$

b représente le seuil par analogie au seuil du déclenchement du neurone biologique. Sa valeur peut être associée à la connexion à une pseudo-entrée appelée " bias" dont l'état d'entrée reste constant et vaut toujours 1. La valeur de ce seuil peut s'ajuster en modifiant son poids synaptique durant la phase d'apprentissage. V_j est le potentiel post-synaptique. Pour calculer la sortie finale x_j une fonction d'activation (ou de transfert) f est appliquée au potentiel post-synaptique. Le choix de la forme de cette fonction dépend du modèle et du codage des états d'activité des neurones. Par exemple, dans le cas des neurones avec des états d'activité binaires (+1/-1), il peut s'agir d'un seuillage par la fonction signe (ou Heaviside pour les neurones binaires (0/1)).

$$x_j = Signe(V_j) \qquad \text{(I.2)}$$

Lorsque les états d'activité des neurones ne sont plus binaires, la fonction d'activation est souvent sigmoïdale.

$$x_j = \frac{1}{1 + \exp(-V_j)} \qquad \text{(I.3)}$$

IV.3 Techniques d'apprentissage

L'apprentissage est la propriété la plus intéressante des RNA car elle permet de réaliser avec succès la tâche qui leur est demandée. Cette propriété permet aux RNA

d'apprendre de leur environnement et d'améliorer leur performance afin d'atteindre le comportement désiré. La phase d'apprentissage se fait à travers un processus itératif d'ajustement des poids synaptiques w_{ij}. Les changements successifs qui vont intervenir au sein de la structure interne du RNA vont lui permettre d'adapter progressivement sa réponse aux stimuli qu'il reçoit de son environnement. A l'issu de cette phase d'apprentissage, les poids synaptique sont fixés : c'est alors la phase d'utilisation, où le RNA pourra être utilisé pour répondre à des stimuli d'entrée inconnus c'est-à-dire non rencontrés durant l'apprentissage. Cette faculté de généralisation donne aux RNA une robustesse intrinsèque face aux fautes.
De manière très générale, le changement de poids $\Delta w_{ij}(t)$ peut s'exprimer de la façon suivante :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w(t) \qquad \text{(I.4)}$$

w_{ij}(t+1) et w_{ij}(t) représentent respectivement la nouvelle et l'ancienne valeur du poids.

Les règles qui permettent d'adapter les poids synaptiques sont organisées dans un processus itératif dynamique ce qu'on appelle *algorithme d'apprentissage*. Il existe deux grandes classes d'algorithme selon que l'apprentissage est supervisé ou non supervisé.

IV.3.1 Apprentissage supervisé

L'apprentissage supervisé illustré dans la figure I.7 est caractérisé par la présence d'un «professeur» qui possède une connaissance de la fonction à faire réaliser par le réseau de neurones. En pratique, le rôle de l'expert est de fournir un ensemble de couples (entrée, sortie désirée) qu'on appelle *base d'apprentissage*. A chaque pas d'apprentissage les poids sont modifiés selon un algorithme donné pour minimiser une fonction d'erreur, de façon à obtenir la réponse désirée pour les entrées de la base d'apprentissage. Parmi les algorithmes les plus répandus, nous pouvons citer la *règle de Delta* [Wid90] et la *rétro-propagation du gradient* [Rum86], ces deux algorithmes sont décrits dans la partie algorithmes d'apprentissage (section III.4).

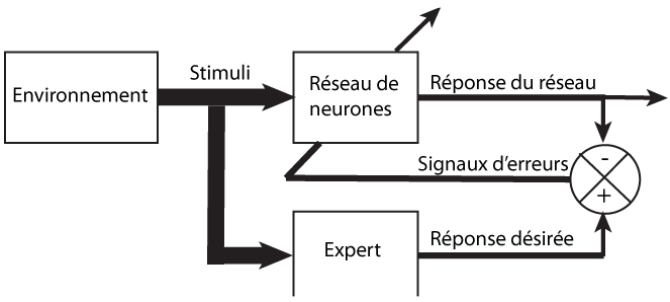


Figure I.7 : Illustration d'apprentissage supervisé

IV.3.2 Apprentissage non supervisé

L'apprentissage non supervisé ou compétitif est aussi appelé auto-organisation [Koho90]. Contrairement à l'apprentissage supervisé qui nécessite la connaissance des sorties désirées, ce type d'apprentissage n'a pas besoin de professeur pour apprendre, comme le montre la figure I.8. Le RNA ne dispose donc que d'un stimulus d'entrée et doit apprendre sans intervention externe (signal d'erreur ou indice de satisfaction), en assimilant ce stimulus à une représentation interne permettant d'encoder ses caractéristiques. Ce type d'apprentissage est fondé généralement sur un processus compétitif, où tous les neurones du réseau apprennent simultanément et de la même façon le stimulus d'entrée; seul le « vainqueur » bénéficiera de l'adaptation de poids synaptique. Ce principe est inspiré de la loi de *Hebb* [Heb50] où seule la connexion reliant deux neurones activés en même temps est renforcée. De cette façon, à chaque présentation de stimuli, le réseau adapte ses poids en renforçant l'efficacité des synapses qui relient des neurones activés en même temps et réduit les autres. Ce type d'apprentissage est utilisé pour élaborer une représentation interne de la connaissance de l'environnement, identifiant une structure simple ou plus explicite de stimuli (analyser les relations entre les variables, clustering, réduction de dimension...), comme dans les cartes de *Kohonen* pour la reconnaissance de forme [Koho90].

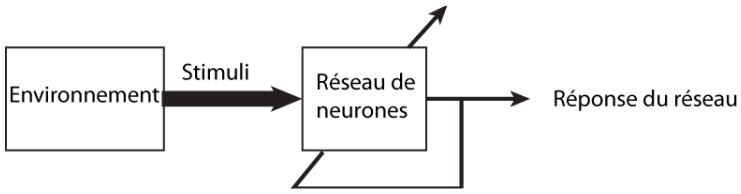


Figure I.8 : Illustration de l'apprentissage non supervisé

IV.3.3 Apprentissage temporel : STDP (Spike-Timing-Dependent Plasticity)

STDP est un modèle d'apprentissage permettant l'émergence de dynamiques plus proches de celles mesurées en biologie sur la plasticité des synapses [BiPo98, Froe02, Levy83] (voir figure I.9).

L'apprentissage par STDP est une forme temporelle asymétrique de l'apprentissage Hebbien. Il se base généralement sur un apprentissage non-supervisé. L'apprentissage dépend comme dans le système biologique du décalage entre l'arrivé des potentiels d'action (spike) pré- et post-synaptiques. Quand le signal post-synaptique atteint la synapse avant le potentiel pré-synaptique, la synapse montre une dépression à long terme (LTD) : la connexion entre les deux neurones s'affaiblit (diminution du poids synpatique) en proportion inverse du temps qui sépare les deux signaux pré- et post-synaptique (i.e. plus le délai entre les deux signaux est court, plus la variation est importante). Inversement, si le potentiel d'action post-synaptique atteint la synapse après le potentiel d'action pré-synaptique, le poids de la synapse

subit une potentiation à long terme (LTP) : la connexion entre les deux neurones augmente en proportion inverse du délai entre les signaux pré- et post-synaptique.

Soit $W_{pré\_post}$ le poids synaptique de la connexion reliant le neurone pré-synaptique au neurone post-synaptique. Si le neurone pré-synaptique émet un potentiel d'action au temps $t_{pré}$ et le neurone post-synaptique émet un autre potentiel d'action au temps t_{post}, alors le délai entre les deux potentiels d'action est donné par : $\Delta t = t_{post} - t_{pré}$. Selon le signe de Δt il y'aura une LTP ou LTD :

- Potentiation ($\Delta t > 0$) ou dépression ($\Delta t < 0$) : $W_{ij}(t+1) = W_{ij}(t) + k/\Delta t$.

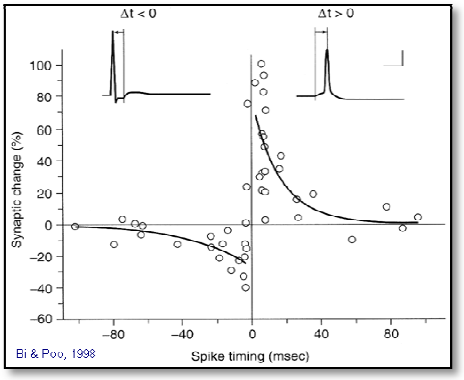
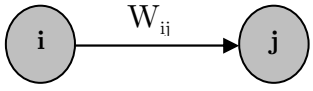


Figure I.9 : Illustration du mécanisme de la STDP montrant le pourcentage de modification de l'efficacité synaptique en fonction du délai entre le potentiel d'action émit par les neurones pré- et post synaptiques.

IV.4 Algorithmes d'apprentissage

La plupart des algorithmes d'apprentissage peuvent être considéré comme des variations de la **règle d'apprentissage hebbien** décrit par *Hebb* dans son livre *Organization of Behaviour*, 1949 [Heb50], où la force de la connexion entre deux neurones augment si il ya corrélation d'activité (c'est-à-dire si deux neurones sont activés en même temps). Si on considère la synapse reliant la sortie du neurone i, dont l'activité est x_i, à l'entrée du neurone j d'activité x_j, alors la règle de *Hebb* exprime la variation de poids synaptique entre le neurone i et j en fonction de la corrélation entre x_i et x_j.



$$\Delta W_{ij} = \alpha x_i x_j \qquad \text{(I.5)}$$

où α est une constant positive qui représente le pas de l'apprentissage.

Une autre règle d'apprentissage que nous trouvons dans plusieurs algorithmes d'apprentissage est basée sur l'actualisation du poids synaptique en fonction de la

corrélation entre l'activité de l'entrée x_i et en fonction de l'erreur mesurée sur la sortie, définie comme la différence entre la sortie actuelle x_j avec la sortie désirée y_j du neurone j.

$$\Delta W_{ij} = \alpha x_i (y_j - x_j) \qquad \text{(I.6)}$$

Cette règle consiste à appliquer la règle de *Hebb* pour les synapses associées aux neurones en erreur (y_j-x_j≠0) et ne rien faire pour les autres. Cette règle est connue sous le nom de ses fondateurs *Widrow et Hoff* en 1960 [Wid60], ou sous le nom de la **règle de Delta** (Adaline) ou encore la règle de LMS (Least mean square).

Les algorithmes basés sur les deux règles d'apprentissage décrites précédemment sont très simples à implémenter, ils ne demandent qu'un simple test de la sortie du neurone pour calculer l'erreur et pour mettre à jour le poids synaptique dans le cas où l'erreur n'est pas nulle (Eq. I.6). Le mécanisme de l'algorithme est illustré par la figure I.10 et les séquences décrites ci-dessous.

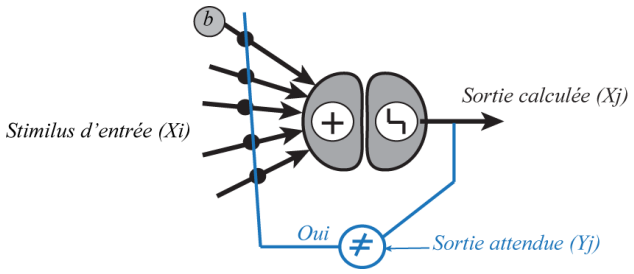


Figure I.10 : Mécanisme d'apprentissage de perceptron

Nous disposons d'un ensemble X de K vecteurs d'entrées de taille I : $x^k = \left\{x_0^k, ..., x_i^k, ...x_{I-1}^k\right\}$ avec $k \in \{0, ...k-1\}$, et d'un ensemble Y de K vecteurs de sortie de taille J : $y^k = \left\{y_0^k, ..., y_j^k, ...x_{J-1}^k\right\}$ dont les composants y_j^k sont binaires. Les ensembles X et Y forment la base d'apprentissage.

L'apprentissage de cette base en utilisant un perceptron simple doit suivre les séquences suivantes :

1. Initialisation des poids synaptiques à des valeurs aléatoires, et du pas d'apprentissage α.
2. Sélection d'un vecteur d'entrée x^k de la base d'apprentissage.
3. Calcul de la sortie des neurones y_j^k selon l'Eq. I. 1.
4. Si la sortie calculée est différente de la sortie désirée, alors :
   - Calculer la correction des poids Δw_{ij} selon l'Eq. I.5 ou I.6.
   - mettre à jour de poids w_{ij} selon l'Eq. I.4.
5. Retour à l'étape 2 tant que l'erreur calculée sur l'ensemble de la base d'apprentissage est supérieure à un seuil.

Les travaux de Warren *McCulloch et Walter Pitts* dans les années 1940 [McC43] ont montré théoriquement qu'un RNA est capable de réaliser n'importe quelle fonction arithmétique ou logique (linéairement séparable). Vers la fin des années 1950 *Frank Rosenblatt* et son équipe de recherche ont réalisé un perceptron capable de reconnaitre des formes [Ros62]. Cependant l'arrivé de livre de *Marvin Minsky et Seymour Papert* [Min69] publié vers la fin des années 1960 a démontré les limites de Perceptron, notamment l'impossibilité de traiter des problèmes non linéairement séparables. En effet, un simple Perceptron ne peut apprendre à distinguer deux classes qu'à la condition que les vecteurs d'entrée associés à l'une des classes puissent être séparés des vecteurs d'entrée de l'autre classe par un hyperplan. Un exemple est donné sur la figure I.11 (a) et (b) pour l'apprentissage des fonctions logiques OR et AND une seule droite de séparation suffit. Cependant un simple Perceptron est incapable de classer correctement l'ensemble des points correspondants à la fonction XOR qui n'est pas linéairement séparable (figure I.11 (c)). Néanmoins, il est possible d'apprendre la fonction XOR avec plusieurs Perceptrons. Cela a été démontré à la fin des années 1980 avec l'apparition de réseaux multicouches (MLP) et l'algorithme de **rétro-propagation des erreurs** (Back-propagation) inventé simultanément par *David Rumelhart et James McClelland* et par *Paul Werbos* et par *Yann LeCun*. Ils ont montré qu'avec un tel réseau, nous pouvons traiter n'importe quel problème non linéairement séparable dont l'exemple le plus élémentaire est le ou-exclusif (XOR) [Rum86]. Comme le montre la figure I.11 (d), l'ajout d'un neurone caché entre l'entrée et la sortie a résolu ce problème.

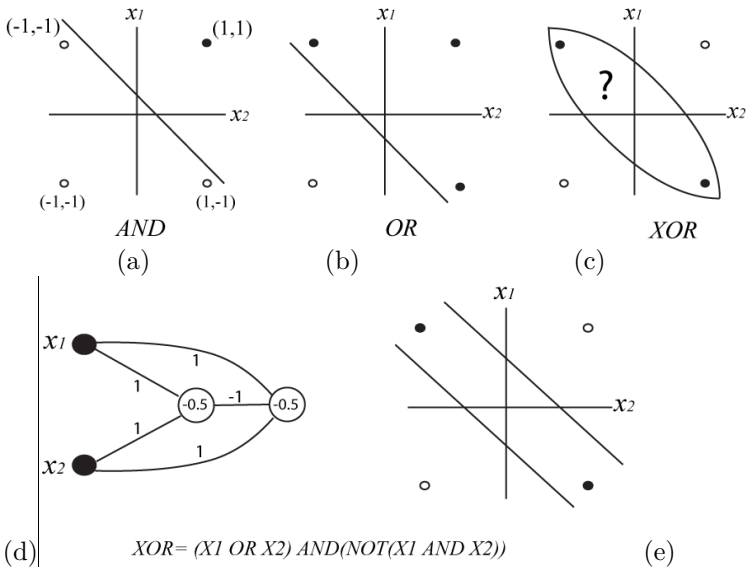


Figure I.11 : (a) Séparation de l'espace des entrées en deux classes par une droite pour la fonction AND, (b) pour la fonction OR, (c) incapacité d'un Perceptron simple à séparer deux classes d'une fonction XOR, (d) la solution pour le problème de XOR consiste à ajouter un neurone caché entre la couche d'entrée et la couche de sortie, les valeurs numériques indiquent les poids synaptiques W_{ij}, et le seuil est indiqué avec la valeur dans le cercle, (e) séparation des quatre points de la fonction xor avec deux droites.

La difficulté pour l'apprentissage d'un tel type de réseau réside dans le calcul de l'erreur associée aux neurones cachés, car la sortie désirée pour ces neurones n'est pas connue. La rétro-propagation de l'erreur permet de résoudre ce problème en calculant l'erreur des neurones cachés à partir de l'erreur des neurones de sortie. L'erreur se propage donc de la sortie vers l'entrée (figure I.12), d'où son nom.

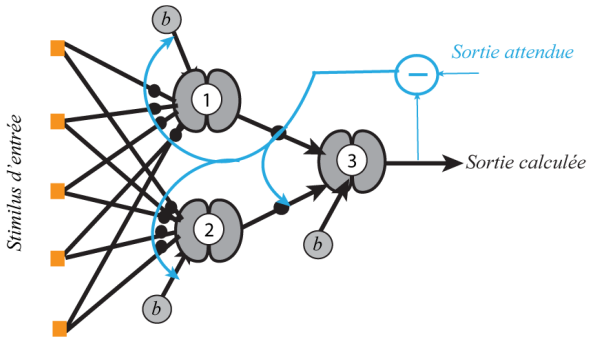


Figure I.12 : Apprentissage d'un perceptron multicouche par rétro-propagation du gradient

La rétro-propagation des erreurs (gradient) consiste à trouver une configuration de poids qui minimise un critère d'erreur. On définit donc une fonction de coût :

$$C(w) == \left[\sum_j e_{kj}^2(w)\right], \quad avec \quad e_{kj} = (y_j - x_j) \qquad (I.7)$$

où, j est l'indice pour les neurones de sortie et k indique un exemple d'apprentissage.

On réalise donc la somme quadratique des erreurs obtenues pour chacun des exemples de la base d'apprentissage.

L'algorithme du gradient permet de trouver une configuration minimale pour la fonction de coût. Pour cela, le poids synaptique est ajusté avec Δw_{ij} d'une manière itérative suivant la formule :

$$\Delta w_{ij} = -\alpha \frac{\partial C(w(t))}{\partial w_{ij}} \qquad (I.8)$$

où ∂C(t) est la dérivée partielle de C par rapport à tous les wij.

Cette règle s'applique à des réseaux multicouches avec des états de neurones continus, et la fonction d'activation est souvent une sigmoïde ($f(x) = \frac{1}{1+e^{-x}}$) qui présente l'avantage d'être particulièrement simple à dériver : $f(x)' = f(x).[1 - f(x)]$.

IV.5 Implémentation des réseaux de neurones

IV.5.1 Implémentation : Analogique, ASIC, FPGA

Les circuits électroniques simulant des réseaux de neurones sont appelés systèmes neuromorphiques ou encore neuromimétiques car ils miment le

fonctionnement des neurones. Il y a eu un intérêt continu dans le développement de ce type de circuits dans le but de transférer les capacités de calcul cognitif de système biologique sur des puces de silicium, en utilisant la technologie CMOS en particulier. Dans ce contexte, les premiers travaux ont été marqués par la réalisation d'un premier modèle de rétine sur une puce de silicium par l'équipe de *Carver Mead* [Mead89] en 1989 et les travaux de l'équipe de *Patrick Garda* sur les neurocomputer analogiques pour les machines synchrones de *Boltzmann* [Pujo94]. Depuis, l'ingénierie neuromorphique n'a cessé de créer des nouvelles implémentations de systèmes biologiques. Selon *Hammerstrom* [Ham08] il y a trois catégories d'implémentation des RNA:

- Circuits analogiques VLSI (very-large system intégration): comme les RNA sont basés sur des calculs analogiques et ils sont peu sensibles au bruit, ils offrent la possibilité d'être implémentés avec des composants analogiques [Cavi94, Bo98, Bose91, Bel92]. L'utilisation de la technologie VLSI permet d'intégrer une grande densité de synapses et de neurones dans la même puce et par conséquence d'avoir un circuit neuromorphique performant.
- Circuit numérique avec un fonctionnement parallèle spécialisé pour des applications de calcul neuronal. Différentes puces neuronales spécialisées ont été réalisées, tels que la CNAPS [ham90], le SYNAPSE-1 [Ram93] et l'ETANN [holl89].
- Circuits DSP (Digital Signal Processing) utilisés pour le traitement de données multimédia. Les opérateurs de calcul de multiplication accumulation de ce type de circuits sont optimisés, ce qui facilite l'implémentation des réseaux de neurones dans ces derniers.

D'autres réalisations ont suivi les travaux de *Carver Mead.* Dans [Hasl06] par exemple, il a été démontré que 1000 neurones ont pu être intégrés dans la même puce en utilisant la technologie 0.35µm. En utilisant la même technologie un neurone a été intégré avec ses buffers dans une puce de 2800µm^2 [Wij08]. Un autre circuit est réalisé avec la technologie 0.25µm dans [Hyn07], contenant 7200 neurones implémentés dans une surface de 11.5mm^2. La majorité des réalisations que nous venons de citer souffrent du manque de parallélisme et de la faible densité d'intégration pourtant nécessaire à l'exploitation efficace des RNA. L'apparition des circuits dédiés à des applications spécifiques (ASIC : Application Specific Integrated Circuit), a permis d'implémenter les RNA avec une performance accrue par rapport aux implémentations analogiques, comme l'amélioration du temps de calcul, parallélisme, taille et consommation [Ruc02, Bud02, Dan02]. Parmi les meilleures réalisations dans ce domaine, nous pouvons citer le neuro-ordinateur *Neuricam Totem NC3001* [DaSh, Lind96] qui utilise 16 bits pour les entrées, 8 bits pour coder les poids synaptiques et 32 bits pour les sorties. Une comparaison des capacités de calcul de *Neuricam* avec un réseau de neurone simulé sur Matlab avec une station Intel (microprocesseur Pentium II de 400-MHz, une mémoire de 128 Mo) à été effectuée dans [Dan02] montrant que le

Neuricam est trois fois plus performant. Ce résultat de comparaison ne tarde pas à s'éteindre avec l'évolution rapide des microprocesseurs. L'implémentation des RNA à base des circuits dédiés ont montré quelques inconvénients et limitations (la majorité étant rencontrée déjà dans le cas des circuits analogiques) [Girau06]:

1. Manque de souplesse et d'adaptabilité : une configuration matérielle figée de modèle RNA, où les fonctions sont gravées dans le silicium, ne permet pas de changer rapidement les modèles utilisés comme le permet la configuration logicielle.
2. Les coûts et temps de développement ne cessent d'augmenter avec les technologies de plus en plus fines.
3. Une phase de test est nécessaire après fabrication de circuit car des défauts matériels non prévus peuvent apparaitre.

Malgré toutes ces limitations, l'implantation des RNA sur des puces de silicium (analogiques ou numériques) reste utile pour certaines applications. La plupart des défauts cités précédemment peuvent être contournés en utilisant des circuits reconfigurables comme les FPGA (*Field Programmable Gate Arrays*). Les FPGA utilisent des blocs reconfigurables (*CLB*) hautement interconnectés pouvant donc s'adapter aux applications visées, sans être profondément modifiés contrairement aux implantations VLSI. L'implantation des RNA avec ce type d'architecture offre beaucoup d'avantages par rapport aux ASIC ou aux circuits analogiques [Torr06, Girau00] ; les principaux sont cités ci-dessous :

1. Le coût et le temps de développement des circuits reconfigurables est faible par rapport à celui des circuits dédiés ;
2. Un prototypage direct est possible par programmation de circuit ;
3. La possibilité d'implanter dans le même circuit, l'application souhaitée et d'autres phases de calcul supplémentaires pour une application embarquée ;
4. La rapidité et l'accessibilité pour les non-experts des circuits intégrés.

Ces avantages ont permis à beaucoup d'équipes de recherche de s'orienter vers cette piste dans le but d'aboutir à des circuits plus puissants, tels que l'implantation de neuro-processeurs sur FPGA [Gsch96, Ros96, Gros07], l'implantation de systèmes dont la structure est plus proche d'un véritable neuro-ordinateur [Ken97, Klr07], l'implantation de multiplieurs rapides [Sala94] et des implantations qui exploitent le parallélisme des RNA [Boum05, Sma06].

Malgré toutes ces implantations et les avantages cités plus haut, les FPGA sont eux aussi confronté à quelques obstacles qui limitent leur utilisation comme :

1. La structure de communication des FPGA est hautement complexe, grâce à l'utilisation des couches métal multiples. Néanmoins, une telle topologie reste modeste devant le graphe de connexion des réseaux de neurones. Le

nombre de liens d'entrée parallèles est élevé dans la plupart des modèles, et l'implantation de ces liens dans un FPGA oblige à mobiliser certaines *CLB* afin d'augmenter les capacités de routage, ce qui se traduit par une perte de ressources de calcul.

2. Quelques applications de réseau de neurones demande une grande précision ce qui demande des opérateurs coûteux en surface.
3. Les poids synaptiques sont codés et stockés dans les *LUT* (Look Up Table). Le FPGA ne peut pas assurer ce stockage de poids pour des RNA de grande taille, ce qui demande une gestion des accès mémoire. Ce problème est résolu partiellement dans les nouveaux FPGA grâce aux blocs mémoires qu'ils contiennent.

IV.5.2 Vers une implémentation nanométrique des RNA

Les FPGA ont bénéficié de plusieurs progrès en taille et en vitesse, dû aux progrès de la microélectronique et la miniaturisation qui permet d'intégrer de plus en plus de composants dans le même circuit. De multiples ressources de calcul sont implantées dans le même circuit, telles que des multiplieurs, des blocs mémoire et des DSP. Cette évolution a permis des gains importants en performance. Néanmoins cela ne règle pas les problèmes des FPGA cités auparavant, le problème ne provenant pas tant de l'architecture des FPGA que de l'approche connexionniste qui doit prendre un nouveau relief, tel que le calcul distribué par exemple. Dans ce contexte, les recherches s'orientent vers le connexionnisme bio-inspiré [Torr06] et l'étude de phénomènes complexes d'émergence [berr06].

Deux idées pertinentes peuvent se présenter actuellement pour l'amélioration des implantations basées sur les architectures FPGA:

La première idée consiste à changer notre façon de penser et de passer de la question « comment implémenter un algorithme à partir d'un ensemble de ressources logiques » à la question « comment faire coopérer un ensemble d'unités de calcul pour résoudre une tâche donnée ». L'exploitation des avancés de la nanotechnologie (*Tour : Nanocell, André dehon : NanoPLA, Goldstein : MLA, Likharev : CMOL*) [Tour03, Chri03, Deh03, Gold01, Likha05] peut aider à surmonter cette question en intégrant de plus en plus d'unités de calcul dans la même puce et en réglant les différents problèmes d'interconnexions. Cependant, le taux de défauts et de variabilité élevée présents dans ces nouvelles architectures nécessitent encore des efforts pour développer des techniques de test et de tolérance aux défauts permettant de surmonter cette contrainte [Niko02, Chab10].

La deuxième idée est de contourner le problème de codage des poids synaptiques, qui demande de plus en plus de mémoire dans les FPGA, par l'implémentation directe des synapses grâce à des composants multi-niveaux [Gil07, Li04, Kund05]. Au lieu de chercher à implémenter les RNA à partir d'un circuit FPGA composé de portes logiques, nous allons utiliser les RNA pour réaliser un

FPGA, en remplaçant les CLB par des petits RNA qui vont s'interconnecter entre eux pour construire un réseau complexe. Cela va permettre d'exploiter les propriétés des RNA, comme leur grande puissance de calcul, ainsi que leur robustesse intrinsèque qui va permettre de régler les problèmes de défaut et des variations de caractéristiques.

IV.5.3 Implémentation de la Synapse

IV.5.3.1 Plasticité de la Synapse biologique

Evoqué précédemment dans la section d'introduction aux réseaux de neurones, un neurone est constitué d'une arborescence dendritique, qui reçoit les informations provenant d'autres neurones. Le neurone « calcule » : par sa morphologie, ses propriétés membranaires, il traite et transmet l'information vers d'autres neurones ou vers des cellules motrices, par exemple pour contrôler une activité musculaire. La transmission de l'information se fait grâce à l'axone et par l'intermédiaire des synapses qui représentent le point de transfert de l'information entre les terminaisons axonales (pré-synaptique) et les cellules cibles (post-synaptique) (figure I.13). L'échange entre la membrane pré et post-synaptique peut se faire par le biais de canaux de chacune de ces membranes. Sous certaines conditions (potentiel suffisant) la membrane pré-synaptique libère les substances chimiques (neurotransmetteurs) qu'elle contient dans la fente synaptique. Ces substances chimiques vont agir sur la membrane post-synaptique qui provoque une dépolarisation de la membrane de la cellule cible. Cette dépolarisation est supportée par des courants électrochimiques proportionnels à la différence de potentiel entre les milieux intracellulaires des deux cellules pré et post-synaptiques. Ces courants peuvent être entrants (post-synaptique dépolarisé) ou sortants (post-synaptique hyperpolarisé) selon que la synapse soit excitatrice ou inhibitrice. Cela nous montre que les synapses ne font pas que la transmission d'influx nerveux mais elles ont aussi la capacité de le moduler, c'est ce que nous appelons la plasticité synaptique [Masa94]. Une synapse peut donc augmenter (potentiation) ou diminuer (dépression) son efficacité. Cette efficacité va jouer un rôle sur la façon dont le réseau traite l'information.

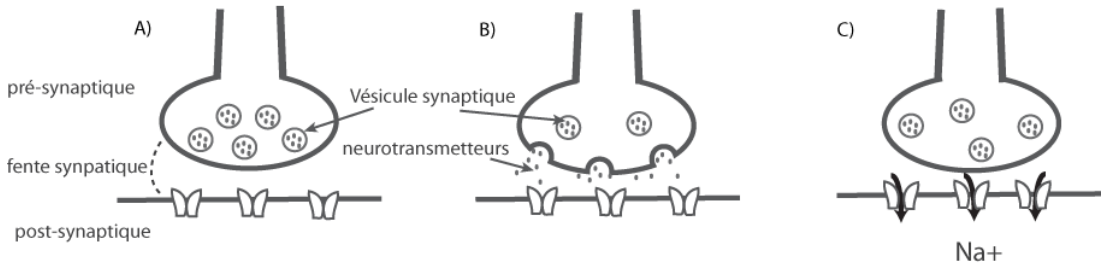


Figure I.13 : Synapse électrochimique. A) un potentiel d'action arrive a travers l'axone à la cellule pré-synaptique. B) il provoque la libération des neurotransmetteurs dans la fente synaptique. C) les neurotransmetteurs provoque l'ouverture des récepteurs post-synaptique et donc une dépolarisation de la synapse.

IV.5.3.2 Conditions sur l'implémentation électronique de la Synapse

La synapse peut être vue comme une conductance électrique qui relie deux neurones, ce qui rend son implémentation électronique possible mais sous certaines conditions. Voici quelques conditions à remplir pour implémenter la synapse :

1. Le poids synaptique doit être mémorisé en permanence y compris en l'absence d'apprentissage, et ne doit pas changer pendant la phase de fonctionnement;
2. Le composant qui implémente la synapse doit respecter la plasticité de la synapse : multi-niveau de conductance (ou résistance);
3. La sortie de la synapse doit pouvoir évaluer le calcul de la somme pondérée des signaux d'entrée avec les poids synaptiques;
4. Chaque synapse doit occuper un minimum de surface pour un maximum d'intégration;
5. Chaque synapse doit fonctionner avec une faible consommation pour ne pas poser de contraintes de puissance consommée par le circuit dédié à l'intégration de ces synapses;
6. Le circuit doit pouvoir implémenter l'un des algorithmes d'apprentissage Hebbien ou rétro-propagation.

IV.5.3.3 Implémentation à base de CMOS

L'idée de l'implémentation des synapses n'est pas nouvelle, elle a déjà été développée dans la thèse de *C. Chao* [Chao90] dans le début des années 1990, et plusieurs autres tentatives d'implémentations ont été réalisées à l'époque en utilisant la technologie CMOS. Nous citons le modèle du transistor à grille flottante [Bilo66, Dior02] réalisé par *Hasler et Diorio* [Hasl95, Dior96] avec des caractéristiques biomimétiques. Huit transistors CMOS ont été proposés dans [Shi03] pour implémenter une synapse. La conception d'une synapse analogique a été étudiée aussi par *Pinto et al* [Pint00] et Lee et al [Lee05]. *Volkovskii* a proposé une synapse à *PLL* (phase-lock loop) dans [Volk05]. Tous les travaux que nous venons de citer ont point commun : la synapse est réalisée à partir de plusieurs transistors CMOS ce qui limite la densité d'intégration.

IV.5.3.4 Implémentation à base de nanocomposants

Les circuits électroniques d'aujourd'hui peuvent fonctionner à des fréquences de l'ordre du Ghz qui dépassent largement celles mesurées dans les systèmes biologiques, qui restent de l'ordre de la dizaine de Hz. Malgré cette rapidité obtenue avec des systèmes de plus en plus sophistiqués, nous n'arrivons pas à réaliser des systèmes artificiels dotés des capacités de traitements équivalentes à celles de notre système nerveux. Nous avons vu précédemment que le défi que doit surmonter les FPGA par exemple n'était pas lié à la vitesse de fonctionnement mais surtout à la complexité des connexions entre les neurones dans la plupart des modèles d'ANN. Les connexions

synaptiques sont considérées comme la partie la plus complexe dans la modélisation neuro-mimétique. Le cerveau humain fonctionne avec 100 milliards de neurones et chaque neurone possède de 10 000 jusqu'à 100 000 connexions synaptiques [haw04, She04]. Avec une telle densité la connectivité entre les neurones sera un défi majeur. L'apparition des nouveaux nanocomposants devrait améliorer ce problème. La dernière avancée de l'intégration nanométrique montre que nous pouvons atteindre des grandes densités d'intégration, estimée à 10^{10} nanocomposants/cm^2 pour les memristors réalisés à l'université *Michigan* [Jo09]. Cela est très prometteur pour réaliser des architectures neuromorphiques à très haute densité d'intégration. Plusieurs nanocomposants multi-niveaux permettent d'implémenter les synapses. Les nouveaux composants et systèmes memristifs développés actuellement par plusieurs équipes de recherche sont parmi les plus prometteurs. Ces systèmes seront développés dans la section suivante.

De grands projets ont été lancés dernièrement dans la modélisation du cerveau et l'implémentation des réseaux de neurones avec les technologies émergentes. Parmi ces grands projets, on peut citer celui du cerveau artificiel lancé en 2005 par *IBM* en collaboration avec *EPFL* (Ecole Polytechnique Fédérale de Lausanne) le « *Bluebrain project* » [Brmin]. L'objectif principal de ce projet est de reproduire de manière la plus réaliste possible avec une approche neuro-mimétique, la structure du cerveau humain. Le projet *Facets « Fast analog computing with emergent transient states »* (2005-2010) [Fact] est le plus grand consortium intégré Européen dans le domaine du neuromorphique. Par rapport à l'approche développée dans Blue Brain, qui était plutôt du type neuromimétique, la stratégie de *Facets* est de s'appuyer sur les dernières avancées de la technologie d'intégration VLSI, pour construire des architectures de calcul neuromorphiques qui s'inspirent du vivant sans chercher à recréer un cerveau. Le projet *DARPA SyNAPSE (IBM, HP, HRL)* lancé en 2009 est également basé aussi sur le développement d'un cerveau électronique imitant le cortex biologique.

V. Nanocomposants

Les difficultés rencontrées pour fabriquer les transistors CMOS de petite tailles, la hausse des coûts de production des puces, l'augmentation de la puissance dissipée et d'autres problèmes que rencontre actuellement l'industrie des semi-conducteurs sont les raisons qui vont mettre fin à la technologie CMOS pour la prochaine décennie [ITRS07].

Afin de résoudre ces problèmes, les chercheurs ont commencé à explorer des nouveaux dispositifs comme des compléments au CMOS pour certains, voire de remplacer le CMOS pour d'autres. Ces nanocomposants offrent la possibilité d'atteindre de plus hauts niveaux de miniaturisation avec des coûts de fabrication très faibles [Tchen06].

Il existe une grande variété de nanocomposants développés dans le cadre des domaines « More than Moore » et « Beyond CMOS ». Le lecteur pourra consultés la publication [Har09] pour voir les différentes classes de nanocomposants développés actuellement.

Dans la suite de cette section, nous allons nous concentré sur l'une de ces variétés : les nanocomposants memristifs. Nous présenterons le fonctionnement de base d'un memristor idéal, pour présenter ensuite ce qu'est un système memristif en général et quelles sont leurs principales applications.

V.1 Memristor

Imaginé pour la première fois en 1971 par le professeur *Leon Chua*, le memristor représente le quatrième élément passif de base en électronique, aux cotés de la résistance (R), du condensateur (C) et de l'inductance (L) [Chua71]. Selon le professeur *Chua* ce composant va permettre de compléter la symétrie conceptuelle, qui relie les deux variables fondamentales de l'électronique (la tension V, le courant I) et leur intégrale (la charge q et le flux φ) par les composants (R, L, C, M) (Figure I.14 (b)). Il définit ainsi la relation manquante M qui relie le flux φ et la charge q.

Le mot memristor vient de la composition des deux mots anglais *memory* et *resistor* car le memristor est considéré comme une résistance variable (généralement non linéaire) avec un effet mémoire. Il est considéré comme un cas particulier des systèmes memristifs [Chua76] (voir prochaine section). Un memristor idéal est un système memristif dont la variable memristance M ne dépend que de la charge totale q qu'il a traversé (figure I.15).

D'une manière simplifiée, pour bien comprendre le fonctionnement d'un memristor, nous pouvons faire l'analogie avec un tuyau qui se dilate ou se contracte lorsqu'une certaine quantité d'eau (courant) le traverse (figure I.14 (a)). Le diamètre du tuyau (conductance) augmente quand l'eau le traverse dans un sens, permettant ainsi à l'eau de le traverser plus rapidement, et diminue quand l'eau le traverse dans le sens inverse. Si la pression de l'eau diminue ou s'arrête (coupure de l'alimentation), le tuyau conserve son dernier diamètre (mémorisation).

Le memristor est représenté par sa memristance M, qui exprime le taux de changement de flux $\varphi(q)$ comme une fonction de la charge électrique q qui traverse le composant (Eq. I.9).

$$M(q) = \frac{d\varphi(q)}{dq} \qquad \text{(I.9)}$$

L'unité de M est la même qu'une résistance, ainsi à chaque instant, le memristor se comporte comme une résistance. La valeur instantanée de M peut s'exprimer en

fonction de la tension appliquée $V = \frac{d\varphi}{dt}$ aux bornes du memristor et du courant $I = \frac{dq}{dt}$ qui le traverse. Il en résulte la relation suivante : $M(q(t)) = \frac{V(t)}{I(t)}$.

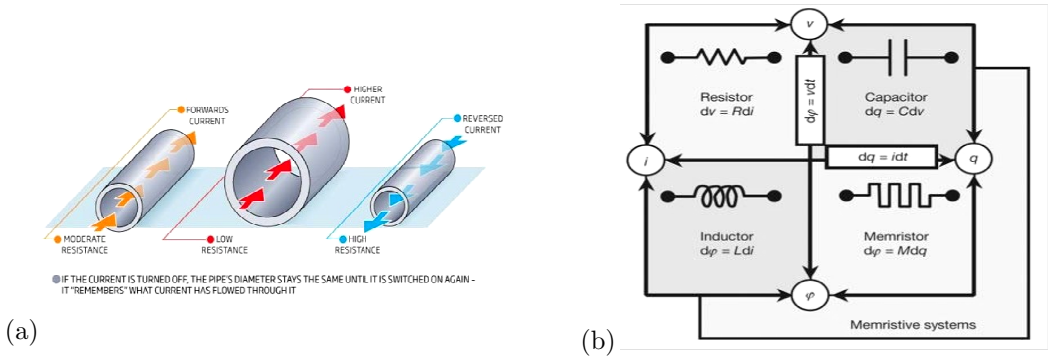


Figure I.14 : (a) Analogie de comportement du memristor, le memristor est représenté par le tuyau, dont le diamètre varie avec le débit d'eau qui le traverse (b) Les quatre principaux composants bipolaires (R, C, L, M) reliant les quatre variables fondamentales de l'électronique (V, I, Q, ϕ) [Stru08].

Depuis 1971 le memristor restait considéré comme un composant purement théorique jusqu'à ce que l'équipe de *Stanley Williams* de laboratoire HP démontre l'existence d'un dispositif réel correspondant au memristor dans le papier « The missing memristor found » publié en 2008 dans *Nature* par *Strukov et al* [Stru08]. Le composant proposé dans ce papier est nanométrique composé de couche mince (5nm) de dioxyde de Titane (TiO_2) pris en sandwich entre deux électrodes de platine (Pt). La couche de Titane contient deux régions séparées par une frontière (figure I.15 (a)), une région fortement dopée (w) qui représente une faible résistance et l'autre non-dopée (D-w) avec une grande résistance. L'application d'un champ électrique va provoquer un déplacement de charges positives d'Oxygène O^+ suivant le sens de la tension appliquée. Si les charges se déplacent de la région dopée vers la région non-dopée, la variable d'état w augmente, et ainsi la résistance R du memristor diminue. R diminue avec le temps d'application de la tension jusqu'à atteindre la valeur minimale R_{on}. Si les charges se déplacent dans le sens contraire, la variable d'état w diminue ce qui provoque l'augmentation de R qui, tant que la tension est appliquée, continue d'augmenter jusqu'à atteindre la valeur maximum R_{off} (figure I.15 (a)).

Un modèle simplifié de fonctionnement du memristor a été proposé par *Sturkov et al*, basé sur deux résistances en série (Figure I.15 (b)) décrivant les deux équations suivantes :

$$w(t) = \mu_v \frac{R_{on}}{D} q(t) \qquad \text{(I.10)}$$

$$M(q) = R_{off}\left(1 - \frac{\mu_v R_{on}}{D^2} q(t)\right) \qquad \text{(I.11)}$$

Avec μ_v la Mobilité de déplacement des ions O^+, w la variable d'état comprise entre *0* et *D* l'épaisseur de la couche mince.

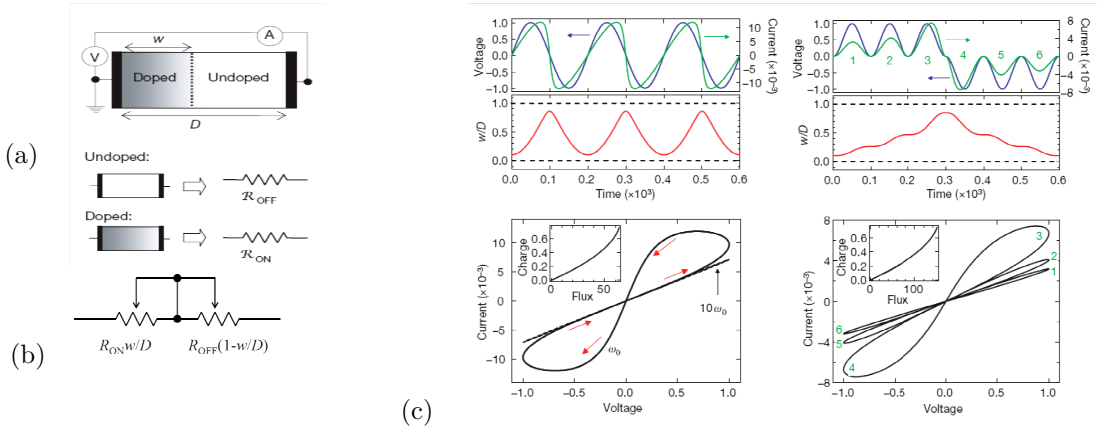


Figure I.15 : (a) schéma équivalent du memristor idéal montrant le passage de l'état *on* vers l'état *off*, V est un voltmètre et A est un ampèremètre. (b) schéma équivalent du memristor avec deux résistances en série normalisées par rapport à la variable d'état w/D. (c) à droite : boucle d'hystérésis I-V correspondant à l'application d'une tension (et d'un courant)) sinusoïdale au bornes de memristor, à gauche : boucle d'hystérésis I-V correspondant à l'application d'une tension sinusoïdale au bornes de memristor, pour plusieurs valeurs d'intensité du courant 1,2,3,4,5 et 6. Les différentes boucles d'hystérésis correspondant aux différentes intensités du courant appliquées [Stru08].

L'équipe de *Strukov et al* a publié dans [stru09] un modèle amélioré du memristor, plus réaliste. D'autres explications de fonctionnement du memristor qui relient sa commutation à d'autres phénomènes, comme la commutation thermique [Stra11], ou la commutation basée sur la création de filaments par ionisation entre les deux électrodes [Wuj09] ont été proposées. Un modèle de commutation basée sur des phénomènes électrochimiques d'oxydoréduction a été également proposé dans [Jeon09]. Bien que l'explication du phénomène de commutation et de mémorisation observée soit toujours l'objet de débats, d'autres équipes de recherche développent de nouveaux types de systèmes et composants memristifs : un composant memristif basé sur le spin présenté dans [Xiao09], ou d'autres types comme le memindictor ou le memcapacitor [Vent09].

V.2 Composants et système memristif

De nombreux systèmes présentent un comportement memristif mais avec des mécanismes physiques qui varient d'un système à un autre. Ces mécanismes peuvent provenir par exemple d'un changement de résistivité, d'un changement thermique, d'un effet relié à une réaction chimique, d'un transfert ionique, d'un changement de phase (cristallisation, amorphisation) ou d'une polarisation de spin.

Le système memristif correspond à une généralisation du memristor (IV.1) a été définit par *Leon Chua* et son étudiant *Kang* dans [Chua76] comme tout système défini par la relation entre le flux φ et la charge électrique q par l'équation suivante :

$$\varphi(t) = M(F(t))q(t) \qquad \text{(I.12)}$$

où M représente la réponse fonctionnel, et F la fonction de dépendance temporelle. Dans le cas du memristor idéal F(t)=q(t).

Cependant comme nous l'avons mentionné précédemment, un système memristif peut dépendre de différents mécanismes physiques. La fonction de réponse F des systèmes memristifs réels ne dépend pas seulement de la charge électrique q, mais peut dépendre aussi d'une ou plusieurs variables d'état qui déterminent l'état du système à chaque instant t. Cela pourrait être par exemple la position des ions d'oxygène O^+ qui détermine la résistance dans les couches minces de TiO_2 [Yan08], ou la température de thermistance [Chua76], ou le degré de polarisation de spin dans certaines structures [Pers08, Pers09]. Si nous considérons le vecteur x comme étant l'ensemble des variables d'états possibles dans un système memristif donné, nous connaissons leurs évolutions au court du temps grâce à l'équation suivante :

$$\frac{dx}{dt} = f(x, I, t) \qquad \text{(I.13)}$$

où f est une fonction continue à n-dimension.

L'équation I.13 montre une dépendance de l'évolution de la variable d'état avec le courant I. Ce type de système memristif est dit à *courant contrôlé* selon *Chua et Kang* [Chua76]. Dans ce cas la relation entre la tension et le courant peut s'écrire de la manière suivante :

$$V(t) = R(x; I; t)I(t) \qquad \text{(I.14)}$$

avec R une fonction de réponse qui a la même unité qu'une résistance et qui dépend du courant I.

Un autre système memristif est dit à *tension contrôlée* et dans ce cas la fonction de réponse dépendra de la tension V. Les deux équations I.13 et I.14 s'écrivent de la manière suivante :

$$\frac{dx}{dt} = f(x, V, t) \qquad \text{(I.15)}$$

$$I(t) = G(x; V; t)V(t) \qquad \text{(I.16)}$$

G est appelée memdictance (conductance à mémoire).

Un memristor idéal qui ne dépend que de la charge q ou que du flux φ est probablement rare. C'est pour cette raison que la plupart des auteurs ne font pas très attention aux appellations «memristif» ou «memristor». La propriété caractéristique des systèmes memristifs s'observe quand ils sont soumis à une entrée périodique : leurs caractéristiques courant-tension présentent tous une boucle d'hystérésis qui passe par l'origine (V=0, I=0), plus ou moins pincée en fonction de la fréquence

[Chua76]. Ce passage de la boucle d'hystérésis par l'origine montre une propriété qui différentie le système memristifs des autres systèmes. Le fait que le courant soit nul quand la tension devient nulle (et vis versa) montre que les systèmes memristifs ne stocke pas l'énergie électrique à la différence de la capacité ou de l'inductance. Cependant, l'énergie fournie conduit à un changement de caractéristiques physiques de ce type de composants. La figure I.16 (a) montre une dépendance de la caractéristique I-V de ce type de systèmes avec la fréquence de la tension d'entrée. Un système memristif se comporte comme une simple résistance linéaire pour des fréquences infinies (w_1), il présente une boucle d'hystérésis qui passe par l'origine pour des fréquences intermédiaires (w_2), et une caractéristique non linéaire pour des fréquences très faibles (w_3).

La caractéristique de variation de la conductance G en fonction de la tension appliquée au composant memristif est illustrée par la figure I.16 (b) et (c). Les deux figures montrent deux types de composants memristifs possibles: composants à seuil ou sans seuil. Les composants memristifs à seuil sont ceux qui représentent une caractéristique proche de celle de la figure I.16 (b), où la variation de la conductance est négligeable quand la tension appliquée est faible et inférieure au seuil (V_{th}), et la conductance varie (augmente ou diminue) sensiblement quand la tension appliquée dépasse le seuil. Cette caractéristique les distingue des composants memristifs sans seuil, où la conductance varie même pour des tensions faibles et inférieures au seuil (figure I.16 (c)).

L'existence d'un effet de seuil est une caractéristique très importante pour les systèmes nécessitant une plage de lecture sans modification de la conductance, et une plage de programmation où l'information est codée par la valeur de la conductance. En effet, cela présente un avantage particulièrement pour nos méthodes d'apprentissage supervisé.

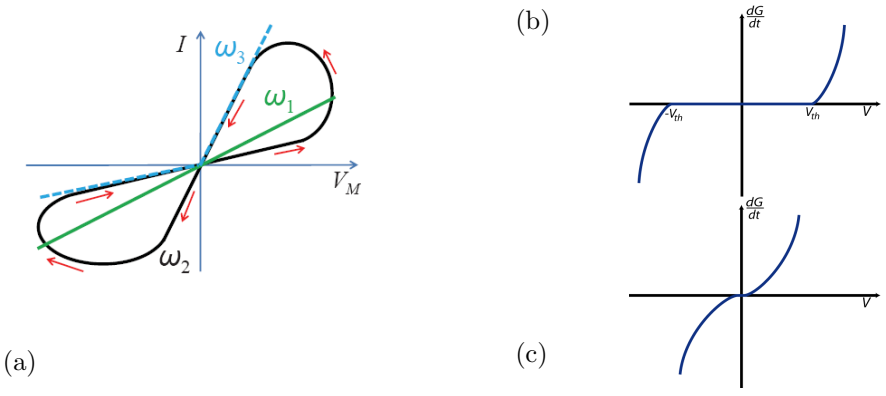


Figure I.16 : (a) Schéma de la boucle d'hystérésis d'un système memristif soumit à une entrée périodique avec différentes fréquences : $w_1 > w_2 > w_3$. (b) caractéristique de variation de la conductance en fonction de la tension appliquée dans le cas où le composant memristif présente un seuil V_{th}. (c) dans le cas sans seuil.

Récemment, l'intérêt pour les dispositifs memristifs a été porté sur le développement de nanostructures pour différentes applications. Nous en citons quelques unes :

V.2.1 Développement des Mémoires

V.2.1.1 Mémoires résistives ou ReRAM (Resistive Random Access Memory)

Les nano dispositifs memristifs permettent de développer des mémoires résistives denses et très rapides dans le but de remplacer dans les années qui viennent les mémoires flash ou les SRAM [Lai08]. Une comparaison des nouvelles mémoires émergentes avec les mémoires classiques est donnée dans le tableau I.1.

Les mémoires ReRAM sont basées sur la commutation résistive, la faible résistance représente l'état « 1 », et la haute résistance représente l'état « 0 ». Ce domaine de recherche a été initié pour la première fois par *Hickmott* en 1962, en observant un comportement d'hystérésis dans l'oxyde isolant [Hick62]. Plus tard, la commutation de la résistance a été démontrée avec des couches minces de monoxyde de silicium (SiO) entre des électrodes métalliques par *Simmons et Verderber* [SiVe67] et avec du TiO_2 par *Argall* [Arg68]. Cependant, c'est seulement en 2008 que ces dispositifs ont été reconnus comme des systèmes memristifs [Stru08]. Les ReRAM se basent sur plusieurs mécanismes de commutation. Selon ITRS 2009, ces mécanismes de commutation sont généralement reliés à des phénomènes nano-ioniques (migration des cations ou d'anions et formation de filaments conducteurs) ou des phénomènes nano-thermiques (fusible et anti fusible thermochimique, mémoire à changement de phase...). Toutefois, le mécanisme de commutation n'est pas clairement identifié pour certain type de ReRAM. La structure des ReRAM est composée généralement d'un isolant électrique pris en sandwich entre deux électrodes. En jouant sur les matériaux des électrodes (Pt, Ti, Ag, Cu) et la nature de l'oxyde utilisé pour l'isolant (TiO2, CuO, NiO, CoO, Fe2O3, MoO, VO2), une grande variété de ReRAM peut être obtenue avec différentes caractéristiques et différents mécanismes de commutation [Lee07, Dris09, Yan08, Shim07]. L'application d'une tension entre les deux électrodes permet de créer des chemins métalliques entre les deux électrodes (isolant) provoquant la commutation de la résistance de la cellule mémoire (figure I.17 (a)). Souvent cette commutation présente des niveaux de résistance intermédiaires entre l'état *On* et l'état *Off*, ce qui permet de fabriquer des cellules mémoires multi-niveaux afin de coder plusieurs bits par cellule, ou d'implémenter les synapses pour avoir des circuits neuromorphiques. Parmi les mémoires basées sur ce principe de fonctionnement on peut citer, les cellules à métallisation programmable (*PMC)* et les cellules à métallisation électrochimique (ECM) connue aussi sous le nom *CBRAM (conductive-bridging random access memory)* [Diet07, Koz05]. Plusieurs équipes de recherche se sont lancées dans le développement et la fabrication des ReRAM. Cela

est dû à leurs hautes performances qui sont meilleures que celles des dernières mémoires flash (voir table I.1). Un exemple de caractéristique de commutation d'une ReRAM basée sur le TiO_2 est illustré par la figure I.17 (b) [Yan08], le même dispositif a été réalisé par *Gergel-Hackett et al* [Hack09]. Ce type de mémoire montre un rapport On/Off supérieur à 10^4. Un autre exemple de ReRAM compatible avec le CMOS, basé sur la commutation d'un dispositif composé de Silicium amorphe pris en sandwich entre deux électrodes a été réalisé et est rapporté dans plusieurs publications [Dongy, Jo08, Jo09]. Les vitesses d'écriture sont très rapides (<10ns), l'endurance est raisonnable (>10^5) et la durée de vie est de 7 ans. Le lecteur pourra se référer au tableau I.1 pour voir plus de détails sur les performances des mémoires émergentes comparées à celles des mémoires actuelles.

(a) (b)

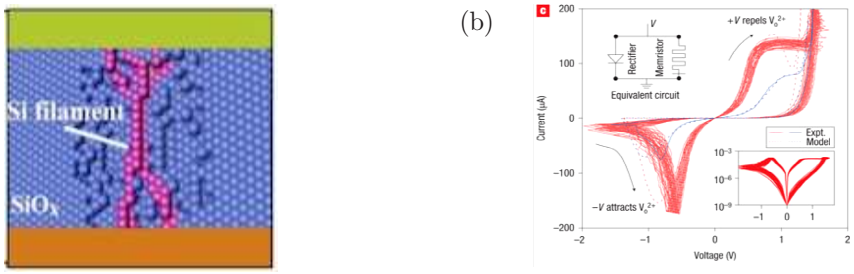


figure I.17 : (a) exemple de composant memristif de type ReRAM, l'application d'une tension aux bornes de composant induit à la formation de filament entre les deux électrodes, et le composant passe de l'état *Off* à l'état *On* [Raou08]. (b) caractéristique expérimentale I-V d'un composant memristif de type ($Pt/TiO_2/Pt$) [Yan08].

V.2.1.2 Mémoire à changement de phase

Les mémoires à changement de phase (ou *PCM, PRAM, PCRAM, CRAM,* Ovonic Unified Memory, Chalcogenide RAM) est un type de mémoire memristives qui peut remplacer aussi les mémoires flash [Raou08, Raou09, Wut07]. Ce type de mémoire se base sur l'utilisation d'un matériau à changement de phase (figure I.18 (a)), comme le verre de chalcogénure contenant un ou plusieurs éléments de la colonne 16 du tableau de Mendeleiev ($Ge_2Sb_2Te_5$, Ge-Te, Ag-In-Sb-Te). La circulation d'un courant de quelques centaines de µA dans la cellule mémoire va induire une hausse de température. Si la température dépasse la température de fusion (~600-700 °C) du matériau, ce dernier se solidifie à l'état amorphe après l'arrêt de l'impulsion. Sa résistance devient donc faible, et un état « 0 » est codé. Le passage de l'état cristallin à l'état amorphe est désigné par l'opération RESET. Le matériau revient à son état cristallin (opération SET) par un échauffement à une température inférieure à celle de la fusion mais supérieure à la température de cristallisation (figure I.18 (b)). Le matériau cristallise et sa résistance devient alors faible, un état « 1 » est codé. Les points forts de ce type de mémoires sont les suivants :

1. Rapidité d'écriture et de lecture, de l'ordre de quelques dizaines de ns (plus rapide que les DRAM) [Lee09].
2. Possibilité d'avoir plusieurs niveaux de résistance [Duyg11, Burr08, Lee09].

3. Le nombre de cycle d'écriture est autour de 10^8-cycles [Zhou09, Lee09].
4. Une haute densité d'intégration, la miniaturisation des PCRAM peut aller jusqu'à 9nm, contrairement au DRAM qui peut s'arrêter à 40nm [Raou08, Lee09].

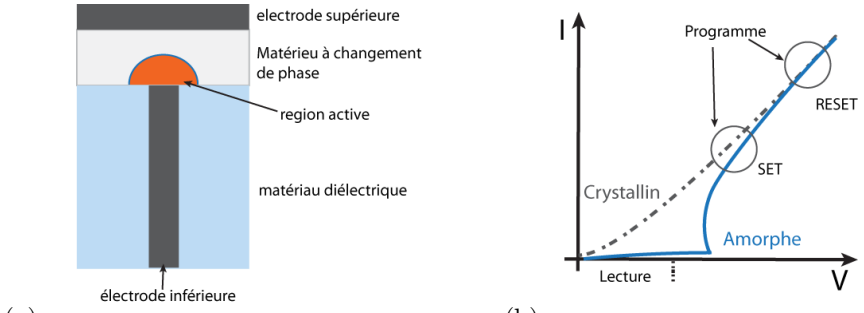


Figure I.18 : (a) Schéma de la vue en coupe de la cellule mémoire à changement de phase, (b) les courbes de caractéristique I-V de l'état cristallin et amorphe de matériau à changement de phase.

Les mémoires PCM se positionnent aujourd'hui à coté des mémoires Flash avec des meilleures performances. La caractéristique multi-niveau des PCM a été exploitée dernièrement par *Duygu Kuzum et al* pour implémenter des synapses [Duyg11]. Parmi les problèmes qui peuvent constituer un obstacle pour l'avenir des PCM on trouve l'intensité du courant élevée et la puissance dissipée importante : 480µw pour le RESET et 90µw pour le SET et 40µw pour la lecture. Cela n'a pas empêché les grands fabricants de se lancer dans la production de ce type de mémoire. *Samsung* a déjà lancé la production de modules de 512Mb, et dernièrement le spécialiste de la mémoire *Numonix* a fabriqué un module de 1Gb de PCM.

V.2.1.3 Magnétorésistance RAM

La magnétorésistance appelée aussi mémoire magnétique **(MRAM)**, est une mémoire non volatile qui exploite les jonctions tunnel magnétiques (MTJ). La figure I.19 illustre un schéma d'une MTJ. Une MTJ est composée de deux couches ferromagnétiques séparées par une fine barrière isolante (oxyde). L'une des couches ferromagnétique est appelée couche de référence ou couche fixe, car elle conserve une direction d'aimantation fixe, tandis que l'autre couche est dite libre car sa direction d'aimantation peut prendre deux orientations possibles : parallèle (résistance faible) ou antiparallèle (résistance élevée) à la couche de référence. Ce phénomène est connu sous le nom de TMR (Tunneling Magneto-Resistance) [Dongx, Burr08, Kry09, Ws09], il permet de bien discriminer les deux états de résistance correspondant aux valeurs numériques « 0 » et « 1 ».

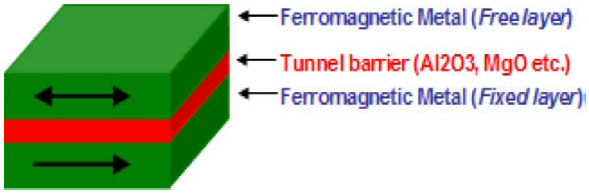


Figure I.19 : Jonction tunnel magnétique MTJ [WS09]

La méthode classique initialement utilisée pour écrire l'information en changeant l'orientation de la couche libre est basée sur l'application d'un champ magnétique (FIMS-MRAM pour Field Induced Magnetic Switching MRAM). Cette méthode souffre de problèmes de scalabilité du fait que le champ à appliquer est inversement proportionnel à la taille du bit, ce qui oblige à augmenter la taille des transistors utilisés pour générer le courant d'écriture. Afin de régler ce problème, une méthode d'écriture thermiquement assistée (TA-MRAM pour Thermally Assisted MRAM) est proposée. Cette méthode consiste à préchauffer la cellule MTJ avant d'appliquer le champ d'écriture, assurant une meilleure sélectivité et une réduction du champ d'écriture. Cependant, ces améliorations ne suffisent pas pour atteindre une consommation raisonnable. Une troisième méthode basée sur l'écriture par un courant polarisé en spin (STT-MRAM pour Spin Transfert Torque MRAM) est donc proposée. Cette méthode semble être la meilleure pour écrire dans les MRAM. Elle permet d'améliorer la densité d'intégration des MRAM ($37F^2$) qui reste toujours inférieure aux autres technologies (6-$7F^2$ pour la DRAM, $5F^2$ pour les ReRAM). Les STT-MRAM promettent aussi des temps de lecture et d'écriture très courts (lecture<10ns, écriture~12.5ns) et une consommation très faibles (lecture~0.026nJ, écriture~2.833 nJ). L'endurance de ce type de mémoires est du même ordre que les SRAM ($>10^{15}$) [Dongx]. La miniaturisation des STT-MRAM passe par une diminution de l'épaisseur de la barrière tunnel qui rend difficile de maintenir la stabilité thermique de la cellule mémoire. Pour cela plusieurs travaux de recherche développent ce point dans le but d'améliorer la stabilité thermique en utilisant, par exemple, des structures perpendiculaires au lieu de structures planaires [Naka08], ou des techniques d'écriture STT assistées thermiquement (STT-TAS) [Ws11].

Ci-dessous le tableau montrant la comparaison entre les mémoires actuelles (SRAM, DRAM, Disk, Flash) et les mémoires émergentes (PCRAM, ReRAM, MRAM).

	SRAM	DRAM	Disk	NAND Flash		PCRAM	RRAM (Memristor)	MRAM (STT-RAM)
Maturity	Product	Product	Product	Product		Advanced development	Early development	Advanced development
Cell Size	>100 F^2	6-8 F^2	(2/3) F^2	4-5 F^2		8-16 F^2	>5 F^2	37 F^2
Read Latency	<10 ns	10-60 ns	8.5 ms	25 µs		48 ns	<10 ns	<10 ns
Write Latency	<10 ns	10-60 ns	9.5 ms	200 µs		40-150 ns	~10 ns	12.5 ns
Energy per bit access	>1 pJ	2 pJ	100-1000 mJ	10 nJ		100 pJ	2 pJ	0.02 pJ
Static Power	Yes	Yes	Yes	No		No	No	No
Endurance	$>10^{15}$	$>10^{15}$	$>10^{15}$	10^4		10^8	10^5	$>10^{15}$
Nonvolatility	No	No	Yes	Yes		Yes	Yes	Yes
	Current Memory Technologies					Emerging NVM Technologies		

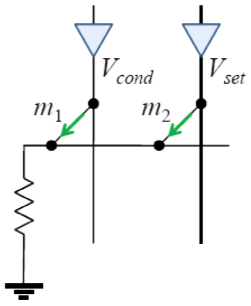
Table I.1 : Comparaison des performances des mémoires actuelles avec les mémoires émergentes [Taci10].

V.2.2 Développement des circuits logiques

Une autre application intéressante des composants memristifs, est le développement de circuits logiques. D'une part ils peuvent être utilisés dans des réseaux reconfigurables comme des commutateurs binaires pour le routage par exemple. D'autre part ils peuvent être utilisés pour implémenter les portes logiques. *Strukov et Likharev* ont montré une grande performance de l'utilisation des CMOL (*Cmos+MOLecular-scale devices*) pour réaliser des circuits de type FPGA [Stru05] ou des circuits dédiés aux traitements d'images [Stru07]. Une comparaison intéressante a été effectuée dans [Cab09], montrant que les FPGA basés sur des composants memristifs ($Pt/TiO_2/Pt$) sont plus rapides et consomment moins d'énergie que les FPGA classiques basés sur la technologie CMOS. La capacité d'auto reconfiguration des circuits memristifs a été démontrée par *Borghetti et al* [Borg09]. Le circuit développé par *Kuekes et al* montre la capacité des composants memristifs à effectuer des opérations booléennes, et la capacité de mémoriser un bit logique leur permettant d'implémenter les bascules « LATCH ». La combinaison de cette fonctionnalité avec des portes logiques basées sur des résistances et des diodes permet de réaliser des circuits logiques performants capables d'effectuer n'importe quelle opération logique. L'inversion et la restauration d'un niveau logique ont été démontrées [Kuek01]. Des travaux récents de *Borghetti et al* ont démontrés expérimentalement la possibilité d'implémenter des fonctions booléennes avec ce type de circuit [Borg10]. *Lehtonen et al* ont démontré que deux composants memristifs sont capables d'implémenter n'importe quelle fonction logique sans qu'il soit nécessaire de rajouter des transistors CMOS [Leht09, Leht10]. Nous allons expliquer le fonctionnement de ce circuit afin de donner au lecteur une idée sur les implémentations logiques à base de composants memristifs.

La figure I.20 (a) présente un circuit memristif qui réalise une implication logique [Leht09]. Le circuit est composé de deux composants memristifs à seuil m_1 et m_2,

reliés à la masse au travers d'une résistance de charge. L'état « 0 » correspond à la haute résistance du composant memristif, et l'état « 1 » à la faible résistance. Le circuit fonctionne à partir des tensions $|V_{cond}|<|V_{set}|$ et le résultat de l'opération effectuée est sauvegardé dans m_2. Le choix des deux tensions V_{cond} et V_{set} est effectué de telle sorte que m_1 ne dépasse jamais le seuil pour que son état reste inchangé. Supposons maintenant que m_1 et m_2 ont tous les deux des hautes résistances $m_1=m_2=0$. Quand les deux tensions V_{cond} et V_{set} sont appliquées, la tension aux bornes de m_2 est supérieure au seuil ce qui conduit à une commutation de m2 de l'état « 0 » vers l'état « 1 ». Cependant, sous les mêmes conditions mais avec $m_1=1$, la commutation de m_2 de « 0 » à « 1 » ne peut pas avoir lieu, car la chute de tension aux bornes de m_2 diminue. Ce type de *LATCH* est appelé *IMP-LATCH* car il est capable d'effectuer l'opération d'implication logique, une opération fondamentale de la logique booléenne [Borg10]. Cette opération est symbolisée par $(m_1 \Rightarrow m_2)=m_2$ (tableau présent dans la figure I.20 (a)) et peut s'écrire aussi sous la forme : $(NON(m_1))OU(m_2)$. A partir de ce circuit et sous réserve de pouvoir imposer un état zéro sur une entrée, il est toute à fait possible de réaliser n'importe quelle opération logique, comme l'inversion de m_1. Elle peut s'effectuer en fixant m_2 égale à 0 et en effectuant une opération d'implication logique $(m_1 \Rightarrow m_2)=m_2$, le résultat de $NON(m_1)$ se trouve donc dans m_2. Toutefois, certaines opérations logiques comme le *OU* ou le *ET* logique nécessitent un troisième memristor m_3 et plusieurs séquences [Leht09] (figure I.20 (b)).



(a)

m_1	m_2	$(m_1 \rightarrow m_2)=m_2$
0	0	1
0	1	1
1	0	0
1	1	1

$NON(m_1)=m_2;\ m_2=0;\ (m_1 \rightarrow m_2)=m_2$

$OU(m_1,m_2)=m_2;\ m_3=0;\ (m_1 \rightarrow m_3)=m_3;\ (m_3 \rightarrow m_2)=m_2$

$ET(m_1,m_2)=m_2;\ m_3=0;\ (m_2 \rightarrow m_3)=m_3;\ (m_1 \rightarrow m_3)=m_3;\ m_2=0;\ (m_1 \rightarrow m_2)=m_2$

(b)

Figure I.20 : (a) circuit d'implication logique basé sur des composants memristifs, l'état « 0 » correspond à la haute résistance de composant memristif et l'état « 1 » correspond à la faible résistance. Le symbole de triangle à l'entrée représente le circuit de contrôle des tensions d'entrée V_{cond} et V_{set}. La notation $(m_1 \Rightarrow m_2)=m_2$ indique que le résultat de l'opération d'implication logique entre m_1 et m_2 est sauvegardé dans m_2. (b) quelques séquences pour implémenter les fonctions NON, OU et ET, nous notons qu'il est nécessaire de rajouter un troisième composant memristif m_3 pour implémenter les fonctions OU, ET, NON-ET..., (figure tirée de [Leht09]).

L'opération logique NON-ET a été démontrée dans [Borg10] avec un IMP-LATCH detrois composants memristifs (Pt/TiO2/Pt) avec trois séquences. *Pershin et Di Ventra* ont amélioré l'IMP-LATCH en rajoutant une sortie constituée d'un composant memcapacitif qui va permettre de mémoriser le résultat de l'opération logique effectuée par les composants memristifs situés à l'entrée, ce qui conduit à une

réduction du nombre de séquences nécessaires pour programmer une fonction logique [Pers10]. Nous tenons à souligner que plusieurs circuits hybrides capables d'effectuer ce genre d'opérations logiques ont été brevetés [Mou06, Snid04p]. La propriété analogique intrinsèque des composants memristifs permet de les exploiter pour réaliser des circuits dédiés à des applications de logique Floue [Klir95].

V.2.3 Application neuromorphique

Une autre application potentielle des composants memristifs est l'implémentation des circuits neuromorphiques inspirés du fonctionnement du cerveau humain (section IV). Grâce à leur caractéristique analogique, leur possibilité d'avoir plusieurs niveaux de résistance et leur haute densité d'intégration, les composants memristifs sont capables d'implémenter les synapses permettant de relier les neurones entre eux et de stocker l'information.

Plusieurs composants memristifs ont montré un comportement de STDP (Spike Timing Dependent Plasticity) [BiPo98]. Comme nous l'avons vu dans la partie où nous avons introduit le memristor, l'augmentation ou la diminution de la résistance (conductance) de composant memristif dépend du sens du courant, et donc de la différence de potentiel appliquée entre ses deux électrodes (figure I.21 (d)). Nous avons vu aussi que le changement de cette résistance dépend de l'intégrale de la tension appliquée. Si maintenant, nous considérons le composant memristif à seuil (figure I.21 (c)), la conductance (poids synaptique) augmente ou diminue non-linéairement selon que la tension appliquée V_M est supérieur au seuil V_{th} ou inférieure

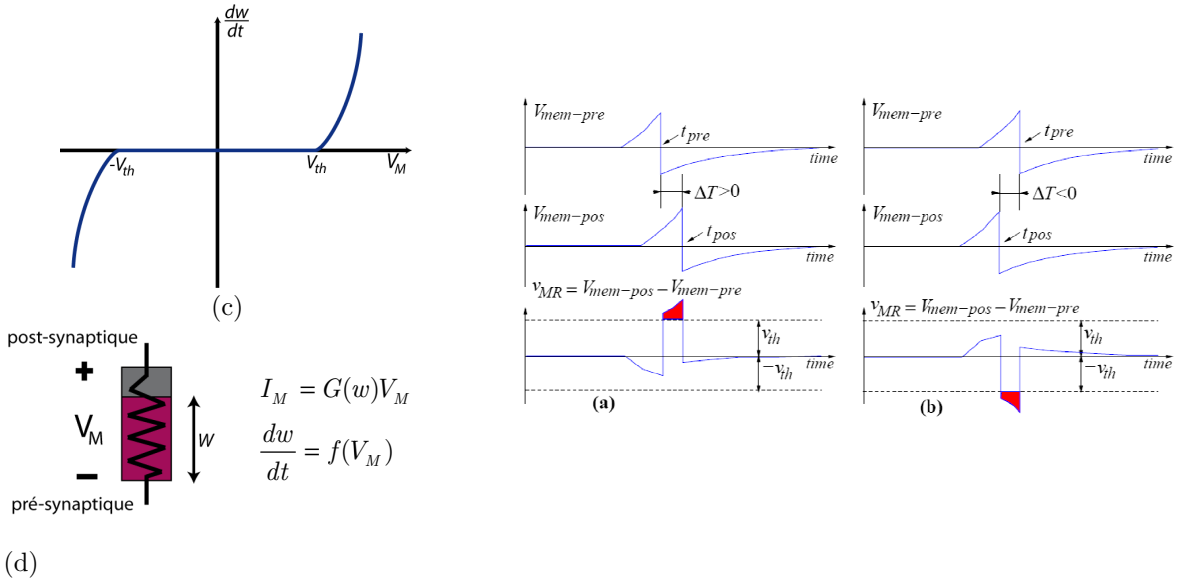


Figure I.21 : (a) quand le spike post-synaptique arrive avant le spike pré-synaptique ($\Delta t>0$) la tension appliquée au memristor dépasse le seuil, le poids synaptique augmente suivant la courbe présente dans (c), et quand $\Delta T<0$ la tension appliquée dépasse le seuil dans le sens négatif et le poids synaptique diminue selon (c), (d) schéma du memristor avec w la position de la frontière qui bouge avec la tension appliquée selon (c).

à - V_{th}. La conductance évolue selon un incrément Δw qui dépend de la différence de temps Δt entre l'impulsion post-synaptique à t_{pos} et l'impulsion pré-synaptique à $t_{pré}$ (figure I.21 (a) (b)). Le poids synaptique subit une potentiation pour un $\Delta w>0$ ($\Delta t>0$) et une dépression pour un $\Delta w<0$ ($\Delta t<0$).

Plusieurs travaux de recherche ont montré expérimentalement l'effet STDP avec différents types de composants memristifs, des composants à deux électrodes dans [Jo10, Choi09, Pers10] et à trois électrodes dans [Alib10, Lai10, Ws10]. *Choi et al* [Choi09] ont démontré en 2009 avec leur crossbar composé de mémoires memristives à deux-pôles (GdO_x/Cu dopé MoO_x) la possibilité de faire la somme des poids synaptiques, qui constitue une opération importante pour l'implémentation des RNA. Récemment *Jo et al* [Jo10] ont montré l'effet STDP (figure I.22 (e)) avec un crossbar de composants memristifs à deux-pôles basés sur l'Ag (figure I.22 (c)). Ils ont utilisé dans leur architecture une partie CMOS qui permet de mesurer la différence de temps Δt entre l'impulsion post-synaptique et pré-synaptique afin de générer des impulsions de programmation, positives dans le cas où $\Delta t>0$ et négatives dans le cas où $\Delta t<0$. Leur composant montre plusieurs niveaux de résistance (figure I.22 (d)) ce qui permet d'implémenter la plasticité de la synapse avec une bonne résolution. La partie CMOS, utilisée ici pour la génération d'impulsions de programmation, est remplacée par des systèmes memristifs capables de générer ces impulsions intrinsèquement dans [Pers10].

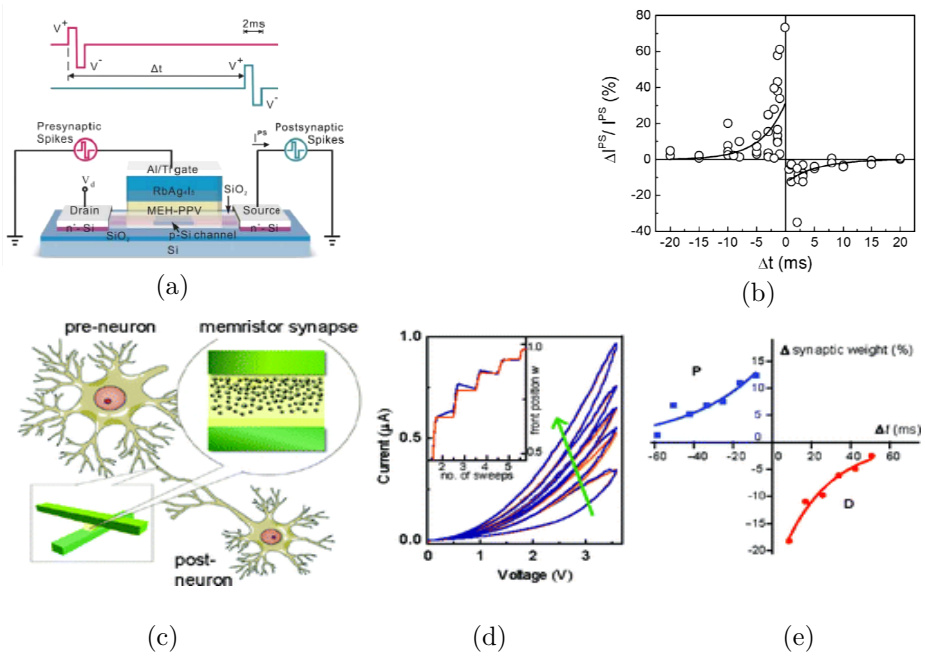
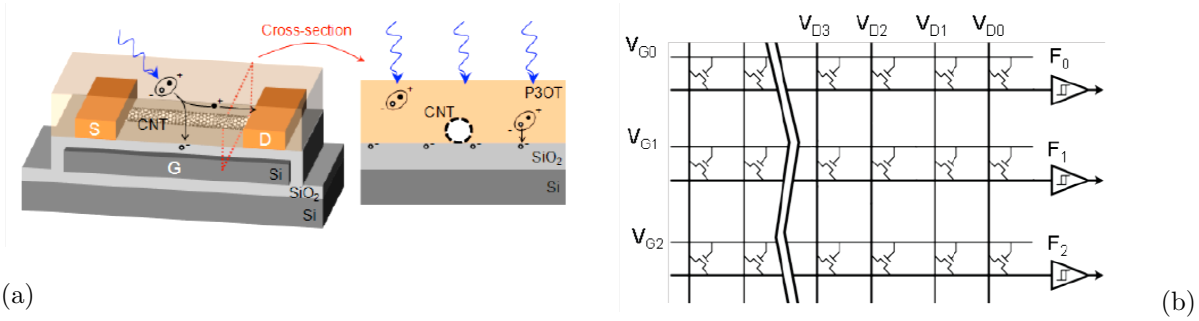


Figure I.22 : (a) transistor memristif de type MOS. (b) ses points mesurés de poids synaptiques [Lia10]. (c) schéma montrant l'implémentation d'une synapse biologique à base du memristor. (d) Caractéristique I-V du memristor, courbe bleu montre la caractéristique mesurée et la courbe orange est calculée à partir d'un modèle, la petite fenêtre montre les niveaux normalisés de la position w atteints en fonction du courant DC [Lai10], (e) les points mesurés de poids synaptique du memristor en fonction du délai entre les impulsions Δt. [Jo10]

La plupart des composants memristifs à trois électrodes ont la structure d'un transistor, où la grille est utilisée pour contrôler le changement de la résistance à travers le contrôle de l'intensité du courant qui passe dans le canal. Récemment, *Lai et al* ont fabriqué un transistor memristif [Lai10] montrant un comportement synaptique STDP (figure I.22 (b)). Comme l'illustre la figure I.22 (a), leur transistor memristif est un MOS conventionnel composé de deux couches au dessous de l'électrode de la grille, une couche de polymère conjugué (MEHPPV) et une couche ionique conductrice de $RbAg_4I_5$. Le comportement synaptique STDP a été démontré expérimentalement avec des impulsions de programmation positives ou négatives de 1ms.

Le NOMFET (nanoparticle organic memory field effect transistor) est un transistor avec un effet de mémoire, qui est dû au chargement des nanoparticules d'Or encapsulées, fixées dans le canal et recouvertes d'une fine couche de pentacène. *Alibart et al* ont démontré avec ce NOMFET un comportement de dépression et de potentiation synaptiques. Cependant, ce modèle souffre de quelques problèmes liés à la fuite des charges qui influent sur le temps de chargement de l'information, pouvant aller de quelques seconde à des centaines de secondes, et des tensions importantes sont nécessaires pour faire fonctionner le circuit (~50v). Une autre réalisation récente présentée par l'équipe de Vincent Deryck dans le papier de *Zhao et al* [Ws10] montre un crossbar composé de transistor à nanotube de carbone à commande optique (OG-CNTFET)[Borg06] (Figure I.23 (a)), avec des caractéristiques semblables à celles des mémoires non-volatiles multi-niveaux, ce qui donne la possibilité d'implémenter des synapses. Le fonctionnement de ce type de transistor est très similaire à celui des MOSFET, la différence se situe au niveau du canal composé ici d'un CNT (caractéristiques I-V est la même qu'un PMOS) recouvert d'un film mince de polymère photo-sensible (poly 3-octyl thiophène, P3OT) qui permet d'absorber un maximum de lumière pour piéger les charges dans le canal, conduisant à une augmentation de la conductance du canal. Cette opération permet de faire un RESET-Optique qui met le transistor à l'état *On* quelque soit la polarisation positive de la grille. L'OG-CNTFET est commandé aussi électriquement. L'application d'une tension négative par la grille permet de libérer les charges piégées dans le canal et diminuer la conductance du canal jusqu'à l'état *Off* [Liao11].



(a) (b)

Figure I.23 : (a) schéma d'un transistor OG-CNTFET, une paire d'électron trou est générée après une photo illumination, traverse le P_3OT pour atteindre la couche SiO_2 ce qui induit à une augmentation de la conductivité de canal, (b) Schéma d'un crossbar à base d'OG-CNTFET, V_D est l'entrées, F est la sortie, l'OG-CNTFET peut être programmé par des impulsions négatives appliquées à la grille V_G [Liao11].

Un intérêt particulier à l'OG-CNTFET a été porté dans le projet PANINI (Programme Architectures Nanoélectronique Intégrées Neuro-Inspirées) sur le développement d'architectures crossbar neuro-inspirées utilisant un apprentissage supervisé basé sur la règle de Delta Binaire (figure I.23 (b)). Nous y reviendrons en détail dans le chapitre quatre pour décrire les techniques d'apprentissages associées à cette architecture.

V.3 Architecture Crossbar pour les nanocomposants

L'architecture crossbar est une approche initialement dédiée aux circuits moléculaires [Chen03, Adeh05, Gold01, Kuek01, Stan03]. Le crossbar moléculaire (figure I.24) est composé de deux plans parallèles, contenant chacun des fils moléculaires ou des nano-fils, séparés par une couche mince « interlayer » composée d'une substance chimique. Les nano-fils sont identiques dans chaque plan, ils se croisent avec un angle droit en un point appelé « jonction ». Une variété de nanocomposants électroniques peut être implémentée en variant le matériau utilisé pour fabriquer les nano-fils et les jonctions. Nous pouvons implémenter des résistances, des diodes, des FET ou des composants memristif en général [Kuek01]. Ces architectures vont être utilisées par la suite pour développer des matrices de commutation programmables à l'échelle nanométrique. La fabrication des crossbars est très simple et mois coûteuse. Deux éléments nanométriques sont essentiels à la fabrication des crossbars : les commutateurs nanométriques implémentés au niveau de la jonction et les interconnexions nanométriques permettant d'interconnecter ces commutateurs entre eux et avec le circuit externe. Les techniques les plus répandues pour la fabrication de ce type d'architectures sont l'auto-assemblage et la nano-impression [Tour03, Chen03].

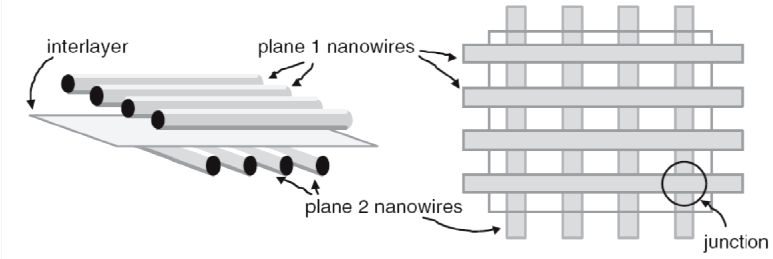


Figure I.24 : schéma de crossbar moléculaire, chaque intersection (junction) doit être configurée pour implémenter un composant électronique [Hogg06].

La réalisation de l'ordinateur reconfigurable TERAMAC par le laboratoire *Hewlett-Packard* fait parti des étapes qui ont marqué les architectures crossbars [Culb97]. Cet ordinateur est composé de petits blocs (LUT), basés sur des matrices de commutateurs programmables, qui sont reliés entre eux pour former une architecture de type FPGA. Le TERAMAC a montré une rapidité de calcul 100 fois supérieure à celle des ordinateurs sophistiqués basés sur l'architecture de *Von Neumann*, et une capacité de pouvoir fonctionner en présence d'un taux de défauts considérable

[Tehr08]. Ce projet n'a pas abouti à un produit commercialisable, mais il a permis de valider le concept des architectures crossbar et leur capacité de tolérer les fautes.
L'équipe de *Stanley Williams* (directeur de HP-lab) a orienté la majorité de ses travaux vers le développement des architectures crossbar et leurs techniques de fabrication basée sur la chimie et l'auto-assemblage, et beaucoup d'autres travaux basés sur le développement de techniques de tolérance aux fautes permettant de compenser le taux de défauts élevé dans ce type d'architectures [Hogg06].

Les premières fabrications dans ce domaine ont été marquées par la fabrication de la mémoire moléculaire basée sur la molécule de rotaxane en 2003 [Chen03]. Cette mémoire a montré une densité record de 6,4 Gbit/cm^2 (figure I.25 (b)). Chaque intersection entre deux nano-fils vertical et horizontal représente une cellule mémoire programmable avec deux niveaux logiques «0 » et « 1 » (Figure I.25 (a)). Cette fabrication a été suivie par la fabrication d'un crossbar de 1Kb fabriqué en 2004 et de 16Kb en 2005. *Jung et al* ont fabriqué en 2004 un crossbar 34x34 avec une distance minimale de 50nm entre chaque cellule (half pitch) [Jung04]. En 2006 *Zhaoning et al* ont utilisé la technique de lithographie à faisceau d'électron (Electron beam lithography) pour fabriquer un crossbar avec 30nm de pitch et une largeur des lignes d'environ 17nm [Zha06]. Les crossbars fabriqués récemment sont composés de composants memristifs au lieu de molécules de rotaxane. Cela permet d'exploiter les propriétés des composants memristifs pour réaliser des mémoires non-volatiles à haute densité d'intégration et à faible coût. L'équipe de HP a déclaré dans MRS-2011 (RERAM R&D Advances Reported at MRS Meeting) la possibilité de fabriquer des crossbars avec des cellules de 50nmx50nm composées de matériau bistable pris en sandwich entre des nano-fils de Ti/Pt.

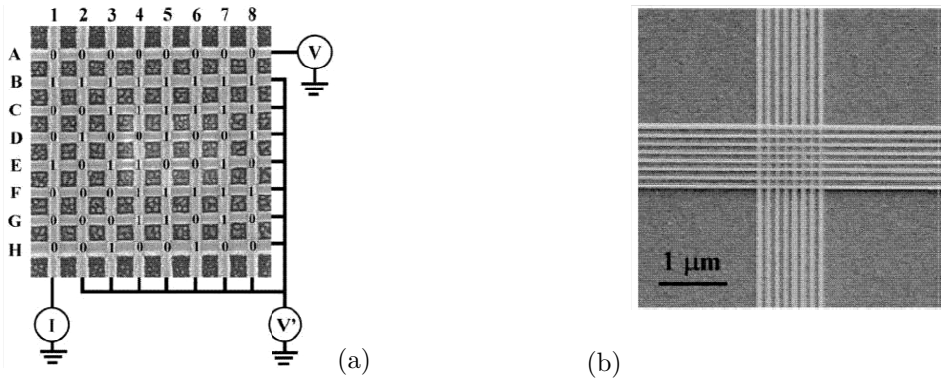


Figure I.25 : images prises par microscopie électronique à balayage (SEM), montrant une intersection de nanofils [Chen03], (b) résistance programmable à chaque intersection.

Malgré les avantages des architectures crossbar, il reste beaucoup de problèmes à régler, comme la dégradation du signal lors de la lecture ou l'écriture, et la difficulté d'adressage des cellules causée par les courants de fuite, qui peuvent prendre plusieurs chemins lors de la lecture ou l'écriture d'une ligne (mot) ou d'une colonne (bit). Une tentative pour régler le problème a été proposée par l'équipe de *Likharev*. Elle consiste à rajouter des étages de circuits (transistor ou diode) pour un adressage

individuel de chaque cellule. Ce type d'architecture hybride, connu sous le nom de CMOL (*Cmos+MOLecular-scale devices*) [Likha05, Stru09b], permet de régler plusieurs problèmes, mais la densité d'intégration et la consommation restent toujours dépendantes des circuits CMOS présents dans l'architecture, ce qui fait perdre les avantages en intégration attendus par l'utilisation des nano dispositifs. Une autre solution récemment proposée [Wang10] consiste à commuter une seule cellule à la fois en appliquant un potentiel V/2 sur la ligne (mot) et la colonne (bit) correspondant à cette cellule, et 0v ailleurs. Ainsi, seule la cellule visée se retrouve avec une différence de potentiel V et les autres cellules sont à ±V/2 ou 0v. Cette approche considère des cellules memristives à effet de seuil qui ne commutent pas sous un seuil de V/2 [Snid08, Stru09a]. Comme nous l'avons mentionné précédemment cette architecture présente plusieurs inconvénients notamment le courant de fuite puisque le courant doit circuler tout au long des cellules de la ligne et de la colonne sélectionnées, ce qui augmente la puissance dissipée. *Chaudhuri et al* ont proposé une approche basée sur la lecture et l'écriture parallèle, permettant d'éliminer les courants parasites par effet de compensation [Chau10]. Cependant, cette approche ne prend pas en compte la présence de défauts. Un seul composant endommagé dans le crossbar élimine l'effet de la lecture parallèle et perturbe l'information lue. Pour ce qui concerne le problème d'atténuation de signal dans l'architecture crossbar, *Dimitrios et al* ont proposé une solution qui consiste à ajouter des buffers dans le cas d'un crossbar de taille 32x32 [Dimi08].

VI. Architectures tolérantes aux fautes

Aujourd'hui, nous arrivons à des dimensions de composant électronique nanométriques. La fabrication de ces composants devient de plus en plus complexe et de moins en moins fiable. Les techniques utilisées actuellement s'approchent de leurs limites, ce qui rend très difficile la fabrication de circuits nanométriques sans aucun défaut. Par conséquence, il faudra trouver une solution afin de compenser ce manque de fiabilité et d'augmenter le rendement. Les techniques de tolérance aux fautes peuvent répondre à ces problèmes [Kuek05, Han05]. Ces techniques sont déjà utilisées pour les circuits CMOS. Elles permettent de tolérer les erreurs qui peuvent intervenir pendant le fonctionnement des circuits, ces mêmes techniques peuvent être utilisées pour tolérer les défauts dans les circuits nanométriques, à condition qu'elles prennent en considération la haute densité de composants intégrés dans le même circuit et le taux élevé de défaut. Le coût et la surface nécessaires pour implémenter cette technique doit être raisonnable. Le temps de test et de réparation des défauts et fautes ne doivent pas influencer la rapidité de ces systèmes.

Parmi les techniques envisagées, on trouve notamment : la redondance, le multiplexage, la reconfiguration et l'approche neuronale.

VI.1 Redondance RMR (R-fold modular redundancy)

La technique de redondance consiste à dupliquer R fois la fonction puis à prendre une décision majoritaire. La **RMR** est la technique la plus utilisée généralement, elle utilise des modules à R composants identiques (R généralement égale à 3) et un vote majoritaire M. Pour avoir plus de fiabilité la technique **CRMR** (Cascaded R-fold modular redundancy) est proposée et consiste à regrouper des modules redondants RMR d'une manière redondante. Comme le montre la figure I.26 (a), la sortie de module est calculée à partir du vote de majoritaire M après avoir comparé toutes les entrées.

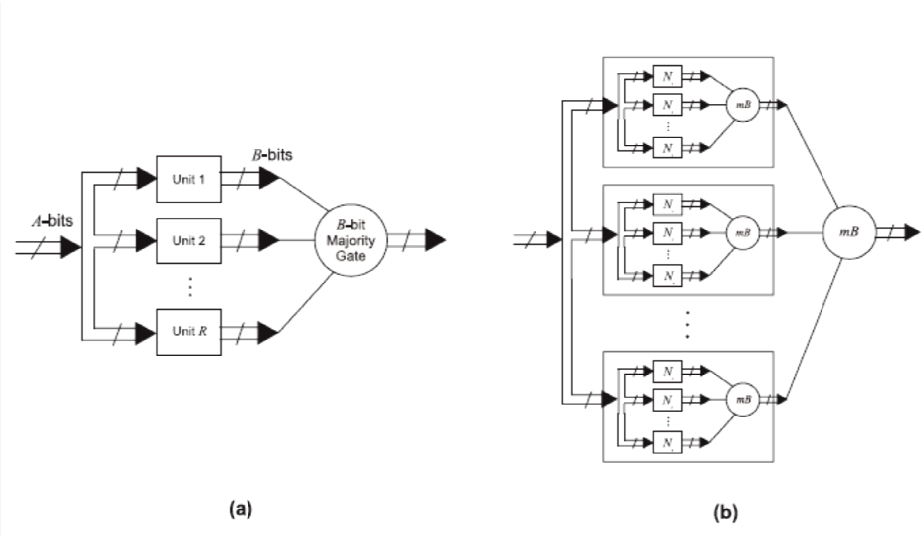


Figure I.26 : (a) Redondance modulaire avec R modules (RMR), les sorties des blocs identiques passent dont passent par un vote majoritaire (M) qui choisit la valeur de la sortie finale ; (b) un CRMR (Cascaded R-fold modular redundancy).

Dans le cas où la sortie est codée sur b bits la comparaison est effectuée sur toutes les configurations possibles. Ceci montre bien, que cette technique nécessite une surface considérable [Niko02, Kuek05, Bhad05]. Une autre limitation pour la RMR est le manque de fiabilité de bit de sortie provenant de n'importe quelle entrée qui peut être bruitée. Le vote aussi ne peut pas être toujours fiable.

VI.2 Multiplexage NAND de Von Neumann

Von Neumann a proposé une technique améliorée de la RMR. Elle est basée sur la redondance et le **multiplexage** des entrées des portes NAND pour réaliser une seule fonction *NAND* correcte [Neu56]. La technique de base est de remplacer à l'intérieur d'un circuit les blocs de traitement par des blocs multiplexées contenant un nombre N de lignes d'entrée pour une sortie unique (figure I.27). Chaque bloc est composé de portes NAND redondantes avec une probabilité de défaut ε, et d'une unité U qui distribue aléatoirement en parallèle les N lignes aux portes NAND. Un opérateur majoritaire M est placé à la sorties des portes NAND pour calculer par vote la sortie finale de bloc. Le but du multiplexage est de s'assurer que la sortie du

bloc de traitement ne soit pas affectée par la défaillance d'un petit nombre d'unités élémentaires. Cette technique permet de tolérer un taux de défaut considérable mais avec un taux très élevé de redondance [Han04, Sad04, Niko01]. L'unité U et le majoritaire M sont supposés fiables dans cette technique ce qui est irréaliste. *Sadek et al* ont développé une architecture tolérante au bruit, basée sur le multiplexage de *Von Neumann*, nommée restitution parallèle. Cette architecture permet de surmonter les problèmes de propagation de bruit et d'erreurs depuis les entrées [Sad04].

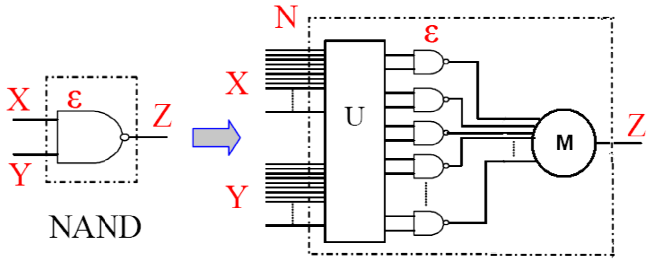


Figure I.27 : Structure à base de multiplexage NAND [Han04].

VI.3 Reconfiguration

Le principe de technique de **reconfiguration** est basé sur la modification de la configuration de l'architecture de telle sorte qu'elle soit mieux adaptée au contexte de l'exécution. Cette approche utilise des architectures composées de modules redondants reconfigurables, où chaque module peut être activé ou désactivé. Une étape de test et de reconfiguration sont nécessaires avant de mettre en marche le système. L'étape de test consiste à détecter les modules défaillants et l'étape de reconfiguration permet d'exclure ces modules défaillants et de les remplacer par des modules redondants qui fonctionnent correctement. Cette technique permet d'augmenter l'efficacité des architectures en associant les structures algorithmiques avec les architectures appropriées. Cependant, le temps de test et de reconfiguration peut être pénalisant en particulier en présence de défaillances dynamiques survenant à des instants aléatoires pendant le fonctionnement de la puce. L'exemple le plus connu d'architectures reconfigurables est le système Teramac montrant la possibilité de tolérer 75% de défauts statiques [Culb97]. D'autres architectures basées sur la reconfiguration dynamique ont été développées pour des circuits nanométriques, comme l'architecture CONAN (COfigurable Nanostructures for reliAbles Nano electronics) développée par *Cotofana et al* [Cot05] ou l'architecture cell Matrix développée par *Durbeck et Macias* [Dur01].

Plusieurs travaux de recherche développent des techniques de tolérance aux fautes dédiées aux architectures nanométriques, basées sur une ou plusieurs des trois techniques de tolérance que nous venons de citer [Chen05, Heat98, Gold03, Rao06]. *Jie Han et al* ont proposé de combiner la technique de multiplexage NAND avec la technique de reconfiguration afin de tolérer les fautes transitoires et les défauts de fabrication [Han03]. Ils ont proposé aussi dans [Han05] une combinaison de la

technique de redondance RMR avec le multiplexage sous l'appellation TIR (Triplicated Interwoven Redundancy) et NIR (N-tuple Interwoven Redundancy). Le but ici est de diminuer le taux de redondance nécessaire pour une meilleure efficacité. Il a été démontré que 5-NIR par exemple nécessite 10 fois moins de redondance que la technique de reconfiguration [Han05].

Nikolic et al ont étudié l'efficacité des trois techniques (RMR, Multiplexage NAND, reconfiguration) avec un circuit hypothétique contenant N=10^{12} nanocomposants regroupés dans des groupes logiques. Chaque groupe logique comporte R unité identiques contenant Nc composants. Une étude est présentée dans [Niko02] montrant le niveau de redondance R (surface de circuit) nécessaire pour maintenir 90% de fiabilité en fonction de taux de défauts P_f des 10^{12} éléments constituant le circuit. La technique de reconfiguration a montré une meilleure tolérance par rapport aux autres (RMR, Multiplexage). Elle tolère entre 1%-10% de défauts. Cependant, le taux de redondance reste très élevé pour cette approche (pour Nc=10^5, R=10^5).

VI.4 Approche neuronale

L'approche **neuronale** ou neuromorphique est une technique basée sur la programmation d'un ensemble de composants par apprentissage (§). En présence de composants défaillants dans l'architecture, l'apprentissage va permettre de les contourner et de les compenser par le partage de la tache à exécuter entre tous les composants. Un fonctionnement inspiré de cerveau humain rend cette approche intrinsèquement redondante et tolérante aux défauts et aux fautes transitoires [chey97, Phat92]. Plusieurs équipes de recherche ont adopté cette propriété pour les nanoarchitectures [likha04, MiHe08, Chap10, Ws08]. *Likharev et al* ont développé par exemple le Crossnet (Distributed Crosspoint Networks) réalisé à base de circuit hybride CMOL (CMOS/ nanowires /Molecular) où les neurones sont réalisés par la couche CMOS et les synapses par les dispositifs moléculaires de type transistor Single Electron Tunneling [Likh04]. Un autre exemple de *Michel He* qui a proposé dans sa thèse un modèle de nanoarchitecture crossbar neuromorphique tolérante aux fautes, utilisant des CBRAM pour implémenter les synapses [MiHe08]. Il a montré qu'il est possible de tolérer les défauts en connectant des petits réseaux des neurones artificiels réalisant la même fonction qu'un module TMR dans un réseau multicouche. Une comparaison des performances de l'approche neuromorphique par rapport aux trois autres techniques de tolérance aux fautes (RMR, Multiplexage, Reconfiguration) est donnée dans le chapitre III avec l'étude détaillée de la tolérance aux défauts d'une architecture neuro-crossbar.

VII. Conclusion

Dans ce chapitre d'état de l'art, de nombreux travaux de recherche dans le domaine de nanotechnologie ont été présentés comme étant une solution aux différentes limitations actuelles de la technologie CMOS, menaçant son arrêt dans quelques décennies. Les différentes nanoarchitectures présentées dans ce chapitre ont pour but de remplacer ou de pousser à l'extrême la technologie CMOS, en offrant la possibilité d'avoir des grandes densités d'intégration avec des faibles coûts de fabrication. Cependant, le taux de défauts de fabrication élevé dans ces architectures peut s'opposer à leur évolution. Par conséquent, le développement de nouvelles techniques de tolérance aux fautes et aux défauts de fabrication va permettre de réaliser d'une façon économique des circuits fiables contenant des milliards de dispositifs.

Les architectures basées sur les réseaux de neurones artificiels sont présentées dans ce chapitre comme l'une des approches prometteuses pour tolérer intrinsèquement un grand taux de défauts. Afin de voir comment cela pourrait être possible et comprendre les différents travaux présentés dans les chapitres suivants, nous avons introduit les RNA et donné ensuite les différentes possibilités de leur implémentation. Nous avons commencé par l'implémentation analogique puis l'implémentation numérique (ASIC, FPGA), en terminant avec l'implémentation basée sur les nanocomposants.

Afin de situer l'approche neuronale par rapport aux autres techniques de tolérance aux fautes, nous avons présenté à la fin de ce chapitre les techniques de tolérance aux fautes les plus connues : la RMR, le Multiplexage de Von Neumann, la Reconfiguration. L'approche neuromorphique sera étudiée d'avantage dans les chapitres suivants, montrant la faisabilité d'une architecture neuromorphique à base de nanocomposants (chapitre II), ainsi que l'étude de fiabilité de ce type d'architecture qui sera étudié dans le chapitre III.

CHAPITRE II

Méthode d'apprentissage pour architectures neuro-inspirées basées sur les nanocomposants

I. Introduction

La conception d'une architecture neuronale nécessite l'intégration de synapses, et de neurones. La réalisation matérielle de ces deux derniers pose généralement beaucoup de difficultés liées à la complexité du processus d'apprentissage et du choix de composants adéquat pour imiter une synapse ou un neurone. Qu'elle soit ou non basée sur des nanocomposants, la conception d'une architecture neuronale dédiée au calcul logique se résume en deux étapes pour la détermination de l'état du neurone de sortie. La première étape est une combinaison linéaire des entrées du système pondérée par un facteur adaptatif (poids synaptique). La deuxième étape est une fonction non linéaire du résultat de la première étape qui donne la sortie finale. Ces deux étapes imposent le choix des éléments suivants :

- Eléments passifs linéaires pour la première étape, qui doivent présenter une caractéristique de multiplication et d'adaptation. La conductance (ou résistance) est la variable d'état de ces éléments. Elle doit être ajustable à multi-niveau.
- Eléments actifs non-linéaires pour la deuxième étape permettant d'effectuer le seuillage de la sortie, correspondant à une restauration de niveau logique.

Comme nous l'avons vu dans le chapitre I (§I section IV.2), il existe une gamme importante de nanocomposants représentant le comportement d'une synapse. Ces composants ont montré un effet mémoire non-volatile et une capacité de varier graduellement leur conductance [Gil07, Li04, Kund05, Jo09]. Grace aux techniques d'autoassemblage ces composants peuvent être intégrés avec des matrices crossbar à haute densité d'intégration [Jo09], permettant d''implanter un très grand nombre de synapses et de diminuer la surface synaptique dominante dans un réseau de neurone. Des nanocomposants tels que la *CBRAM* et l'*OG-CNTFET* ont été utilisés en tant que synapses pour démontrer la faisabilité d'un modèle d'une architecture neuromorphique [MiHe08, Liao11].

II. Architecture crossbar neuro-inspirée

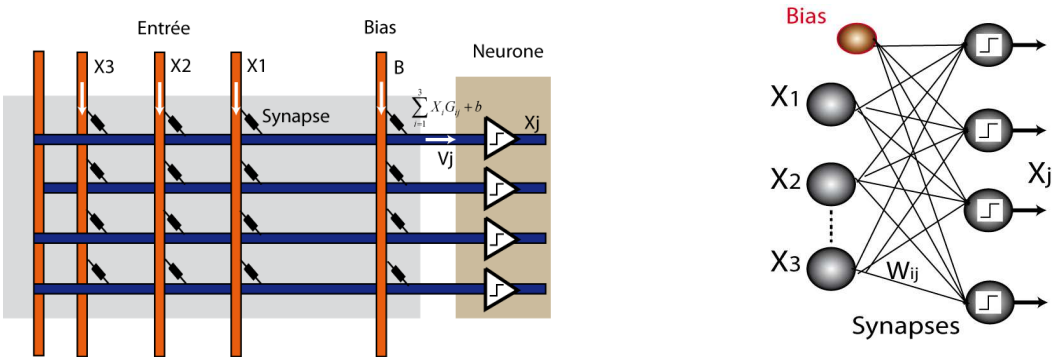
II.1 Neuro-crossbar

Tel qu'illustré par la figure II.1 (a), une architecture crossbar peut être exploitée pour réaliser un réseau de neurone [Wid60] (figure II.1 (b)), nommée ici neuro-crossbar. Un **neuro-crossbar** est une matrice de nanocomposants interconnectés entre eux par des nano-fils. Chaque intersection implémente une synapse, et la conductance G_{ij} de ce nanocomposant correspond au poids synaptique (W_{ij}). Les tensions d'entrées X_i du réseau sont appliquées aux nanofils pré-synaptique verticaux

(figure II.1 (a)). Selon le théorème de *Millmann*, une somme de tensions X_i pondérée par les conductances G_{ij} va être générée à la sortie de chaque ligne horizontale correspondant à la tension post-synaptique V_j. Le neurone réalisé grâce à l'élément actif non-linéaire implémenté à la sortie de la ligne permet de comparer la tension poste-synaptique (V_j) à un seuil d'activation pour donner un état final X_j (-1 ou 1) (Eq. II.1).

$$X_j = f(V_j)\begin{cases} 1 \ si \ V_j \geq 0 \\ -1 \ si \ V_j < 0 \end{cases} \qquad \text{(II.1)}$$

Une entrée de seuil «*Bias*» supplémentaire figée à un potentiel constant (V ou -V) est ajoutée au réseau. Elle permet d'ajuster le seuil du neurone. Selon la fonction programmée par ligne, cette valeur est ajustée en modifiant le poids synaptique (conductance) de la synapse W_b liée à l'entrée de *Bias* X_b de sorte à obtenir la bonne sortie désirée (Y_j).



(a) (b)

Figure II.1 : (a) neuro-crossbar composé d'une matrice de synapses (à l'intersection des lignes) reliant les entrées (colonnes) aux neurones (à droite), le « Bias » est l'entrée seuil, connectée à une tension fixe B. (b) réseau de neurones artificiel équivalent au neuro-crossbar présenté dans (a).

Contrairement aux nombreuses démonstrations montrant la possibilité d'implémenter les synapses à base de nanocomposants, les travaux sur l'implémentation des neurones sont actuellement basés sur la technologie CMOS classique. Les portes de type TLG (Threshold Logic Gate) par exemple, qui sont nombreuses en technologie VLSI, montrent le même comportement qu'un neurone [Beiu0]. Dans ce contexte, les travaux de *Likharev* ont montré la possibilité de réaliser une architecture hybride (Crossnet) constituée de synapses réalisées par des nanocomposants et des neurones à base de la technologie CMOS [Likh03, Turel05]. Sans nier l'intérêt d'une telle architecture, il semble qu'il est plus intéressant d'implémenter les neurones en utilisant des nanocomposants, tels que les Latchs moléculaires qui peuvent être utilisés pour restaurer des niveaux logiques [Kuek01]. En conséquence, les neurones réalisés à partir de nanocomposants vont certainement occuper moins de surface et donc présenter une consommation inférieure. Cependant,

le manque de fiabilité de ces dispositifs limite leur intégration comme éléments de décision, car cela peut entrainer des erreurs supplémentaires à celles provoquées par les défauts présents au niveau des synapses. En attendant l'amélioration des techniques de fabrication de nanocomposants, l'architecture CMOL représente actuellement la piste la plus prometteuse pour fabriquer des architectures neuromorphiques avec une meilleure fiabilité.

II.2 Neuro-crossbar à entrées différentielles

L'architecture crossbar présentée précédemment par la Figure II.1 a montré que chaque nanocomposant passif joue le rôle d'une seule synapse. Cependant, les poids synaptiques W_{ij} dans un réseau de neurones sont habituellement signés alors qu'un élément passif ne peut pas simuler une résistance négative. Le poids synaptique dans le cas de la figure II.1 ne peut être que positif, ce qui permettra d'implémenter que les fonctions logiques utilisant un poids synaptique positif. En règle générale, pour un réseau composé de N entrées, il existe 2^{2^n} fonctions logiques. Dans le cas de N=3 entrées par exemple, sur les 256 fonctions possibles, seulement 11 fonctions logiques sont implantables avec des poids synaptiques positifs. Les autres fonctions nécessitent des poids synaptiques signés [MiHe08]. Les poids synaptiques positifs n'implémentent donc qu'un petit ensemble de fonctions logiques. En conséquence, il est nécessaire d'avoir des poids synaptiques signés. La solution pour avoir des poids synaptiques signés est de représenter la synapse par une paire de composants avec des **entrées différentielles** (positives et négatives) (figure II.2). Le poids synaptique W_{ij} dans ce cas là est égal à la différence de conductance entre les deux composants constituant la paire synaptique (Eq II.2).

$$W_{ij} = K(G_{ij}^{+} - G_{ij}^{-}) \qquad \text{II.2}$$

avec K le facteur de normalisation.

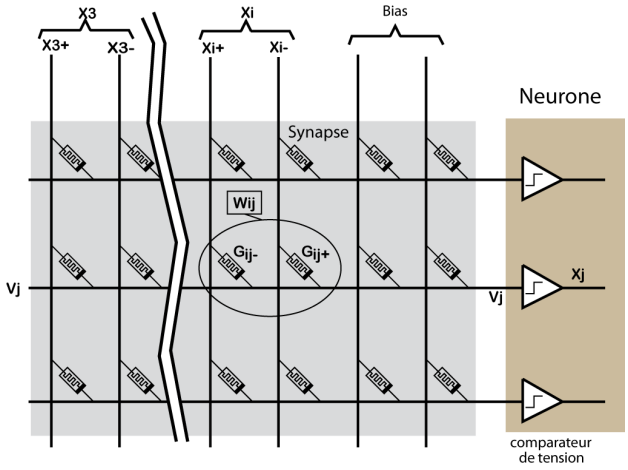


Figure II.2 : Architecture neuro-crossbar avec des entrées différentielles pour simuler des poids synaptiques signés.

III. Techniques d'apprentissage pour les neuro-crossbars

III.1 Modèles de synapses possibles

La figure II.3 montre la classification des différents modèles de variation de la conductance des nanocomposants pouvant jouer le rôle de la synapse. Ces modèles ont été classifiés par *J-O. Klein* et *E. Belhaire* dans [Jok09b] avec des codes sous forme d'une chaine de caractères représentant la forme de variation de la conductance (+0+, -0-, +0-, ...). Les caractères '+', '-' et '0' sont respectivement utilisés pour désigner une variation de conductance positive, négative ou nulle. Le zéro, toujours présent au centre, symbolise la zone neutre (ou zone de lecture) où la conductance est censée rester invariable pour des tensions faibles. Cette zone permettra au système d'être exploité sans déprogrammer les synapses.

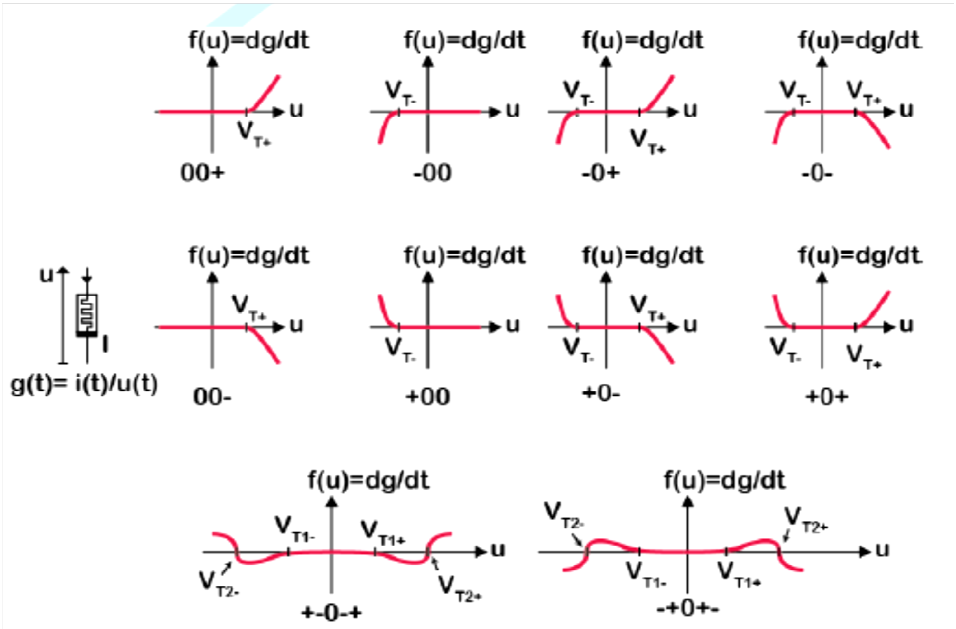


Figure II.3 : Classification des différents modèles de synapses.

Plusieurs réalisations de nanocomposants ont montré des caractéristiques semblables à l'un des modèles de la figure II.3. Par exemple, le dispositif de type «-0+» où la conductance peut augmenter en appliquant une tension positive et diminuer en appliquant une tension négative est une caractéristique semblable à celle d'un memristor [Stru09, kund05] (§I. section IV.1). Le transistor basé sur les nanotubes de carbone à commande optique (OGCNTFET) montre une caractéristique de variation de conductance semblable au type « 00-», où la conductance, suite à l'application d'une tension positive par la grille, ne peut que diminuer [Liao11] (§I section IV). Un nanocomposant memristif de type ReRAM (TaO_x) a été publié récemment dans [Myo11] montrant les mêmes caractéristiques

que le modèle « +0+». Un nanocomposant à commutation résistive à deux seuils possédant les mêmes caractéristiques que le modèle «-+0+-» a été proposé dans [Eike10], comme une solution permettant de régler le problème du courant de fuite dans les mémoires crossbar.

La variation de la conductance dans la plupart de ces démonstrations expérimentales est généralement différente de celle présente dans les modèles théoriques. Parfois, la zone neutre présente des variations de conductance ce qui peut causer des problèmes de déprogrammation lors de l'exploration de circuit programmé, nécessitant un réapprentissage pour corriger. Le nombre de niveaux de conductance existant entre l'état *On* et l'état *off* définit la résolution de nanocomposant déterminant ainsi la plasticité de la synapse. Les composants possédant des résolutions faibles peuvent causer des difficultés au réseau pour converger lors de l'étape d'apprentissage. Nous allons étudier dans le chapitre suivant, l'influence de chacune des variations et de dispersions des caractéristiques de nanocomposants qui peuvent influencer l'apprentissage.

I.1 Implémentation de la règle de Delta

L'apprentissage est l'étape qui consiste à configurer les synapses d'une manière intelligente pour implémenter une fonction logique. Comme nous l'avons évoqué dans le chapitre précédent. Il existe différents techniques d'apprentissage pour différentes applications de réseau de neurones. Dans le cas d'une architecture simple dédiée au calcul et l'apprentissage de fonctions logiques, l'apprentissage supervisé basé sur la règle de *Delta* est généralement utilisé. Cette simple règle permet d'ajuster les poids synaptiques ΔW_{ij} (conductance, ΔG_{ij}) en fonction de l'erreur exprimée par la différence entre la réponse désirée(Y_j) et la réponse calculée (X_j), et le signe de l'entrée X_i (§règle de Delta). Le poids synaptique est incrémenté ($+\alpha$) quand l'erreur est positive et décrémenté ($-\alpha$) dans le cas d'une erreur négative (tableau II.1). Le fonctionnement « en boucle fermée » (puisqu'il n'y a d'action que sur les synapses d'un neurone en erreur) permet d'avoir une programmation synaptique rapide qui peut s'arrêter d'elle-même.

X_i	Y_j	X_j	ΔW_{ij}	N° configuration
L	L	L	0	(C0)
L	L	H	$+\alpha$	(C1)
L	H	L	$-\alpha$	(C2)
L	H	H	0	(C3)
H	L	L	0	(C4)
H	L	H	$-\alpha$	(C5)
H	H	L	$+\alpha$	(C6)
H	H	H	0	(C7)

Tableau II.1 : Table de vérité de changement de poids synaptique suivant la règle de Delta. L représente le niveau bas « -1 », et H est le niveau haut « 1 ».

A l'échelle nanométrique le problème posé est la façon d'implémenter la technique d'apprentissage sans perdre l'intérêt de l'utilisation des nanocomposants en termes de densité d'intégration. Il est donc nécessaire de simplifier au maximum la procédure permettant de faire varier les poids synaptiques afin de réaliser la règle d'apprentissage, et d'optimiser le matériel associé.

III.1.1 Apprentissage par impulsion de programmation

Un circuit de programmation multi-niveau est nécessaire pour implémenter la règle de *Delta*. Ce circuit permettra de générer une commande en tension ou en courant pour faire varier la conductance des nanocomposants [MiHe08]. Plusieurs études relatives à la variation de la conductance de certains nanocomposants montrent une dépendance en fonction du temps, et une possibilité d'avoir plusieurs niveaux de conductance, par application d'impulsions relativement courtes (dizaines de ns), avant d'atteindre l'état extrême [Kozi05, Jo10]. Un exemple est illustré par la figure II.4 montrant l'évolution de la conductance d'un composant memristif réalisé à l'université de *Michigan* par *Jo et al* [Jo10]. Il montre la possibilité d'incrémenter et de décrémenter la conductance de ce dispositif en appliquant des impulsions de tension positives ou négatives.

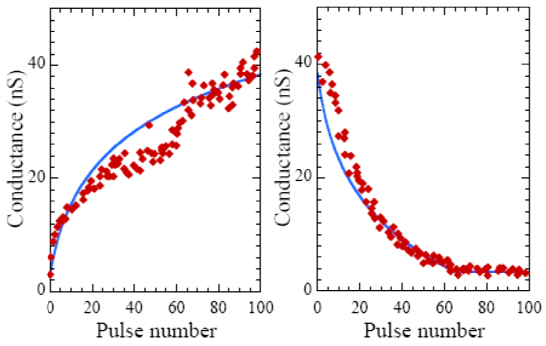


Figure II.4 : Evolution de la conductance d'un composant memristif en fonction d'impulsions de programmation appliquées au dispositif. (a) la conductance est mesurée avec une tension appliquée de 1v (<seuil), une série d'impulsions négative (=-3.2V) est appliquée au dispositif afin d'incrémenter sa conductance, (b) des impulsions de 2.8V sont appliquées pour décroitre la conductance [Jo10].

III.1.2 Cellule d'apprentissage avec sélection individuelle

Un circuit d'apprentissage implémentant la règle de *Delta* a été proposé dans la thèse de *Michel He* [MiHe08]. Ce circuit est composé d'éléments de sélection placés au niveau de chaque colonne permettant l'adressage de chaque synapse (CBRAM) et des éléments de programmation associés à chaque neurone constituant l'unité d'apprentissage (Figure II.5). Cette unité est composée de deux parties. Une première partie analogique permet le contrôle de l'augmentation et de la diminution de la conductance du composant CBRAM par l'injection d'un courant de programmation positif ou négatif à partir d'un circuit d'injection de charge [Zhu94]. Cette partie est conçue à partir de modèles de transistors CNTFET ainsi que par des condensateurs.

Une deuxième partie numérique est conçue à partir d'un schéma de porte logique tout à fait classique. Cette partie génère deux signaux de commande envoyés vers la partie analogique permettant d'incrémenter (INC) ou de décrémenter (DEC) la conductance en suivant la règle de *Delta* (tableau II.1). Ces signaux de commande vont être générés à partir des valeurs de sortie obtenue X_j et souhaitée Y_j, mais aussi à partir de l'état d'une entrée X_i sélectionnée. La dépendance de la règle de *Delta* par rapport à l'entrée X_i nécessite l'insertion dans la matrice d'éléments de sélection qui permettent de connecter l'entrée considérée à son unité d'apprentissage, conduisant ainsi à une programmation séquentielle, colonne par colonne. Cette approche n'est pas compatible avec les grandes densités d'intégration et nécessite une surface relativement importante pour l'implémentation des circuits de sélection et des cellules de programmation. L'étude menée dans la thèse de *Michel He* constituait une première étape qui a permis de démontrer la faisabilité de l'implémentation d'une technique d'apprentissage pour configurer intelligemment des nanocomposants. Grâce aux travaux de *J.O klein et E. Belaire,* cette approche a été optimisée par l'élimination de circuit de sélection. Ils ont démontré la possibilité de configurer les éléments de la matrice de synapses en ne jouant que sur les conducteurs accessibles à partir du pourtour de la matrice [JoB07, Jok09]. Les travaux présentés dans ce manuscrit sont basés sur cette dernière approche, développée encore ici en démontrant la possibilité d'effectuer un apprentissage parallèle de plusieurs neurones à la fois. Nous verrons aussi plus loin qu'une telle approche offre la possibilité de mettre les neurones en compétition, ce qui en présence de redondance permet d'améliorer la fiabilité (chapitre III).

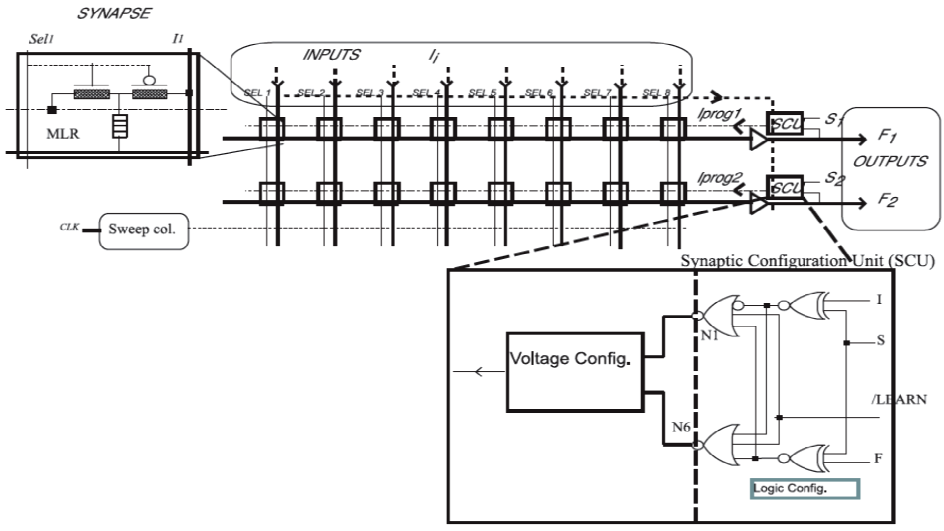


Figure II.5 : Architecture d'un réseau de neurone comportant un apprentissage on-chip utilisé dans la thèse de Michel He. Comme le sens dans lequel le courant de programmation, définissant le sens de variation de la conductance, dépend de l'état de l'entrée (suivant la règle de Hebb ou Delta), il est nécessaire d'isoler une colonne de synapses, de la déconnecter de son entrée pour la connecter aux unités d'apprentissage. En appliquant une version binaire de la règle Delta, il est ainsi possible de réaliser un apprentissage, séquentiellement, colonne par colonne.

III.1.3 Apprentissage parallèle sans sélection individuelle

Contrairement à l'architecture d'apprentissage proposée dans la thèse de *Michel He,* basée sur des éléments de sélection de colonne par colonne, l'architecture auto-configurable proposée dans [Jok09] montre la possibilité de configurer les éléments de la matrice crossbar sans recourir à des circuits de sélection (figure II.6 (a)). La sélection s'effectue en jouant juste sur la différence de potentiel appliqués entre les entrées et les sorties. Les synapses verront leur conductance modifiée selon que le potentiel appliqué à leurs bornes dépasse le seuil ou pas. Un tel comportement d'**apprentissage conditionnel** requiert des nanocomposants non-volatiles dont la conductance varie pour des plages de tensions qui dépassent le seuil, et reste inchangée pour des plages de tension inférieures au seuil. Des niveaux de tensions V_H (niveau haut) et V_L (niveau bas) suffisamment inférieurs à la tension de seuil sont utilisés pour lire les conductances sans les déprogrammer.

L'apprentissage par la règle de *Delta* peut être réalisée par un neurone conçu à partir de porte logique (figure II.6 (b)) permettant, selon l'erreur effectuée par le neurone correspondant (Y_j-X_j), d'appliquer un potentiel de programmation positif ou négatif V_p/-V_p au niveau du nœud post-synaptique. La dépendance à l'entrée X_i s'effectue par un jeu d'un effet de seuil (figure II.6 (c)).

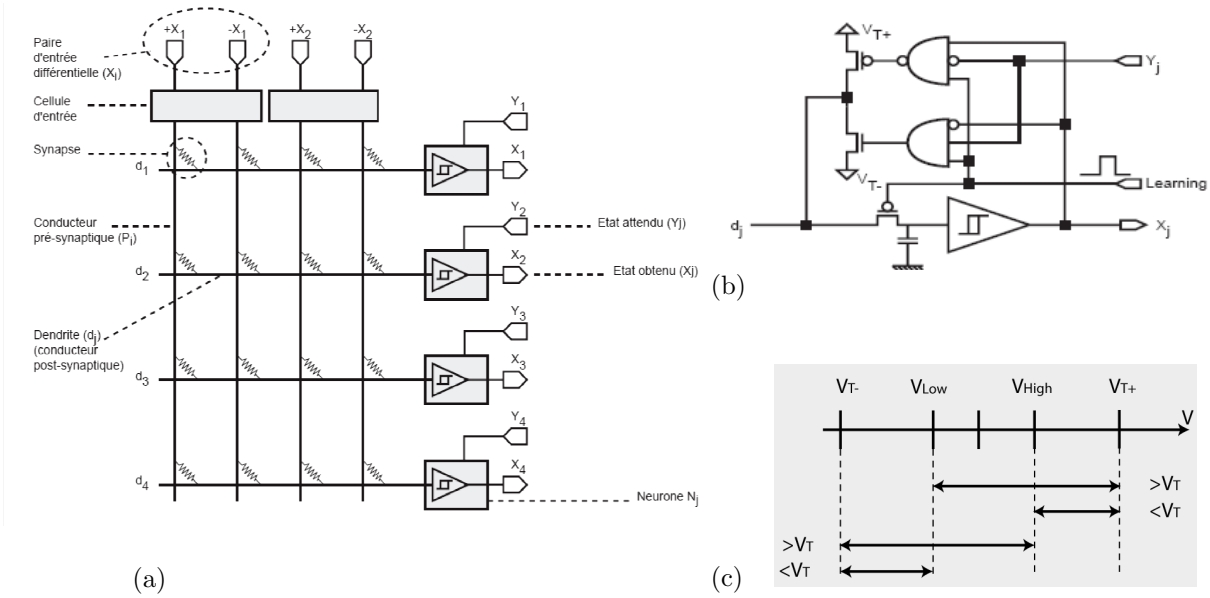


Figure II.6 : (a) Architecture d'un réseau avec apprentissage, sans éléments de sélection des synapses. La cellule neurone réalise l'apprentissage en imposant un potentiel sur les conducteurs post-synaptiques (dendrites). Les synapses verront leur conductance modifiée en fonction de la différence entre ce potentiel et celui imposé par les entrées. (b) Schéma de neurone avec apprentissage susceptible de réaliser la règle DELTA pour des synapses dont la conductance croît lorsqu'on applique une impulsion de tension d'amplitude supérieur aux seuils VT+. (c) Diagramme de la différence de potentiel aux bornes de la synapse. En appliquant une tension de programmation au nœud post-synaptique V_{post}=V_T, la différence de potentiel $V_{pré}$-V_{post} est inférieure ou supérieure au seuil dépendamment de la tension appliquée au noeud pré-synaptique.

III.1.3.1 Fonctionnement de l'apprentissage conditionnel

Comme illustré sur la figure II.6 (c), les nanocomposants utilisés ici doivent avoir un comportement à seuil. Leurs conductances ne peuvent être modifiées que si la tension appliquée est supérieure à un seuil (V_T). L'apprentissage conditionnel consiste à mettre à jour les poids synaptiques (conductance) en jouant sur cette différence de potentiel appliquée aux noeuds pré et post-synaptique. Afin d'expliquer le fonctionnement de cette technique, nous allons considérer un modèle de nanocomposants de type « -0+» (figure II.3) où la conductance est supposée augmenter pour une tension supérieure à V_T et diminuer pour une tension inférieure à $-V_T$. La variation de la conductance est supposée négligeable pour des tensions appliquées à l'entrée V_H et V_L inférieurs au seuil (Eq. II.3).

$$\begin{aligned} V_H - V_{P-} &> V_{T+} \quad \rightarrow \text{G augmente (+)} \\ V_L - V_{P-} &< V_{T+} \quad \rightarrow \text{G stable (0)} \\ V_L - V_{P+} &< V_{T-} \quad \rightarrow \text{G diminue (-)} \\ V_H - V_{P+} &> V_{T-} \quad \rightarrow \text{G stable (0)} \end{aligned} \qquad \text{(II.3)}$$

Ce mode de fonctionnement est caractéristiques d'un memristor à seuil (§I section V.2). C'est pour quoi, sous l'hypothèse que la conductance du nanocomposant peut être contrôlée par des impulsions de programmation ($V_P=V_T$) appliquée au post-synaptique, le modèle de nanocomposant utilisé dans cette partie est appelé « memristor».

Le schéma de la figure II.7 montre les différents cas possibles d'erreur à la sortie de crossbar donnés par la table de vérité de la règle de Delta (tableau II.1). Le schéma est composé de deux entrées montrant les deux cas où l'entrée différentielle est au niveau haut X_1=H ($X_{1+}=V_H$, $X_{1-}=V_L$) ou au niveau bas X_1=L ($X_{1+}=V_L$,$X_{1-}=V_H$). Les quatre sorties montrent les différents cas possibles de la sortie calculée X_j par rapport à la sortie désirée Y_j. Deux types d'erreur sont possibles : le premier cas correspond au cas où on veut obtenir une sortie à l'état bas (Y_j=L) mais on a obtenu une sortie à l'état haut (X_j=H), ce type d'erreur est noté XY=HL. Le deuxième cas correspond au cas où la sortie désirée est l'état haut (Y_j=H) mais la sortie obtenue est à l'état bas (X_j=L), ce type d'erreur est noté XY=LH. Selon que l'entrée est à l'état haut ou à l'état bas on distingue quatre configurations C1, C2, C5 et C6 permettant d'éliminer les deux types d'erreurs (tableau II.1). La configuration C1 correspond à l'augmentation de poids synaptique (W_{ij}) dans le cas de l'erreur XY=HL avec une entrée à l'état bas (X_i=L). La configuration C5 correspond à la diminution de poids synaptique (W_{ij}) dans le cas de l'erreur XY=HL avec une entrée à l'état haut (X_i=H). Les configurations C2 et C6 correspondent respectivement à la diminution et l'augmentation de poids synaptique dans d'une erreur de type XY=LH pour une entrée à l'état bas pour C2 et une entrée à l'état haut pour C6.

Les quatre configurations d'erreurs décrites précédemment nécessitent l'application de potentiels de programmation positifs ou négatifs sur le post-synaptique afin d'augmenter ou de diminuer les conductances (G_{ij}). Seuls les memristors dont l'entrée est de signe opposée au potentiel appliqué verront une tension suffisante (supérieur ou inférieur à V_T) pour que leur conductance augmente ou diminue. Les autres memristors verront une tension inférieure au seuil de programmation ce qui permet de garder leur conductance inchangée.

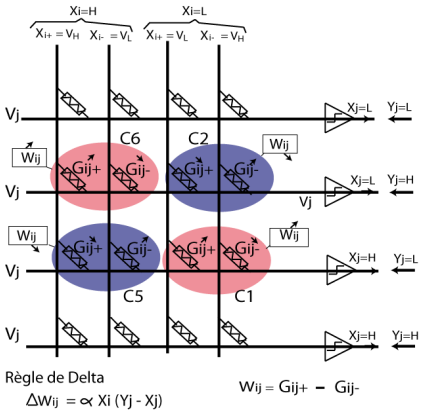


Figure II.7 : Les différentes configurations (C1, C2, C5, C6) pour éliminer les deux types d'erreur XY=HL et XY=LH, dans les deux cas possibles de niveau de potentiel d'entrée Xi=H et Xi=L.

Il a été démontré dans [Jok09] qu'il est impossible d'appliquer les quatre configurations en même temps. L'application d'un potentiel positif ou négatif au memristor «-0+» permet soit de diminuer soit d'augmenter sa conductance quand la tension dépasse le seuil dans un sens ou dans un autre. Ceci conduit à n'appliquer que deux configurations sur les quatre (C2 et C6) (figure II.8 (a)). Afin d'appliquer les deux autres configurations (C1 et C5) il est nécessaire d'inverser la polarité des entrées (figure II.8 (b)). Comme décrit dans le tableau II.2, deux séquences et une étape d'inversion sont donc nécessaires pour quatre configurations possibles pour éliminer toutes les erreurs, et cela quelque soit la polarité de l'entrée (H ou L). Les séquences sont notées S1, S2 et l'étape d'inversion est notée Ip. Ip_+ désigne une polarité normale des entrées et Ip_- une polarité inversée.

Seq #	Erreur (Y_j-X_j)	Polarité d'entrée	V_P	G_{ij+}	Config	G_{ij-}	Config
S1	> 0 (XY=LH)	I_{P+}	V_{P-}	augmente si $X_{i+}=V_H$	C6	augmente si $X_{i-}=V_H$	C2
S2			V_{P+}	Diminue si $X_{i+}=V_L$	C6	Diminue si $X_{i-}=V_L$	C2
S1	< 0 (XY=HL)	I_{P-}	V_{P-}	augmente si $X_{i+}=V_H$	C1	Augmente si $X_{i-}=V_H$	C5
S2			V_{P+}	Diminue si $X_{i+}=V_L$	C5	Diminue si $X_{i-}=V_L$	C1

Tableau II.2 : Séquences de programmation pour les différentes configurations.

Il existe deux manières d'augmenter ou diminuer le poids synaptique ($W_{ij}=G_{ij+}-G_{ij-}$), selon que l'on agit sur la conductance G_{ij+} ou sur G_{ij-}. Les deux séquences S1 et S2 représentent les deux façons d'éliminer les erreurs de type C2 et C6 sans inversion de polarité d'entrées (Ip_+), et C1, C5 après inversion de polarité.

- **S1** consiste à appliquer une impulsion de programmation négative ($V_P=-V_T$) au potentiel post-synaptique (V_j) d'une sortie en erreur de type XY= LH, permettant d'augmenter la conductance des memristors connectés à des entrées de potentiel positif. Les memristors connectés à des entrées de potentiel négatif, verront un potentiel inférieur au seuil, leur conductance restera donc inchangée. Comme le montre la figure II.8 (a), l'effet de l'augmentation de la conductance peut se présenter différemment selon que l'entrée soit à l'état haut ($X_i=H$) ou à l'état bas ($X_i=L$). Dans le cas où $X_i=H$, le memristor concerné est G_{ij+}connecté à $X_{i+}=V_H$. L'augmentation de sa conductance induit à une augmentation de poids synaptique ($W_{ij}=G_{ij+}-G_{ij-}$) permettant d'appliquer la configuration C6. Dans le cas où $X_i=L$, le memristor concerné est G_{ij-} connecté à $X_{i-}=V_H$, l'augmentation de sa conductance induit une diminution de poids synaptique permettant d'appliquer la configuration C2.

- **S2** consiste à appliquer une impulsion positive ($V_P=V_T$) afin de diminuer la conductance des memristors connectés à des entrées de potentiel négatif. Cette séquence permet de diminuer la conductance du memristor (G_{ij-}) connecté à $X_{i-}=V_L$. Par conséquent, le poids synaptique correspondant diminue. On peut ainsi réaliser la configuration C6. La séquence S2 permet aussi de réaliser la configuration C2 en diminuant la conductance du memristor connecté à X_{i-} ce qui induit l'augmentation de poids synaptique.

Afin d'effectuer C1 et C5 il est nécessaire d'inverser la polarité des entrées (Ip_-) et d'appliquer ensuite soit la séquence S1 ou soit la séquence S2. Dans ce cas l'application de la séquence S1 fait augmenter la conductance G_{ij-} du memristor connecté à $X_{i-}=V_H$ associée à l'entrée $X_i=H$ ce qui conduit à une diminution de poids synaptique (configuration C5). S1 permet aussi d'augmenter la conductance G_{ij+} du memristor connecté à $X_{i+}=V_H$ associé à l'entrée $X_i=L$, ce qui conduit à une augmentation du poids synaptique correspondant (configuration C1) (figure II.8 (b)).

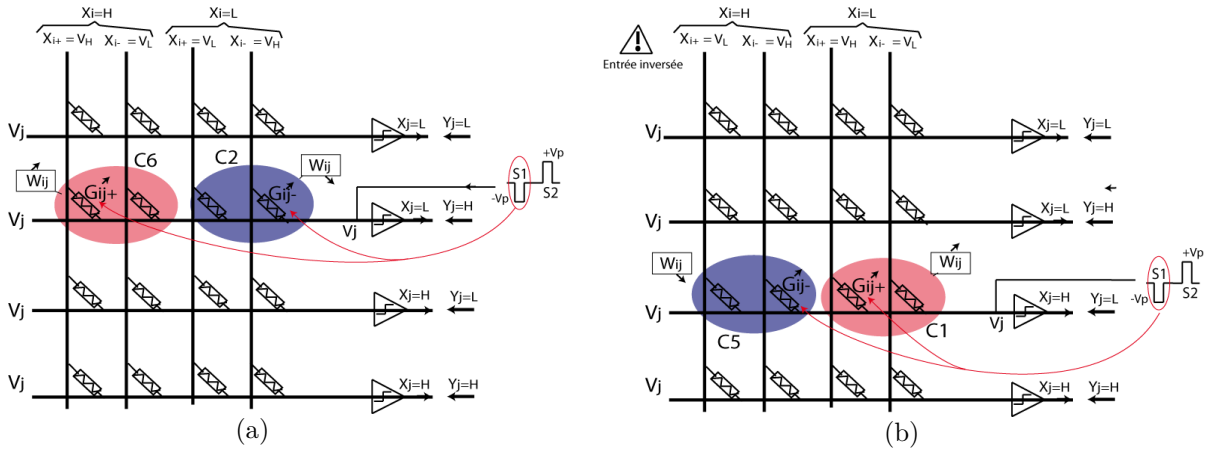


Figure II.8 : (a) effet de la séquence S1 pour éliminer les erreurs XY=LH respectivement dans le cas où Xi=H et Xi=L. (b) effet de la séquence S1 pour éliminer les erreurs XY=LH respectivement dans le cas où Xi=H et Xi=L avec une polarité inversée des entrées différentielles (Ip_-)

III.1.3.2 Mise en œuvre

III.1.3.2.1 Cellule d'apprentissage en CMOS

La cellule d'apprentissage permettant l'application des séquences S1 et S2 en cas d'erreur peut être réalisée à l'aide de porte logique de type NAND. Comme le montre la figure II.9, la sortie calculée (X_j) est mémorisée grâce à une bascule D, permettant de la réinjecter dans la cellule d'apprentissage afin de la comparer avec la sortie désirée Y_j et générer ensuite la commande permettant de commuter le transistor SwV_{p+} pour l'envoi de l'impulsion positive (V_{p+}) ou de commuter le transistor SwV_{p-} pour l'envoi de l'impulsion négative (V_{p-}). Selon le type d'erreur produit soit XY=LH ou XY=HL, la cellule d'apprentissage génère une commande permettant l'inversion de polarité d'entrée (Ip_-) ou pas (Ip_+). Pour appliquer cette commande, des Switchs d'inversion de polarité sont nécessaires au niveau des entrées

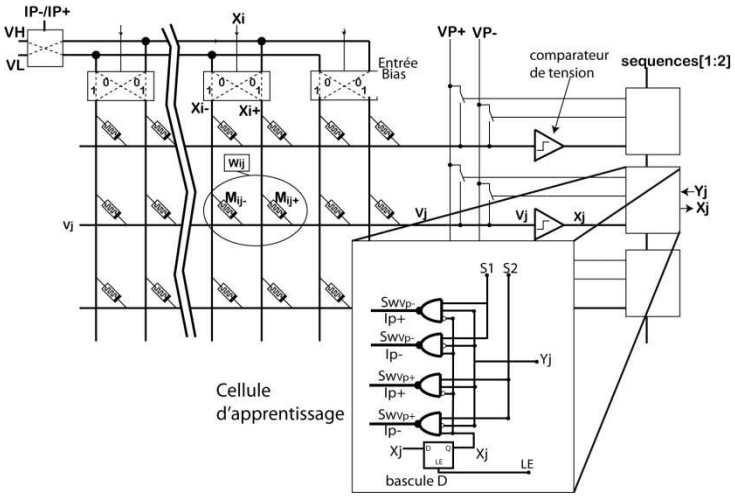


Figure II.9 : schéma d'un neuro-crossbar composé d'une cellule d'apprentissage réalisée par des portes NAND.

III.1.3.2.2 Cellule d'apprentissage en Latch Crossbar

Les travaux de *Philip Kuekes* du laboratoire HP ont montré la possibilité d'inversion et de restauration d'un signal à partir d'un latch crossbar programmable basé sur seulement deux switchs moléculaires. Ces deux opérations fondamentales sont suffisantes pour implémenter n'importe quelle fonction logique [Kuek05b]. Nous allons démontrer dans cette partie la possibilité d'utiliser ce type de circuit pour réaliser un neurone auto-configurable, capable de réaliser la fonction d'activation et d'implémenter les séquences de programmation S1 et S2 à partir d'un neurone composé de seulement deux transistors et de deux memristors.

Le latch à base de crossbar proposé dans [Kuek05b] est composé de deux lignes de contrôle C_A et C_B reliées à une ligne de signal L à travers deux commutateurs bistables S_A et S_B (figure II. 10 (a)) disposés suivant une configuration antiparallèle, où les tensions appliquées à C_A et C_B sont opposées. La caractéristique I-V des commutateurs S_A et S_B est illustré par la figure II. 10 (b), montrant une boucle d'hystérésis. Les commutateurs se comportent comme des résistances pour des tensions inférieures au seuil ($<$0.7v). La commutation s'effectue avec des tensions qui dépassent le seuil, dans le sens positif pour fermer le switch « 1 » ou dans le sens négatif pour ouvrir le switch « 0 ».

Le passage de signal par des diodes ou des résistances fait dégrader son état, et par conséquent l'état logique peut devenir faible. On parle alors d'un 1 ou d'un 0 faible et d'un 1 ou un 0 fort. Comme le montre la figure II. 10 (c), la **commutation inconditionnelle** consiste à appliquer une tension (positive ou négative) suffisamment grande pour ouvrir (0) ou fermer (1) les commutateurs. Cette commutation est effectuée à l'état initial par une application d'une tension positive à C_A pour ouvrir S_A et une tension négative à C_B pour ouvrir S_B, permettant de réaliser l'équivalent d'une opération RESET.

La **commutation conditionnelle** consiste à appliquer une tension (cond open ou cond close) permettant de passer le commutateur de son état logique faible vers un état logique fort. Si par exemple la tension appliquée à la ligne *L* est comprise entre un état «0» faible et un état «0» fort, l'application de la tension d'ouverture conditionnelle (cond open) va permettre d'ouvrir le commutateur. Par contre, si la tension de la ligne *L* est située entre un état «1» faible et un état «1» fort, cette tension d'ouverture conditionnelle ne permettra pas d'ouvrir le commutateur. Dans cet état une tension de fermeture conditionnelle est nécessaire pour fermer le commutateur et son état passe donc d'un état faible à un état fort. Ces propriétés sont exploitées dans la suite pour réaliser un neurone basé sur ce type de latch (figure II.11).

Le neurone basé sur les portes Nand présenté précédemment comprenait une quarantaine de transistors. Il peut être remplacé ici par un neurone composé de seulement deux memristors (latch) et deux transistors (Sw_{vj}, Sw_{yj}) (figure II.11 (a)).

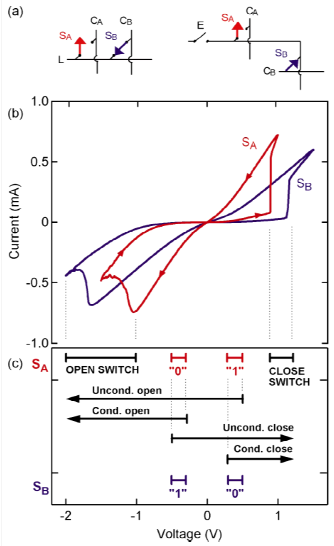


Figure II.10 : d'après [Kuek05] (a) schéma de crossbar de type latch, L est la ligne qui reçoit le signal, C_A et C_B sont les lignes de contrôle. Les commutateurs bistables S_A et S_B, permettant de relier les deux lignes de contrôle à la ligne du signal, ont une polarité opposée des impulsions d'ouverture et de fermeture des deux commutateurs. (b) boucle d'hystérésis I-V expérimentale des deux commutateurs bistables S_A et S_B. (c) schéma illustrant les niveaux de tension nécessaire pour la logique (0,1) et la commutation (open switch, close swotch), et le contrôle de potentiel pour une ouverture ou une fermeture conditionnelle ou inconditionnelle des switchs.

Les memristors constituant le latch sont de même type que les memristors implémentant les synapses « -0+ », à la différence que les tensions de seuil des memristors composants le latch V_{TH+} et V_{TH-} doivent être plus grandes que celles des memristors synaptiques et ceci pour ne pas affecter les memristors composant les neurones lors de la programmation des synapses. Les deux transistors commandés par le signal Sw_{vj} et Sw_{yj} permettent de connecter respectivement le potentiel post-synaptique (V_j) et la sortie désirée (Y_j) au potentiel du neurone de sortie observé (X_j). Les deux memristors de latch fonctionnent en mode commutateur binaire permettant de relier les lignes de contrôle S_A et S_B au potentiel de neurone Xj.

Le processus d'apprentissage est illustré par la figure II. 11 (b). Initialement, le potentiel X_j est connecté à la réponse désirée inversée Y_j grâce au switch Sw_{YJ}. Tous les memristors des neurones sont ouvert inconditionnellement au début. Ensuite, une commande conditionnelle est générée permettant de fermer l'un des memristor S_A ou S_B dans le cas où la différence entre X_j et Y_j est non nulle. Sinon si la différence de potentiel vu par les memristors est nulle (X_j=Y_j), S_A et S_B reste ouverts. Quand Y_j est négatif (Y_j=L) par rapport à X_j (potentiel de référence) le memristor S_A est conditionnellement fermé, et dans le cas d'un Y_j positif (Y_j=H) le memristor S_B est conditionnellement fermé. Après cette étape de fermeture conditionnelle de S_A et S_B, la sortie X_j est déconnectée de Y_j et connectée à V_j (grâce à Sw_{VJ}) afin de l'échantillonner et de mettre à jour le potentiel V_j. X_j est déconnecté à la fin pour isoler les synapses des impulsions provenant de C_A et C_B et pour mettre à jour le potentiel. La dernière étape consiste à envoyer des impulsions V_{p+}=V_{TH+} ou V_{p-}=V_{TH-} à travers C_A et C_B pour fermer conditionnellement l'un des memristors S_A ou S_B selon

le niveau de potentiel de la ligne du signal X_j et Y_j. C_A est connecté à toutes les sorties en erreur XY=HL et C_B est connecté à toutes les sorties en erreur XY=LH. Grâce à cette fermeture conditionnelle, X_j étant connecté à V_j l'envoie d'impulsions positives ou négatives à travers C_A et C_B permet d'implémenter les deux séquences de programmation S1 et S2. Les configurations de changement de poids synaptique C2 et C6 sont effectuées avec une polarité normale, tandis que C1 et C5 sont effectuées avec une polarité d'entrées inversée (tableau II.2).

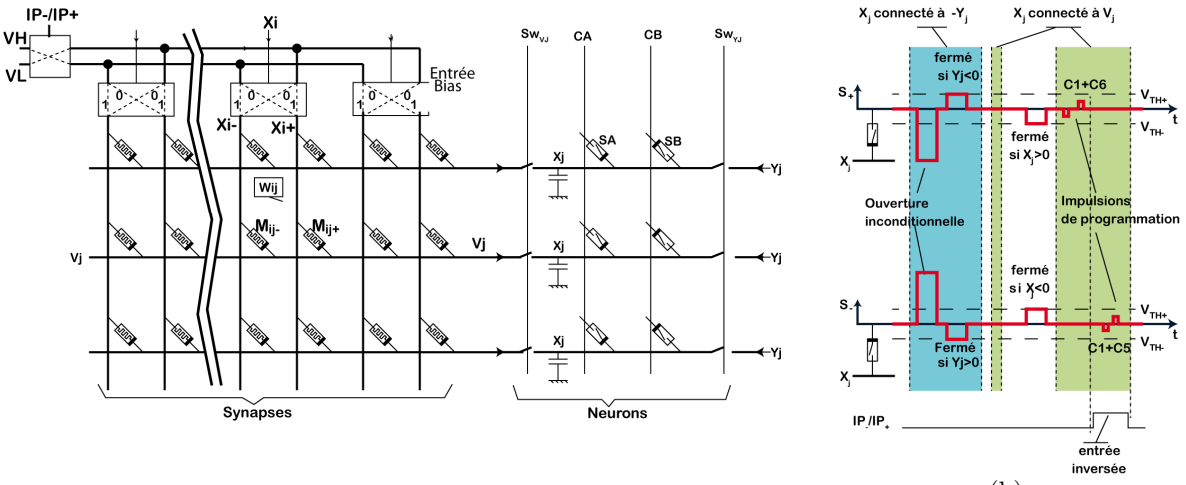


Figure II. 11 : (a) schéma d'un neuro-crossbar composé d'une cellule d'apprentissage basée sur memristor, (b) processus d'apprentissage d'un neurone basé sur un latch.

La cellule d'apprentissage proposée dans cette partie peut être réalisée en peu d'étapes lithographiques, et avec la même technologie que les memristors implémentant les synapses. Cet aspect, déterminant dans la densité d'intégration d'une puce et le coût de fabrication, permet d'envisager la fabrication des architectures neuro-crossbar à très grande densité, compacts et moins coûteuses. Néanmoins, comme il été mentionné déjà, le manque de fiabilité des ces dispositifs limite leur intégration comme éléments de décision.

IV. Modèle de neuro-crossbar en simulation

IV.1 Apprentissage de fonctions logiques linéairement séparables

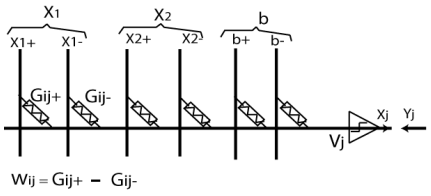
IV.1.1 Modèle fonctionnelle d'un neuro-crossbar en Matlab

Comme nous l'avons évoqué dans le chapitre I, un simple perceptron (figure 12 (c)) est capable d'apprendre n'importe quelle fonction logique à condition qu'elle soit linéairement séparable (§I section IV.2). Nous allons présenter dans cette partie une simulation fonctionnelle développée sous Matlab, réalisant un apprentissage de

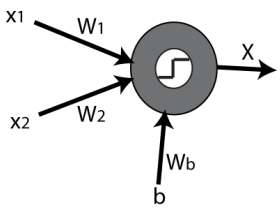
fonctions logiques à partir de l'équivalent d'un perceptron en crossbar composé d'un modèle de memristors «-0+» supposé idéal (figure II. 12 (b)).

x_1	x_2	Y_1	
1	1	1	(1)
-1	1	-1	(2)
1	-1	-1	(3)
-1	-1	-1	(4)

(a)



(b)



(c)

Figure II.12 : (a) base d'exemples d'apprentissage (b) a droite, schéma d'un simple perceptron à deux entrées, b est l'entrée bias fixée à 1 ou -1. A gauche, équivalent d'un perceptron simple en crossbar composé de memristors.

Ce modèle fonctionnel d'apprentissage consiste à calculer les potentiels électriques des nœuds importants du réseau (pré et post-synaptique). La modification de la matrice de conductance est décrite de façon algorithmique (par une incrémentation/décrémentation) à partir de la tension appliquée aux bornes de chaque memristor lors de l'application d'une impulsion de programmation. Plusieurs époques d'apprentissage sont nécessaires pour faire converger le réseau (NB_EPOCH), 20 époques dans le cas de cet apprentissage. Chaque époque est elle-même constituée de plusieurs itérations correspondant à la présentation de chaque pattern. Le nombre de pattern est égal à 2^N avec N le nombre d'entrées logiques.

```
for epoch = 1:NB_EPOCH,
for pattern_num = 1:2^N,
Xi = tabPattern (pattern_num,:);
Yj = ExpectedOutput(pattern_num,:);
…
```

Les conductances des memristors sont représentées par un vecteur de valeurs numériques initialisées avant de commencer l'apprentissage. Le poids synaptique est déduit des valeurs de conductances.

$G = \{ G_{x1+}, G_{x1-}, G_{x2+}, G_{x2-}, G_{xb+}, G_{xb}\}$.

$W = \{W_1, W_2, W_b\} = \{ G_{x1+} - G_{x1-}, G_{x2+} - G_{x2-}, G_{xb+} - G_{xb}\}$.

Le nombre de memristors dépend du nombre d'entrées et de sorties nécessaires pour implémenter un neuro-crossbar :

$$N_m = N_0 \times (2 \times N_{li} + 2).$$

Où N_m est le nombre de memristors reliant les entrées physiques au neurone de sortie, N_{li} le nombre d'entrées logiques et N_0 le nombre de sorties. Le terme "+2" correspond à l'introduction d'un neurone de seuil. Pour un réseau à deux entrées logiques (X_1, X_2) le vecteur de conductance contient six valeurs de conductance correspondant à la

structure différentielle des entrées où chaque entrée logique est représentée par deux entrées différentielles, plus les deux entrées différentielles de seuil.

La première étape de l'apprentissage consiste à calculer le potentiel post-synaptique (V_j) en multipliant les potentiels des entrées (x_i) par la matrice de conductance G, après normalisation par ligne.

```
Vj = Xi*normalize_W(G);
```

Le niveau logique de la sortie X_j est alors calculé par seuillage à partir du signe du potentiel post-synaptique (V_j) :

```
Xj = sign_fun (Vj);
```

L'application d'impulsions de programmation consiste à imposer au potentiel post-synaptique (V_j) un potentiel V_p ou $-V_p$ selon l'erreur entre la sortie calculée (X_j) et la sortie désirée (Y_j).

```
% séquence S1
Vj( (Xj == L) & (Yj == H)) = -VP;
% séquence S2
Vj( (Xj == L) & (Yj == H)) = VP;

%Inversion des entrées (Ip-=1)
Xi = -Xi;

% séquence S1
Vj( (Xj == H) & (Yj == L)) = -VP;
% séquence S2
Vj( (Xj == H) & (Yj == L)) = VP;
```

Après chaque modification de V_j (et éventuellement inversion de X_i) un appel à la fonction Update_G() met à jour la matrice des conductances synaptiques :

```
G = Update_G (G,Xi, Vj, VT, '-0+', Ginc);
```

La fonction Update_G() calcule la modification des valeurs de conductances à partir de la matrice courante G, du vecteur de potentiel d'entrée X_i et de potentiel post-synaptique V_j pour calculer la différence de potentielle appliquée à chaque point de la matrice (Gvoltage = X_i-V_j), de la tension de seuil V_T, du type de composant « -0+, 00+, -0-, ... » et du pas d'incrémentation G_{inc} (DeltaG) qui égal à 1.

Après avoir calculé la matrice des tensions aux bornes de chaque memristor (Gvoltage) les modifications de conductances sont calculées selon que Gvoltage dépasse ou non le seuil et selon le modèle de synapse envisagé :

```
if strcmp(type,'+0+')
G(abs(GVoltage)>VT) = G(abs(GVoltage)>VT)+DeltaG;
elseif strcmp(type,'-0-')
G(abs(GVoltage)>VT) = G(abs(GVoltage)>VT)-DeltaG;
```

```
elseif strcmp(type,'-0+')
G(GVoltage>VT) = G(GVoltage>VT)+DeltaG;
G(GVoltage<-VT) = G(GVoltage<-VT)-DeltaG;
elseif strcmp(type,'00+')
G(GVoltage>VT) = G(GVoltage>VT)+DeltaG;
```

IV.1.2 Résultats de simulation

IV.1.2.1 Neuro-crossbar à une seule sortie

L'apprentissage de la fonction logique AND présentée par la table de la figure II. 12 (a) a été réalisé à partir d'un modèle de crossbar à base de memristor (-0+) composé de deux entrées et une sortie (figure II. 12 (b)). Le résultat après convergence de l'apprentissage est présenté sur la figure II. 13, montrant deux graphes. Le premier correspond à l'évolution du nombre d'erreur (en rouge) et de bonne réponse (en vert) que le réseau effectue en fonction du nombre d'itération d'apprentissage. Le deuxième graphique est un histogramme des conductances synaptiques obtenues en fin d'apprentissage. Le premier graphique montre une convergence très rapide de réseau pour apprendre la fonction logique AND. Seulement deux époques d'apprentissage sont suffisantes pour converger.

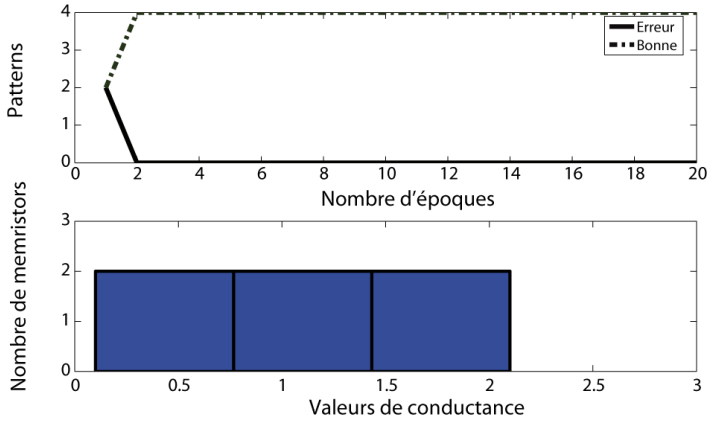


Figure II. 13: En haut, convergence de l'algorithme d'apprentissage pour apprendre la fonction AND en utilisant le neuro-crossbar de la figure II. 12 (b). La courbe discontinue représente le nombre de bonnes réponses, la courbe continue le nombre d'erreurs commises à chaque époque d'apprentissage. En bas, distribution des valeurs de conductances obtenues à la fin de l'apprentissage.

Le tableau II.3 suivant montre les valeurs de conductances et de poids synaptiques déduits, récupérés après convergence de l'apprentissage des fonctions logiques AND, NAND, OR et NOR. Dans le modèle de simulation considéré dans cette partie, la conductance est supposée variable (incrémentation/ décrémentation) entre deux valeurs G_{min}=0 et G_{max}=10 avec un pas d'incrémentation de G_{inc}=1.

	G_{x1+}	G_{x1-}	G_{x2+}	G_{x2-}	G_{b+}	G_{b-}
	W_{x1}		W_{x2}		W_b	
Initialisation	0	2	1	0	2	0
	-2		1		2	
AND	2	0	1	1	0	2
	2		0		-2	
NAND	0	2	0	2	2	1
	-2		-2		1	
OR	2	0	1	1	2	0
	2		0		2	
NOR	0	2	0	2	1	2
	-2		-2		-1	

Tableau II.3 : Valeurs de conductances (G) et de poids synaptiques (W) de neuro-crossbar de la figure II.12 (b) après convergence de l'apprentissage d'une fonction logique.

IV.1.2.2 Neuro-crossbar à plusieurs sorties

Nous avons montré précédemment qu'il est possible d'éliminer l'erreur en quelques époques d'apprentissage grâce à l'application d'un potentiel positif ou négatif au niveau du potentiel post-synaptique. Ainsi, deux circuits ont été proposés pour l'implémentation de la cellule d'apprentissage, permettant d'appliquer les séquences d'apprentissage en fonction des erreurs constatées sur les neurones considérés.

Les architectures crossbars sont habituellement fabriquées en forme de matrice contenant plusieurs entrées et plusieurs sorties. Dans le cas d'un neuro-crossbar multi-sortie, nous pouvons répliquer les cellules d'apprentissage pour toutes les sorties du réseau (figure II.9 ou figure II.11 (a)). Un apprentissage parallèle est donc possible en désignant à chaque cellule la sortie désirée à apprendre. La table de vérité et les fonctions désirées peuvent être stockées dans une mémoire RAM externe. A chaque tic d'horloge un pattern de la table de vérité est envoyé aux entrées du réseau et un pattern de fonctions désirées est envoyé aux sorties du réseau.

L'apprentissage de l'ensemble des 104 fonctions logiques linéairement séparables à trois entrées a été effectué avec un réseau à trois entrées et une entrée « *bias* ». La figure II.14 montre le nombre d'époques d'apprentissage pour apprendre la totalité de ces fonctions qui ne dépasse pas 10 époques.

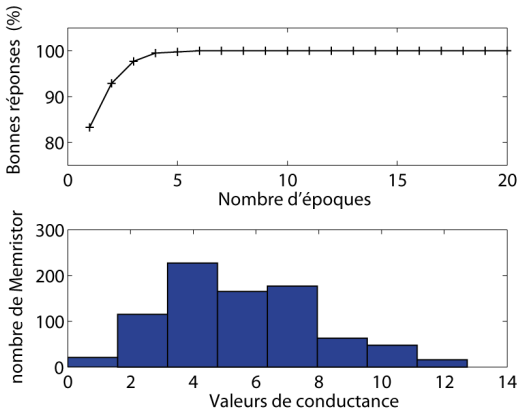


Figure II.14 : Apprentissage de 104 fonctions linéairement séparables à trois entrées avec le modèle fonctionnel du neuro-crossbar. Le graphe d'en haut présente la convergence de ces fonctions en fonction du nombre d'époques d'apprentissage, et le graphe d'en bas montre la distribution des valeurs de conductance de neuro-crossbar lors de l'apprentissage des 104 fonctions.

IV.2 Apprentissage de fonctions logiques non-linéairement séparables

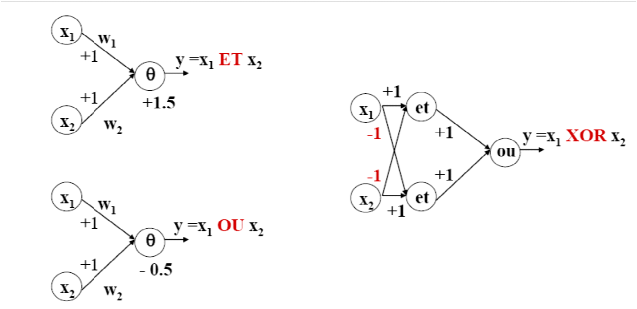
Les simulations effectuées précédemment ont montré la possibilité d'apprendre au neuro-crossbar monocouche (perceptron) des fonctions logiques. L'utilisation de la règle de *Delta* a montré des convergences d'apprentissage très rapides. Néanmoins, un simple perceptron ne peut apprendre que les fonctions linéairement séparables [Min69]. Par conséquence, le neuro-crossbar à monocouche présenté précédemment sera limité à l'apprentissage de fonctions logiques linéairement séparables. Selon *Rumelhart et McClelland* pour implémenter des fonctions non-linéairement séparables, il est nécessaire d'utiliser un réseau de neurones **multicouche** composé d'un réseau de perceptrons simples [Rum86]. Cependant, la principale difficulté de l'implémentation physique d'un tel réseau réside dans la trop grande complexité du matériel à mettre en œuvre afin de réaliser on-chip l'apprentissage classiquement utilisé pour ce type de réseau (rétro-propagation du gradient).

IV.2.1 Neuro-crossbars monocouche en cascade formant un réseau multicouche

Afin d'avoir un réseau multicouche sans recourir à l'implémentation complexe de l'algorithme de rétro-propagation du gradient, nous proposons ici de **cascader des neuro-crossbars monocouches** pour réaliser un réseau multicouche (figure II. 16 (a)). Chaque neuro-crossbar monocouche va pouvoir apprendre des fonctions logiques linéairement séparables en utilisant la règle de *Delta*, et l'association de ces neuro-crossbar pourrait réaliser des fonctions non-linéairement séparables.

Prenons l'exemple de la fonction XOR. Cette fonction peut être décomposée en éléments simples et linéairement séparables : XOR(x_1, x_2)= ((x_1 ET (NON(x_2))) OU (NON(x_1) ET x_2)). Comme le montre la figure II. 16 (a), chaque perceptron réalise

une fonction linéairement séparable, et la fonction finale XOR est réalisée par l'association des perceptrons en un seul réseau multicouche (figure II.16 (b)).

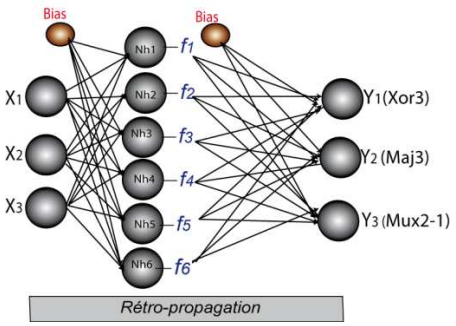


(a) (b)

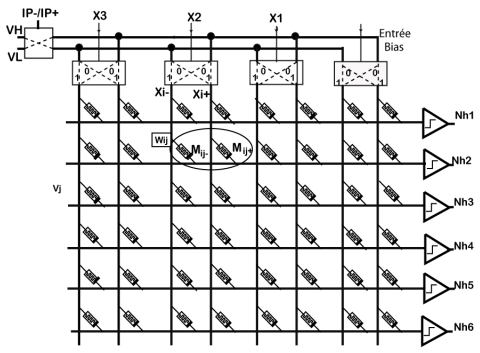
Figure II.15 : décomposition du problème de XOR en éléments simple, (a) deux perceptrons simples permettant d'apprendre les deux fonctions logiques « ET » et « OU ». (b) réseau multicouche associant permettant d'apprendre la fonction XOR, en associant les deux perceptrons simples (ET et OU), la fonction « NON » est réalisée grâce aux poids signé « -1 ».

Comme nous l'avons montré dans la partie précédente. L'apprentissage d'un crossbar monocouche s'effectue par l'application de potentiels de programmation à travers les sorties. Il est donc nécessaire d'avoir des accès aux sorties de chaque crossbar. Pour cela, nous proposons l'apprentissage **couche par couche** (figure 16 (b)) qui consiste à isoler le premier crossbar reliant l'entrée à la couche cachée afin de lui apprendre les fonctions correspondantes (fonctions cachées).

(a)



(b)



(c)

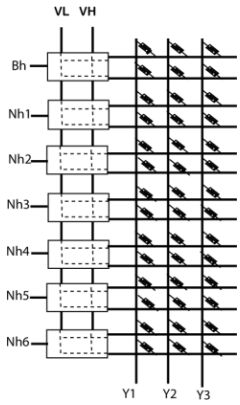


Figure 16 : (a) schéma du réseau MLP montrant la détermination des fonctions de neurones cachés par un apprentissage logiciel « rétro-propagation », (b) première étape de l'apprentissage matériel des fonctions (f1...f6) déterminées par l'apprentissage logiciel de rétro-propagation, (c) deuxième étape de l'apprentissage matériel des fonctions de sortie.

Une fois l'apprentissage de premier crossbar terminé, sa matrice synaptique est configurée et prête à calculer les fonctions de neurones cachées. Ce sont ces derniers qui vont alimenter les neurones d'entrée du crossbar suivant.

Un étage d'adaptation est nécessaire pour relier les deux blocs. Cet étage va permettre de transformer les sorties binaire de la couche cachée en entrées différentielles pour alimenter la couche de sortie. Pour l'apprentissage des neurones de sortie, les patterns sont toujours présentés aux entrées mais ils ne servent ici qu'à fournir l'état des neurones cachés, en multipliant les patterns d'entrées par la première matrice synaptique (entre l'entrée et la couche cachée). La sortie est calculée en multipliant les potentiels de la couche cachée par la deuxième matrice synaptique (entre la couche cachée et la couche de sortie). Les cellules d'apprentissage placées à la sortie permettent de configurer cette matrice comme s'il s'agissait d'un crossbar monocouche.

Toutefois, cette approche d'apprentissage couche par couche nécessite de fixer des fonctions explicites aux neurones cachés, or habituellement le terme de couche cachée souligne le fait que cette couche n'est pas directement observable à la différence de l'entrée ou de la sortie qui sont nécessaires pour apprendre au réseau la fonction souhaitée. Afin de définir les fonctions des neurones cachés et démontrer la faisabilité de l'apprentissage couche par couche, nous procéderons à un pré-apprentissage purement logiciel basé sur la Rprop (retro-propagation du gradient) [Ried94, Igel05] (§I section IV.4). Une fois que l'apprentissage Rprop a convergé, nous récupérons les fonctions des neurones cachées (et non les poids synaptiques associés) qui vont servir à effectuer l'apprentissage matériel couche par couche basé sur le modèle fonctionnel de neuro-crossbar permettant finalement de déterminer les poids synaptiques des deux matrices cachée et de sortie.

IV.2.2 Résultats de simulation

Nous avons choisi l'exemple de réseau multicouche illustré dans la figure précédente (figure II. 16 (b)). Il est composé d'une couche d'entrée contenant trois entrées (x_1, x_2, x_3), une seule couche cachée constituée de six neurones cachés (Nh_1...Nh_6) et une couche de sortie contenant trois neurones de sortie (Y_1, Y_2, Y_3). Chaque neurone de sortie va apprendre une fonction parmi les trois fonctions : Xor3, Maj3, Mux2-1 (Tableau II. 4). Afin de récupérer les fonctions des neurones cachés, ce réseau a été simulé grâce à un programme réalisant l'algorithme de rétro-propagation que nous avons décrit sous Matlab®. Les fonctions des neurones de la couche cachée récupérées après convergence de réseau sont présentées dans le tableau suivant :

Couche d'entrée			Couche cachée						Couche de sortie		
X_3	X_2	X_1	Nh_1	Nh_2	Nh_3	Nh_4	Nh_5	Nh_6	Y_1(Xor3)	Y_2(Maj3)	Y_3(Mux2-1)
-1	-1	-1	-1	1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	-1	1	-1	1	-1	-1
-1	1	-1	-1	1	1	1	1	-1	1	-1	-1
-1	1	1	1	-1	1	-1	-1	1	-1	1	1
1	-1	-1	-1	-1	1	1	1	-1	1	-1	1
1	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1
1	1	-1	1	-1	1	1	-1	-1	-1	1	1
1	1	1	1	-1	-1	1	-1	1	1	1	1

Tableau II.4 Fonctions logiques des neurones de différentes couches.

Après avoir déterminé les fonctions des neurones de la couche cachée par l'algorithme de rétro-propagation, nous pouvons maintenant procéder à l'apprentissage couche par couche. Commençons en premier par l'apprentissage des fonctions cachées, en utilisant le modèle fonctionnel de neuro-crossbar monocouche à plusieurs sorties développées dans les parties précédentes. Le neuro-crossbar est composé de trois entrées logiques (huit entrées différentielles) et six sorties correspondant aux six neurones cachés. Le type de composant utilisé ici est le memristor (-0+) avec une conductance qui varie entre G_{min}=0 et G_{max}=10 par un pas d'incrémentation G_{inc}=1. La matrice de conductance est initialisée à des valeurs aléatoires autour de la valeur G_{max} :

```
G =randn(SIZETT,NB_cach)+Gmax;
```

La figure II 17 montre les résultats de convergence des six neurones cachés, où chaque neurone Nh_i a appris la fonction qui lui est associée (tableau II.4) après environ trois époques d'apprentissage. Cela montre qu'en utilisant la règle de Delta la convergence est très rapide. L'apprentissage des six neurones se fait ici en parallèle en envoyant des impulsions de programmation à tous les neurones en erreurs ce qui permet de réduire le nombre d'époques d'apprentissage à environ 3 époques pour que tous les neurones convergent.

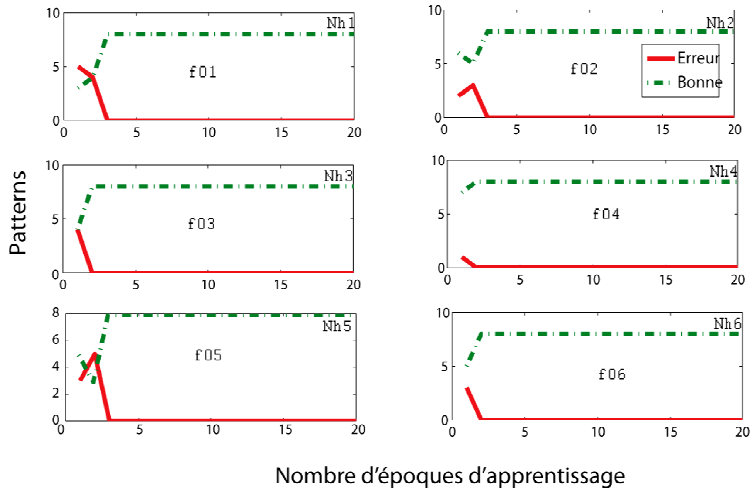


Figure II.17 : Six graphes montrant la convergence d'apprentissage de chaque neurone de la couche cachée durant les 20 époques d'apprentissage. La courbe discontinue montre le nombre de réponses correctes et la courbe continue montre le nombre de réponses en erreur.

Les valeurs de conductances de la matrice de synapses située entre la couche d'entrée et la couche cachée après convergence sont présentées dans le tableau suivant:

G	Nh1	Nh3	Nh4	Nh6	Nh7	Nh8
B_{+}	8	2	8	6	4	2
X_{3+}	8	2	4	6	4	4
X_{2+}	10	8	8	8	2	8
X_{1+}	10	4	2	4	2	8
B_{-}	8	10	4	6	6	10
X_{3-}	7.48	10	6	4	8	8
X_{2-}	4	8	5.16	6	10	2
X_{1-}	4	8	10	8	8	2

Tableau II.5 : Valeurs de conductance de la matrice située entre la couche d'entrée et la couche cachée.

Les valeurs des poids synaptiques correspondant sont calculées à partir de l'expression $W=G_{+} - G_{-}$ et sont présentées par le tableau suivant :

W	Nh1	Nh2	Nh3	Nh4	Nh5	Nh6
B	-8	0	-2	0	-8	4
X3	-8	0.51	-4	2	-4	-2
X2	0	6	-8	2	6	2.83
X1	-4	6	-6	-4	6	-8

Tableau II. 6 : Valeurs de poids synaptique (W) déduit de la matrice des conductances du tableau II.5 ($W=G_{+}-G_{-}$).

Une fois la matrice synaptique entre la couche d'entrée et la couche cachée configurée, la deuxième étape consiste à configurer la deuxième matrice synaptique située entre la couche cachée et la couche de sortie, en apprenant aux neurones de sortie Y_1, Y_2 et Y_3 les fonctions Xor3, Maj3, Mux2-1 (tableau II. 5).

Le modèle de neuro-crossbar considéré ici est composé de deux couches. La première couche, située entre l'entrée et la couche cachée (configurée pendant la première étape), va servir à calculer l'état des neurones cachés pour alimenter la deuxième couche et calculer l'état des neurones de sortie. La configuration de la deuxième matrice synaptique située entre la couche cachée et la couche de sortie se fait à travers l'application d'un potentiel de programmation appliqué aux sorties (Y_1, Y_2, Y_3). La figure II.18 montre les résultats de simulation de la convergence des trois neurones de sortie, où chaque neurone a appris la fonction logique qui lui est associée en quelques époques d'apprentissage.

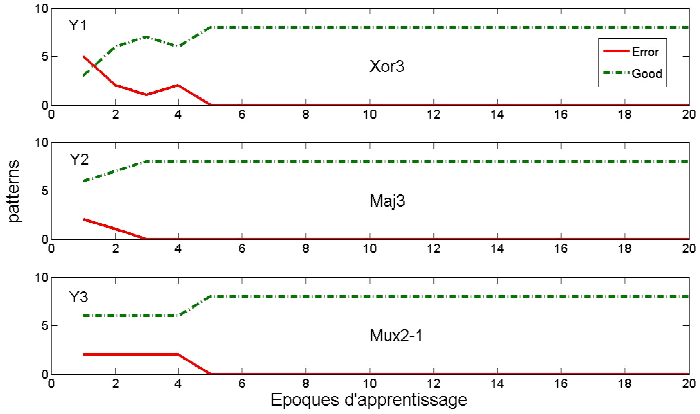


Figure II.18 : Trois graphes montrant la convergence d'apprentissage de chaque neurone de la couche de sortie (Y_1, Y_2, Y_3). La courbe continue montre l'erreur sur les patterns durant la phase d'apprentissage et la courbe discontinue montre l'état des patterns bons.

Les valeurs de conductances de la matrice de synapses située entre la couche cachée et la couche de sortie après convergence sont présentées par le tableau suivant:

G	Bh_+ Bh_-	$Nh1_+$ $Nh1_-$	$Nh2_+$ $Nh2_-$	$Nh3_+$ $Nh3_-$	$Nh4_+$ $Nh4_-$	$Nh5_+$ $Nh5_-$	$Nh6_+$ $Nh6_-$
Y1	8 4	5.52 10	4 10	10 4	8 2	8 4	2 10
Y2	7.55 10	8 10	10 2.14	4 10	8 10	10 8	6.72 10
Y3	6 8	0 10	8 10	8.46 8	10 2	8 5.7	10 8

Tableau II. 7 : Valeurs de conductance de la matrice située entre la couche cachée et la couche de sortie.

Les valeurs des poids synaptiques correspondant sont présentées par le tableau suivant :

W	Bh	Nh1	Nh2	Nh3	Nh4	Nh5	Nh6
Y1	4	-4.48	-6	6	6	4	-8
Y2	-2.45	0	7.86	-6	-2	2	-3.28
Y3	-2	-10	-2	0.46	8	2.3	2

Tableau II. 8 : Valeurs de poids synaptique (W) déduit de la matrice des conductances du tableau II. 7 ($W=G_+-G_-$).

Nous avons montré par nos simulations la possibilité d'apprendre des fonctions non linéairement séparables, tout en utilisant un apprentissage basé sur une version binaire de la règle de Delta, compatible avec un apprentissage on-chip pour des nanocomposants. Dans la suite de ce chapitre, nous allons voir à quoi peuvent servir ces modèles neuro-crossbars.

V. Architecture de calcul à base de neuro-crossbar

L'intérêt de l'approche neuro-inspirée est de permettre une utilisation des nano-composants pour construire des architectures qui soient :

– reconfigurables et tolérantes aux fautes ;
– flexibles ;
– robustes.

Une architecture basée sur ce principe reposerait en fait sur un apprentissage de fonctions élémentaires qui seraient interconnectées pour composer une architecture de calcul. Au niveau matériel ces fonctions élémentaires utilisent un petit ensemble de nano-composants dans une topologie de type "crossbar" agissant comme des connexions synaptiques. Comme nous l'avons vu précédemment ces connexions synaptiques peuvent être "reconfigurées" à volonté grâce à l'apprentissage delta binaire à couches multiples. Ceci permet d'avoir une architecture flexible proposant un haut niveau d'intégration. La densité d'intégration potentiellement élevé augmente grandement les possibilités de redondance ce qui améliore la tolérance de l'architecture aux pannes. La flexibilité intrinsèque de l'apprentissage en ligne associée à cette importante disponibilité de connexions synaptiques (memristors par exemple) rend l'architecture encore plus robuste et auto-reconfigurable. En effet, si une connexion synaptique devient inutilisable, elle peut être remplacée par un voisin grâce à la flexibilité de cette configuration par apprentissage.

Il est donc théoriquement possible d'obtenir une architecture de calcul qui soit auto adaptative et puisse se reconfigurer en cas de problème.

Nous allons présenter dans ce qui suit quelques résultats que nous avons obtenus en collaboration avec l'équipe de *Michel Paindavoine,* montrant la possibilité de concevoir des unités arithmétiques à l'aide de modèle de neuro-crossbars présenté précédemment. La plupart des simulations présentées dans la section suivante a été mené par *Olivier Brousse* en utilisant les modèles que nous avons développé dans ce cadre de cette collaboration.

V.1 Unité arithmétique et logique

L'unité arithmétique et logique (UAL) permet de réaliser des opérations logiques de base comme le ET, OU, NON et le XOR. Elle permet également de réaliser des opérations arithmétiques comme l'adition, la soustraction, la division et la multiplication sur des données numériques. Afin de montrer la possibilité d'implémenter une UAL à base de modèle de neuro-crossbar, il est nécessaire de démontrer qu'un neuro-crossbar puisse réaliser ces opérations. Nous avons déjà montré précédemment la capacité d'un neuro-crossbar à apprendre n'importe quelle fonction logique linéairement séparable ou non-linéairement séparable. Il reste à démontrer la possibilité d'apprendre des fonctions arithmétiques d'adition ou de

multiplication en suivant la même méthode d'apprentissage que les fonctions logiques (règle de Dela). Néanmoins, la complexité des fonctions arithmétiques fait apparaitre une limitation forte de cette approche à cause du nombre de patterns à apprendre, qui croit d'une manière exponentielle avec l'augmentation de la largeur des mots binaires à traiter.

Pour cela, deux approches ont été proposées. La première approche est basée sur l'apprentissage de la fonction complète et la deuxième approche consiste à décomposer cette fonction en grain de description plus fin pour s'affranchir des limites de la première approche.

V.1.1 Apprentissage de la fonction complète

Cette approche consiste à apprendre à un réseau multicouche la fonction complète, en se basant sur l'apprentissage hors-ligne (purement logicielle) de rétro-propagation pour définir les fonctions de neurones cachés. Cette approche souffre d'une limitation liée aux nombre de patterns à apprendre qui peut atteindre des valeurs très grandes, selon la complexité de la fonction à apprendre. Cela peut influencer l'architecture du réseau de neurone qui devient de plus en plus complexe, nécessitant un très grand nombre de neurones dans la couche cachée et une très grande résolution des poids synaptiques.

Par exemple un additionneur de deux mots de 3 bits (noté 2x3 par la suite) est définissable par ces $2^{2\times3} = 64$ patterns de couple entrées/sorties. Un additionneur de deux mots de 4 bits (noté 2x4 par la suite) quant à lui possède $2^{2\times4} = 256$ patterns et un additionneur de deux mots de 8 bits (noté 2x8 par la suite) en a $2^{2\times8} = 65536$ patterns.

L'apprentissage d'un additionneur 2x3 bits requiert un réseau possédant 8 neurones dans la couche cachée tandis qu'un additionneur 2x4 bits a besoin de 16 neurones cachés. La Figure II. 19 illustre ainsi une combinaison de valeurs de conductance des 200 memristors pour les deux crossbars en cascade qui permettent d'obtenir un additionneur 2x3 bits. La matrice synaptique de crossbar entre la couche d'entrée et la couche cachée possède 8x(2x(7+1))=128 memristors et la matrice de la couche de sortie en possède 4x(2x(8+1))=72.

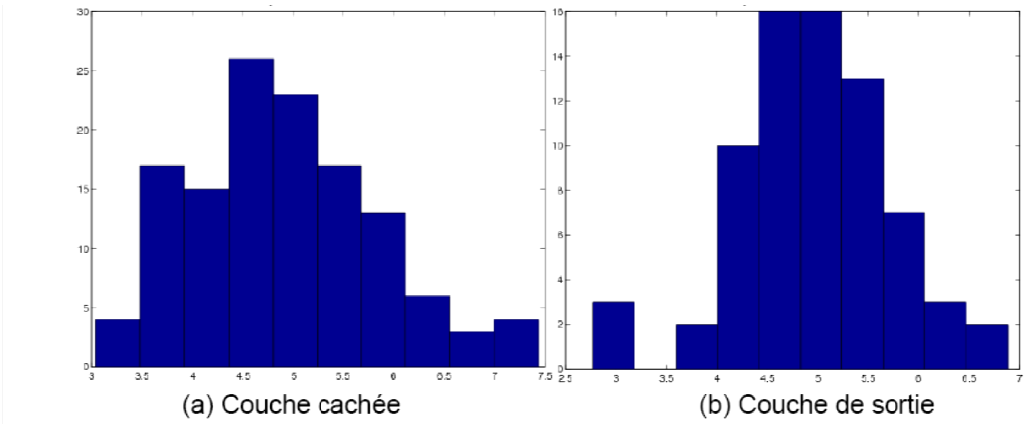


Figure II. 19 : Addittionneur 2x3 bits : histogramme par couche de 200 valeurs de conductances obtenue en simulation sous Matlab, en utilisant le modèle fonctionnel de neuro –crossbar réalisant un apprentissage couche par couche: (a) couche cachée contenant 8 neurones cachés, l'équivalent d'un crossbar à 8x(2x(7+1))= 128 memristors. (b) couche de sortie contenant 4 neurones de sortie l'équivalent d'un crossbar de 4x(2x(8+1))=72 memristors. Toutes les valeurs de conductance sont comprises entre 3 et 7.5.

Un multiplieur de la même taille soit 2x3 bits a besoin de 15 neurones dans la couche cachée ce qui se traduit lors de l'apprentissage matériel par l'utilisation de 402 conductances : la matrice entre la couche d'entrée et la couche cachée contient 15x(2x(6+1))= 210 memristors et la couche de sortie contient 6x(2x(15+1))=192 memristors. L'une des combinaisons possible de valeurs de conductance pour les deux matrices synaptiques est représentée en figure II. 20

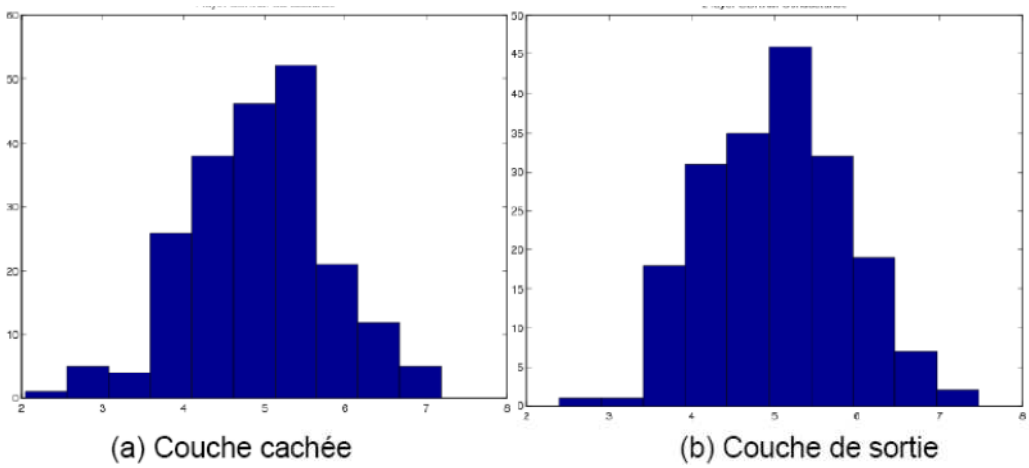


Figure II. 20 : Multiplicateur 2x3 bits : histogrammes par couche de 402 valeurs de conductances obtenue en simulation sous Matlab, en utilisant le modèle fonctionnel de neuro –crossbar réalisant un apprentissage couche par couche, (a) distribution des valeurs de conductances (entre 2 et 7) pour les 210 memristor de la couche cachée, (b) distribution des valeurs de conductance (entre 3 et 7) pour les 192 memristors de la couche de sortie.

La complexité des fonctions, comme par exemple la multiplication, rend cette approche peu susceptible de passer à l'échelle et ne permet pas aisément d'obtenir des solutions optimales. Pour s'affranchir des limitations constatées de cette approche, une approche à grain de description plus fin est proposée.

V.1.2 Approche grain fin

Cette seconde approche utilise des fonctions élémentaires que l'on assemble pour obtenir des fonctions plus complexes. Afin de comparer les deux approches : fonction complète et fonction à grain fin, nous reprenons dans cette partie les deux exemples d'additionneur et de multiplieur entamés précédemment.

Comme le montre la figure II. 21, l'additionneur 2x3 peut être réalisé facilement en cascadant trois additionneurs 2x1 bit (avec retenue d'entrée et de sortie). Seulement 40 connexions synaptiques sont nécessaires pour implémenter un additionneur 2x1 en neuro-crossbar. Cela ne nécessite que 120 connexions synaptiques pour implémenter l'additionneur 2x3. Soit un gain de 40% par rapport à l'approche « apprentissage de la fonction complète » qui nécessite 200 connexions synaptiques.

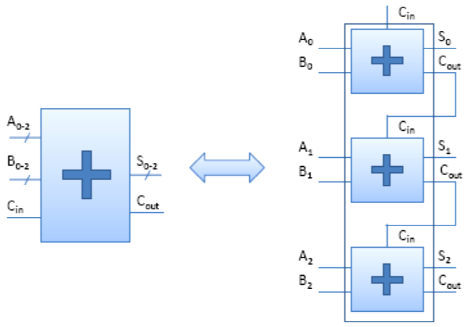


Figure II.21 : Structure d'un additionneur 2x3 avec l'approche grain fin.

Suivant le même principe un multiplieur 2×3 peut être simplifié en utilisant six additionneurs complet 2×1 et neuf fonctions logiques "ET" (figure II.22). L'implémentation de ce multiplicateur en neuro-crossbar ne nécessite que 294 connexions synaptiques. Cela revient à un gain d'environ 26% de nanocomposants par rapport à l'approche « apprentissage de la fonction complète ».

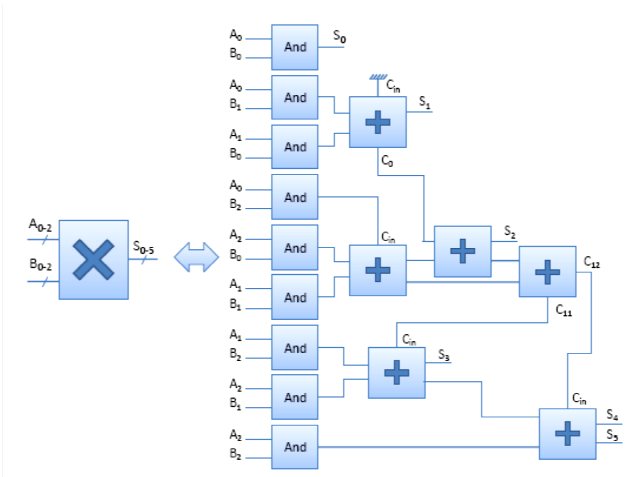


Figure II.22 : Structure d'un Multiplieur 2×3 en utilisant l'approche grain fin.

Ces résultats de simplification que nous venons de présenter montre que l'approche basée sur le grain fin est encore plus performante car elle permet de s'affranchir des problèmes de scalabilité rencontrés lors de la première approche. A partir des résultats de cette approche, nous proposons dans la section suivante une architecture basée sur l'interconnexion de plusieurs éléments de base (neuro-crossbar monocouche) permettant de réaliser une fonction logique complexe.

V.2 Architecture reconfigurable neuro-inspirée : Field Trainable Neural Array (FTNA)

V.2.1 Structure et fonctionnement

L'architecture FTNA proposée ici est l'une des architectures qui peut bénéficier des neuro-crossbars. Elle est Inspirée des architectures reconfigurables FPGA. Elle est donc composée d'un ensemble de blocs neuronaux programmables nommés NLB (Neural Logic Bloc), interconnectés entre eux pour réaliser une fonction complexe (figure II. 23), comme le fait un réseau de neurones multicouche.

Chaque bloc NLB est représenté comme une LUT reconfigurable (Configurable Logic Bloc, CLB) d'un FPGA. Le neuro-crossbar composant le NLB va permettre d'implémenter des fonctions logiques dans chaque bloc. Une fois que le bloc NLB configuré, il est interconnecté aux autres blocs grâce à des switchs programmables nommés PRS (Programmable Routing Switch). Ces derniers sont réalisés à base de la technologie SRAM qui augmente la surface d'occupation, diminuant ainsi l'intérêt de l'utilisation des nanocomposants dans les NLB. Cependant, des travaux sont en cours pour développer des PRS à base de memristors [Cong11]. L'intégration de ce type de switch est compatible avec les blocs NLB qui sont composé de nanocomposants de type memristor.

Au lieu de charger, comme le cas classique les fonctions à programmer dans des LUT-SRAM, le bloc NLB permet de les apprendre grâce à la règle de Delta binaire implémentée dans l'unité d'apprentissage (Figure II.23). Le schéma simplifié du modèle FTNA présenté sur la Figure 5, montre les blocs nécessaires à l'apprentissage des fonctions logiques par les blocs NLB. Ce circuit a besoin de deux étapes avant qu'il soit opérationnel : une étape d'apprentissage des blocs NLB et une étape de routage. Pendant l'étape d'apprentissage, un bloc NLB est sélectionné pour lui apprendre des fonctions logiques (Y_j) qui sont déjà chargées dans la RAM avec la table de vérité correspondante. La machine à état finie (automate) permet de présenter à chaque étape une entrée X_i, de la sortie obtenue X_j avec la sortie désirée Y_j, et d'envoyer par la suite des impulsions (V_{p+}/V_{p-}) afin de modifier les conductances des memristors en se basant sur la règle de Delta binaire décrite dans la section I. Un seul bloc est sélectionné par l'unité d'apprentissage pour être configuré. Avant de passer à la configuration du bloc suivant, l'unité d'apprentissage configure

le Switch PRS qui va relier chaque bloc au bloc suivant afin de réaliser par exemple un perceptron multicouche.

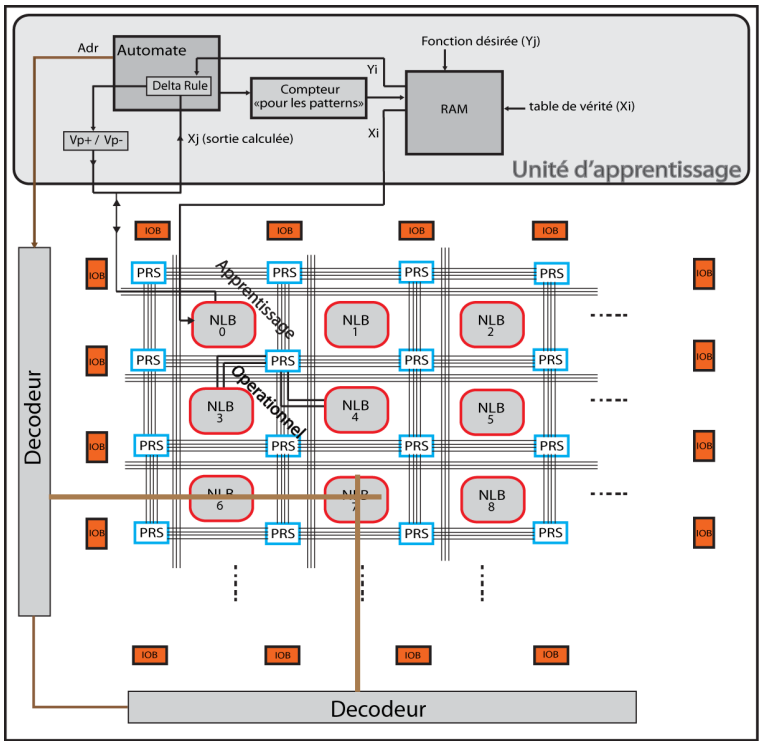


Figure II. 23 : Schéma de l'architecture Field Trainaible Neural Array (FTNA)

Chaque NLB est considéré comme un simple perceptron avec plusieurs sorties. Il ne va pouvoir apprendre que des fonctions logiques simples (linéairement séparables). Pour apprendre une fonction complexe avec ce type d'architecture, une étape de synthèse hors ligne de la fonction à apprendre est nécessaire. Nous avons montré au paragraphe précédent qu'un apprentissage basé sur la rétro-propagation peut être utilisé, mais il est également possible d'adapter des outils de synthèse standard afin de décomposer cette fonction en éléments simple et linéairement séparables (Figure II.24).

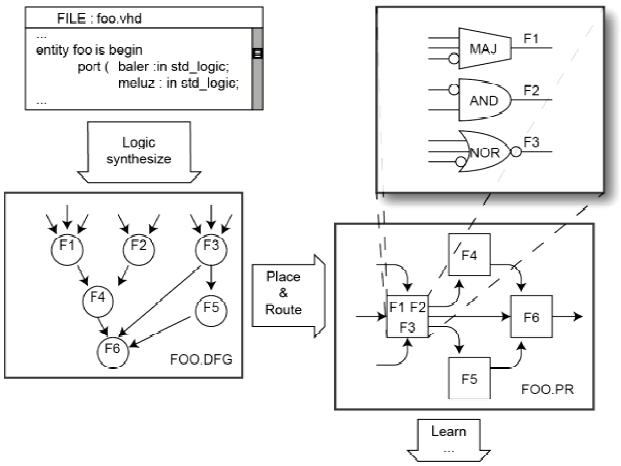


Figure II. 24 : Schéma simplifié d'outils de synthèse des fonctions à implémenter dans l'FPNA.

I.1.1 Comparaison entre un bloc CLB-SRAM et un bloc NLB-memristor :

Un bloc NLB est composé d'un neuro-crossbar. Comme nous l'avons évoqué dans la section 1, un neurocrossbar est composé de deux parties, une partie contenant les synapses et l'autre partie contenant les neurones. L'implémentation de la partie synapses est réalisée à l'aide des nanocomposants (éléments memristifs). Cependant, l'implémentation de la partie neurones comporte deux possibilités. La première possibilité est l'implémentation à base de la technologie CMOS, où le crossbar dans ce cas là est composé d'une partie memristor et d'une partie CMOS, on l'appelle ici « MemCmos » (figure II. 25 (a)). La deuxième possibilité est l'implémentation à base de Latchs memristifs. Contrairement à la première implémentation, le neuro-crossbar dans ce cas là est composé principalement par des nanocomposants memristifs. Cette implémentation est nommée dans cette partie « Mem » (figure II. 25 (b)).

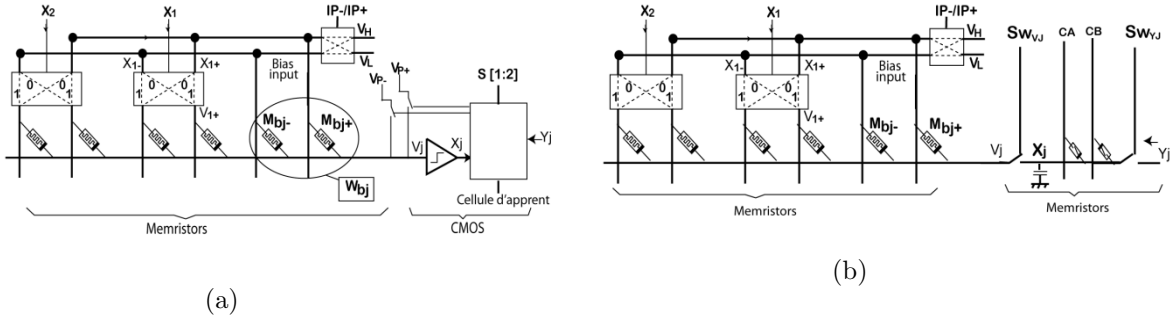


Figure II.25 : schéma d'un neuro-crossbar. (a) MemCmos. (b) Mem.

Pour les deux types d'implémentations, le même nombre de transistors est nécessaire pour réaliser les switchs situés aux niveaux des entrées. Quatre transistors par entrée sont nécessaires pour réaliser chaque switch d'entrée (X_i), ayant comme rôle l'inversion de la polarité des entrées différentielles (X_{i+}, X_{i-}) quand il y a besoin. Quatre autres transistors sont nécessaires pour réaliser le switch permettant d'envoyer la commande (Ip_+/Ip_-) d'inversion de la polarité des entrées. Au total, nous avons besoin de « 4×Nombre d'entrée+4 » transistors pour réaliser le circuit d'entrée pour les deux cas d'implémentation « Mem » ou « MemCmos ».

Concernant la sortie de neuro-crossbar et dans le cas de l'implémentation « MemCmos » le nombre de transistors est de sept par sortie : un transistor permettant de commuter entre la programmation et la lecture de la sortie, deux transistors permettant d'appliquer les impulsions de programmation Vp_+ et Vp_- et au moins quatre transistors pour réaliser le comparateur dédié au seuillage de la sortie. Dans le cas de l'implémentation « Mem », le nombre de transistors par sortie est réduit à deux, plus les deux memristors de latch.

Le nombre de memristors dans le cas d'un crossbar implémenté avec « MemCmos » est représenté par le nombre de connexions synaptiques reliant les

entrées aux sorties du réseau. Etant donné que les entrées sont différentielles, deux memristors par entrée sont nécessaires plus deux memristors pour réaliser l'entrée de seuil, ce qui donne un total de memristors de « 2×nombre d'entrée +2 ». Dans le cas d'un crossbar implémenté avec « Mem » il faut rajouter à ce nombre les deux memristors de Latch placés dans chaque sortie.

Afin de comparer la surface que peut occuper un neuro-crossbar à celle que peut occuper un bloc CLB basé sur les SRAM, nous avons calculé le nombre de transistors (Tr) et le nombre de memristors (M), nécessaires pour implémenter un neuro-crossbar pour les deux types d'implémentation. Ces résultats sont comparés aux nombre de transistors nécessaires pour implémenter une cellule LUT-SRAM basique (figure II. 26) [Ws09].

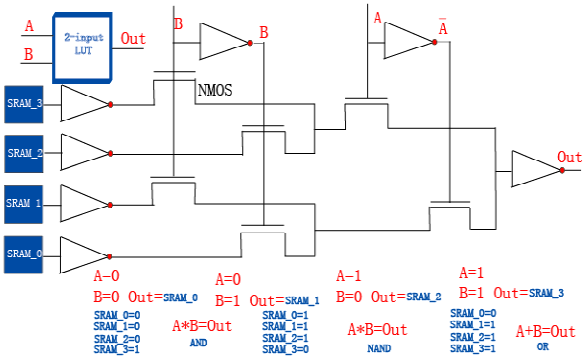


Figure II. 26: Exemple d'une LUT-SRAM à deux entrées.

Les différents résultats de calcul sont présentés dans le tableau II.9 pour plusieurs nombres d'entrée/sortie. Nous précisons que les différents calculs effectués ici pour définir le nombre de transistors et de memristors de neuro-crossbar restent une estimation. La réalisation d'une étude précise déterminant de manière définitive le nombre d'éléments nécessaires pour réaliser cette architecture devrait voir le jour prochainement.

En prenant en compte la taille nanométrique des nanocomposants utilisés pour notre approche neuro-inspirée, et la possibilité d'avoir des grandes densités d'intégration avec une flexibilité de configuration obtenue grâce à l'apprentissage neuronal, l'approche neuro-inspirée que nous venons de proposer est très compétitive par rapport aux approches basées sur la technologie CMOS. Le remplacement des cellules CLB d'un circuit FPGA par des cellules NLB va permettre de diminuer la surface occupée par le circuit FPGA.

Config	2 entrées	3 entrées	4 entrées	5 entrées	
Mem	2Tr+8M	2Tr+10	2Tr+12	2Tr+14	1 sortie
MemCmos	19Tr+6M	23Tr+8M	27Tr+10M	31Tr+12M	
SRAM	34	66	132	262	
Mem	2Tr+16M	2Tr+20M	2Tr+24M	2Tr+28M	2 sorties
MemCmos	26Tr+12M	30Tr+16M	34Tr+20M	38Tr+24M	
SRAM	68	132	264	524	
Mem	2Tr+24M	2Tr+30M	2Tr+36M	2Tr+42M	3 sorties
MemCmos	33Tr+18M	37Tr+24M	41Tr+30M	45Tr+36M	
SRAM	102	198	396	786	
Mem	2Tr+32M	2Tr+40M	2Tr+48M	2Tr+56M	4 sorties
MemCmos	40Tr+24M	44Tr+32M	48Tr+40M	52Tr+48M	
SRAM	136	264	528	1048	

Tableau II. 9 : comparaison entre le nombre de transistors dans une CLB fabriquée à base de SRAM avec le nombre de transistors et de memristors d'un NLB. Tr : transistors CMOS, M : memristor, pour les deux implémentations possibles, Mem et MemCmos.

V.2.2 Modèle FTNA sous VHDL

Un modèle de circuit FTNA a été décrit en VHDL. Il réalise un réseau de neurone multicouche composé de trois couches NLB (NLB0, NLB1, NLB2) mises en cascade. Un bloc d'adaptation est placé entre les blocs permettant de brancher les sortie du bloc NLB0 (ou NLB1) aux entrées différentielles du bloc NLB1 (ou NLB2).

L'objectif de ce modèle est de montrer la faisabilité d'un circuit à base de crossbar capable d'implémenter des fonctions logiques non linéairement séparables avec une grande rapidité d'apprentissage. Un tel modèle peut être amélioré et utilisé dans les études dédiées à l'amélioration de la capacité de traitement et la fiabilité des coprocesseurs et des circuits reconfigurables en général.

Chacun des trois blocs NLB est composé de six entrées logiques et six neurones de sortie. Les fonctions à apprendre à chaque bloc sont présentés dans le tableau II.10. Les neurones de sortie de NLB0 vont apprendre des fonctions logiques linéairement séparables. Ces fonctions correspondant à la couche cachée d'un perceptron multicouche, définies auparavant grâce à un apprentissage logiciel basé sur la rétro-propagation du gradient, de façon à apprendre ensuite aux neurones des couches suivantes (NLB1, NLB2) les trois fonctions « Xor3, Maj3, Mux2-1 ». Les neurones excédentaires vont apprendre la fonction zéro. Etant donné que les fonctions que nous avons choisies d'apprendre aux différents blocs sont à trois entrées « sur huit bits », la table de vérité des entrées de bloc NLB0 va être sur 3 bits (X_3, X_2, X_1). Les entrées qui ne sont pas utilisées (X_6, X_5, X_4) sont mises à «-1».

Fonctions désiré NLB0	Fonctions désiré NLB1	Fonctions désiré NLB2
NLB0_Y1 <= "11101000"	NLB1_Y1 <= "01101001"	NLB2_Y1 <= "01101001"
NLB0_Y2 <= "11100000"	NLB1_Y2 <= "00010111"	NLB2_Y2 <= "00010111"
NLB0_Y3 <= "00010111"	NLB1_Y3 <= "00011011"	NLB2_Y3 <= "00011011"
NLB0_Y4 <= "11000100"	NLB1_Y4 <= "00000000"	NLB2_Y4 <= "00000000"
NLB0_Y5 <= "01000101"	NLB1_Y5 <= "00000000"	NLB2_Y5 <= "00000000"
NLB0_Y6 <= "00001010"	NLB1_Y6 <= "00000000"	NLB2_Y6 <= "00000000"

Tableau II.10 : Fonctions désirées à apprendre par les trois blocs NLB.

Les valeurs des trois matrices de conductances ont été récupérées après convergence de l'apprentissage :

Matrice de conductance du bloc NLB0 :

Entrées / Sorties	X_{b+}	X_{6+}	X_{5+}	X_{4+}	X_{3+}	X_{2+}	X_{1+}	X_{b-}	X_{6-}	X_{5-}	X_{4-}	X_{3-}	X_{2-}	X_{1-}
Nlb0_Y_1	10	9	9	9	8	8	8	8	9	9	9	10	10	10
Nlb0_Y_2	8	9	9	9	6	8	8	10	9	9	9	12	10	10
Nlb0_Y_3	11	9	9	9	11	11	11	7	9	9	9	7	7	7
Nlb0_Y_4	7	9	9	9	7	5	11	11	9	9	9	11	13	7
Nlb0_Y_5	8	9	9	9	10	6	12	10	9	9	9	8	12	6
Nlb0_Y_6	9	9	9	9	11	9	7	9	9	9	9	7	9	11

Matrice de conductance du bloc NLB1 :

Entrées / Sorties	X_{b+}	Nlb0 Y_{6+}	Nlb0 Y_{5+}	Nlb0 Y_{4+}	Nlb0 Y_{3+}	Nlb0 Y_{2+}	Nlb0 Y_{1+}	X_{b-}	Nlb0 Y_{6-}	Nlb0 Y_{5-}	Nlb0 Y_{4-}	Nlb0 Y_{3-}	Nlb0 Y_{2-}	Nlb0 Y_{1-}
Nlb1_Y_1	9	11	9	7	7	13	9	9	7	9	11	11	5	9
Nlb1_Y_2	9	7	9	11	9	9	9	9	11	9	7	9	9	6
Nlb1_Y_3	8	8	8	10	6	8	12	10	10	10	8	12	10	8
Nlb1_Y_4	6	8	10	10	8	10	10	12	10	8	8	10	8	8
Nlb1_Y_5	6	8	10	10	8	10	10	12	10	8	8	10	8	8
Nlb1_Y_6	6	8	10	10	8	10	10	12	10	8	8	10	8	8

Matrice de conductance du bloc NLB2 :

Entrées / Sorties	X_{b+}	Nlb1 Y_{6+}	Nlb1 Y_{5+}	Nlb1 Y_{4+}	Nlb1 Y_{3+}	Nlb1 Y_{2+}	Nlb1 Y_{1+}	Xb_	Nlb1 Y_{6-}	Nlb1 Y_{5-}	Nlb1 Y_{4-}	Nlb1 Y_{3-}	Nlb1 Y_{2-}	Nlb1 Y_{1-}
Nlb2_Y_1	9	13	9	9	9	9	9	9	5	9	9	9	9	9
Nlb2_Y_2	9	9	13	11	9	9	9	9	9	5	7	9	9	9
Nlb2_Y_3	9	9	9	11	9	9	9	9	9	9	7	9	9	9
Nlb2_Y_4	9	9	9	9	9	9	9	9	9	9	9	9	9	9
Nlb2_Y_5	9	9	9	9	9	9	9	9	9	9	9	9	9	9
Nlb2_Y_6	9	9	9	9	9	9	9	9	9	9	9	9	9	9

VI. Conclusion

Une large gamme de nanocomposants est en voie de développement actuellement dans le but de réaliser dans le future des architectures électroniques plus performantes et moins coûteuses. Dans ce chapitre, une architecture neuro-inspirée basée sur un modèle idéal de nanocomposant (memristor) a été simulée, montrant la possibilité d'implémenter des fonctions logiques et arithmétiques. Cette architecture nommée neuro-crossbar est composée de deux parties, la première partie contient la matrice de nanocomposants reconfigurable à l'image des synapses dans un réseau de neurones artificiel et la deuxième partie contient les neurones de sortie et la cellule d'apprentissage permettant d'implémenter la technique d'apprentissage pour configurer la matrice des synapses. Deux circuits ont été proposés pour l'implémentation de ces cellules d'apprentissage, la première est basée sur la technologie CMOS classique et la deuxième est basée sur le latch composé de memristors. Afin de démontrer la faisabilité de la conception dans ce contexte idéal, nous avons effectué des simulations de l'apprentissage de quelques fonctions logiques à partir d'un modèle fonctionnel de neuro-crossbar développé sous Matlab. Les résultats de simulation ont montré la capacité d'un neuro-crossbar monocouche à apprendre des fonctions logiques linéairement séparables avec une convergence très rapide. La mise en cascade de ces neuro-crossbar monocouche a permis de réaliser un réseau multicouche capable d'apprendre les fonctions non linéairement séparables. Ces résultats ont démontré la possibilité d'interconnecter des petits blocs de neuro-crossbar pour réaliser des fonctions plus complexes. Dans ce contexte, une architecture nommée FTNA a été proposée permettant d'assembler des blocs neuro-crossbar (NLB) dans un même circuit.

Les simulations de ce chapitre ont été réalisées à partir d'un modèle de neuro-crossbar supposé idéal sans défauts ni de dispersions de caractéristiques des nanocomposants. Cependant, le but principal d'utilisation des réseaux de neurones pour ce type d'architecture est leur robustesse intrinsèque permettant l'autocorrection des défauts et des dispersions de caractéristiques des nanocomposants qui peuvent être présent dans l'architecture neuro-crossbar. Cet aspect sera considéré dans le chapitre suivant, en étudiant le degré de tolérance à plusieurs types de défauts et de variations des neuro-crossbar.

CHAPITRE III

Fiabilité des architectures neuro-crossbar

I. Introduction

Les nouvelles techniques de fabrication nanométriques semblent apporter des solutions aux limitations des technologies CMOS. Elles permettent de fabriquer des circuits à haute densité, en utilisant des techniques de fabrication chimiques d'auto-assemblage qui offrent des possibilités de réduire les coûts élevés des technologies de fabrication classiques [Gree01, Gold01, Brown07]. L'auto-assemblage et l'auto-alignement sont généralement utilisés pour fabriquer des architectures crossbars à deux dimensions [Huan04, Taho05]. Malgré les avantages de ces nouvelles techniques, le taux de défauts de la nanofabrication reste élevé (de l'ordre de 10%) par rapport à celui présent dans les circuits basés sur le CMOS [Stan03, Wang07, Gold03], ce qui diminue le rendement et la fiabilité de ce type d'approche.

Il est donc primordial de prendre en compte de la tolérance aux fautes et de trouver les techniques qui permettent de fiabiliser ces futurs dispositifs. Plusieurs techniques de tolérance aux fautes ont été développées afin d'augmenter la fiabilité des architectures basées sur les nanocomposants [Heat98, Niko02, Han03, Huan04, Bhad05, Chen05, Kuek05, Taho05, Hogg06, Huan06, Rao06, Tehr08]. Ces méthodes sont fondées sur des travaux déjà anciens tels que la redondance modulaire (TMR ou CMR) [Niko02, Kuek05, Bhad05] et le multiplexage de Von Neumann [Han04, Sad04, Niko01]. L'inconvénient majeur de ces techniques réside dans leur mise en œuvre qui nécessite des surfaces importantes et limite donc leur intérêt en nanoarchitectures. Une autre technique de tolérance aux fautes basée sur la reconfiguration est mieux adaptée aux nanoarchitectures. Cette technique se devise en deux étapes : une première étape de test permettant de détecter et localiser les modules défaillants et une deuxième étape de reroutage qui se base sur la carte de défauts conçue pendant la première étape pour restructurer l'architecture et réaliser un système fiable. Ces deux étapes peuvent s'effectuer soit avant de mettre le circuit en route, c'est le cas de la reconfiguration statique [Heat98, Chen05, Gold03, Taho05], soit pendant la phase opérationnelle du circuit, c'est le cas de la reconfiguration dynamique [Cot05, Dur01]. Cependant la haute densité des nanocomposants constitue un défi majeur pour réaliser la carte des défauts. Le temps nécessaire pour l'étape de test et la mémoire nécessaire pour stocker la carte des défauts rend cette technique de reconfiguration peu exploitable. Une autre façon de faire face à ces difficultés consiste à utiliser des codes correcteurs d'erreur ou des techniques d'auto-test intégré, comme l'approche « BIST- Built In Self-Test » [Kar89, Wang07] permettant de réduire les temps de test, mais là encore il est difficile de réaliser une structure de « BIST » compacte permettant l'intégration dans les architectures nanométriques en plus de ça, ces techniques ne sont pas dédiées à tolérer des taux de défaut élevés comme dans le cas des nanocomposants.

Compte tenu de ces limitations, nous sommes amenés à étudier une autre alternative de tolérance aux fautes dans les architectures nanométriques. Déjà évoqués dans les deux chapitres précédents, les réseaux de neurones artificiels (RNA) sont considérés intrinsèquement redondants et tolérants aux défauts [Emm91, Ker94, Ass97].

Nous allons présenter tout au long de ce chapitre l'étude de la robustesse de notre architecture neuro-crossbar (NC). La plupart des défauts et variations technologiques considérés sont des « défauts statiques » ou des variations technologiques qui peuvent intervenir soit au niveau des neurones soit au niveau des synapses. L'influence de chacun de ces défauts et dispersions sur la convergence de NC va être traitée séparément. Les simulations Monte-Carlo développées sous Matlab® vont permettre de mesurer le taux de convergence de l'apprentissage de NC pour apprendre une fonction logique en présence de l'un de ces défauts et variations technologiques. Un modèle probabiliste de prédiction de convergence de l'apprentissage à été proposé pour chaque type de défauts ou de dispersion.

II. Robustesse des Réseaux de Neurones Artificiels (RNA)

II.1 Robustesse

La robustesse ou la tolérance aux fautes d'un système signifie que celui-ci est capable de fonctionner et fournir une réponse correcte en présence de fautes. Une faute peut être une panne ou un défaut d'un élément matériel ou logiciel, provoquant ainsi une erreur dans le système.

L'étude de la robustesse des RNA revient généralement à étudier leur comportement dans deux cas :

1/ Les défauts et variations survenant sur leurs composants internes, les neurones et les synapses.

2/ Le bruit survenant sur les informations présentées à l'entrée ou sur les poids synaptiques.

Pour les architectures classiques, les erreurs peuvent provoquer des défaillances. Dans le cas des architectures basées sur les RNA, les erreurs conduisent plutôt à des dégradations progressives des performances qui s'expliquent par les propriétés spécifiques aux RNA [Ass97].

II.1.1 Redondance

La redondance, aussi bien au niveau des neurones que des synapses, est considérée comme le facteur le plus important pour augmenter la fiabilité des RNA [Emm91]. En effet, à l'image des réseaux de neurones biologiques nous trouvons deux types de redondance : la redondance spatiale où les éléments (neurones, synapses) sont répliqués, et la redondance temporelle où les opérations de calcul sont répétées [Moor88].

II.1.2 Nature distribuée de l'information

Le traitement de l'information dans un RNA est distribué partout dans le réseau. Chaque neurone et chaque synapse contribue d'une façon directe ou indirecte à l'élaboration d'une petite partie de la réponse finale. Par conséquent, la perte d'un neurone ou d'une synapse en présence de redondance suffisante ne devrait pas affecter de manière significative la réponse finale du réseau. En effet, la nature distribuée de l'information fait que les fautes vont être supportées par tous les éléments du réseau au lieu de l'être par un élément isolé. Une dégradation à un endroit donné du réseau est compensée par le traitement effectué dans toutes les autres parties du réseau, conduisant ainsi à une dégradation progressive plutôt qu'à une défaillance catastrophique [Emm92].

II.1.3 Présence de rétroaction

La présence d'élément de contre-réaction va faire apparaitre une redondance temporelle qui permet à posteriori une correction de la valeur de sortie du réseau, contribuant ainsi à une amélioration de la tolérance du RNA [Moor88].

II.2 Techniques d'amélioration de la robustesse des RNA

Diverses études ont été effectuées en vue d'améliorer la tolérance aux fautes des RNA grâce à l'apprentissage [Ass97, Ker94]. Par exemple, des études ont montré que l'ajout du bruit sur les entrées de RNA lors de l'apprentissage augmente sa tolérance [Mats94]. En effet, lorsque l'apprentissage est fait en présence de bruit en entrée, les poids synaptiques sont ajustés de telle sorte à éliminer ce bruit en phase de reconnaissance de l'information [Cun89]. Une autre méthode pour améliorer la tolérance des RNA est le réapprentissage. En cas d'erreur, telle que la défaillance d'un neurone ou une synapse, un réapprentissage permet de restaurer la précision du réseau. Dans ce contexte, différentes études ont montré que le perceptron multicouches (MLP), via un réapprentissage adéquat, est capable de tolérer des taux de défauts considérables et d'atteindre sa performance initiale [Tana89, Bed88]. Une autre technique a été proposée dans [Emm91] qui consiste à augmenter la taille des couches cachées en dupliquant les neurones et les synapses. Une meilleure robustesse y est observée pour cette technique due à la compensation des erreurs sur les nœuds sensibles par les nœuds ajoutés. Les nœuds sensibles sont ceux qui ont des poids

synaptiques plus grand puisque ces derniers peuvent « amplifier » les erreurs survenant dans le réseau. Dans ce contexte, une approche a été proposée dans [Dar89] permettant de diminuer la sensibilité de ces nœuds en distribuant leur sensibilité sur tout le réseau. De ce fait, la sensibilité concentrée sur un nœud, se trouvera répartie sur plusieurs après réapprentissage. Pour cela, certains auteurs proposent de limiter les poids des connexions synaptiques à des valeurs faibles, et d'autres proposent de limiter la précision des poids [Choi93, Saka93].

III. Robustesse des Neuro-Crossbars (NC)

III.1 Neurones défaillants dans un NC multicouches

Afin d'étudier l'influence de la présence de neurones défaillants dans l'architecture neuro-crossbar, nous avons choisit de simuler l'apprentissage couche par couche des mêmes fonctions logiques présentées dans le chapitre II (tableau II.4), mais cette fois ci avec un réseau contenant des neurones défaillants dans la couche cachée (figure III.1 (a)). L'une des difficultés que nous rencontrons ici provient de la méthode d'apprentissage utilisée, qui consiste à désigner pour chaque neurone caché la fonction à apprendre. Pour éviter que la défaillance d'un neurone caché ne soit rédhibitoire pour l'apprentissage, nous avons choisi de ne pas fixer a priori de placement de chaque fonction cachée sur un neurone caché, mais au contraire de laisser le réseau faire cette association en réalisant un **apprentissage compétitif** (figure III.1 (d)), où chaque fonction est apprise en concurrence sur tous les neurones disponibles jusqu'à ce que l'un d'eux présente un fonctionnement correct. Ce dernier sera figé et retiré de la liste des neurones disponibles, alors que les autres fonctions seront apprises par les neurones disponibles. Ainsi, si quelques neurones défaillants sont supportés par quelques neurones redondants, ils resteront simplement «disponibles». L'algorithme de l'apprentissage compétitif est donné ci-dessous :

initialement, l'ensemble de neurones A =N_0 et l'ensemble de fonction P=F

- Tant que (P est non vide){
- sélectionner une fonction f_i de P ;
- Tous les neurones de A essaye d'apprendre la même fonction f_i de P ;
- Quand n_j, l'un des neurones disponible réussit à apprendre la fonction sélectionnée f_i, le neuron n_j est assigné à cette fonction f_i ;
- Suppression de neuron n_j de l'ensemble A ;
- Suppression de la fonction f_i de l'ensemble P }.

Il reste à s'assurer que ces neurones défaillants ne perturberont pas les neurones de la couche suivante « de sortie », ce que nous allons vérifier ici. Nous supposons

dans cette partie que les défauts qui affectent les synapses (conductances) ou les entrées du réseau, sont représentés soit par un collage à «-1 » ou « 1 » d'un neurone soit par un neurone dont l'état est aléatoire noté « ? ».

Dans les simulations dont les résultats suivent, nous avons introduit de la redondance au niveau des neurones de la couche cachée (figure III.1) et simulé des défauts sur plusieurs neurones et synapses afin de démontrer la robustesse de neuro-crossbar multicouche. Grâce à l'apprentissage matériel (règle de Delta) couche par couche, les neuf neurones de la couche cachée (Nh1...Nh9) vont se mettre en concurrence pour apprendre les six fonctions cachées (f1...f6). Parmi ces neuf neurones, trois neurones sont défaillants (Nh2, Nh5, Nh9) (figure III.1 (b)). Une fois l'apprentissage de la couche cachée terminé, les trois neurones de sortie Y1, Y2 et Y3 vont apprendre respectivement les trois fonctions Xor3, Maj3 et Mux2-1 (figure III.1 (c)).

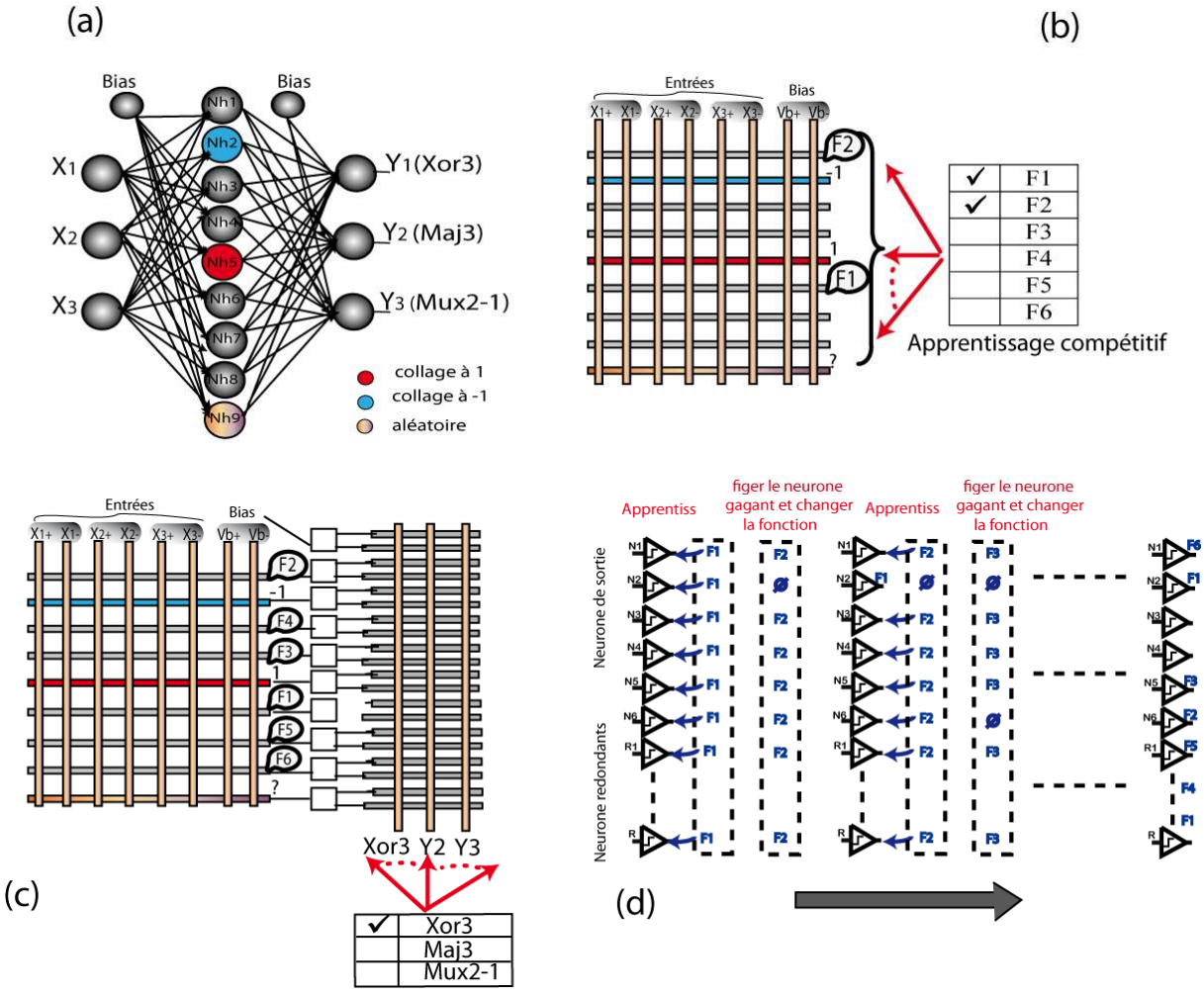


Figure III.1 : (a) schéma d'un réseau de neurone MLP contenant des neurones cachés défaillants. (b) apprentissage hardware des neurones de la première couche de neuro-crossbar. (c) apprentissage hardware de la couche de sortie d'un neuro-crossbar à deux couches mises en cascade en présence de neurones défaillants dans la couche cachée. (d) Schéma des étapes de l'apprentissage compétitif.

Pour la réalisation des simulations de cette partie, nous avons adapté le modèle fonctionnel de l'apprentissage couche par couche de neuro-crossbar déjà développé dans le chapitre précédent. Nous avons ajouté une partie permettant d'intégrer les défauts dans les synapses et dans les neurones, et une autre partie permettant d'intégrer l'apprentissage compétitif afin d'éviter les différents neurones défaillants.

Un neuro-crossbar est caractérisé par sa taille ($N_i \times N_o$) où $N_i=2\times N_{li}+2$ est le nombre d'entrées physiques incluant le seuil et N_{li} est le nombre d'entrées logiques et N_o est le nombre de sorties totale incluant les sorties redondantes. Dans la suite un neuro-crossbar à N_i entrées et N_o sortie sera noté *NC* $N_i \times N_o$.

Le même modèle de synapse basé sur le memristor « -0+ » est utilisé dans toutes les simulations qui suivent. Dans la plupart des cas, la conductance des memristors G varie dans un intervalle positif entre une valeur $G_{min}\approx 0$ et $G_{max}\approx 10$ avec un pas d'incrémentation $G_{inc}=1$. Dans le cas d'une initialisation aléatoire ou dans le cas de l'étude de variation de la conductance, la valeur de la conductance est tirée au sort (par la fonction Matlab *randn*). Nous appliquons dans un cas pareil la fonction $f^+(x)$ qui permet de ramener les valeurs de conductance négatives à $G_{min}\approx 0$.

$$f^+(x)=\frac{1}{2}(x+|x|) \qquad \text{(III.1)}$$

III.1.1 Résultats de simulation de l'apprentissage des neurones cachés

La première étape de l'apprentissage couche par couche est la configuration de la matrice située entre la couche d'entrée et la couche cachée en apprenant aux neurones cachées Nhi les différentes fonctions f_i. Les paramètres de simulation de cet apprentissage sont décrits dans le tableau suivant :

Taille de neuro-crossbar	8×9
Nombre de fonction à apprendre Nombre de neurones redondants	Fh=6 (f_1, f_2, f_3, f_4, f_5, f_6). $Rh=N_h-F_h=9-6=3$.
Matrice de conductance G	initialisée à $G=f^+(randn(N_i, N_h)+G_{max})$, $G_{min}=0$, $G_{max}=10$, $G_{inc}=1$.
Nombre de répétitions d'apprentissage (époques)	$N_{epoq}=20$.
Modèle de composant synaptique	Memristor « -0+ », tension de seuil V_T/ $-V_T=1/-1$, tension de programmation $V_P/-V_P=1/-1$, tension de lecture $V_H/V_L=0.4/-0.4$.
Défauts considérés	Toutes les conductances de la ligne Nh2 sont collées à G_{min} ($G(2\ ,\ :)=G_{min}$). Les conductances positives de la ligne Nh5 sont collées à G_{max} et les conductances négatives sont collées à zéro. ($G(5,\ 1:4)=G_{max}$, $G(5,\ 5:8)=G_{min}$). L'état du neurone Nh2 est collé à « -1 ». L'état du neurone Nh5 est collé à « 1 ». L'état du neurone Nh9 est aléatoire (probabilité d'avoir V_H ou V_L est de $1./(1+\exp^{(-Vhj(9)/T)})$ avec T=100).

Tableau III.1 : Paramètres de simulation de l'apprentissage des neurones cachés.

La figure III.2 montre l'évolution de l'erreur sur les huit patterns (2^{Ni}) présentés pour chaque neurone durant 20 époques d'apprentissage. L'apprentissage est très rapide et la présence de neurones défaillants n'influe pas sur l'apprentissage. Grâce à l'apprentissage compétitif, les six neurones opérationnels ont appris les six fonctions et les trois neurones défaillants (Nh2, Nh5, Nh9) ont été évités. Nous pouvons voir sur la figure III.2 que l'apprentissage des fonctions *fi* ne s'est pas fait dans l'ordre des neurones, Nh1 par exemple a appris f_2, Nh6 a appris f_1, du fait de l'apprentissage compétitif.

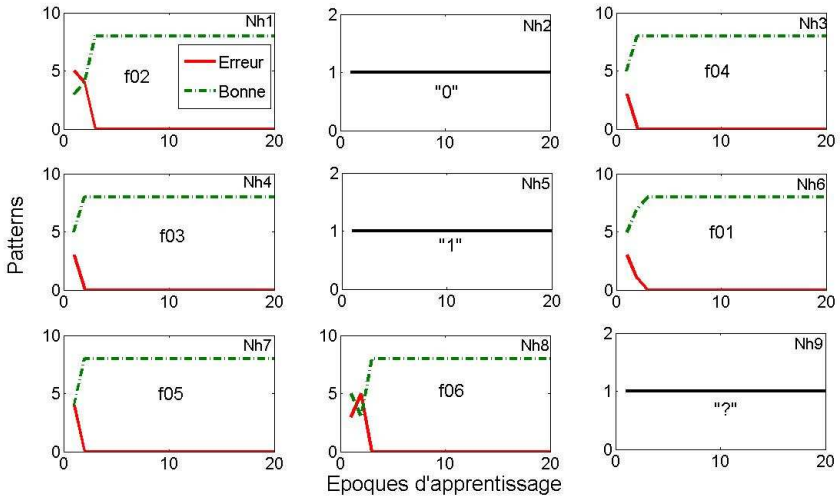


Figure III.2 : résultats de simulation de l'apprentissage des neurones cachés

Le tableau III.2 montre les valeurs de conductances de la matrice des synapses situées entre la couche d'entrée et la couche cachée à la fin de l'apprentissage.

	f_2	« -1 »	f_4	f_3	« 1»	f_1	f_5	f_6	« ? »
	N_{h1}	N_{h2}	N_{h3}	N_{h4}	N_{h5}	N_{h6}	N_{h7}	N_{h8}	N_{h9}
B_+	2	G_{min}	6	8	G_{max}	8	6	2	4
X_{3+}	2	G_{min}	10	4	G_{max}	10	4	4	6
X_{2+}	8	G_{min}	10	10	G_{max}	10	2	8	4
X_{1+}	4	G_{min}	4	2	G_{max}	8	2	8	10
B_-	10	G_{min}	6	4	G_{min}	8	6	10	6
X_{3-}	10	G_{min}	3.65	6	G_{min}	5.81	6	8	4
X_{2-}	7.04	G_{min}	2	4.40	G_{min}	6	8.81	2	6
X_{1-}	8	G_{min}	8	10	G_{min}	6	8	2	0

Tableau III.2 : Valeurs de conductance de la matrice située entre la couche d'entrée et la couche cachée, G_{max}=10 et G_{min}=0.

Les poids synaptiques (W) correspondant à la matrice de conductance précédente sont calculés grâce à l'expression $W=G_{+}-G_{-}$ et donnés par le tableau suivant :

	N_{h1}	N_{h2}	N_{h3}	N_{h4}	N_{h5}	N_{h6}	N_{h7}	N_{h8}	N_{h9}
B	-8	W_{min}	0	4	W_{max}	0	0	-8	-2
X3	-8	W_{min}	6.35	-2	W_{max}	4.19	-2	-4	2
X2	0.95	W_{min}	8	5.6	W_{max}	4	-6.81	6	-2
X1	-4	W_{min}	-4	-8	W_{max}	2	-6	6	10

Tableau III.3 : Valeurs de poids synaptique (W) déduit de la matrice des conductances de tableau III.2 grâce à l'expression $W=G_{+}-G_{-}$. Dans le tableau W_{max} correspond à la valeur 10 et W_{min} à 0.

III.1.2 Résultats de simulation de l'apprentissage des neurones de sortie

Le modèle de neuro-crossbar présenté dans cette partie est composé des deux couches. La première couche située entre la couche d'entrée et la couche cachée, déjà configurée dans la partie précédente, est utilisée ici pour calculer l'état des neurones cachés. Ces neurones cachés vont alimenter la deuxième couche pour calculer l'état des neurones de sortie. La deuxième étape de l'apprentissage couche par couche est la configuration de la matrice située entre la couche cachée et la couche de sortie en apprenant aux neurones de sortie Y_1, Y_2 et Y_3 les trois fonctions Xor3, Maj3 et Mux2-1. Les paramètres de simulation de cet apprentissage sont décrits dans le tableau suivant :

Taille de neuro-crossbar	20×3
Nombre de fonctions à apprendre Nombre de neurones redondants	F=3 (Xor3, Maj3 , Mux2-1.). $R_h=N_o-F=3-3=0$.
Matrice de conductance G	initialisée à $G=f^{+}(randn(N_h,N_o)+G_{max})$, $G_{min}=0$, $G_{max}=10$, $G_{inc}=1$.
Nombre de répétitions d'apprentissage (époques)	$N_{epoq}=20$.
Modèle de composant synaptique	Memristor « -0+ », tension de seuil V_T/ $-V_T=1/-1$, tension de programmation $V_P/-V_P=1/-1$, tension de lecture $V_H/V_L=0.4/-0.4$.
Défauts considérés	Trois entrées défaillantes : L'entrée N_{h1} est collé à « -1 ». L'entrée N_{h5} est collé à « 1 ». L'entrée N_{h9} est aléatoire (probabilité d'avoir V_H ou V_L est de $1./(1+\exp^{(-Vhj(9)/T)})$ avec T=100).

Tableau III.4 : Paramètres de simulation de l'apprentissage des neurones de sortie.

L'évolution de l'erreur commises par les trois neurones de sortie durant les vingt époques d'apprentissage est donnée par la figure III.3. Les trois neurones ont convergé

avec un nombre d'époques très petit, peu différent du cas idéal sans défaut (§1 section IV.2.2).

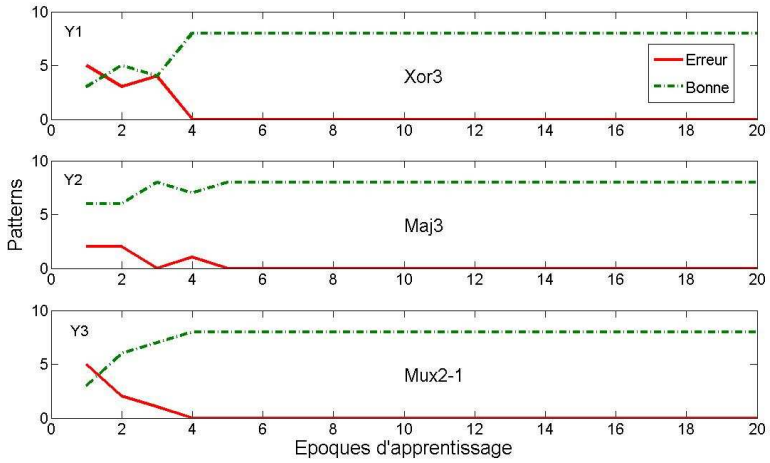


Figure III.3 : Résultats de simulation de l'apprentissage des neurones de sortie.

Les conductances de la matrice entre la couche cachée et la couche de sortie après convergence sont données dans le tableau suivant :

	Bh_+ Bh_-	$Nh1_+$ $Nh1_-$	$Nh2_+$ $Nh2_-$	$Nh3_+$ $Nh3_-$	$Nh4_+$ $Nh4_-$	$Nh5_+$ $Nh5_-$	$Nh6_+$ $Nh6_-$	$Nh7_+$ $Nh7_-$	$Nh8_+$ $Nh8_-$	$Nh9_+$ $Nh9_-$
Y1	10 6	6 8	6 10	10 0	0 10	10 5.81	2 8	8 2	10 4	6 8
Y2	7.42 10	8 8	10 7.16	10 8	7.85 8	8 10	10 0	0 10	10 7.12	8 10
Y3	8 8	2 10	8 8	8 2	10 0	8 8	10 5.29	6 10	10 1.57	4 10

Tableau III.5 : Valeurs de conductance de la matrice située entre la couche cachée et la couche de sortie.

Les poids synaptiques (W) correspondant à la matrice de conductance précédente sont donnés par le tableau suivant :

	Bh	Nh1	Nh2	Nh3	Nh4	Nh5	Nh6	Nh7	Nh8	Nh9
Y1	4	-2	-4	10	-10	4.19	-6	6	6	-2
Y2	-2.58	0	2.83	2	-0.15	-2	10	-10	2.88	-2
Y3	0	-8	0	6	10	0	4.7	-4	8.43	-6

Tableau III.6 : Valeurs de poids synaptique (W) déduit de la matrice des conductances de tableau III.4 grâce à l'expression$W=G_+-G_-$.

Le calcul de la matrice synaptique entre la couche cachée et la couche de sortie nous a permis de comparer les différentes valeurs de poids synaptiques reliant les

neurones cachés aux neurones de sortie. Nous observons sur le tableau précédent que le neurone Nh5 collé à « 1 » se comporte comme un seuil : le jeu de poids qui lui associé est sensiblement identique à ceux associés au neurone de seuil (Bh), et le neurone Nh2 collé à « -1 » se comporte comme le complémentaire du seuil. Les défauts de collage de Nh2 et Nh5 sont donc compensés entre eux et n'influencent pas la convergence du réseau. Concernant l'état aléatoire du neurone Nh9, on pourrait s'attendre à trouver un jeu de poids associés nuls. Ce n'est pas le cas, la flexibilité de l'apprentissage permet donc de trouver une combinaison de poids synaptiques qui autorise la convergence de l'apprentissage en dépit de cette entrée aléatoire.

Nous concluons de cette partie de simulation que la redondance introduite par l'apprentissage compétitif est efficace sur les neurones cachés. La méthode d'apprentissage par concurrence entre les neurones a permis au réseau de contourner les neurones défaillants.

III.2 Défauts et variations des caractéristiques des synapses

III.2.1 Modèle de memristor non-idéal

Comme nous l'avons déjà évoqué dans le premier chapitre, les techniques utilisées pour la fabrication des memristors ou des nanocomposants en général ne permettent pas actuellement d'avoir des composants fiables. Une matrice crossbar de memristors peut contenir des défauts de collage des connexions ou des dispersions entre les caractéristiques des memristors. La présence de ces défauts et dispersions dans les composants constituant le neuro-crossbar peuvent engendrer des erreurs durant l'apprentissage conduisant à la divergence dans certain cas. Afin d'étudier l'influence de ces défauts et dispersions sur l'apprentissage d'un neuro-crossbar, nous considérons dans le cas de notre étude un modèle de memristor «-0+» (figure III.4 (a) et (b)) non-idéal qui peut contenir plusieurs types de défauts ou des variations de caractéristiques par rapport à d'autres memristors de la même matrice synaptique. La figure III.4 illustre les défauts et la variation considérés dans notre cas.

- Défaut de collage à G_{min} équivalent à un memristor en circuit-ouvert. La conductance reste collée à une valeur minimale G_{min} tout au long de l'apprentissage (zéro) (figure (III.4 (c)).
- Défaut de collage à G_{max} équivalent à un memristor en court-circuit. La conductance reste collée à une valeur maximale G_{max} tout au long de l'apprentissage (figure (III.4 (c)).
- Dispersion de la plage de variation de la conductance [G_{min}, G_{max}]. Chaque memristor de la matrice synaptique peut avoir une plage de variation de sa conductance différente des autres (figure III.4 (d)).
- Variation des seuils V_{T+} et V_{T-}, où les memristors de la matrice synaptique peuvent avoir des valeurs de seuil décalées par rapport à la valeur moyenne de seuil (V_{T0}) (figure III.4 (e)).

- Dispersion de la vitesse de variation de la conductance. Pour une impulsion de programmation donnée, l'incrémentation de la conductance des memristors de la même ligne peut être différente. Ce type de dispersion est représenté dans ce chapitre par une variation sur le pas d'incrémentation de conductance (G_{inc}) (figure III.4 (f)).

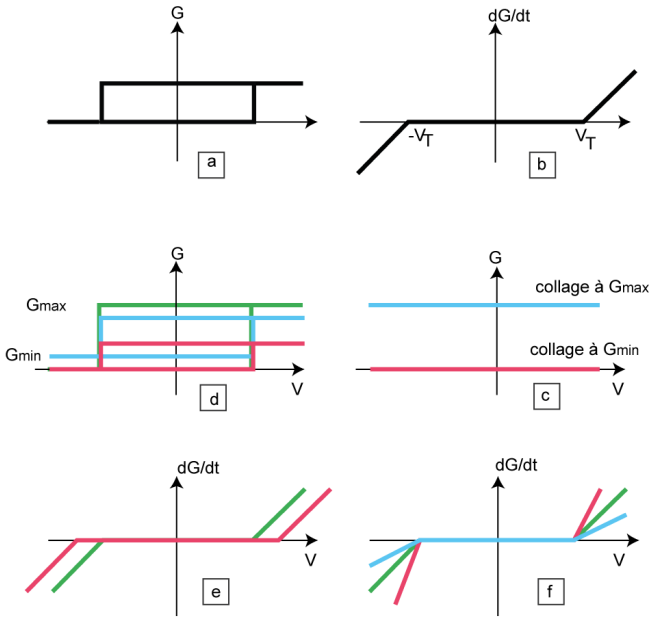


Figure III.4 : Illustration des modèles de défauts et de variations que peut avoir un memristor non-idéal.

III.2.2 Mesure de la robustesse en simulation

En se basant sur le modèle de simulation de neuro-crossbar et les méthodes d'apprentissage développées précédemment, nous avons effectué des simulations Monte-Carlo permettant de mesurer le taux de convergence d'un neuro-crossbar pour apprendre une ou plusieurs fonctions logiques. L'apprentissage de ces fonctions est répété N_{epoq} fois. Comme nous l'avons déjà vu dans les simulations précédentes, l'apprentissage est très rapide et peut converger en quelques époques. Dans les simulations qui suivent, nous considérons que l'apprentissage diverge dans le cas où le nombre d'époques d'apprentissage atteint le nombre maximal d'époques d'apprentissage (N_{epoq}) fixé au préalable. Un bloc neuro-crossbar (NC $N_i \times N_0$) est considéré globalement convergeant, s'il a réussit à apprendre toutes les fonctions souhaitées E_f. Inversement, l'apprentissage du crossbar sera considéré totalement défaillant dès que l'une des fonctions n'aura pas être apprise. Ces simulations sont répétées N_{exp} fois pour estimer la probabilité de convergence de NC, réalisant ainsi des statistiques Monte-Carlo.

III.2.3 Effet de défaut de collage des memristors dans le cas d'un NC 8×1

III.2.3.1 Défauts de collage pré-placés

L'effet de défaut de collage à G_{min} et à G_{max} sur l'apprentissage d'un neuro-crossbar NC 8×1 (figure III.5) est étudié ici. Un défaut de collage d'une connexion est considéré dans notre modèle de simulation comme un composant qui n'évolue pas durant l'apprentissage et sa conductance reste collée à G_{min}=0 ou à G_{max}=12.

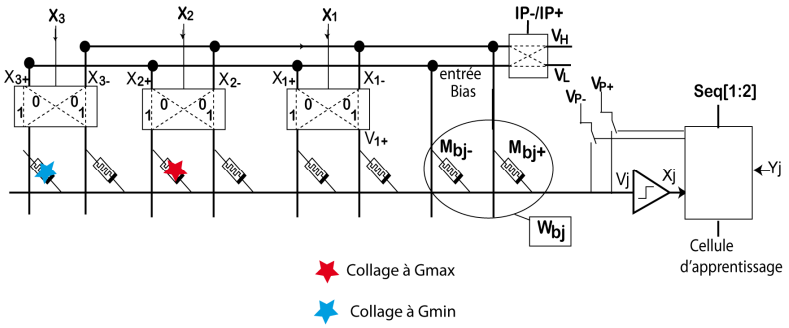


Figure III.5 : schéma d'un neuro-crossbar à une seule sortie en présence de défauts de collage à G_{max} et G_{min}.

La simulation développée dans cette partie consiste à placer un défaut de collage sur l'un des memristor (M_{X3+}, M_{X3-}, M_{X2+}, M_{X2-}, M_{X1+}, M_{X1-}, M_{b+}, M_{b-}) de NC 8×1 de la figure III.5, et réaliser ensuite l'apprentissage de la fonction F=X_1andX_2 (tableau III.7). Afin de connaitre l'effet de ces défauts sur l'apprentissage, nous supervisons la convergence de neurone de sortie pour chaque cas. Dans le cas où le neurone converge en présence d'un défaut de collage sur un memristor donné, l'apprentissage de la fonction F est considéré insensible à ce memristor. L'apprentissage n'a pas besoin de configurer ce memristor pour apprendre la fonction F. Dans le cas contraire, le memristor qui contient le défaut est considéré comme un memristor crucial pour apprendre la fonction F.

X_3	X_2	X_1	F
-1	-1	-1	-1
-1	-1	1	-1
-1	1	-1	-1
-1	1	1	1
1	-1	-1	-1
1	-1	1	-1
1	1	-1	-1
1	1	1	1

Tableau III.7 : Table de vérité de la fonction (X_1)AND(X_2) **à trois entrées**.

Pour un apprentissage de la fonction F nous pouvons en déduire que les poids synaptiques de NC 8×1 doivent satisfaire les inégalités suivantes :

$$\begin{aligned}
&W_b - W_{X1} - W_{X2} - W_{X3} < 0\\
&W_b + W_{X1} - W_{X2} - W_{X3} < 0\\
&W_b - W_{X1} + W_{X2} - W_{X3} < 0\\
&W_b + W_{X1} + W_{X2} - W_{X3} > 0\\
&W_b - W_{X1} - W_{X2} + W_{X3} < 0\\
&W_b + W_{X1} - W_{X2} + W_{X3} < 0\\
&W_b - W_{X1} + W_{X2} + W_{X3} < 0\\
&W_b + W_{X1} + W_{X2} + W_{X3} > 0
\end{aligned} \qquad \text{(III.2)}$$

Nous commençons par l'apprentissage de la fonction F dans le cas de défaut de type collage à G_{min}. Les résultats de simulation de cet apprentissage sont montrés dans le tableau III.8 où les valeurs de conductance présentées dans ce tableau sont récupérées après la fin de l'apprentissage ($N_{époq}$=50). Les conductances des memristors sont initialisées avant l'apprentissage à des valeurs aléatoires autour de la valeur G_{min}=0.01 ($G=f^{+}(randn(N_h,N_o)+G_{min})$. Elles peuvent varier, avec un pas d'incrémentation de G_{inc}=1, entre cette valeur initiale jusqu'à une valeur fixée à G_{max}=11.99. Dans le cas où la valeur de la conductance dépasse les extrémités de l'intervalle défini, une instruction dans le programme va permettre de la ramener vers l'une des extrémités G_{min} ou G_{max} ($G(G>G_{max})=G_{max}$, $G(G<G_{min})=G_{min}$). Les valeurs 0 et 12 sont réservées respectivement aux défauts de collage de telle manière que si la conductance est égale à l'une de ces valeurs, l'apprentissage n'ajuste pas la conductance.

Memristors / Défaut de Collage à G_{min}=0	M_{X1+}	M_{X1-}	M_{X2+}	M_{X2-}	M_{X3+}	M_{X3-}	M_{b+}	M_{b-}	Convergence (N_{epoq})
aucun	2.05	1.02	4.02	0.08	1.03	1.01	0.05	3.03	Converge(3)
$\mathbf{G_{X1+}}$	**0**	0.01	2.01	0.01	1.00	1.08	1.06	1.02	**Diverge (50)**
$\mathbf{G_{X1-}}$	3.02	**0**	3.05	1.05	1.06	1.04	0.07	3.00	Converge (4)
$\mathbf{G_{X2+}}$	2.03	0.09	**0**	0.04	1.09	1.03	1.05	1.09	**Diverge (50)**
$\mathbf{G_{X2-}}$	2.06	1.01	3.07	**0**	1.09	1.02	0.01	3.03	Converge (4)
$\mathbf{G_{X3+}}$	1.06	0.08	1.05	0.05	**0**	0.00	0.06	1.03	Converge (2)
$\mathbf{G_{X3-}}$	1.01	0.01	1.03	0.04	0.1	**0**	0.00	1.04	Converge (2)
$\mathbf{G_{b+}}$	2.04	1.01	3.06	0.03	1.00	1.05	**0**	3.01	Converge (4)
$\mathbf{G_{b-}}$	3.00	0.01	3.02	1.02	0.03	3.09	0.05	**0**	**Diverge (50)**
$\mathbf{G_{X1-}}$, $\mathbf{G_{X2-}}$, $\mathbf{G_{X3-}}$, $\mathbf{G_{X3+}}$, $\mathbf{G_{b+}}$	0.09	**0**	1.07	**0**	**0**	**0**	**0**	1.01	Converge (2)

Tableau III.8 : Localisation des memristors cruciaux pour l'apprentissage de la fonction F=(X_1) And (X_2) en présence de défaut de type collage à G_{min}=0.

La première ligne du tableau précédent montre les valeurs de conductances des memristors permettant d'apprendre la fonction F en absence de défauts. Les poids synaptiques correspondants sont les suivants :

$$W_{X1} = G_{X1+} - G_{X1-} = 2.05-1.02=1.03$$
$$W_{X2} = G_{X2+} - G_{X2-} = 4.02-0.08=3.94$$
$$W_{X3} = G_{X3+} - G_{X3-} = 1.03-1.01=0.02$$
$$W_{b} = G_{b+} - G_{b-} = 0.05-3.03=-2.98$$

Ces poids synaptiques respectent bien les inégalités de l'équation III.2. Il est intéressant de noter que l'apprentissage neutralise le poids synaptique de l'entrée X_3 ($W_{X3}\approx 0$) car la fonction F n'utilise pas cette entrée.

Les résultats présentés dans le tableau III.8 montrent bien qu'il y a un ensemble de memristors nécessaires pour apprendre la fonction F. Il s'agit des trois memristors $\{M_{X1+}\ M_{X2+}\ M_{b-}\}$. L'apprentissage de la fonction F diverge si un défaut est présent sur l'un de ces memristors alors qu'un défaut présent sur les autres peut être compensé par l'apprentissage.

Nous allons présenter maintenant les résultats de simulation dans le cas de défaut de collage à G_{max}. Nous avons gardé les mêmes paramètres que la simulation précédente. Les résultats de l'apprentissage sont présentés dans le tableau suivant :

Memristors / Défaut de Collage à G_{max}=12	M_{X1+}	M_{X1-}	M_{X2+}	M_{X2-}	M_{X3+}	M_{X3-}	M_{b+}	M_{b-}	Convergence ($N_{époq}$)
aucun	4.02	0.03	3.08	0.02	1.08	1.09	1.02	2.04	Converge(3)
G_{X1+}	**12**	5.06	5.05	0.08	1.01	0.04	0.09	5.03	Converge (4)
G_{X1-}	11.99	**12**	3.04	0.03	2.05	1.02	1.09	3.04	**Diverge(50)**
G_{X2+}	5.04	0.08	**12**	5.08	1.04	0.06	0.05	5.02	Converge(4)
G_{X2-}	3.07	0.04	11.99	**12**	2.08	1.05	1.06	3.02	**Diverge(50)**
G_{X3+}	4.06	1.05	6.05	0.09	**12**	10.05	0.07	5.03	Converge(4)
G_{X3-}	5.01	0.01	5.02	0.01	11.08	**12**	1.09	5.04	Converge(4)
G_{b+}	4.00	0.01	4.07	1.02	0.01	3.02	**12**	11.99	**Diverge(50)**
G_{b-}	5.03	0.07	5.07	0.06	0.09	1.00	5.09	**12**	Converge(4)
G_{X1+}, G_{X2+}, G_{X3-}, G_{X3+}, G_{b-}	**12**	0.03	**12**	0.09	**12**	**12**	0.02	**12**	Converge(1)

Tableau III.9 : Localisation des memristors cruciaux pour l'apprentissage de la fonction F=(X_1) And (X_2) en présence de défaut de type collage à G_{max}=12.

Les résultats présentés dans le tableau III.9 montrent les mêmes observations que dans le cas de collage à G_{min}. La convergence de l'apprentissage de la fonction F est sensible à trois memristors $\{M_{X1-}\ M_{X2-}\ M_{b+}\}$. L'apprentissage de la fonction F diverge si un défaut est présent sur l'un de ces memristors alors qu'un défaut présent sur les autres memristors peut être compensé par l'apprentissage. Les trois

memristors cruciaux trouvés ici sont des memristors complémentaires à ceux trouvés dans le cas de collage à G_{min}.

Les résultats de simulations des deux cas de défauts (collage à G_{max} et collage à G_{min}) ont montré que le neuro-crossbar 8×1 peut apprendre la fonction F même en présence de plusieurs memristors défaillants (voir dernière ligne des deux tableaux III.8 et III.9). Nous avons vu que le nombre de défauts présent dans les memristors n'est pas très important dans la mesure où l'apprentissage trouve des memristors fonctionnels pour les compenser. Néanmoins, la position de ces défauts semble jouer un rôle important pour la convergence. L'apprentissage d'une fonction donnée nécessite de faire varier quelques poids synaptiques dans le sens positifs et d'autres dans le sens négatifs et quelques uns restent neutres. En effet, quand un défaut affecte un memristor crucial pour l'apprentissage de cette fonction, cela peut empêcher un poids synaptique de varier dans le bon sens pour converger. Cela explique pourquoi le neuro-crossbar ne peut pas tolérer les défauts de type collage à G_{min} sur les memristors $\{M_{X1+}, M_{X2+}, M_{b-}\}$, et ne tolère pas les défauts de collage à G_{max} sur les memristors complémentaires $\{M_{X1-}, M_{X2-}, M_{b+}\}$.

La dépendance de l'apprentissage à des memristors critiques pour converger peut être vue comme un point faible car un défaut sur un seul memristor peut empêcher le réseau de converger. D'un autre coté, cela peut être vu comme un avantage dans le sens où le neuro-crossbar n'a pas besoin de tous les memristors pour converger.

III.2.3.2 Défauts de collage distribués aléatoirement

Les défauts de collage des synapses à G_{min} ou à G_{max} sont injectés ici aléatoirement suivant une probabilité de défauts prédéfinie (§Annexe). La simulation réalisée ici consiste à apprendre au NC 8×1 les trois fonctions logiques : $F_1=X_1$, $F_2=X_1$ And X_2 et $F_3 = (X_1$ And $X_2)$And(notX_3), en présence d'un taux de défaut P_f qui varie de 0% à 100%. Le choix des fonctions logiques F_1, F_2 et F_3 a été fait par rapport au nombre de memristors cruciaux (N_m) qui est différent pour les trois fonctions. L'apprentissage de la fonction F_1 nécessite un poids synaptique W_{x1} positif (G_{x1+}-G_{x1-}>0). L'apprentissage de cette fonction dans le cas d'un collage à G_{min} sur le memristor $\{M_{x1+}\}$ ou d'un collage à G_{max} sur le memristor $\{M_{x1-}\}$. Un seul memristor est crucial (N_m=1) pour l'apprentissage de la fonction F_1. Trois memristors sont cruciaux (N_m=3) pour l'apprentissage de la fonction F_2 :$\{M_{X1+}, M_{X2+}, M_{b-}\}$ dans le cas de collage à G_{min}. Les memristors cruciaux pour l'apprentissage de la fonction F_3 ont été déterminés de la même façon que la fonction $F_2=X_1$ And X_2 présentée précédemment. Nous avons trouvé que l'apprentissage de la fonction F_3 est sensible aux quatre memristors (N_m=4) $\{M_{X1+}, M_{X2+}, M_{X3-}, M_{b-}\}$ dans le cas d'un collage à G_{min}, et à leurs complémentaires dans le cas d'un collage à G_{max}.

La détermination de nombre de memristors cruciaux N_m va permettre, en présence d'une probabilité de défauts P_f sur un memristor, de calculer la probabilité de succès pour un neurone P_{neuro} à apprendre une fonction donnée (Eq. III.3).

$$P_{neuro} = (1 - P_f)^{Nm} \qquad \text{(III.3)}$$

Afin de comparer ce modèle prédictif de succès de neurone à apprendre une fonction logiques, nous avons développé des simulations Monte-Carlo consistant à injecter aléatoirement un taux de défaut P_f de collage à G_{min} dans les synapses de NC 8×1 et à mesurer le taux de convergence (succès) de l'apprentissage des trois fonctions F_1, F_2 et F_3. L'expérience consiste à faire varier le taux de défauts injecté (0% à 100%) et superviser la convergence du neurone pour l'apprentissage de chaque fonction. Le taux de succès d'apprentissage est mesuré pour chaque taux de défauts injecté en effectuant la moyenne de convergence sur un nombre de répétitions N_{exp}=500. Les courbes continues de la figure III. 6 montrent les résultats de simulation des taux de succès d'apprentissage des trois fonctions F_1, F_2 et F_3 en fonction des taux de défaut injecté P_f. Ces courbes sont comparées aux courbes discontinues qui présentent la probabilité de succès du neurone pour les trois fonctions, calculées à partir de l'équation III.3.

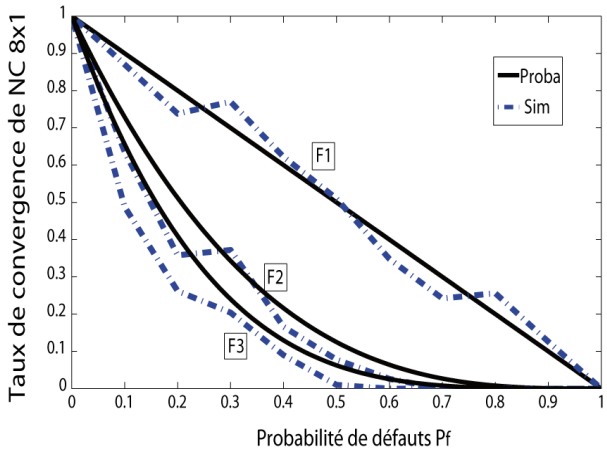


Figure III.6 : Probabilité de succès d'un NC 8×1 taux de défauts de collage des memristors à G_{min}.

Les trois courbes de simulation montrent un taux de convergence qui se dégrade progressivement avec l'augmentation du taux de défauts P_f injecté. Cependant, l'allure de la dégradation des trois courbes n'est pas la même pour les trois fonctions. La fonction F_3 par exemple se dégrade plus rapidement que la fonction F_2 ou F_1. Cela est dû au nombre de memristors nécessaires pour apprendre la fonction F_3 qui est plus grand (N_m=4) que les deux autres fonctions.

Les courbes issues des simulations sont en accord avec les courbes théoriques déterminées à partir de l'équation III.3 et de façon plus nette pour la fonction F_1.

La solution la plus naïve pour améliorer la tolérance aux défauts de neuro-crossbar est de dupliquer les entrées pour augmenter les chances de trouver le nombre de memristors nécessaires pour apprendre la fonction souhaitée. Une autre solution qui peut être plus bénéfique est celle évoquée déjà dans la section 2.1. Elle consiste à dupliquer les neurones pour avoir un nombre de neurones supérieur au nombre de fonction à apprendre. Nous allons voir dans la partie suivante que la mise en concurrence de ces neurones (apprentissage compétitif) pour apprendre un ensemble de fonctions logiques en présence de neurones redondants peut augmenter la tolérance aux défauts.

III.2.4 Effet de défaut de collage des memristors dans le cas d'un NC 8×(6+R)

III.2.4.1 Simulation

Nous considérons dans cette partie un neuro-crossbar à six sorties NC 8×6 où chaque neurone de sortie est supposé apprendre une fonction logique F parmi E_f fonctions. En présence de défauts dans les synapses certains neurones ne vont pas réussir à apprendre la fonction souhaitée. Par conséquent, le neuro-crossbar diverge car nous considérons dans le cas de cette étude qu'un NC $N_i \times N_0$ par exemple converge si et seulement si les N_0 neurones ont collectivement réussit à apprendre l'ensemble de fonctions souhaitées E_f. Le rajout de R neurones redondants devrait compenser les neurones défaillants et améliorer la tolérance.

La simulation réalisée ici consiste à apprendre au NC 8×(6+R) un ensemble de six fonctions logiques en utilisant l'apprentissage compétitif (figure III.7).

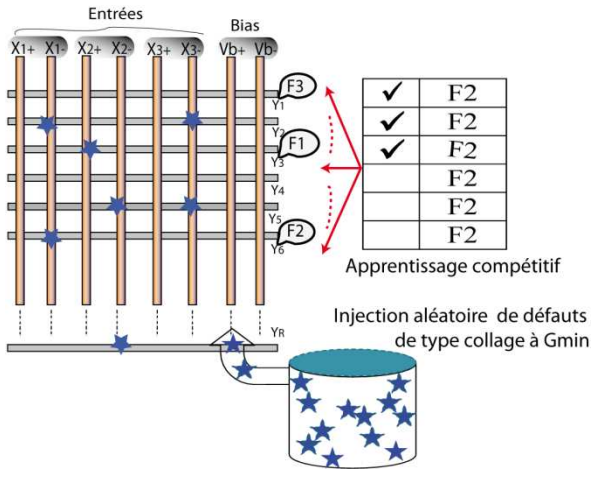


Figure III.7 : Apprentissage compétitif de la fonction F2 avec un Neuro-crossbar à trois entrées, six neurones de sortie et R neurones redondants, en présence de défauts de type collage à G_{min} injectés aléatoirement.

Pour des raisons de simplification et afin de pouvoir calculer la probabilité de convergence de NC, de manière à ce que les neurones aient la même probabilité de

réussir l'apprentissage de la fonction souhaitée, nous avons choisi la même fonction $F_2=X_1$ And X_2 à apprendre par les six neurones. L'apprentissage de chaque neurone se fait en présence d'un taux de défaut de collage à G_{min} qui varie entre 0 et 100%. Pour estimer la probabilité de convergence, l'apprentissage est répété 500 fois pour chaque taux de défauts injecté aléatoirement dans la matrice des synapses. Le modèle de memristor utilisé ici est le même que celui utilisé dans les simulations précédentes (-0+) avec une tension de seuil $V_T/-V_T=1/-1$. Les paramètres de simulation sont les suivants :

- La tension de programmation $V_P/-V_P=1/-1$.
- La tension de lecture $V_H/V_L=0.4/-0.4$.
- La conductance varie entre 0 et 10 avec une initialisation $G=rand(N_i,N_o)*0.1$.
- Le pas d'incrémentation $G_{inc}=1$.
- Les défauts $G_{min}=0$ et $G_{max}=12$.

La simulation est réalisée dans le cas d'absence de neurones redondants (R=0), où les six neurones vont apprendre les six fonctions F. Elle est réalisée pour un cas de 100% de redondance (R=6), où douze neurones vont être utilisés pour apprendre collectivement les six fonctions F, et finalement pour le cas de 200% de redondance (R=12), où dix-huit neurones sont utilisés pour apprendre les six fonctions F. La figure III.8 montre la probabilité de convergence estimée en réalisant la moyenne de convergence de NC sur les 500 itérations. Les courbes sont représentées en fonction du taux de défauts injectés pour les trois taux de redondance considérés.

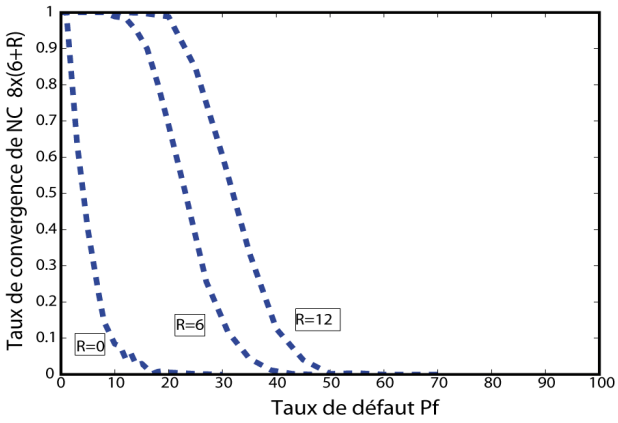


Figure III.8 : Résultats de simulation Monte-Carlo, montrant le taux de convergence de NC 8×(6+R) en fonction de taux de défaut de type G_{min}.

La première courbe correspondant à R=0 montre que le NC commence à diverger pour des faibles taux de défauts. Cela s'explique par le fait que la défaillance d'un seul neurone suffit pour que l'apprentissage du NC soit considéré comme ayant échoué. Les deux autres courbes (R=6, R=12) montrent que l'ajout de neurones redondants améliore d'une manière considérable la tolérance aux défauts. Pour R=6 (100% de redondance) l'apprentissage fonctionne parfaitement jusqu'à un taux de

défaut de 10% et pour R=12 (200% de redondance) le réseau tolère jusqu'à 20% de défauts.

III.2.4.2 Modélisation

La probabilité de convergence d'un NC $N_i \times N_0$ dépend de la probabilité de succès de ses neurones pour apprendre l'ensemble des fonctions désirées *Ef*. Si l'apprentissage d'une seule fonction diverge alors le neuro-crossbar va globalement échouer. La probabilité de succès du neuro-crossbar P_{NC} est alors donnée par l'équation suivante :

$$P_{NC} = (P_{neuro})^{Ef} \qquad \text{(III.4)}$$

avec P_{neuro} la probabilité de convergence d'un neurone donnée par l'équation III.3.

En présence de défaut, le neuro-crossbar peut contenir des neurones fonctionnels que nous notons ici N_{fon} et des neurones défaillants notés ici $N_{déf}$.

$$N_o = N_{fon} + N_{déf} \qquad \text{(III.5)}$$

Pour que le neuro-crossbar converge le nombre de neurones fonctionnels doit être supérieur au nombre de fonctions à apprendre ($N_o \geq N_{ok} \geq E_f$). Dans le cas contraire, les neurones redondants vont permettre de compenser les $N_{déf}$ neurones défaillants. Le nombre de neurones redondants dans un NC $N_i \times (Ef+R)$ est donné par la relation suivante :

$$R = N_o - Ef \qquad \text{(III.6)}$$

Le taux de redondance est défini par la relation suivante :

$$R_{ratio} = \frac{N_o}{Ef} - 1 = \frac{R}{Ef} \qquad \text{(III.7)}$$

Si P_{neuro} est la probabilité qu'un neurone fonctionne alors la probabilité qu'un neurone soit défaillant est $Q_{neuro} = 1 - P_{neuro}$. Le neuro-crossbar NC $N_i \times (Ef+R)$ converge si les (Ef+R) neurones ont réussi à apprendre les Ef fonctions désirées. La probabilité P_{NCR} que le neuro-crossbar converge en présence de R neurones redondants est calculée par la loi binomiale donnée par l'équation III.8.

$$P_{NCR}(N_{fon} = Ef) = \sum_{N_{fon}=Ef}^{Ef+R} C(P_{neuro})^{N_{fon}} (Q_{neuro})^{N_{def}} \qquad \text{(III.8)}$$

Avec C le facteur binomial : $C = \begin{pmatrix} N_o \\ N_{fon} \end{pmatrix}$ qui représente le nombre de configurations différentes correspondant exactement à $N_{fon}=Ef$.

Dans notre exemple le NC est composé de dix neurones de sortie (No=10) pour apprendre six fonctions (Ef=6). Le nombre de neurones redondants dans ce cas là est de quatre (R=4). La probabilité que le NC converge est donnée par la somme des probabilités hachurées de la figure III .9 (dans le cas où Pneuro=Qneuro=0.5): PNCR(6) +PNCR(7) +PNCR(8) +PNCR(9) + PNCR(10).

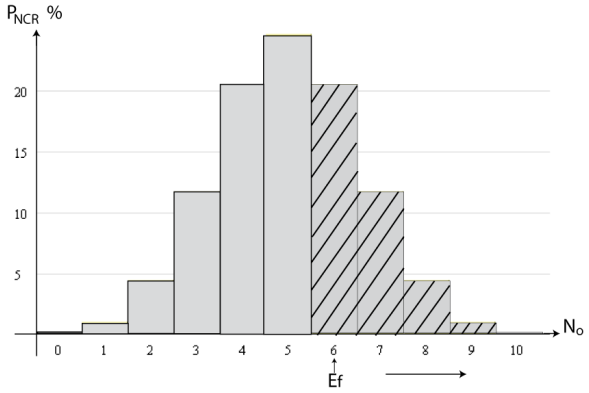


Figure III.9: histogramme de la loi binomiale, nous considérons dans ce cas un nombre de neurones N_o qui vari de 0 à 10 avec une probabilité de succès de neurone P_{neuro}=0.5, ce qui donne un maximum de probabilité de converger en N_o =5.

Afin de comparer le modèle de prédiction de la convergence que nous venons de proposer avec les courbes de simulation de la figure III.8, nous avons calculé la probabilité de convergence de NC 8×(6+R) à partir de l'équation III.8, cela pour une probabilité de succès du neurone P_{neuro} calculée à partir de l'équation III.3 avec N_m=3 et une probabilité de défaut sur un memristor P_f variable entre 0 et 1. Le calcul a été développé pour les trois cas de redondance R=0, R=6 et R=12. Les courbes obtenues sont superposées avec les courbes de simulation ce qui confirme l'efficacité de notre modèle (figure III.10).

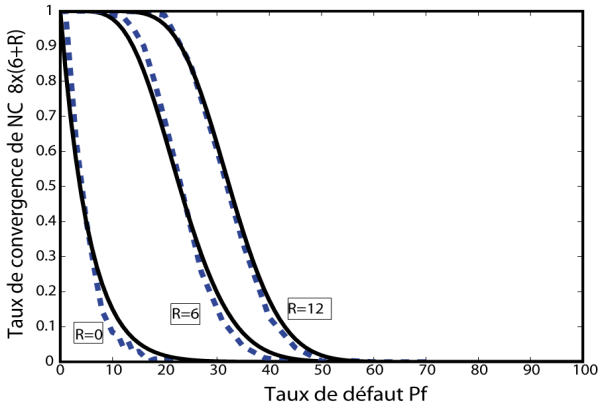


Figure III.10 : Comparaison des probabilités de convergence de neuro-crossbar 8×(6+R) (courbes continues) avec les courbes de simulation (courbes discontinues).

Après avoir valider notre modèle prédictif, nous allons tracer maintenant le nombre minimum de neurones redondants qu'il faut ajouter au NC 8×(6+R) pour avoir, en présence d'une probabilité de défaut des memristors *Pf*, une convergence de

100% de l'apprentissage de la fonction F. La courbe obtenue est présentée dans la figure III.11, montrant une évolution exponentielle du nombre de neurones redondants requit pour l'apprentissage en fonction de σ. Pour des valeurs de probabilité de défauts Pf<0.3, la redondance requise reste tolérable.

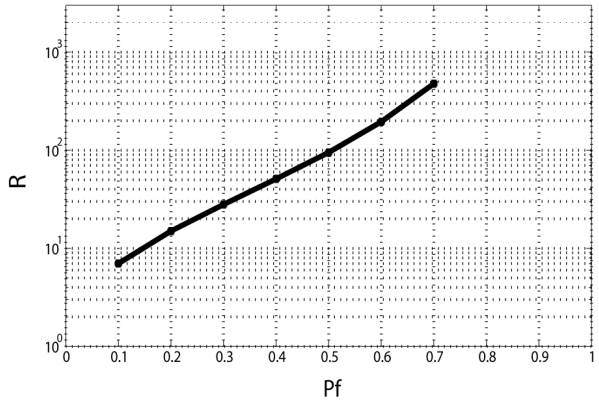


Figure III.11 : Nombre de neurones redondants nécessaires pour apprendre la fonction F avec un NC 8×(6+R) en présence d'un taux de défaut de collage Pf

III.2.5 Variation du seuil V_T des memristors

Comme nous l'avons évoqué précédemment (III.2.1), la variation de seuil est parmi les paramètres caractérisant un memristor non idéal. La présence de ce type de variation dans un neuro-crossbar peut causer l'échec de l'apprentissage.

Des mesures expérimentales de caractéristiques des nanocomposants de type memristif ont montré que la tension de seuil présente une variabilité qui peut être modélisée selon une loi de distribution normale autour d'une tension de seuil moyenne [Borg10] (figure III.12). Dans cet exemple, la distribution des tensions de seuil de l'état *Off* se caractérise par une moyenne V_{T0+}=0.61 et un écart type relatif à la moyenne de σ_{off}=0.25. De même la distribution des tensions de seuil de l'état *On* se caractérise par une moyenne V_{T0-}=-2.07 et un écart type relatif à la moyenne de σ_{on}=0.10.

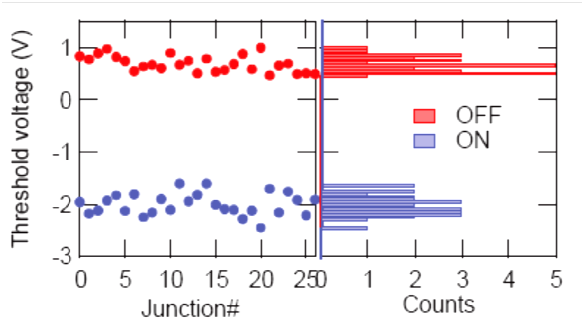


Figure III.12 : Mesure expérimentale de la tension de seuil pour 26 memristors [Borg10].

L'étude réalisée dans la suite de cette partie consiste à mesurer le degré de tolérance de notre modèle de neuro-crossbar, en réalisant des simulations Monte-

Carlo pour apprendre une fonction logique en présence d'une variation de seuil des memristors. Les résultats de simulation vont être comparés à un modèle asymptotique prédictif de probabilité de convergence.

III.2.5.1 Effet des tensions de seuil extrêmes

Dans le cas d'une distribution Gaussienne de la variation de la tension de seuil un certain nombre de memristors peuvent avoir des tensions de seuil extrêmes qui peuvent causer la divergence du neuro-crossbar. Pour connaitre l'effet des tensions de seuil extrêmes, nous remplaçons dans les conditions de l'apprentissage (§II Eq II.3), la tension de seuil utilisée habituellement V_T=1 par deux cas de tensions de seuil : la tension de seuil faible V_T=0.5 et la tension de seuil élevée V_T=2. Prenons un exemple de paramètres de simulation utilisés par défaut : tensions de programmations de V_{p+}/V_{p-}=1/-1 et tensions de lectures appliquées aux entrées de V_H/V_L=0.4/-0.4.

- Dans le cas d'une faible tension de seuil V_{T+}/V_{T-}=0.5/-0.5

Situation idéale	Situation réelle
V_H-V_{P-} >V_{T+} ➔ G augmente (+)	0.4 + 1=1.4 >0.5 ➔ G augmente (+)
V_L-V_{P-} < V_{T+} ➔ G stable (0)	-0.4 + 1=0.6 >0.5 ➔ G augmente (+)
V_L-V_{P+} < V_{T-} ➔ G diminue (-)	-0.4 - 1=-1.4 <-0.5 ➔ G diminue (-)
V_H-V_{P+} > V_{T-} ➔ G stable (0)	0.4 - 1= -0.6 <-0.5 ➔ G diminue (-)

La vérification des conditions d'apprentissage avec les paramètres utilisés en simulation nous montre qu'un memristor possédant un seuil faible ne contient pas de plage neutre où la conductance est stable. La programmation de la conductance du memristor est indépendante de la polarité des entrées. Elle ne dépend que de la tension de programmation V_P. Elle augmente quand la tension de programmation est négative V_{P-} et diminue quand elle est positive V_{p+}. Ceci peut provoquer une oscillation de la conductance autour de la valeur de conductance initiale ($G=G_{min}$) tout au long de l'apprentissage.

- Dans le cas d'une tension de seuil élevée V_{T+}/V_{T-}=2/-2

Situation idéale	Situation réelle
V_H-V_{P-} >V_{T+} ➔ G augmente (+)	0.4 + 1=1.4 <2 ➔ G stable (0)
V_L-V_{P-} < V_{T+} ➔ G stable (0)	-0.4 + 1=0.6 <2 ➔ G stable (0)
V_L-V_{P+} < V_{T-} ➔ G diminue (-)	-0.4 - 1=-1.4 >-2 ➔ G stable (0)
V_H-V_{P+} > V_{T-} ➔ G stable (0)	0.4 - 1= -0.6 >-2 ➔ G stable (0)

La vérification des conditions d'apprentissage dans le cas d'un memristor possédant une tension de seuil élevée montre que la programmation de sa conductance n'est pas possible, car la tension de programmation appliquée à ses bornes ne peut pas atteindre le seuil. Dans ce cas la conductance du memristor restera collée à sa valeur initiale (G_{init}).

Maintenant que nous avons étudié l'effet des seuils extrêmes sur l'apprentissage, nous allons évaluer par simulation l'effet de ce type de variation sur le modèle NC. Pour cela, nous avons repris la simulation réalisée dans le cas de l'étude de défauts de collage pré-placés (section III.2.2.1). Cependant, au lieu de placer un défaut de collage sur l'un des huit memristor de NC 8×1, nous mettons à la place, une tension de seuil différente de la tension de seuil par défaut V_{T0}=1, soit une tension de seuil élevée (V_T=2) soit une tension de seuil faible (V_T=0.5).

Le tableau III.10 montre les résultats de simulation Monte-Carlo montrant la moyenne de convergence (100 répétitions) du NC 8×1 pour apprendre la fonction F=X_1 And X_2, ceci dans le cas où l'un des huit memristors est mis en difficulté avec une tension de seuil soit faible (V_T=0.5) soit élevé (V_T=2). L'apprentissage a été réalisé pour deux initialisations différentes de la conductance (G= rand(N_i,N_o)*G_{min} et G= rand(N_i,N_o)*G_{max}).

	M / V_T	M_{X1+}	M_{X1-}	M_{X2+}	M_{X2-}	M_{X3+}	M_{X3-}	M_{b+}	M_{b-}
$G_{init}=G_{min}$	V_T=0.5	0.02	1	0.02	1	0.99	1	0.99	0.04
$G_{init}=G_{max}$	V_T=0.5	1	0.01	1	0.02	1	1	0.05	1
$G_{init}=G_{min}$	V_T=2	0	1	0	1	0.99	0.99	0.97	0.07
$G_{init}=G_{max}$	V_T=2	1	0	1	0	1	1	0.02	1

Tableau III.10: Taux de convergence de l'apprentissage de la fonction F=X_1 And X_2 avec un NC 8×1 dans le cas où la tension de seuil de l'un des huit memristors est faible (V_T=0.5) ou élevée (V_T=2).

Les résultats de simulations présents dans le tableau III.10 montrent que le taux de convergence de l'apprentissage devient très faible (≈0.02) quand l'un des memristors critiques pour l'apprentissage de la fonction F=X_1 And X_2 est touché par la variation de seuil. L'effet d'un seuil élevé V_T=2 est très similaire à un défaut de collage. Quand la conductance est initialisée à G=G_{min} les trois memristors cruciaux (N_m=3) identifiés sont les mêmes que ceux trouvés dans le cas de collage à G_{min} {M_{X1+}, M_{X2+}, M_{b-}}. Quand la conductance est initialisée à G=G_{max} les memristors trouvés sont les mêmes que ceux trouvés dans le cas de collage à G_{max} {M_{X1-}, M_{X2-}, M_{b+}}. Ces résultats de simulation confirment bien l'effet des seuils extrêmes déterminés précédemment à partir des conditions d'apprentissage conditionnel.

III.2.5.2 Simulation de la variation gaussienne de seuil

La simulation réalisée ici consiste à mesurer le taux de convergence de neuro-crossbar 8×(6+R) pour apprendre la fonction F=X_1 And X_2 en présence d'une variation aléatoire de la tension de seuil des huit memristors avec une distribution gaussienne. Nous gardons les paramètres de simulation utilisés dans la simulation

réalisée dans la partie précédente (2.4.1) qui concerne l'étude de l'impact des défauts de collage (V_{T0}=1, $V_P/V_{P-}=V_{T0}/-V_{T0}$,$V_H/V_L$=0.4/-0.4, G_{init}=rand(8,6+R)*0.1). Une erreur ξ aléatoire est ajoutée aux seuils des memristors de neuro-crossbar mais nous avons conservé leur caractère symétrique ($V_{T+}=-V_{T-}$). La distribution est gaussienne, et nous faisons varier son écart type σ.

$$|V_T| = V_{T0} + \xi$$
$$V_{T+} = |V_T|, \quad V_{T-} = -|V_T|$$

Pour estimer la probabilité de convergence l'apprentissage est répété 1000 fois pour chaque écart type σ. Les résultats de simulation sont présentés par la figure III. 13, montrant le taux de convergence moyen sur les 1000 répétitions pour trois cas de taux de redondance différents R=0, R=6 et R=12.

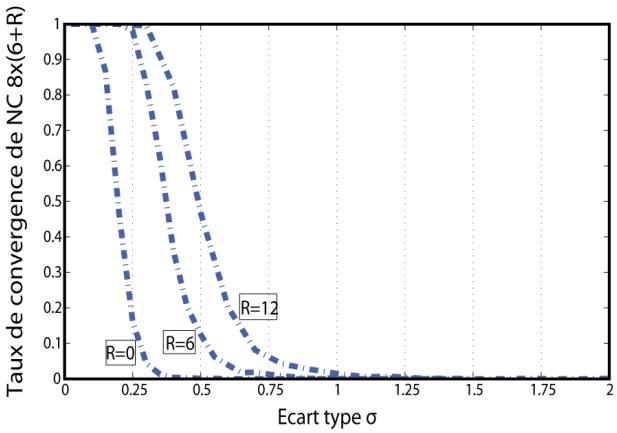


Figure III.13 : Résultats de simulation Monte-Carlo, montrant le taux de convergence de l'apprentissage de la fonction F=X_1 And X_2 avec un NC 8×(6+R) en présence d'une variation de seuil de distribution gaussienne d'écart type σ.

Les courbes de la figure III.13 montrent une allure décroissante de la convergence en fonction de l'augmentation de l'écart type σ de la distribution du seuil V_T. L'ajout de neurones redondants fait déplacer les courbes vers la droite pour être plus tolérantes aux variations de seuil. La première courbe montre qu'en absence de neurones redondants (R=0) l'apprentissage peut tolérer un V_T avec des variations qui peuvent aller jusqu'à σ=0.12 (V_T=1+randn(8,6)*0.12). L'ajout de six neurones redondants a amélioré la tolérance. L'apprentissage peut tolérer des variations jusqu'à σ=0.23. Dans le cas de R=12, l'apprentissage tolère des variations qui peuvent aller jusqu'à un écart type de σ=0.3, ce qui représente des variations de seuil importantes.

Les mesures des tensions de seuil des memristors d'HP [Borg10] ont montré des écarts type de σ_{Off}=0.25 et de σ_{On}=0.10. Une dispersion du même ordre sur le seuil de nos modèles de memristors nécessitera l'ajout d'un nombre raisonnable de neurones redondants (R=6) au NC 8×(6+R) pour tolérer une dispersion gaussienne de σ=0.25 sur la tension V_{T+} et V_{T-}.

III.2.5.3 Modélisation de la variation Gaussienne de seuil

Pour développer le modèle de prédiction de la convergence de l'apprentissage, nous revenons aux conditions de l'apprentissage conditionnel qui vont définir les marges de l'erreur que peut tolérer le neuro-crossbar. Selon ces conditions, un memristor fonctionnel est un memristor qui possède des tensions de seuil positive V_{T+} et négative V_{T-} qui respectent les deux conditions suivantes :

$$V_L - V_{P-} < V_{T+} < V_H - V_{P-} \quad \text{(III.9)}$$
$$V_L - V_{P+} < V_{T-} < V_H - V_{P+} \quad \text{(III.10)}$$

Si une erreur ξ est ajoutée à la tension de seuil V_T, la tension de seuil positive du memristor devient $V_{T+}=V_{T0}+\xi$ et la tension de seuil négative devient $V_{T-}= -(V_{T0}+\xi)$, avec V_{T0} la valeur de la tension de seuil utilisée habituellement dans nos simulations (=1). Pour déterminer la marge d'erreur tolérée par le neuro-crossbar, nous remplaçons les nouvelles valeurs de tension de seuil en présence d'erreur dans les inégalités III.9 et III.10, avec des tensions de programmations $V_P/-V_p=V_{T0}/-V_{T0}$ et des tensions appliquées aux entrées $V_H/V_L=V_i/-V_i$.

$$\text{(III.9)} : -V_i + V_{T0} < V_{T0}+\xi < V_i+V_{T0}$$
$$\text{(III.10)} : -V_i - V_{T0} < -V_{T0}-\xi < V_i-V_{T0}$$

Nous déduisons de ces deux inégalités, la condition donnant l'équation III.11 :

$$-Vi < \xi < Vi \quad \text{(III.11)}$$

L'inégalité III.11 obtenue montre que la marge d'erreur ξ que peut tolérer le neuro-crossbar durant l'apprentissage est compris entre $-V_i$ et V_i qui représente les tensions appliquées aux entrées d'une paire de memristors. Si l'erreur ξ dépasse cet intervalle, l'apprentissage conditionnel ne fonctionne plus et l'apprentissage de la fonction souhaitée peut diverger si cette erreur touche l'un des memristors critiques pour l'apprentissage de cette fonction (voir tableau III.10). Pour augmenter cette marge d'erreur il suffit donc d'augmenter les tensions appliquées aux entrées de neuro-crossbar ($V_i/-V_i$). Cependant, la tension d'entrée doit rester inférieure à la tension de seuil V_T du memristor pour ne pas atteindre le seuil lors de la lecture et par conséquent déprogrammer les conductances. L'erreur ξ donc est conditionnée par une autre condition, nommée ici condition de lecture, donnée par l'inégalité III.12.

$$|V_i|<|V_T|$$
$$V_i<V_{T+} \quad \& \quad -V_i>V_{T-} \quad \text{(III.12)}$$

Pour trouver la marge d'erreur vérifiant cette condition, nous remplaçons V_{T+} par $V_{T0}+\xi$ et V_{T-} par $-(V_{T0}+\xi)$ ce qui donne l'inégalités III.13:

$$V_i< V_{T0}+\xi \quad \& \quad -V_i>-V_{T0}-\xi$$

$$\rightarrow \quad \xi > V_i - V_{T0} \quad \& \quad \xi > V_i - V_{T0}$$

$$\rightarrow \quad \xi > V_i - V_{T0} \qquad \text{(III.13)}$$

Contrairement à la condition III.11 qui doit être vérifiée par les memristors cruciaux qui doivent être programmés pour converger vers la fonction souhaitée, la condition III.11 doit être vérifiée par les memristors cruciaux et leurs memristors complémentaires. L'erreur tolérée par les memristors complémentaires ne dépend que de la condition III.13. Cependant, celle des memristors cruciaux dépend de l'union des deux intervalles correspondant aux deux conditions (III.11 et III .13). Comme le montre la figure III.14 (b) on distingue deux cas possibles selon que Vi est inférieur ou supérieur à $V_{T0}/2$:

- Quand $V_i < V_{T0}/2$: (III.11) U (III.13) ➔ III.11 : $-V_i < \xi < V_i$.
- Quand $V_i > V_{T0}/2$: (III.11) U (III.13) ➔ $V_i - V_{T0} < \xi < V_i$.

Nous considérons maintenant ξ une variable aléatoire de seuil distribuée suivant une loi Gaussienne φ(ξ) (Eq III.14) sur un ensemble de memristors (figure III.14 (a)).

$$\varphi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - V_{T0})^2}{2\sigma^2}\right) \qquad \text{(III.14)}$$

où x (V) est la variable aléatoire, $x = V_{T0} + \xi$, σ représente l'écart type et V_{T0} la valeur moyenne de seuil.

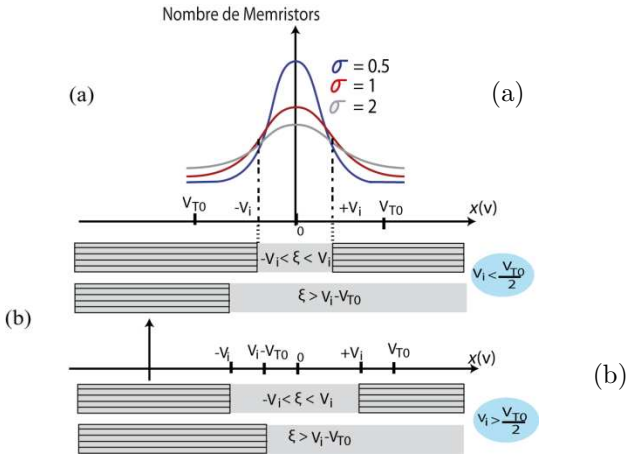


Figure III.14 : (a) Exemple d'erreur ξ de distribution Gaussienne sur le seuil des memristors pour différentes valeurs de σ. (b) en gris: les intervalles de l'erreur que peut tolérer l'apprentissage pour les deux cas $V_i < V_{T0}/2$ et $V_i > V_{T0}/2$, en gris hachuré : les intervalles d'erreur intolérable où le memristor possédant cette erreur est considéré défaillant, soit qu'il ne respecte pas les conditions de l'apprentissage soit qu'il n'est plus volatile lors des lectures.

La probabilité de trouver des memristors fonctionnels P_{mem} en présence d'une distribution gaussienne d'erreur ξ sur les seuils des memristors va dépendre de l'écart type σ et de l'intervalle fixé par III.11 et III.13. Selon ces deux dernières inéquations,

un memristor est considéré fonctionnel si l'erreur ξ présente sur sa tension de seuil V_T est à l'intérieur des intervalles fixés. La probabilité que la variable aléatoire ξ à distribution gaussienne d'écart-type σ prenne une valeur dans l'intervalle $[\xi_a, \xi_b]$ est donnée par la relation suivante:

$$P_{mem}(\xi_a \le \xi \le \xi_b) = \frac{1}{\sqrt{2\pi}\sigma}\int_{\xi_a}^{\xi_b} \exp(-\frac{(\xi)^2}{2\sigma^2})d\xi \qquad \text{(III.15)}$$

Le calcul de cette intégrale devient simple grâce à la fonction d'erreur gaussienne erf(x) donnée par l'équation III.16 suivante :

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x \exp(-t^2)dt \qquad \text{(III.16)}$$

Avec un changement de variable $t = \frac{\xi}{\sqrt{2}\sigma}$, l'intégrale de l'équation III.15 devient comme suit:

$$P_{mem}(\xi_a \le \xi \le \xi_b) = \frac{1}{\sqrt{2\pi}\sigma}\left[\int_0^{\frac{\xi_b}{\sqrt{2}\sigma}} \exp(-t^2).dt\sqrt{2}\sigma - \int_0^{\frac{\xi_a}{\sqrt{2}\sigma}} \exp(-t^2).dt\sqrt{2}\sigma\right]$$

$$P_{mem}(\xi_a \le \xi \le \xi_b) = \frac{1}{2}\left[erf(\frac{\xi_b}{\sqrt{2}\sigma}) - erf(\frac{\xi_a}{\sqrt{2}\sigma})\right] \qquad \text{(III. 17)}$$

L'application des conditions données par III.11 et III.13 dans III.17 donne la probabilité qu'un memristor crucial fonctionne:

- Quand $V_i < V_{T0}/2$:

$$P_{mem}(-V_i \le \xi \le V_i) = erf(\frac{V_i}{\sqrt{2}\sigma}) \qquad \text{(III.18)}$$

- Quand $V_i > V_{T0}/2$:

$$P_{mem}(V_i - V_{T0} \le \xi \le V_i) = \frac{1}{2}\left[erf(\frac{V_i}{\sqrt{2}\sigma}) - erf(\frac{V_i - V_{T0}}{\sqrt{2}\sigma})\right] \qquad \text{(III.19)}$$

L'application de la condition III .13 dans III.17 donne la probabilité P_{memc} qu'un memristor complémentaire ne soit pas déprogrammé et fonctionne correctement (III.20).

$$P_{memc}(\xi > V_i - V_{T0}) = \frac{1}{2}\left[1 - erf(\frac{V_i - V_{T0}}{\sqrt{2}\sigma})\right] \qquad \text{(III.20)}$$

Afin de valider ce modèle, nous allons l'utiliser pour modéliser les courbes de simulation obtenues dans la partie précédente (figure III.13). Nous avons utilisé une tension d'entrée V_i=0.4 ce qui nous place dans le cas de $V_i < V_{T0}/2$. La probabilité qu'un memristor crucial fonctionne est donnée par l'équation III.18 et la probabilité pour qu'un memristor complémentaire ne soit déprogrammé est donnée par III.20. La fonction apprise dans ces simulations est la fonction F=X_1 And X_2 qui nécessite dans le cas de $G_{init}=G_{min}$ les trois memristors $\{M_{x2+}\ M_{x1+}\ M_{b-}\}$ pour converger.

Pour résumé, le neurone converge si seulement si les trois memristors critiques $\{M_{x2+}\ M_{x1+}\ M_{b-}\}$ soient programmables (respectent la condition III.11) pendant l'étape d'apprentissage et que leurs complémentaires $\{M_{x2-}\ M_{x1-}\ M_{b+}\}$ ne soient pas déprogrammés pendant l'étape opérationnelle. Il en résulte de cela que la probabilité que le neurone converge P_{neuro} est exprimée comme le suivant :

$$P_{neuro} = (P_{X1+} \times P_{X2+} \times P_{b-}) \times (P_{X1-} \times P_{X2-} \times P_{b+})$$

$$P_{neuro} = \left[P_{mem}(-V_i \le \xi \le V_i)\right]^{N_m} \times \left[P_{memc}(\xi \ge V_i - V_{T0})\right]^{N_m} \qquad \text{(III.21)}$$

AN : $P_{neuro} = \left[P_{mem}(-0.4 \le \xi \le 0.4)\right]^3 \times \left[P_{memc}(\xi \ge 0.4 - 1)\right]^3$

$$P_{neuro} = \left(erf(\frac{0.4}{\sqrt{2}\sigma})\right)^3 \left(\frac{1}{2} - \frac{1}{2}erf(\frac{0.4-1}{\sqrt{2}\sigma})\right)^3$$

Le NC 8×6 converge si et seulement si les six neurones convergent. Dans ce cas la probabilité de convergence peut être calculée en remplaçant P_{neuro} de l'équation III.20 dans l'équation II.2 avec E_f=6.

$$P_{NC} = (P_{neuro})^6 = \left[\left(erf(\frac{0.4}{\sqrt{2}\sigma})\right)^3 \left(\frac{1}{2} - \frac{1}{2}erf(\frac{0.4-1}{\sqrt{2}\sigma})\right)^3\right]^6 \qquad \text{(III.22)}$$

La probabilité P_{NCR} que le NC 8×(6+R) fonctionne en présence de R neurones redondants peut être calculée en remplaçant P_{neuro} de l'équation III.20 dans l'équation III.8 :

$$P_{NCR}(N_{fon} = 6) = \sum_{N_{fon}=6}^{6+R} C\binom{6+R}{6}(P_{neuro})^{N_{fon}}(1 - P_{neuro})^{N_0 - N_{fon}} \qquad \text{(III.23)}$$

Afin de comparer le modèle que nous venons de développer avec les courbes de simulation, nous avons tracé sur la même figure III.15 les trois probabilités $P_{NC0,}$ P_{NC6}

et P_{NC12} pour un écart type σ variable. Les résultats montrent que les courbes correspondant aux calculs des probabilités (bleu continu) sont presque superposées avec les courbes obtenues par simulation Monte-Carlo (noir discontinu) pour des faibles σ. Pour des grandes valeurs de σ les courbes issues du modèle chute plus rapidement que les courbes de simulation, cela est peut être dû à quelques convergences aléatoires de l'apprentissage en simulation.

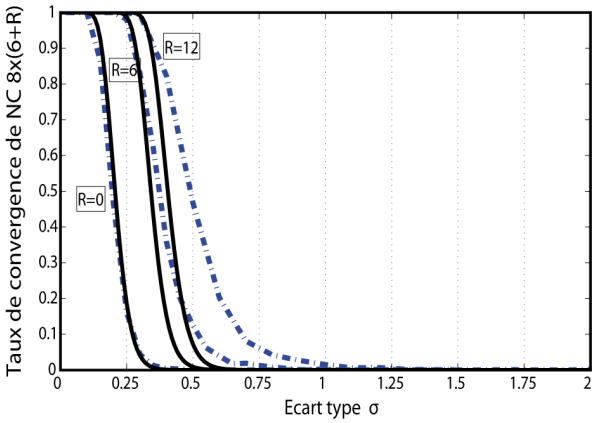


Figure III.15 : Probabilité de convergence de NC 8×(6+R) pour apprendre la fonction F=X_1 And X_2. Les courbes discontinues représentent les résultats obtenus en simulation et les courbes discontinues représentent les résultats de calcul à partir du modèle de probabilité.

Nous allons tracer à partir du modèle précédent, le nombre minimum de neurones redondants qu'il faut ajouter au NC 8×(6+R) pour avoir 100% de convergence de l'apprentissage de la fonction F, en présence d'une variation de seuil d'écart type σ. La courbe de la figure III.16 montre que le nombre de neurones nécessaires pour faire converger l'apprentissage augmente quasi exponentiellement avec l'écart type σ. Pour des valeurs de $\sigma<0.5$, la redondance requise pour converger reste réaliste (≤ 40) et pour des valeurs de $\sigma>0.5$ la redondance requise pour la compensation des variations de seuil devient excessivement coûteuse. Environ 200 neurones redondants sont nécessaires pour faire converger un NC qui contient des memristors avec des variations de seuil de $\sigma=1$, et pour une variation de $\sigma=2$ il faut environ 980 neurones redondants pour converger. Cependant, quelques statistiques réalisées pour des vrais nanocomposants [Borg10, Zhon03] ont montré des variations de seuil des memristors et des CNT-FET avec un écart type relatif qui ne dépasse pas $\sigma=0.5$.

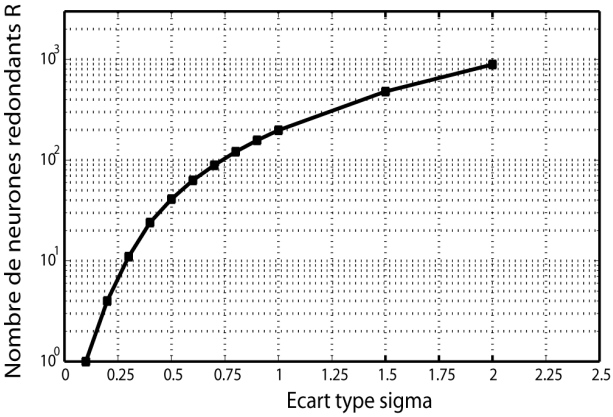


Figure III.16 : Nombre de neurones redondants qu'il faut ajouter au NC 8×(6+R) pour apprendre à 100% la fonction F=X_1 And X_2 en présence d'une variation de seuil des memristor d'un écart type σ.

III.2.5.4 Influence de la tension de lecture sur une variation gaussienne de seuil

Comme nous l'avons vu précédemment dans le modèle de prédiction de convergence dans le cas d'une présence de variation des seuils des memristors, la tension d'entrée V_i peut influencer la tolérance à ces variations. Nous allons voir dans cette partie l'influence de la tension d'entrée V_i sur la tolérance aux variations de seuil. Les courbes discontinues de la figure III.17 représentent les résultats de simulation du taux de convergence de NC 8x1 et de NC 8x6 en fonction de V_i. La fonction à apprendre est F=X_1 And X_2 en présence d'une variation gaussienne des seuils des memristors d'un écart type σ=0.4 et d'une tension moyenne de seuil V_{T0}=2. Chaque point de la courbe bleue discontinue représente la moyenne de convergence sur 500 répétitions pour une tension d'entrée V_i entre 0 et V_{T0}. Le taux de convergence maximum correspond au point optimal $V_i=V_{T0}/2$. Les deux courbes continues représentent la probabilité de convergence de NC 8x6 et de NC 8x1 calculées respectivement à partir de l'équation III.18 et III.19 pour un V_i variable.

- Pour un Vi<$V_{T0}/2$:

$$P_{NC8\times1} = P_{neuro} = \left[P_{mem}(-V_i \le \xi \le V_i)\right]^3 \times \left[P_{memc}(\xi \ge V - V_{T0})\right]^3$$

$$P_{neuro} = \left(erf(\frac{V_i}{\sqrt{2}\times 0.4})\right)^3 \left(\frac{1}{2} - \frac{1}{2}erf(\frac{V_i - 2}{\sqrt{2}\times 0.4})\right)^3$$

$$P_{NC8\times6} = (P_{neuro})^6$$

- Pour un Vi>$V_{T0}/2$:

$$P_{NC8\times1} = P_{neuro} = \left[P_{mem}(V_i - V_{T0} \le \xi \le V_i)\right]^3 \times \left[P_{memc}(\xi \ge V - V_{T0})\right]^3$$

$$P_{neuro} = \left(\frac{1}{2} erf(\frac{V_i}{\sqrt{2}\times 0.4}) - \frac{1}{2} erf(\frac{V_i - V_{T0}}{\sqrt{2}\times 0.4})\right)^3 \left(\frac{1}{2} - \frac{1}{2} erf(\frac{V_i - V_{T0}}{\sqrt{2}\times 0.4})\right)^3$$

$$P_{NC8\times6} = (P_{neuro})^6$$

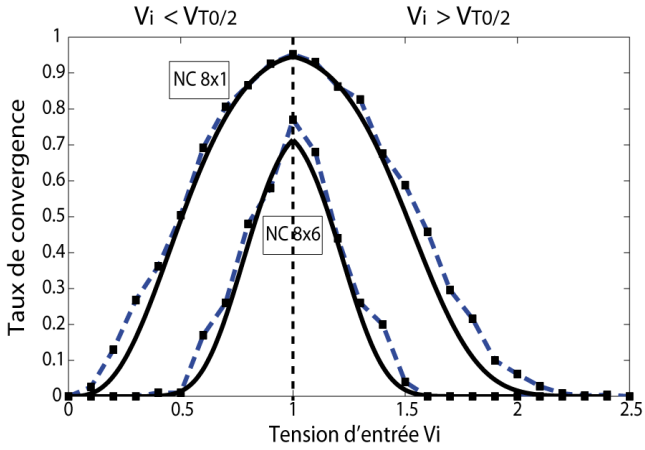


Figure III.17 : Taux de convergence de NC 8x6 et de NC 8x1 en fonction de la tension d'entrée V_i.

Ces résultats montrent que nous avons toujours intérêt à choisir les niveaux logiques utilisés en lecture égaux à la moitié de la tension de seuil (V_i=V_{T0}/2).

III.2.6 Effet de la variation de la résolution des memristor

La résolution des memristor est donnée ici par le nombre de niveau de conductance existant entre l'état *On* (G=G_{max}) et l'état *Off* (G=G_{min}) (Eq III.24). Nous supposons qu'il y a deux paramètres qui peuvent influencer la résolution. Le premier paramètre est le pas d'incrémentation G_{inc}, où la résolution devient importante quand G_{inc} devient petit. Le deuxième paramètre est la plage de variation de la conductance [G_{min} G_{max}] qui peut être différente d'un memristor à un autre.

$$R_{es} = \frac{G_{\max} - G_{\min}}{G_{inc}} \qquad \text{(III.24)}$$

Nous développons dans ce qui suit des simulations permettant de déterminer l'effet de la résolution limitée des memristors sur l'apprentissage et mesurer le taux de tolérance de NC aux dispersions de chacun des deux paramètres influençant la résolution (G_{inc} et la plage de variation de la conductance).

III.2.6.1 Effet de la résolution sur l'apprentissage

La résolution minimale des poids synaptiques nécessaire pour converger l'apprentissage dépend de la complexité de la fonction à apprendre [Ker94] et du nombre d'entrées utilisées. Généralement il est difficile de déterminer la résolution minimale nécessaire pour assurer la convergence de l'apprentissage. Cependant, nous proposons ici une analyse permettant de déterminer la résolution minimale requise

pour un cas particulier. Nous proposons d'apprendre à un seul neurone, avec un nombre d'entrée N_i variable (NC N_ix1), la fonction « F=X_1And X_2 AndX_3...And X_{Ni} ». Comme la fonction « And » est un opérateur commutatif, tous les poids synaptiques (à part le « Seuil ») devraient avoir le même poids synaptique ($W_{ij}=W_0$) ce qui simplifie notre analyse.

Par définition de la fonction *And,* toutes les entrées doivent être actives pour que la sortie le soit :

- $Y_j= X_1W_{X1} + X_2W_{X2} +...+X_{Ni}W_{Ni} +X_bW_b>0=1$ si les N_i entrées X_i sont activées.

De plus, il suffit que l'une des entrées soit inactive pour que la sortie le soit :

- $Y_j= X_1W_{X1} + X_2W_{X2} +...+X_{Ni}W_{Ni} +X_bW_b<0=-1$ si l'une des entrées X_i est désactivée.

L'entrée « bias » doit distinguer entre le cas ou toutes les entrées (N_i) sont activées ($Y_j=N_iW_0+W_bX_b>0$) et le cas précédent où l'une des entrées est désactivée ($Y_j=(N_i-2)W_0+W_bX_b<0$). Par conséquent, le « seuil » doit avoir une valeur appartenant à l'intervalle [(N-2).W_0 ; N.W_0] et une résolution minimale de 2.W_0 est nécessaire pour une plage de variation du poids synaptique de 2.N_i.W_0. Cela revient à une précision de 1/N_i ce qui implique une relation linéaire entre le nombre de niveaux de conductance nécessaires pour apprendre une fonction et son nombre d'entrées N_i.

Afin de tracer la résolution nécessaire en fonction du nombre d'entrées, nous avons simulé l'apprentissage de NC N_ix1 pour un nombre d'entrées N_i variable avec les paramètres de simulation suivants : ($V_{T0}=1$, $V_P/V_{P-}=V_{T0}/-V_{T0}$, $V_H/V_L=0.4/-0.4$, G_{init}=rand(N_i,1)*0.1, $G_{min}=0$ et $G_{max}=10$, nombre de répétitions de l'apprentissage 500, critère d'arrêt de l'apprentissage est fixé à $N_{epoq}=50$).

Nous avons réalisé l'apprentissage avec des memristors à un seul niveau de conductance, ce qui revient à des memristors à deux états *On* et *Off*. Dans le cas où l'apprentissage ne converge pas, nous augmentons le nombre de niveaux N_v jusqu'à ce que l'apprentissage converge ce qui donne le nombre minimum de niveaux nécessaires pour converger l'apprentissage (figure III.18 (b)). Nous mesurons aussi le nombre d'époques d'apprentissage N_{epoq} nécessaire pour que l'apprentissage converge en présence des memristors avec une résolution minimale (figure III.18 (a)).

G_{inc}=ones(N_i,1)*F, F=G_{max}/N_v , (N_v=1 2 3 4 …) ;

Tant que les memristors de NC n'atteignent pas la résolution minimale, l'apprentissage diverge après avoir dépassé le critère d'arrêt fixé à $N_{epoq}=50$. Une fois que les memristors atteignent la résolution minimale qui leur permet de converger, le nombre d'époques d'apprentissage atteint une valeur minimale N_{epoq}. L'augmentation de la résolution des memristors au-delà de la résolution minimale ne fait pas diminuer

ce nombre d'époques. Néanmoins, l'augmentation du nombre d'entrées de NC augmente le N_{epoq} et le Nv_{min} selon les deux graphes de la figure III.18. Comme nous l'avons prédit précédemment, la relation entre le nombre de niveaux minimum (Nv_{min}) nécessaires pour faire converger l'apprentissage semble être linéaire en fonction du nombre d'entrées (figure III.18 (b)).

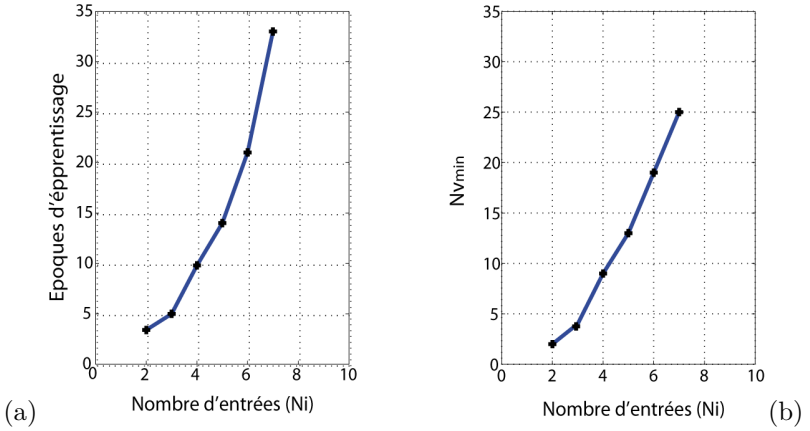


Figure III.18 : (a) nombre d'époques d'apprentissage nécessaire pour faire converger le NC N_ix1 en fonction de N_i en présence d'une résolution Nv_{min}. (b) résolution minimale nécessaire au memristors pour apprendre la fonction « And » à N_i entrées.

Les deux graphes de la figure III.18 s'avèrent intéressants pour déterminer pour un NC N_ix1 donné la résolution minimale et le nombre d'époques minimum nécessaire pour apprendre la fonction « And » ($Nv_{min} \approx 4(N_i-2)$). En se basant sur le graphe (b), la condition que doivent satisfaire les memristors pour faire converger le NC est alors la suivante :

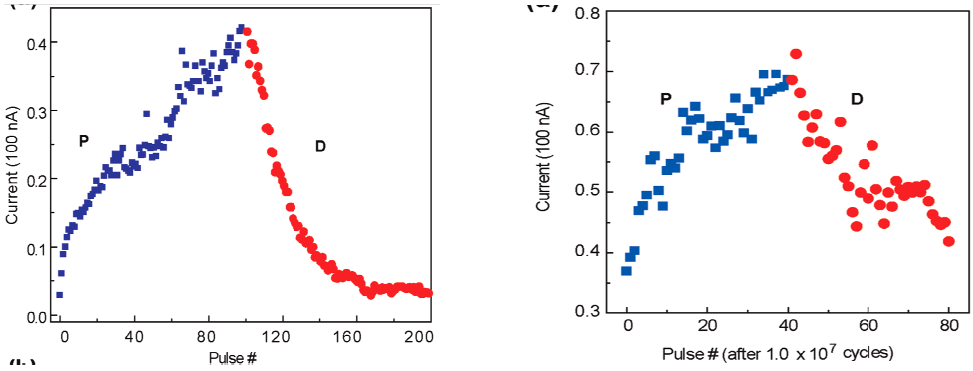
$$\frac{G_{\max} - G_{\min}}{G_{inc}} > Nv_{\min} - 1 \qquad \text{(III.25)}$$

Par conséquent, le pas d'apprentissage G_{inc} ne doit pas dépasser le terme :

$$G_{inc} < G_{inc\max} = \frac{G_{\max} - G_{\min}}{Nv_{\min} - 1} \qquad \text{(III.26)}$$

III.2.6.2 Simulation de la dispersion du pas d'incrémentation Ginc

Avant de montrer les résultats de nos simulations sur l'impact de la variation de pas d'incrémentation sur la convergence, nous souhaitons montrer un exemple de mesures expérimentales du pas d'incrémentation du courant, mesuré après avoir appliqué une série d'impulsions (positives et négatives) sur un nanocomposant de type memristor réalisé par l'équipe de *Wei Lu* de l'université de Michigan [Jo10]. La figure III.19 montre deux graphes correspondant aux cas d'un memristor neuf (a) et d'un memristor programmé 10^7 fois (b) où la dispersion du pas d'incrémentation augmente quand le composant vieillit.



(a) (b)

Figure III.19: (a) Courant mesuré expérimentalement dans un memristor en fonction du nombre d'impulsions appliquées, de 0 à 100 c'est des impulsions positives de (3.2 V, 300 μs) et de 100 à 200 c'est des impulsions négatives de (-2.8, 300 μs) (b) après 10^7 cycles d'écriture [Jo10].

Toutes les simulations des parties précédentes ont été effectuées avec un pas d'incrémentation G_{inc}=1 et une plage de conductance de [G_{min}=0 G_{max}=10] ce qui revient à une résolution de Nv=10 niveaux. Afin de déterminer l'effet de l'augmentation du pas d'incrémentation sur l'apprentissage, nous avons réalisé en première étape la simulation de l'apprentissage de la fonction F3=X1 And X2 And X3 avec un NC 8x1 en variant G_{inc} (1 à 10) de l'un des memristors à chaque fois. Les résultats de cette simulation ont montré que l'augmentation du pas d'incrémentation de l'un des huit memristors n'influence pas la convergence. L'apprentissage compense la présence d'un memristor possédant un grand G_{inc}, en augmentant le nombre d'époques d'apprentissage de quelques itérations et en profitant de la flexibilité des autres memristors l'apprentissage fini par trouver la bonne configuration pour converger.

Si maintenant le pas d'incrémentation de tous les memristors augmente en même temps, la limite de G_{inc} qui permet de faire converger le NC 8x1 pour apprendre la fonction F3 est alors calculée à partir de la formule III.26. Pour une fonction à trois entrées N_{Vmin}≈3.5 (récupéré du graphe b de la figure III.18).

$$G_{inc\,\max} = \frac{G_{\max} - G_{\min}}{Nv_{\min} - 1} = \frac{10}{3.5 - 1} = 4$$

Nous faisons l'hypothèse que la dispersion du pas d'incrémentation suit une loi normale centrée autour d'un pas G_{incr0} et d'un écart type σ donné. Pour cela, nous considérons dans la simulation suivante des pas d'incrémentation G_{inc} des memristors d'un NC 8x(6+R) distribués aléatoirement suivant une loi gaussienne : G_{inc0}=1 et G_{inc} $=f^{+}(G_{inc0}+\xi)$.

Pour déterminer le degré de tolérance de NC 8x(6+R) à ces variations, nous avons mesuré le taux de convergence de l'apprentissage de la fonction F3 en présence de cette dispersion à plusieurs valeurs d'écart type σ. Pour chaque valeur de σ l'apprentissage est répété 500 fois avec une injection aléatoire à chaque fois.

Les paramètres de simulation sont les suivants : V_{T0}=1, V_P/V_{P-}=V_{T0}/-V_{T0},V_H/V_L=0.4/-0.4, G_{init}=randn(8,6+R)+G_{max}, G_{min}=0 et G_{max}=10, N_{epoq}=50, N_{exp}=500.

Les courbes présentées dans la figure III.20 montrent le taux de convergence de NC 8x(6+R) obtenus en fonction de σ dans le cas de plusieurs valeurs de R. L'apprentissage sans redondance (R=0) montre une tolérance aux variations de G_{inc} d'un écart type σ≈1. En utilisant l'apprentissage compétitif, l'ajout de neurones redondants augmente les chances de trouver des neurones gagnants. L'ajout de six (R=6) neurones a amélioré la tolérance aux variations de G_{inc} de 3.5xσ. Cela montre que l'apprentissage n'est pas très sensible aux variations du pas d'incrémentation.

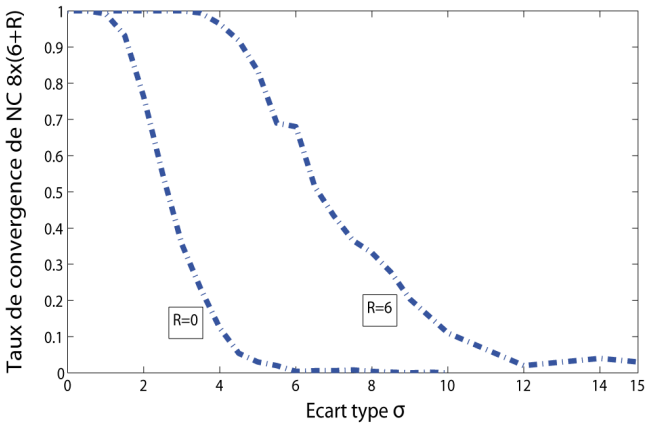


Figure III.20: Résultats de simulation de taux de convergence de NC 8x(6+R) en fonction de l'écart type σ de la distribution gaussienne sur les pas d'incrémentation G_{inc} des memristors.

III.2.6.3 Modélisation de la variation du pas d'incrémentation G_{inc}

Nous avons vu dans les simulations Monte-Carlo précédentes qu'en présence d'une variation gaussienne du pas d'incrémentation, la probabilité de convergence chute à partir d'une certaine valeur d'écart type σ. La limite de pas d'incrémentation permettant la convergence de NC 8x1 pour apprendre la fonction F3 a été déterminée précédemment à partir de l'inégalité III.26, $G_{incmax} \leq 4$. En présence d'une erreur ξ sur le G_{inc}, l'inégalité III.26 devient :

$$G_{inc0} + \xi < G_{inc\,\max} = \frac{G_{\max} - G_{\min}}{Nv_{\min} - 1}$$

$$\xi < G_{inc\,\max} - G_{inc0} = \frac{G_{\max} - G_{\min}}{Nv_{\min} - 1} - G_{inc0}$$

La probabilité qu'un memristor fonctionne en présence d'une dispersion gaussienne de G_{inc} avec un écart type σ, est donnée donc par la relation suivante :

$$P_{mem}(\xi < G_{inc\,\max} - G_{inc0}) = \frac{1}{2} erf\left(\frac{G_{inc\,\max} - G_{inc0}}{\sqrt{2}\sigma}\right) + \frac{1}{2} \qquad \text{(III.27)}$$

Comme nous l'avons vu dans les parties précédentes, l'apprentissage de la fonction F3 nécessite le fonctionnement correcte des quatre memristors cruciaux $\{M_{x1+}, M_{x2+}, M_{x3+}, M_{b-}\}$ (N_m=4). Cependant nous avons effectué des simulations pour déterminer l'influence de l'augmentation du pas d'incrémentation de chaque memristor sur la convergence. Les résultats de ces simulation ont montré que la résolution du memristor M_{b-} lié à l'entrée « Seuil» n'est pas importante pour faire converger l'apprentissage. L'apprentissage n'est donc sensible qu'à la résolution des trois memristors $\{M_{x1+}, M_{x2+}, M_{x3+}\}$ (N_m=3).

La probabilité qu'un neurone apprenne cette fonction est alors donnée par : $P_{neuro} = (P_{mem})^{N_m=3}$ et la probabilité P_{NCR} qu'un NC 8x(6+R) apprenne cette fonction peut être calculée en remplaçant P_{neuro} calculée ici dans l'équation (III.23).

Afin de comparer ces probabilités de convergence avec les courbes de simulation de la figure III.20, nous les avons tracées sur la même figure III.21 (courbes continues).

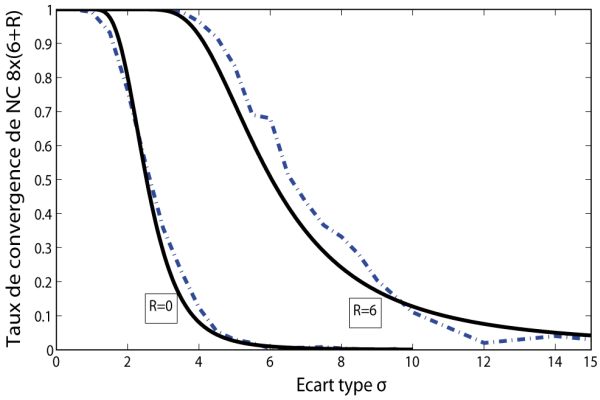


Figure III.21 : Taux de convergence de NC NC 8x(6+R) en fonction de σ de variation de G_{inc}, courbes discontinues issues de la simulation et courbes continues calculées à partir du modèle.

Les courbes continues présentées sur la figure III.21 ont été tracées à partir du modèle prédictif précédent avec un G_{incmax}=5 au lieu de G_{incmax}=4 de telle sorte que les courbes théoriques soient voisines de celles simulées. Ce décalage des courbes de simulation par rapport aux courbes issues du modèle proposé montre que notre modèle est plus pessimiste que la simulation.

Maintenant que nous avons validé le modèle prédictif, nous allons tracer la relation entre le nombre de neurones redondants R qu'il faut ajouter au NC 8×(6+R) pour avoir 100% de convergence de l'apprentissage de la fonction F3, en présence d'une variation gaussienne (σ, G_{inc0}) du pas d'incrémentation de la conductance des memristors. Le résultat est présenté sur la figure III.22 montrant la relation entre R et σ. Cette courbe confirme encore la très bonne tolérance aux variations du pas d'incrémentation G_{inc}. Par exemple, l'apprentissage n'a besoin que de dix neurones redondants pour tolérer des variations de σ=3.

Figure III.22 : Redondance nécessaire pour une convergence de 100% en fonction de l'écart type σ de la variation gaussienne de pas d'incrémentation.

III.2.6.4 Simulation de la dispersion de la plage de variation de la conductance des memristors

Pour avoir une idée sur l'ordre de grandeur des variations des conductances de nanocomposants de type memristors, nous présentons dans la figure III.23 un exemple de mesures expérimentales des résistances de 26 memristors [Borg10]. Les mesures montrent une dispersion des résistances distribuées suivant une loi gaussienne autour d'une moyenne R_{on0}=8 Kohm et d'un écart type relatif σ_{Ron}=0.8 pour l'état *On* et d'une moyenne R_{off0}=1.07 10^4 Kohm et d'un écart type relatif est σ_{Roff}=1.04 pour l'état *off*. La présence d'une telle dispersion sur nos composants a pour conséquence la limitation de la plage de variation de conductance ainsi que la diminution du nombre de niveaux entre l'état *On* et l'état *Off*.

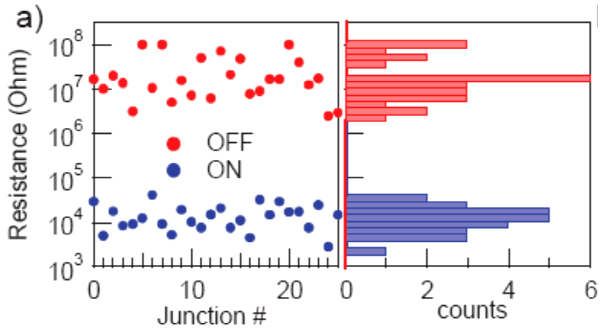


Figure III.23 : Mesure expérimentale de la résistance pour 26 memristors [Borg10].

La plage de variation de la conductance des memristors utilisée par défaut dans les simulations précédentes était comprise entre G_{min}=0 et G_{max}=10, ce qui a permis une variation du poids synaptique ($W=G_+-G_-$) entre $W_{min}= G_{min+}- G_{max-}$=-10 et $W_{max}= G_{max+}- G_{min-}$=10. Avant de simuler la dispersion gaussienne des plages de variation de conductance des memristors, nous avons commencé par la détermination de la limite de la plage de variation des conductances de memristors permettant la

convergence de NC dans le cas de l'apprentissage de la fonction F3 avec un seul neurone (NC 8×1). En limitant à chaque fois la conductance de l'un des memristors et en mesurant le taux de convergence, nous avons pu déterminer la limite tolérée par chaque memristor (tableau III.11). La limitation de la conductance se fait ici en diminuant la conductance maximale G_{max}=10-ξ avec ξ variant de 0 à 10. Nous considérons que la conductance minimale ne change pas (G_{min}=0).

G_{min}=0 G_{max}=10-ξ (ξ vari de 0 à 10)	Condition de convergence
G_{b+}	quelque soit ξ
G_{X3+}	ξ<7.5
G_{X2+}	ξ<8
G_{X1+}	ξ<8
G_{b-}	ξ<7.5
G_{X3-}	quelque soit ξ
G_{X2-}	quelque soit ξ
G_{X1-}	quelque soit ξ

Tableau III.11 : Condition de convergence sur la plage de variation des poids synaptiques pour apprendre la fonction F3

Les résultats du tableau III.11 montrent que l'apprentissage de la fonction F3 diverge dans le cas où la plage de variation de la conductance des memristors cruciaux devient inférieure à 2 dans le cas des deux memristors {M_{x2+}, M_{x1+}} ou inférieure à 2.5 pour {M_{b-}, M_{x3+}}. La limitation de la plage de variation des autres memristors n'empêche pas l'apprentissage de converger.

Nous allons maintenant répéter la même simulation de l'apprentissage de la fonction F3 avec un NC 8×(6+R) en présence d'une limitation de variation de la conductance ξ distribuée aléatoirement sur les memristors suivant une loi gaussienne (σ). Les paramètres de simulation sont les suivants : V_{T0}=1, $V_P/V_{P-}=V_{T0}/-V_{T0}$, V_H/V_L=0.4/-0.4, G_{init}=randn(8,6+R)+10, G_{inc}=1, N_{epoq}=50, N_{exp}=500.

```
ξ= randn(2*3+2,6+R)*σ
Gmax=10+ξ;
Gmin=zeros(2*NB_in+2,NB_out);
```

Les courbes montrant le taux de convergence de NC 8×(6+R) obtenues en simulation en fonction de l'écart type σ sont présentées dans la figure III.24. La courbe correspondant à R=0 montre qu'en absence de neurones redondants le NC 8×6 peut tolérer des limitations de la plage de variation d'un écart type σ=2. Au-delà de cette valeur, le taux de convergence se dégrade progressivement pour devenir très faible pour des valeurs de σ>8. L'ajout de six neurones redondants (R+6) a

amélioré de presque 3 fois la tolérance, ainsi le NC 8×(6+6) peut converger à 100% jusqu'à une valeur de σ=5.5. Ces résultats montrent que l'apprentissage est possible même avec les caractéristiques des memristors de HP (σ_{Roff}=1.04, σ_{Ron}=0.8) vu précédemment dans la figure III.23.

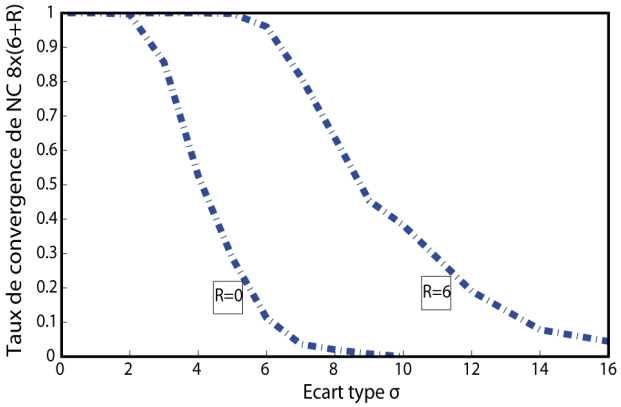


Figure III.24 : Taux de convergence de NC 8x(6+R) en fonction de l'écart type de la limite de variation de la conductance.

Une très grande tolérance aux limitations de la plage de conductance de NC à trois entrées est alors possible. Cela est dû à la petite plage de conductance que nécessite le NC pour apprendre la fonction « And » à trois entrées. Cependant, l'augmentation du nombre d'entrées de NC nécessitera de plus en plus de résolution et de plage de variation de conductance ce qui fait diminuer la tolérance.

Selon les paramètres du memristor utilisé (G_{min}, G_{max}, Nv_{min}, G_{inc}, σ), un travail d'optimisation est nécessaire pour déterminer le nombre d'entrées, le nombre de neurones de sortie et le nombre de neurones redondants que peut utiliser le NC pour apprendre une fonction en gardant un taux de tolérance raisonnable.

III.2.6.5 Modélisation de la dispersion de la plage de variation de la conductance des memristors

Afin de modéliser les courbes de simulation précédentes, nous avons besoin de connaitre la limite de la plage de conductance au-delà de laquelle le NC diverge. Cette limite peut être calculée à partir de l'inégalité III.25 et à partir de tableau III.11 qui montre la limite de convergence pour chaque memristor.

D'après III.25 : $G_{max} - G_{min} > G_{inc}(Nv_{min} - 1)$

Dans le cas d'une perturbation gaussienne ξ sur la conductance G_{max} l'inégalité III.25 devient :

$$(G_{max} + \xi) - G_{min} > G_{inc}(Nv_{min} - 1)$$

$$\xi > G_{min} - G_{max} + G_{inc}(Nv_{min} - 1) \qquad \text{(III.28)}$$

Dans la simulation réalisée précédemment $G_{inc}=1$, $G_{min}=0$ et $G_{max}=10$. D'après le graphe (b) de la figure III.18, pour une fonction à trois entrées (F3) $Nv_{min}=3.5$.

L'application de ces valeurs dans III.28 donne: $\xi > -7.5$.

Ce résultat est cohérant avec les résultats du tableau III.11 qui montrent que les deux memristors cruciaux $\{M_{b-}, M_{x3+}\}$ ne peuvent pas fonctionner pour des conductances G<(10-7.5) ce qui fait diverger l'apprentissage.

Pour une distribution gaussienne de ξ, la probabilité P_{neuro} que le neurone fonctionne est donnée par la probabilité que les quatre memristors cruciaux fonctionnent :

$$P_{neuro} = P_{b-} \times P_{X3+} \times P_{X2+} \times P_{X1+}$$

$$P_{neuro} = \left[P_{mem}(\xi > -7.5)\right]^2 \left[P_{mem}(\xi > -8)\right]^2$$

$$P_{neuro} = \left[\frac{1}{2} + \frac{1}{2} erf(\frac{7.5}{\sqrt{2}\sigma})\right]^2 \left[\frac{1}{2} + \frac{1}{2} erf(\frac{8}{\sqrt{2}\sigma})\right]^2$$

La probabilité P_{NCR} de convergence de NC 8x(6+R) peut être calculée en remplaçant P_{neuro} calculé ici dans l'équation III.23. Les courbes obtenues sont tracées sur la figure III.25 et comparée avec les courbes de simulation. Le modèle que nous avons développé est en accord avec les simulations.

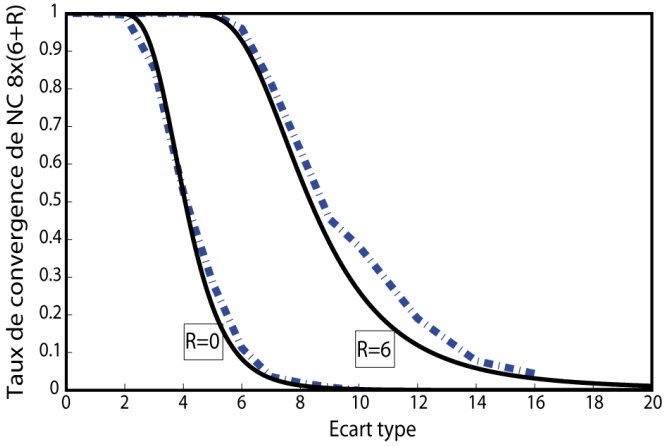


Figure III.25 : Taux de convergence de NC 8x(6+R) en fonction de σ. Les courbes discontinues issues de simulation et les courbes continues calculées à partir du modèle.

La courbe de la figure III.26 a été tracée à partir du modèle développé précédemment, montrant la relation entre le nombre de neurones redondants R qu'il faut ajouter au NC 8×(6+R) pour avoir 100% de convergence de l'apprentissage de la fonction F3, en présence d'une limitation gaussienne (σ) de la plage de conductance (G_{max}) des memristors. La courbe montre que pour un σ<2 l'apprentissage peut converger sans recourir à des neurones redondants (R=0) et pour un σ>2, le nombre

de neurones redondants nécessaires pour faire converger l'apprentissage augmente avec l'augmentation de σ.

Figure III.26 : Redondance nécessaire pour une convergence de 100% en fonction de l'écart type σ de la limitation gaussienne de la plage de variation de conductance.

IV. Conclusion

Le taux de défauts élevé présent dans les nanoarchitectures actuelles requiert le développement de techniques de tolérance et de reconfiguration qui vont permettre de supporter ces défauts. Plusieurs techniques ont été développées pour la tolérance de défauts et de fautes pour les nanoarchitectures [Heat98, Niko02, Han03, Huan04, Bhad05, Chen05, Kuek05, Taho05, Hogg06, Huan06, Rao06, Tehr08]. Cependant, la plupart de ces techniques nécessitent des surfaces et des temps de test importants, ce qui fait perdre l'intérêt des nanoarchitectures.

Nous avons montré dans le chapitre précédent la technique basée sur l'apprentissage neuronal utilisée pour configurer les nano-crossbars. Afin de démontrer la robustesse de cette technique, nous avons étudié tout au long de ce chapitre le degré de tolérance de l'apprentissage neuronal aux défauts et aux dispersions des caractéristiques des nanocomposants. Nous avons vu dans la partie défauts sur les neurones que la présence de neurones cachés défaillants dans un MLP n'empêche pas les neurones de sortie de converger. Grace aux simulations Monte-carlo nous avons mesuré, dans la partie défauts et dispersion des caractéristiques des synapses, le taux de tolérance de l'apprentissage en présence de défauts de collage sur les memristors « collage de la conductance à G_{min} et à G_{max} ». Nous avons aussi mesuré le taux de tolérance aux dispersions des caractéristiques des memristors « tension de seuil, tension d'entrée, résolution, pas d'incrémentation, plage de variation de la conductance ». La plupart de ces simulations ont montré des taux de tolérance importants comparés à des mesures de dispersions expérimentales (memristors d'HP par exemple). L'ajout des neurones redondants a montré une amélioration considérable du taux de convergence pour la plupart des cas étudiés. Nous avons développé pour chaque type de défaut et dispersion un modèle

permettant de calculer la probabilité de convergence de l'apprentissage. Tous les modèles proposés ont été validés par comparaison aux courbes de convergence obtenues en simulation.

Les temps de simulation Monte-Carlo importants rendent difficile l'étude de robustesse de blocs neuro-crossbar à grand nombre d'entrées. Par conséquent, la robustesse a été étudiée pour un NC à trois entrées seulement. Cependant, les modèles probabilistes proposés peuvent être utilisés pour prédire la probabilité de convergence pour un NC à grand nombre d'entrées/sorties.

Afin de comparer la robustesse de l'approche neuronale avec les autres techniques de tolérance aux fautes utilisées dans l'étude de *Nikolic* [Niko], nous avons utilisé le modèle de la section (III.2.4.2) pour prédire le taux de redondance nécessaire pour faire converger à 90% un circuit de $N=10^{12}$ nanocomposants, regroupés dans $d=10^9$ unités, chaque unité contenant $Nc=10^3$ nanocomposants [Chab10]. La comparaison présentée sur la figure III.27 montre une meilleure tolérance aux défauts pour l'approche neuronale avec un apprentissage compétitif.

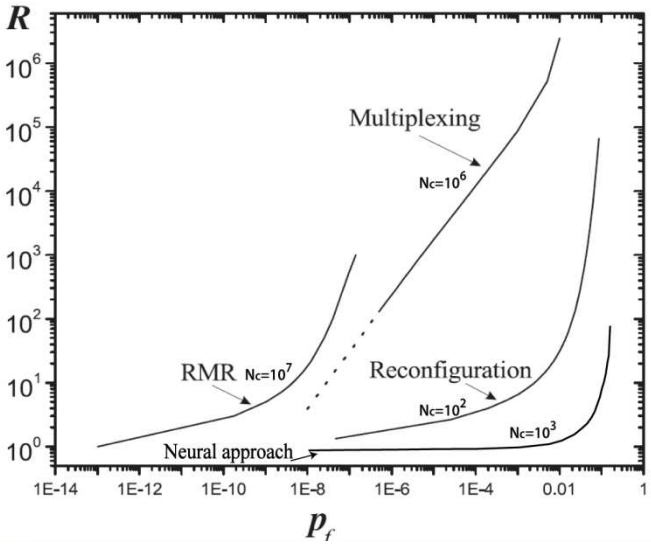


Figure III.27 : Comparaison de l'efficacité de la technique de tolérance aux fautes neuro-inspirée par rapport aux techniques de tolérance aux fautes les plus connues (RMR, Multiplexage de Von Neumann, Reconfiguration), [Chab10].

CHAPITRE IV

Démonstration expérimentale d'apprentissage avec un crossbar d'OGCNTFET

I. Introduction

Les travaux que nous allons présenter dans ce chapitre ont été effectués dans le cadre du projet PANINI (*Programme Architectures Nanoélectronique Intégrées Neuro-Inspirées*). L'une des originalités de ce projet est la volonté de réaliser une démonstration expérimentale d'un apprentissage basé sur des nanocomposants réels. Le choix s'est tourné vers les transistors à nanotube de carbone à commande optique (OG-CNTFET) compte tenue de leur originalité et de leur disponibilité au sein de consortium dès le début du projet. Trois tâches ont été développées en parallèle pour la mise en œuvre de ce démonstrateur.

- La première tâche dédiée au CEA-LEM consiste à réaliser et caractériser les tapis d'OG-CNTFET en crossbar. Les premiers travaux initiés par *Julien Borghetti* ont été poursuivis par *Guillaume Agnus* et finalisé par *Karim Gacem,* sous la direction de *Vincent Derycke*.

- La deuxième tâche est dédiée à notre équipe « NanoArchi » à l'IEF. Elle consiste à développer et implémenter des procédés d'apprentissage neuronal sur un circuit électronique de type FPGA permettant de piloter le processus d'apprentissage du tapis d'OGCNTFET fourni par le CEA-LEM. Ces travaux ont été réalisés par notre ingénieur de recherche *Jean-Marie RETROUVEY* que j'ai poursuivi par la suite, sous la direction de *Jacques-Olivier KLEIN*.

- La troisième tâche est dédiée à l'IMS-Bordeaux et qui consiste à développer un modèle compact de transistor OGCNTFET en se basant sur les caractéristiques électriques mesurées par le CEA-LEM. Les résultats de ces travaux sont exploités pour valider en simulation les procédés d'apprentissage. La majorité de ces travaux ont été réalisés dans le cadre de la thèse de *Si-Yu LIAO* [Liao11b], sous la direction de *Cristell MANEUX*.

Nous allons donner dans ce chapitre plus d'informations sur ces tâches permettant au lecteur de comprendre le fonctionnement du démonstrateur, que nous allons présenter à la fin de ce chapitre avec les derniers résultats obtenus sur l'apprentissage de fonctions logiques à une, deux et trois entrées.

II. Neuro-crossbar à base d'OGCNTFET

II.1 Transistor à effet de champ commandé optiquement (OG-CNTFET)

Comme nous l'avons évoqué en bref dans le chapitre 1 section IV.2.3, l'OG-CNTFET est un transistor basé sur le CNTFET conventionnel de type P avec un oxyde de grille en SiO_2 et une grille arrière [Borg06, Agnus10, WS10, Liao11]. L'OG-CNTFET montre des caractéristiques d'un élément mémoire non-volatile. Il est optiquement commandé grâce à un enrobage de film mince de polymère photosensible P3OT (figure IV.1 (a)). Sous illumination laser, les paires électrons photo-générés dans le polymère photosensible conduisent l'OG-CNTFET, en piégeant les électrons à proximité de son canal en nanotube de carbone, vers un état *On* où la conductance est maximale indépendamment de la tension de grille (Figure IV.1 (b)). Il a été démontré qu'en modulant la puissance du laser, il est possible d'augmenter par pas la conductance de l'OG-CNTFET [Agnus10]. Plus la puissance du laser augmente plus la quantité de charges piégées à proximité du canal augmente (figure IV.1 (c)). L'évacuation de ces charges piégées est directement liée au champ électrique appliqué à l'oxyde de la grille. L'effacement de l'OG-CNTFET est possible électriquement, en appliquant soit des impulsions *négatives* sur la grille (relative à une tension constante sur la source) soit des impulsions *positives* sur le drain ou la source (relative à une tension constante sur la grille).

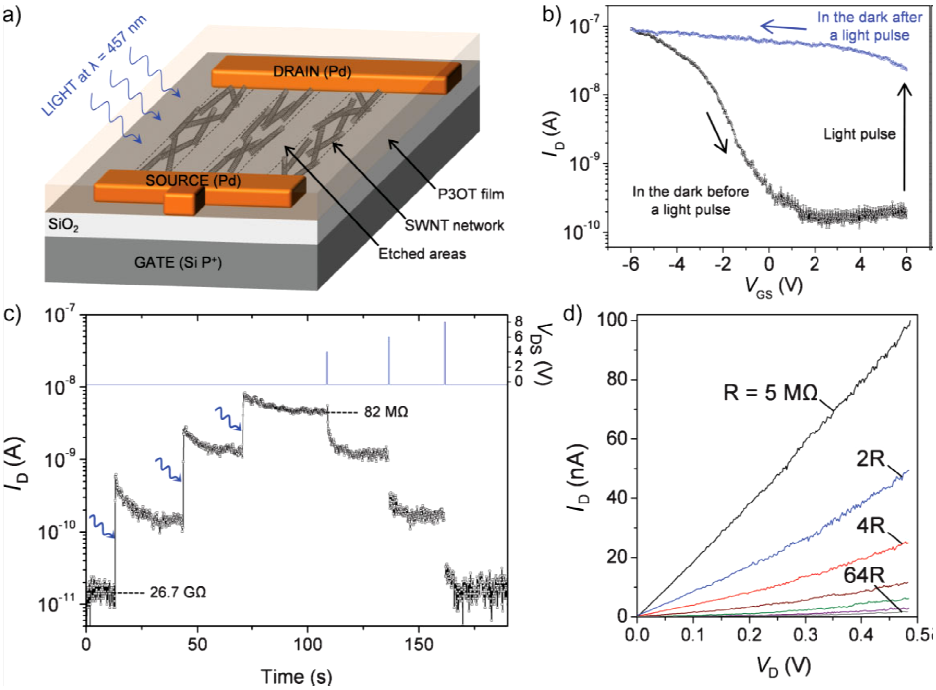


Figure IV.1 : (a) Schéma d'un réseau de nanotubes de carbone formant un OG-CNTFET. (b) caractéristique de transfert $I_D(V_{GS})$ avant (noir) et après (bleu) l'envoie de fuseau lumineux. (c) Courant de drain (I_D) en fonction du temps de l'OG-CNTFET travaillant comme un composant à deux électrodes (à une tension V_{GS}=+2V). Des impulsions optiques et électriques sont utilisées pour programmer la résistance de composant. (d) Courbes $I_D(V_{DS})$ obtenues pour le même OG-CNTFET montrant plusieurs niveaux de résistances programmables possibles (figure tirée de [Agnus10]).

La figure IV.1 (c) montre l'effacement par pas en appliquant des impulsions positives (V_{DS}) d'amplitudes supérieures à la tension de la grille (fixée à V_G=2V) sur le drain. Toutes ces propriétés montrent que l'OG-CNTFET est un très bon candidat pour fabriquer des mémoires ou synapses dans les circuits neuromorphiques.

Comme nous l'avons évoqué dans le chapitre précédent (II.2.6), la résolution des synapses de neuro-crossbar est très importante pour avoir une meilleure flexibilité et une grande capacité d'apprentissage. L'OG-CNTFET montre une très bonne résolution (figure IV.1 (d)) où les niveaux de résistance programmable peuvent aller de R=5 MOhm jusqu'à R=64R. Cependant, comme l'OG-CNTFET est un transistor (à trois électrodes) on doit adapter la méthode d'apprentissage dédiée aux memristors (à deux électrodes) proposée dans le chapitre II (II.2.3). Nous allons voir dans la partie suivante la méthode d'apprentissage permettant de configurer le neuro-crossbar à base d'OG-CNTFET.

II.2 Méthode d'apprentissage

La méthode d'apprentissage utilisée ici est destinée à être appliquée à une matrice de dispositifs OG-CNTFET dont les drains correspondent aux entrées, les sources aux sorties, et dont les grilles sont communes à toutes les synapses connectées à la même sortie (neurone). Les entrées sont différentielles (positives et négatives) de façon à émuler une « conductance signée » au moyen d'un couple de dispositifs OG-CNTFET (figure IV.2). Pour faire varier la conductance de ces transistors, il faut un mécanisme d'apprentissage qui dépend de l'entrée et de la sortie. La seule configuration qui a pu être démontrée expérimentalement [Ws10] correspond à des impulsions de programmation (V_{D_PROG}) appliquées aux entrées (drains), avec un effet de protection par la grille qui permet de protéger l'OG-CNTFET dans le cas où il n'est pas sensé être programmé. Le processus d'apprentissage se base sur la combinaison des propriétés d'OG-CNTFET présentées dans la partie précédente :

1. La commande optique est utilisée comme une opération RESET pour initialiser les transistors de la matrice neuro-crossbar avant l'étape de l'apprentissage. Ils se trouvent alors tout dans leur état de conductance maximum.
2. La commande électrique est utilisée pendant l'apprentissage pour programmer les conductances des transistors. Des impulsions de programmation positives sont appliquées sur les entrées (drains), permettant d'évacuer les charges piégées pendant l'opération RESET et par conséquent diminuent la conductance.

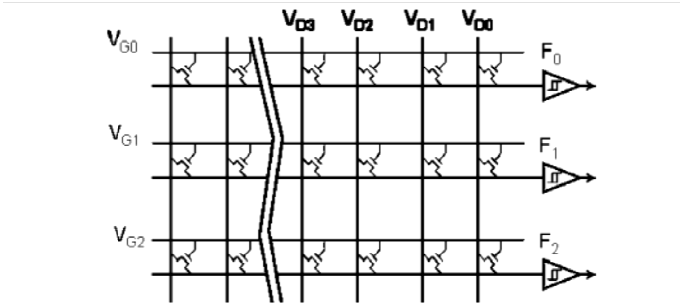
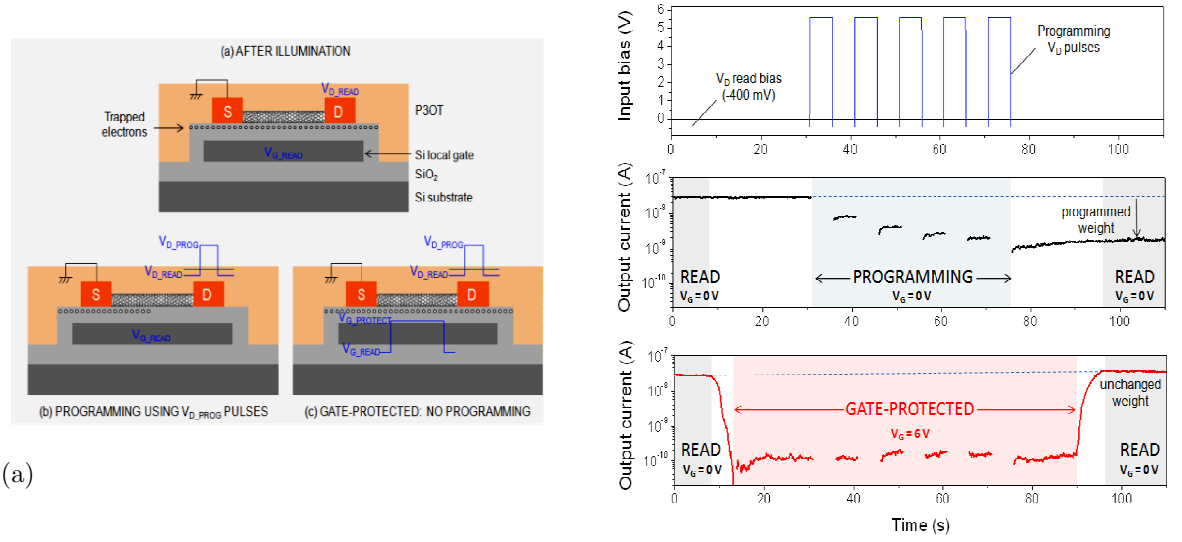


Figure IV.2 : Neuro-crossbar d'OG-CNTFET à trois entrées et trois grilles de sortie (NC 8x3).

Dans le cas d'un neuro-crossbar à plusieurs sorties, l'application des impulsions de programmation par les entrées peut affecter les transistors correspondant à des sorties correctes qui ne demandent pas d'être programmés. Cependant, la technique de protection par la grille de ces sorties correctes permet de protéger leurs transistors. De cette façon seuls les transistors connectés aux sorties entachées par l'erreur considérée ne sont affectés. La protection par la grille (commune par ligne) ce fait par l'application d'un potentiel de grille ($V_{G\_PROTECT}$) proche de celui appliqué aux drains (V_{D_PROG}) lors de l'impulsion de programmation. De cette façon aucun champ n'est appliqué entre la grille et le drain et donc aucun changement de conductance n'affecte l'OG-CNTFET (figure IV.3).

(b)



(a)

Figure IV.3 : (a) Après illumination des charges sont piégées dans le SiO_2 proche de la surface. (b) L'application d'impulsions positives sur le drain fait décroitre la conductance en absence de la protection sur la grille et aucun effet sur la conductance en présence de la protection sur la grille.

La technique de protection de la grille permet non seulement de programmer les neurones en état d'erreur et protéger les neurones en état correcte, mais elle permet aussi un apprentissage parallèle en corrigeant toutes les erreurs de même type en même temps. Les deux types d'erreurs possibles (sortie attendue = 1, sortie obtenue

=0 et réciproquement) sont traitées distinctement suivant la règle de *Delta* par l'une des deux phases d'apprentissage « Alpha » et « Beta » (tableau IV.1).

Configuration	Xi	Yj	Xj	Signe (ΔW)	Séquences
0	-1	-1	-1	0	
1	-1	-1	1	1	Alpha
2	-1	1	-1	-1	Beta
3	-1	1	1	0	
4	1	-1	-1	0	
5	1	-1	1	-1	Alpha
6	1	1	-1	1	Beta
7	1	1	1	0	

Tableau IV.1 : Règle de Delta pour une entrée binaire

Phase Alpha : toutes les grilles sont protégées ($V_{G\_PROTECT}$) sauf celles qui sont reliées en sortie à un neurone dont l'état est positif (X=1) alors qu'il devrait être négatif (Y=-1). Pour éliminer ce type d'erreur, il va falloir diminuer le courant généré par la paire de transistors pour atteindre la sortie X=-1. Pour cela, une impulsion de programmation positive ($V_{Pulse}=V_{D\_PROG}$) est appliquée à toutes les entrées qui sont à l'état haut ($X_{i+}=V_H$) (figure IV.4). Cela est réalisé grâce au multiplexeur (AMuxp) par l'application d'une impulsion sur V_{P2p}, le potentiel $V_+=V_H$ est remplacé par une impulsion positive (V_P) de telle sorte que $V_+=V_P=V_{D\_PROG}$. Dans le cas où l'entrée est négative (X_i=-1 : $X_{i+}=V_L$, $X_{i-}=V_H$), l'impulsion de programmation provoque la diminution de la conductance (G_-) de transistor lié à X_{i-} réalisant par cela la configuration C1 qui nécessite l'augmentation du poids synaptique (ΔW=1). Dans le cas où l'entrée est positive (X_i=1 : $X_{i+}=V_H$, $X_{i-}=V_L$), l'impulsion de programmation provoque la diminution de la conductance (G_+) de transistor lié à X_{i+} réalisant par cela la configuration C5 qui nécessite la diminution du poids synaptique (ΔW=-1).

Phase Beta : toutes les grilles sont protégées ($V_{G\_PROTECT}$) sauf celles qui sont reliées en sortie à un neurone dont l'état est positif (X=-1) alors qu'il devrait être négatif (Y=1). Pour éliminer ce type d'erreur, il va falloir augmenter le courant généré par la paire de transistors pour atteindre la sortie X=1. Pour cela, une impulsion de programmation positive ($V_{Pulse}=V_{D\_PROG}$) est appliquée à toutes les entrées qui sont à l'état bas ($X_{i-}=V_L$) (figure IV.4). Cela est réalisé grâce au multiplexeur (AMuxN), en appliquant une impulsion sur V_{P2m} le potentiel $V_-=V_L$ est remplacé par une impulsion positive (V_P) de telle sorte que $V_-=V_P=V_{D\_PROG}$. Dans le cas où l'entrée est négative (X_i=-1 : $X_{i+}=V_L$, $X_{i-}=V_H$), l'impulsion de programmation provoque la diminution de la conductance (G_+) des transistors liés à X_{i+} réalisant par cela la configuration C2 qui nécessite la diminution du poids synaptique (ΔW=-1). Dans le cas où l'entrée est positive (X_i=1 : $X_{i+}=V_H$, $X_{i-}=V_L$), l'impulsion de programmation provoque la diminution de la conductance (G_-) des transistors liés à

X_{i-} réalisant par cela la configuration C6 qui nécessite l'augmentation du poids synaptique ($\Delta W=1$).

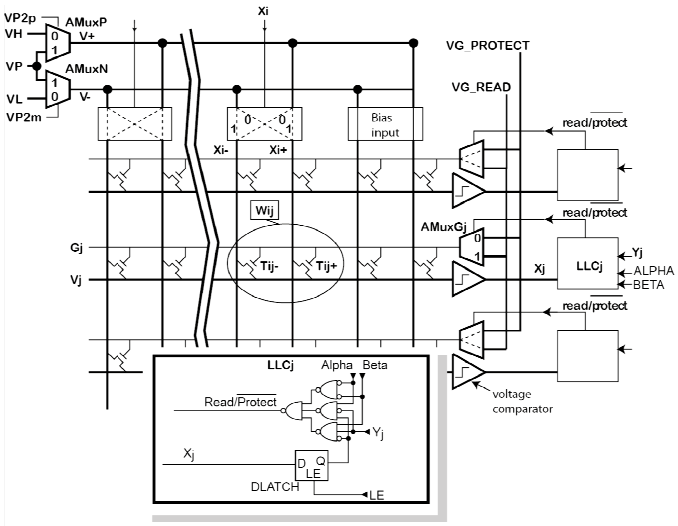


Figure IV.4 : Neuro-crossbar d'OG-CNTFET à trois entrées et trois grilles de sortie (NC 8x3).

En mode opérationnel, pour ne pas déprogrammer les conductances des transistors, les tensions appliquées aux entrées grâce aux deux multiplexeurs (AmuxP, AmuxN) sont ramené à des tensions faibles (V_H/V_L), et celles des grilles sont ramenées à une tension $V_G=V_{G\_READ}$ choisie à partir de la caractéristiques $I_D(V_{GS})$ de telle sorte à avoir une grande sensibilité de la conductance aux charges piégées par le laser lors du RESET.

II.3 Architecture du démonstrateur d'apprentissage d'OGCNTFET

Le circuit d'apprentissage est construit de façon à pouvoir générer les impulsions sur les entrées et appliquer la tension de protection sur les grilles là et quand c'est nécessaire. Les impulsions sur les entrées sont appliquées en remplaçant l'un des niveaux logiques (V_H ou V_L) par la tension de programmation au moyen de multiplexeurs analogiques commandés par les signaux V_{T2p} (pour appliquer la tension de programmation V_{Pulse} sur les entrées positives) ou V_{T2m} (pour appliquer la tension de programmation V_{Pulse} sur les entrées négatives). Les deux signaux V_{T2p} et V_{T2m} sont issus d'un automate. Le signal de protection de grille est généré par un multiplexeur analogique commandé par un signal numérique au niveau haut lorsque la grille doit être protégée (Read/protect). Ce dernier est calculé par un bloc combinatoire à partir des états attendus (Y) et obtenus (X) en sortie des neurones et des signaux A et B actifs respectivement lors des phases Alpha et Beta de l'apprentissage et tous deux également générés par l'automate. Enfin, une mémoire contient les patterns (Y) correspondant à la fonction à apprendre. Les sorties des neurones sont mémorisées

temporairement durant l'apprentissage par une bascule D placée en sortie du comparateur qui implémente la fonction d'activation des neurones.

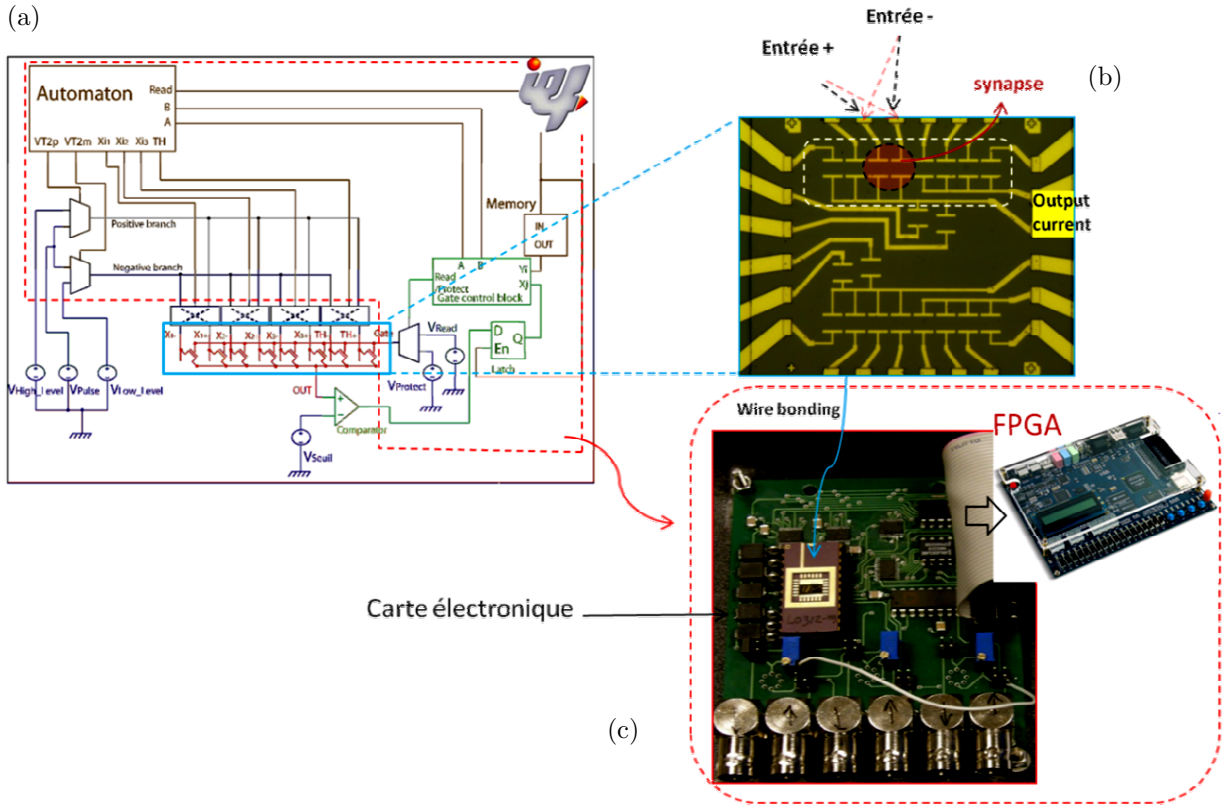


Figure IV.5 : (a) Version simplifiée du circuit mixte pour le démonstrateur d'apprentissage d'OG-CNTFET réalisé par *Jean-Marie RETROUVEY* à l'IEF. Dans cette architecture chaque couple d'OG-CNTFETs joue le rôle d'une synapse. (b) Implémentation physique de la partie nano de ce circuit réalisée par le LEM (zone de 100 µm X 100 µm) ; 8 OG-CNTFETs partageant la même électrode de sortie sont utilisés. (c) Carte électronique pilotée par un FPGA permettant le contrôle des différents signaux d'entrées/sorties du tapis d'OGCNTFET, réalisée à l'IEF.

La plupart des blocs du démonstrateur sont implémentés grâce au circuit FPGA (ALTERA De2-70). L'automate synthétisé dans le FPGA à partir d'une description VHDL effectue la présentation des entrées, la détection des erreurs, la protection des grilles et le déclenchement des deux phases d'apprentissage (figure IV.5 (c)). Un switch analogique contrôlé par un bloc combinatoire est ajouté en sortie du FPGA sur le coté grille du réseau de neurones pour choisir la tension à appliquer à la grille : V_{G_READ} ou $V_{G\_PROTECT}$. La carte électronique permet d'interfacer et de mettre à niveau les signaux logiques entre le FPGA et la matrice d'OG-CNTFET. Elle est connectée au FPGA par une nappe. Un comparateur de tension (AD8564) placé à la sortie de l'OG-CNTFET (source commune) implémente le neurone. Des switchs Quadruples d'Analog Devices (ADG 1634 TSSOP) sont utilisés pour gérer les tensions d'entrée du tapis. Le switch activant l'impulsion de programmation sur les entrées positives ou négative est implémenté par un composant MAXIM (Max 319).

Le tapis d'OG-CNTFET réalisant le neuro-crossbar est placé dans un boîtier DIL-24 par « bonding » (figure IV.5 (b)). Il est ensuite placé sur un support à force d'insertion nulle DIL-24 sur la carte électronique (figure IV.6). Cela permet de tester plusieurs échantillons pendant les manipulations d'apprentissage afin de trouver le bon.

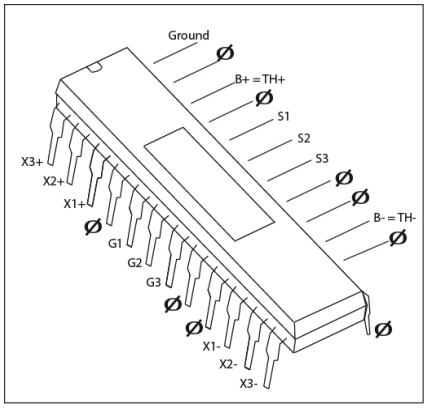


Figure IV.6 : Support DIL-24 permettant d'intégrer le neuro-crossbar d'OG-CNTFET dans le circuit démonstrateur.

II.4 Résultats d'apprentissage

L'expérience présentée ici a été mise au point par l'équipe du CEA-LEM et notre équipe « NanoArchi » (figure IV.7). L'objectif est de démontrer la possibilité d'apprendre des fonctions logiques à partir d'un neuro-crossbar d'OG-CNTFET. Les tapis d'OG-CNTFET utilisés sont de taille 6×1 et 8×1 comportant respectivement deux et trois entrées différentielles plus une entrée « bias » et une seule sortie.

Des régulateurs de tensions (ajustables et fixes), à partir des deux tensions -15V et 15V, sont utilisés pour alimenter la carte électronique. Les impulsions de programmation V_{H_LEVEL}, V_{L_LEVEL}, V_{pulse_input} et V_{pulse_gate} sont apportées via des sources de tensions externes.

La première étape consiste à réaliser grâce au « Laser » une opération RESET permettant l'initialisation la conductance des OG-CNTFET. Le courant de sortie (post-synaptique) du tapis d'OG-CNTFET est visualisé à l'oscilloscope permettant de voir l'allure du courant lors de l'initialisation. Le courant de sortie sature à son maximum signifie que les OG-CNTFET sont à l'état de conduction maximale ce qui nécessite d'arrêter l'envoie du « Laser ». La deuxième étape concerne le réglage des paramètres de l'apprentissage sur le FPGA: la configuration de la fonction logique à apprendre grâce aux switchs SW10 jusqu'à SW17, le réglage de la durée de l'impulsion de programmation (V_{PULSE}) de 10 ms ou 100 ms grâce au switch SW6. Une fois tous les paramètres réglés, l'automate réalisant l'apprentissage est alors lancé grâce au switch SW1.

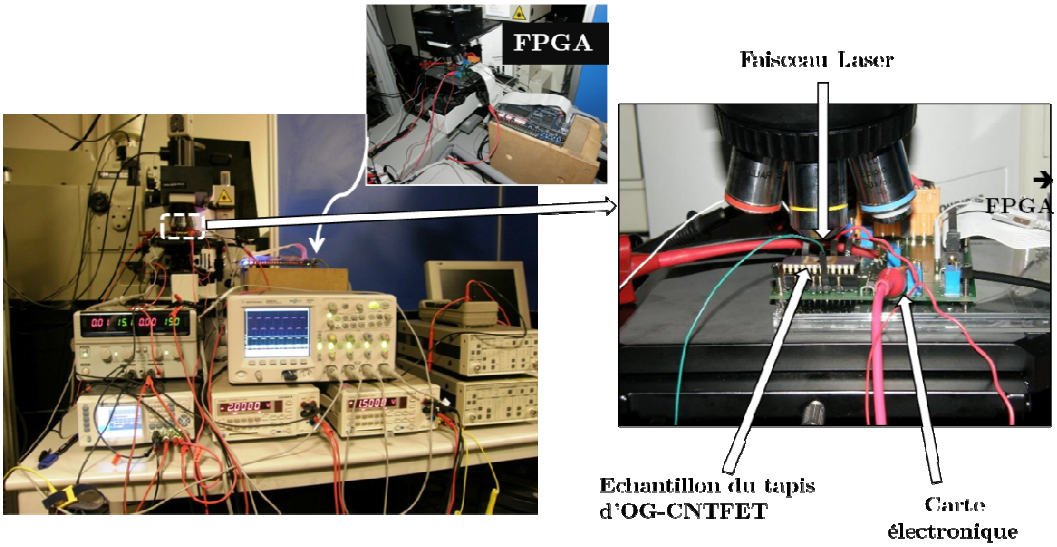


Figure IV.7 : Photo de la manipe de démonstrateur d'apprentissage mise au point par l'IEF et le CEA-LEM.

Nous avons commencé les expérimentations sur un échantillon de taille 6×1 pour démontrer l'apprentissage de fonctions logiques à une entrée et à deux entrées. Nous présentons les résultats de déroulement de l'apprentissage pour la fonction $F=X_1$ And X_2 sur la figure IV.8. Grâce au processus d'apprentissage, le système passe après six époques d'apprentissage d'un état initial dont la fonction obtenue $X=X_2$ And $/X_1$ vers la fonction souhaitée F. La représentation de la tension du nœud post-synaptique par rapport au seuil montre une bonne séparation entre les niveaux. Les niveaux inférieurs au seuil sont transformés grâce au comparateur en état logique « 0 », vu comme un état « -1 » par le FPGA (pour avoir une cohérence avec les résultats obtenus en simulation), et les niveaux supérieurs au seuil sont transformés en état logique « 1 ».

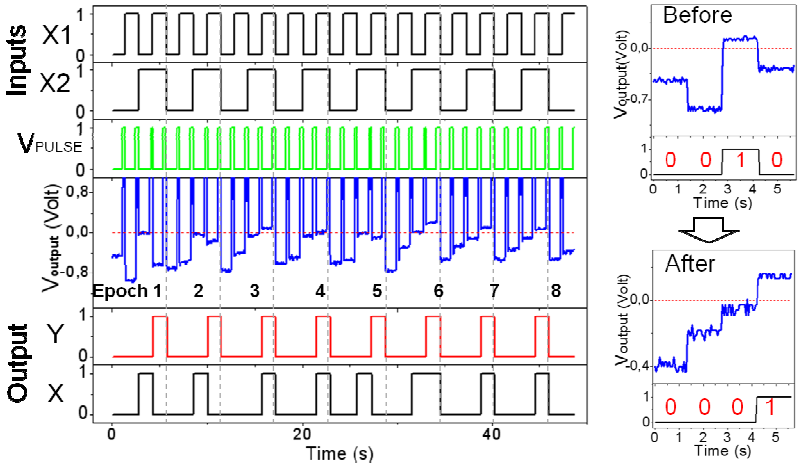


Figure IV.8 : Démonstration de l'apprentissage de la fonction AND, après six époques d'apprentissage la fonction obtenue X est identique à la fonction désirée Y. Les deux petites figures à droite montrent l'état de la tension post-synaptique (V_{output}) par rapport au seuil (placé à zéro) avant et après l'apprentissage.

Avec le même échantillon 6×1 nous avons démontré l'apprentissage de plusieurs autres fonctions à une et à deux entrées (figure IV.9). Avant l'apprentissage de chaque fonction nous envoyons le faisceau « Laser » pour initialiser les conductances et la fonction obtenue à l'état initial n'est pas toujours la même. Nous voyons bien qu'il est possible d'apprendre des fonctions logiques et leurs fonctions complémentaires.

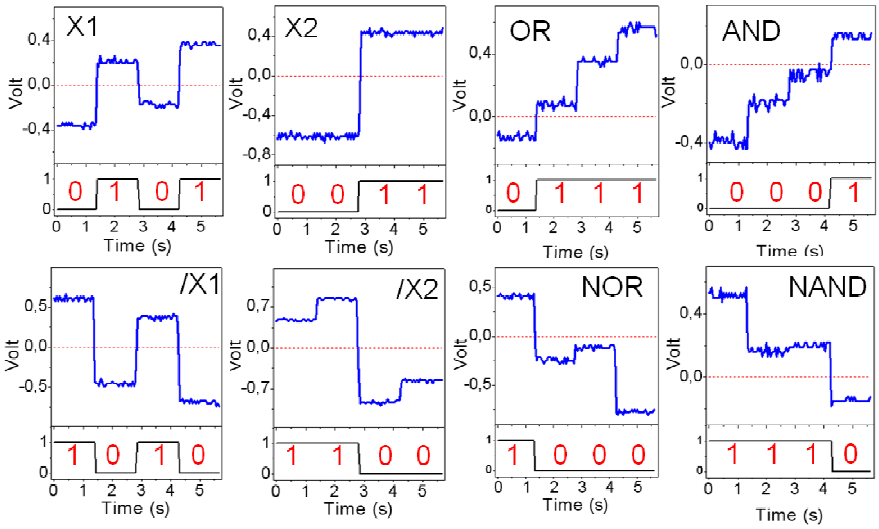


Figure IV.9 : Démonstration de l'apprentissage fonctions logiques à une et deux entrées.

Un autre échantillon de taille 8×1 est utilisé pour démontrer l'apprentissage de fonctions logiques à trois entrées (huit patterns). Nous donnons un exemple des résultats obtenus pour l'apprentissage de quatre fonctions différentes sur la figure IV.9. La représentation des huit patterns mesurés sur le nœud post-synaptique montre que l'apprentissage a convergé pour les quatre fonctions.

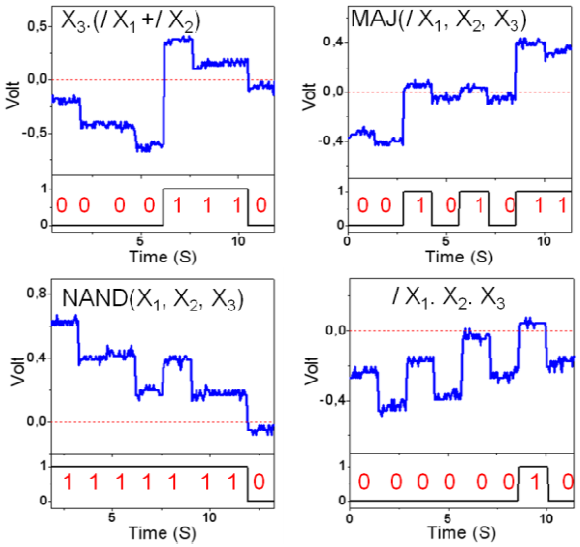


Figure IV.9 : Démonstration de l'apprentissage de fonctions logiques à trois entrées.

III. Conclusion

Ce chapitre présente le démonstrateur d'apprentissage de neuro-crossbar d'OG-CNTFET réalisé dans le cadre du projet PANINI. Le but de ce démonstrateur est de démontrer expérimentalement la faisabilité de l'architecture neuro-crossbar en validant les procédés d'apprentissage sur de vrais nanocomposants. Nous avons évoqué au cours de ce chapitre tous les éléments permettant de comprendre le fonctionnement de ce démonstrateur. Nous avons commencé par une brève explication du fonctionnement de l'OG-CNTFET avec ces différentes caractéristiques électriques. Nous avons expliqué ensuite la méthode d'apprentissage proposée pour ce type de circuit composé de nanocomposants à trois électrodes. Un circuit basé sur un FPGA est utilisé pour implémenter cette méthode d'apprentissage et une carte électronique est réalisée pour communiquer entre le FPGA et le tapis d'OG-CNTFET. La dernière partie de ce chapitre a été consacrée aux résultats des expériences menées par l'équipe du CEA-LEM et notre équipe « NanoArchi ». Nous avons expliqué le montage et les étapes de la démarche expérimentale qui a permis d'appliquer la procédure d'apprentissage implémentée sur un FPGA pour deux échantillons du tapis d'OG-CNTFET 6×1 et 8×1. Les résultats d'apprentissage ont montré la possibilité d'apprendre plusieurs fonctions logiques à une, deux et trois entrées. A notre connaissance, une telle démonstration n'a jamais été réalisée auparavant sur des dispositifs à base de nanotubes de carbone ni aucun autre nanocomposant memristif.

Conclusions et perspectives

Contexte général

Depuis l'annonce de la limite de la loi de Moore, beaucoup de travaux de recherche se sont lancés pour développer de nouvelles architectures et de nouveaux nanocomposants qui pourront remplacer le CMOS et poursuivre l'évolution de l'électronique. Et si le temps était venu pour les architectures neuromorphiques de prendre le relai ?

Depuis l'invention des réseaux de neurones artificiels (RNA) leur développement n'a donné lieu qu'à très peu de réalisation matérielle. Pour qu'ils puissent constituer une alternative crédible, des intégrations réalistes et des réalisations sur puce doivent convaincre des grandes performances des architectures neuromorphiques. Du fait de la complexité de l'implémentation d'une synapse ou d'un neurone en technologie CMOS qui nécessitent plusieurs composants, les tentatives d'implémentations des RNA n'ont pas suscité beaucoup d'intérêts, tout au moins de la part des industriels. Cependant, l'apparition des nouveaux nanocomposants (memristifs), possédant les caractéristiques intrinsèques d'une synapse ou d'un neurone et permettant de viser une très grande densité d'intégration (10^{12} composant/cm^2), devrait simplifier les modèles de synapses et de neurones à implémenter ce qui facilite la réalisation matérielle des RNA. Maintenant que les nanocomposant promettent des grandes densités d'intégration l'intérêt d'utilisation des RNA a des chances d'être démontré matériellement. Un grand champ de recherche et de développement des architectures neuromorphiques est ainsi ouvert pour démontrer leur performance et leur grande robustesse par rapport aux technologies classiques.

Travaux réalisés

Ce manuscrit propose un modèle de nanoarchitecture neuromorphique originale, plus particulièrement dédiée aux applications logiques et arithmétiques. L'architecture est essentiellement basée sur un crossbar composé de nanofils conducteurs et des nanocomposants reconfigurables à multi-niveaux (memristor, OG-CNTFET, CBRAM...). Ces nanocomposants vont jouer le rôle de synapses au sein d'un réseau de neurones artificiels. Nées de la recherche d'une architecture plus performante et plus robuste que les architectures classiques utilisant le CMOS ou la logique programmable, la démonstration présentée dans ce manuscrit montre des méthodes d'apprentissage neuronal permettant de configurer des circuits à très grande densité de nanocomposant avec une flexibilité et une très grande tolérance aux défauts.

Comme point de départ de notre étude, un bilan pragmatique des nanoarchitectures et leur difficulté d'intégration a permis d'identifier le taux de

défaut de fabrication et les dispersions des caractéristiques des nanocomposants comme leurs principaux obstacles. Les méthodes de configuration et les techniques de tolérances aux défauts proposées pour ces architectures révèlent plusieurs faiblesses : des temps de test très long et des méthodes de contournement de défauts couteuses en surface. Ce constat a motivé l'exploration de la voie de l'étude de fiabilité de notre architecture.

Les méthodes d'apprentissage, comme l'apprentissage *compétitif*, que nous avons développé sont ainsi le fruit d'une analyse approfondie visant l'amélioration de la robustesse. L'étude des propriétés des RNA nous a permis d'identifier quelques limitations comme l'apprentissage limité à des fonctions linéairement séparables pour les neuro-crossbars à une seule couche. Une architecture est proposée par la suite permettant d'apprendre des fonctions plus complexes et non-linéairement séparables. Cette architecture est composée de plusieurs neuro-crossbars mis en cascade afin d'avoir un réseau de type multicouches (Multi Layer Perceptron). Cette idée a conduit à proposer l'architecture de circuit reconfigurable FTNA (Field Trainable Neural Array) composé de plusieurs blocs neuro-crossbar (NLB) qui seront reliés comme dans un circuit FPGA (Field Programmable Gate Arraye). Chaque neuro-crossbar est considéré comme une cellule CLB (configurable Logique Block) dont la fonction, programmée dans le cas de CLB, va être apprise dans le cas d'un NLB grâce à la méthode d'apprentissage basée sur la « *règle de Delta binaire* ». Cette proposition va permettre non seulement de réaliser des circuits reconfigurables à très hautes densités, mais surtout, grâce à l'apprentissage neuronal des fonctions peuvent être programmées à partir des blocs contenant des nanocomposants défaillants ou présentant de fortes variations de caractéristiques.

La robustesse des architectures du futur constitue un enjeu majeur pour faire face aux taux élevés de défauts et aux fortes dispersions technologiques des caractéristiques des nanocomposants produits à grande densité à un coût raisonnable. Grâce aux simulations Monte-Carlo, nous avons pu estimer le degré de tolérance de notre modèle de neuro-crossbar en présence de défauts et de fortes variations des caractéristiques des composants. Des modèles de prédiction probabilistes de la convergence d'un neuro-crossbar ont été proposés pour chaque défaut et chaque type de dispersion envisagée. L'étude menée dans cette partie a permis de montrer le degré de robustesse de l'approche neuronale. Le taux de tolérance mesuré en simulation est important et l'ajout de neurones redondants révèle une amélioration considérable de cette tolérance. La taille, mesurée en nombre d'entrées/sorties, des neuro-crossbars étudié dans cette partie est inférieure à la taille réelle que peuvent avoir les nano-circuits, du fait des temps de simulation qui peuvent être très long pour des neuro-crossbars de grande taille. Cependant, grâce aux modèles probabilistes que nous avons développés et confrontés à nos simulations Monte Carlo, il devient possible de généraliser cette étude à des circuits à très grande taille.

La dernière partie de ce manuscrit a été consacrée au démonstrateur d'apprentissage de fonctions logiques avec un vrai neuro-crossbar basés sur des transistors à nanotube de carbone à commande optique (OG-CNTFET). Ce démonstrateur a été réalisé dans le cadre du projet PANINI dans le but de valider les méthodes d'apprentissage que nous avons proposées pour ce type d'architecture. Nous avons démontré la possibilité d'apprendre des fonctions logiques à une, deux et trois entrées à partir d'un tapis d'OG-CNTFET à plusieurs entrées et une seule sortie. A notre connaissance, la démonstration de la possibilité de configurer par apprentissage des nanocomposants pour apprendre des fonctions logiques constitue une première mondiale.

Perspectives

D'un point de vue théorique, l'étude de la fiabilité des neuro-crossbar et les modèles probabilistes développés dans ce manuscrit constituent une base pour poursuivre l'étude sur des réseaux plus larges afin d'étendre la portée et la généralité de nos observations. Le développement d'autres techniques d'apprentissage, dans le même esprit que l'apprentissage compétitif, pourrait encore améliorer la robustesse des neuro-crossbars. La période d'apprentissage est l'étape pendant laquelle le circuit consomme plus d'énergie, il est donc intéressant de prendre en compte cet aspect pendant le développement des nouvelles techniques d'apprentissage. Il reste tout un travail d'optimisation des paramètres d'apprentissage (impulsions de programmation, tensions d'entrée/sortie...) pour faciliter la convergence de l'apprentissage tout en minimisant la consommation énergétique.

Je me suis limité dans cette thèse à l'étude d'une méthode d'apprentissage supervisée, mais d'autres approches sont possibles. En particulier, la combinaison de techniques d'apprentissage supervisé et non-supervisé (STDP par exemple) pourrait s'avérer fructueuse pour identifier les fonctions des couches cachées utiles sans recourir à des méthodes logicielles, dans des neuro-crossbars multicouches. Elle permettrait également d'envisager d'autres applications que la logique, notamment dans le domaine de la reconnaissance de formes ou le traitement d'image et de son.

D'un point de vue pratique, l'étude menée dans ce manuscrit a montré l'intérêt d'utiliser des réseaux de neurones pour tolérer les défauts et les dispersions technologiques. En regard des résultats obtenus, il semblerait intéressant de continuer à développer une architecture complète (FTNA) contenant les blocs neuro-crossbar associés aux circuits périphériques qui permettent d'implémenter la méthode d'apprentissage. Pour cela, il reste beaucoup de travail. Ainsi, une étude pointue est nécessaire pour développer les switchs de routage programmables (PRS), en remplaçant la structure classique basée sur les SRAM par des switchs basée sur des memristors. En particulier, il faudra adapter aux architectures FTNA les outils de synthèse initialement dédiés aux FPGA classiques. Enfin, comme, l'un des principaux problèmes pour les architectures d'aujourd'hui est la consommation, il

paraît stratégique de l'évaluer dans le contexte de l'architecture FTNA pour montrer à coté de sa robustesse son efficacité énergétique.

Un autre point intéressant à développer dans l'architecture neuro-crossbar est la partie neurone dont la conception est jusqu'ici réalisée en technologie CMOS. Comme nous l'avons déjà évoqué dans le chapitre II (section III.1.3.2.1), il serait intéressant d'exploiter les propriétés de quelques nanocomposants (comme le Latch moléculaire (& II section II.1.3.2.2)) pour réaliser des neurones qui consomment moins et qui occupent moins de surface.

Pour finir, maintenant que nous avons démontré expérimentalement la possibilité de programmer des fonctions logiques à base du tapis d'OG-CNTFET, il est temps de réaliser des crossbars à base d'autres technologies memristives composées de dipôles potentiellement plus denses, comme les memristors, les CBRAM ou les MRAM, et de mettre en pratique les différentes solutions proposées pour régler les différents problèmes que rencontrent les architectures crossbar. Cela définit un axe de recherche passionnant qui va permettre de progresser rapidement vers des architectures en rupture dépassant les limitations du CMOS.

Référence :

[Adeh96]: A. DeHon, "Reconfigurable architectures for general-purpose computing", *MIT Artif. Intell. Lab.*, Cambridge, MA, 02139, pages 35-41, 1996.

[Adeh02]: A. Dehon, "Very large scale spatial computing," in *Proceedingsof the 3rd International Conference on Unconventional Models of Computation (UMC '02)*, pp. 27–37, Kobe, Japan, 2002.

[Adeh04]: M.Wilson A.DeHon. "Nanowire-based sublithographic programmable logic arrays. FPGA'04", *Nanotechnology, IEEE Transactions on,* vol. 4 issue:6 pp. 681–687, 2004.

[Adeh05]: André Dehon. "Nanowire-based programmable architectures". *ACM on Emerging Technologies in Computing Systems*, 1(2) :109–162, 2005.

[Agnus10] : G. Agnus, W.S. Zhao, V. Derycke, A. Filoramo, Y. Lhuillier, S. Lenfant, D. Vuillaume, C. Gamrat, and J.P. Bourgoin, "Two-Terminal Carbon Nanotube Programmable Devices for Adaptive Architectures", *Advanced Materials*, vol. 22, no 6, pp.702-706, 2010.

[Alib10] : Alibart, F., S. Pleutin, D. Guerin, C. Novembre, S. Lenfant, K. Lmimouni, C. Gamrat, and D. Vuillaume, "An Organic Nanoparticle Transistor Behaving as a Biological Spiking Synapse". *Advanced. Functional. Materials*, vol. 20, pp. 330, 2010.

[Arg68] : F.Argall, "Switching Phenomena in Titanium Oxide Thin Films" *Solid-State Electronics*, vol. 11, pp. 535-541, 1968.

[Ass97] : A. Assoum, "Etude de la tolérance aux aléas logiques des réseaux de neurones artificiels", *Thèse de doctorat* de l'Institut National Polytechnique de Grenoble, 1997.

[Bed88] : Bedworth M.D. et Lowe D., "Fault Tolerance in Multi-Layer Perceptrons : a Preliminary Study" *RSRE : Pattern Recognition and Machine Intelligence Division*, 1988.

[Bel92] : Eric BELHAIRE, "Contribution à la réalisation électronique de Réseaux de Neurones Formels : Intégration Analogique d'une MACHINE DE BOLTZMANN". *Thèse de Doctorat*, Université Paris sud, Institut d'2lectronique Fondamentale (IEF), soutenue en 1992.

[Beiu03] : V. Beiu, « VLSI implementation of threshold gates - A comprehensive Survey ». *IEEE Transactions on Neural Networks*, vol 14, pp. 1217-1243, 2003.

[Berr06] : H. Berry and M. Quoy. "Structure and dynamics of random recurrent neural networks". *Adaptive Behavior, Vol. 14*, pp 129-137, 2006.

[Bilo66] A. Bilotti, "Operation of a MOS transistor as a variable resistor". *Proceedings of the IEEE*, vol. 54, pp. 1093-1094. 1966.

[Bhad05] : D. Bhaduri and S. Shukla, "NANOLAB- A Tool for Evaluating Reliability of Defect-Tolerant Nano-Architectures". *IEEE Transactions on Nanotechnology*, vol. 4, pp. 381–394, 2005.

[BiPo98] : G. Q. Bi and M. M. Poo, "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type". *Journal of Neuroscience*, Vol. 18, pp. 10464-10472, 1998.

[Bo98] : G. M. Bo, D. D. Caviglia, and M. Valle, "An Analog VLSI Implementation of a Feature Extractor for Real Time Optical Character recognition". *Journal of Solid State Circuits*, vol. 33, no. 4, pp. 556-564, 1998.

[Borg06] : J. Borghetti, V. Derycke, S. Lenfant, P. Chenevier, A. Filoramo, M. Goffman, D. Vuillaume and J. P. Bourgoin, "Optoelectronic switch and memory devices based on polymer functionalized carbon nanotube transistors". *Adv. Mater.*, vol. 18, pp. 2535, 2006.

[Borg09] : Borghetti, J., Z. Li, J. Straznicky, X. Li, D. A. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming". *Proc. Nat. Ac. Sci.*, vol. 106, pp 1699, 2009.

[Borg10] : Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. StewartR. Stanley Williams et al, "Memristive' switches enable 'stateful' logic operations via material implication". *Nature*, vol. 464, pp. 873-876, 2010.

[Bose91] : B. E. Boser, E. Säckinger, Jane Bromley, Y. Le Cun, and L. D. Jakel, "An Analog Neural Network Processor with Programmable Topology". *IEEE Journal of Solid State Circuits*, vol. 26, no. 12, pp. 2017-2025, 1991.

[Boum05]: H. Boumeridja, M. Atencia, G. Joya, and F. Sandoval. "FPGA implementation of hopfield networks for systems identification". In Proc. IWANN, vol 3512 of LNCS, 2005.

[Brmin] : Brain Mind Institute, Ecole Polytechnique Fédérale de Lausanne, http://bmi.epfl.ch/

[Brow07] : Jason G. Brown and R. D Blanton, "A Built-in Self-test and Diagnosis Strategy for Chemically Assembled Elctronic Nanotechnology". *J.Electron. Test*, vol. 23, pp. 144, 2007.

[Brug00] : Brugger, J., J.W. Berenschot, S. Kuiper, W. Nijdam, B Otter, and M Elwenspoek, "Resistless patterning of sub-micron structures by evaporation through nanostencils". Microelectronic Engineering, vol. 53, pp. 402-405, 2000.

[Bud02] : Buddefeld, J. Grosspietsch, K.E. "Intelligent-memory architecture for artificial neural networks", *IEEE Micro*, vol.22, pp.32, 2002.

[Burr08] :Burr, G. W., Kurdi, B. N., Scott, J. C., Lam, C. H., Gopalakrishnan, K., Shenoy, R.S. "Overview of Candidate Device Technologies for Storage-Class Memory". *IBM J. Res. Dev.*, vol. 52, pp. 449-464, 2008.

[Cab09] : Cabe, A. C., and S. Das, "Performance simulation and analysis of a CMOS/nano hybrid nanoprocessor system". *Nanotechn*, vol. 20, pp. 165203, 2009.

[Cavi94] : D. D. Caviglia, M. Valle, A. Rossi, M. Vincentelli, G. M. Bo, P. Colangelo, P. Pedrazzi, and A. Colla, "Feature Extraction Circuit for Optical Character Recognition". *Electronics Letters,* vol. 30, no. 10, pp. 769-760, 1994.

[Chab10] : D. Chabi, J.O. Klein, "Hight fault tolerance in neural crossbar", Design and Technology of Integrated Systems in Nanoscale Era (*DTIS'10*), 5th International Conference on, pp 1 – 6, 2010.

[Chao90] : Chao, C. "Incorporation of Learning within the CMOS Neuron". University of Southern California M.S. Thesis, July, 1990.

[Chau10] : Caudhuri S., Zhao W., Klein J.-O., Chappert C., Mazoyer P. "Design of embedded MRAM Macros for Memory-in-Logic Applications". In Proc. Of IEEE/ACM Great Lakes Symposium on VLSI (*GLSVLSI 2010*), vol. 1, p. 155, 2010.

[Chen96] : R. H. Chen, A. N. Korotov, and K. K. Likharev. "Single electron transistor logic". *Appl. Phys. Lett.,* vol. 68, issue 14, pp 1954, 1996.

[Chen03] : Y. Chen et al., "Nanoscale molecular-switch crossbar circuits". *Nanotechnology*, vol 14, pp. 462–468, 2003.

[Chen05] : Chen He et al. "A Reconfiguration-Based Defect-Tolerant DesignParadigm for Nanotechnologies". *IEEE Design & Test of Computers*, vol. 22, pp. 316-326, 2005.

[Chey97] : Ph. Cheynet, J.-C. Rubio, R. Velazco, and J.-D. Muller, "Evaluating neural network robustness with an architecture built around L-Neuro 2.3". 6th Int. Conference on Microelectronics for Neural Networks (*MICRONEURO'97*), Poster Session 1, pp. 244-250, 1997.

[Choi09] : Choi, H., H. Jung, J. Lee, J. Yoon, J. Park, D.-J. Seong, W. Lee, M. Hasan, G.-Y. Jung, and H. Hwang, "An electrically modifiable synapse array of resistive switching memory". *Nanotechnology*, vol. 20, pp. 345201, 2009.

[Choi93] : J. Choi, S. Oh, R. Marks, "Training Layered Perceptron using Low Accuracy Computation". Proc. International Joint Conference Neural Network, pp. 783-788, 1993.

[Chri03] : Christopher P. Husband, SUmmer M. Husband, Jonathan S. Daniels, and James M. Tour. "Logic and memory with nanocell circuits". *IEEE transactions on electron devices*, vol. 50, no. 9, pp. 1865-1875, 2003.

[Chua71] : L. O. Chua, "Memristor—the missing circuit element". *IEEE Trans. Circuit Theory*, vol. 18, pp. 507–519, 1971.

[Chua76] : L. O. Chua and S. M. Kang, " Memristive devices and systems". *Proc.IEEE*, vol. 64, pp. 209-223, 1976.

[Cong11]: Cong, J.; Bingjun Xiao,"mrFPGA: A novel FPGA architecture with memristor-based reconfiguration". Nanoscale Architectures (*NANOARCH*), 2011 IEEE/ACM International Symposium on pp. 1 – 8, 2011.

[Cot05] : S. D. Cotofana, A. Schmid, Y. Leblebici, A. Ionescu, O. Soffke, P. Zipf, M. Glesner, A. Rubio, "CONAN - A Design Exploration Framework for Reliable Nano-Electronics Architectures". *Proceedings of 16th International Conference on Application-Specific Systems*, Architectures and Processors, pp. 260-267, 2005.

[Cui01] : Y. Cui and C. Lieber. "Functional nanoscale electronic devices assembled using silicon nanowire building blocks". *Science,* vol. 291, pp. 851–853, 2001.

[Cui03] : Yi Cui, Zhaohui Zhong, Deli Wang, Wayne U. Wang, and Charles M. Lieber "High Performance Silicon Nanowire Field Effect Transistors". *Nano let*, vol. 3, no. 2, pp. 149-152, 2003.

[Culb97] : W. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider, "Defect tolerance on the Teramac custom computer". IEEE Symposium on FPGA's for Custom Computing Machines (*FCCM '97*). pp. 116–123, 1997.

[Cun89] : Y. Le Cun, "Modèles Connexionnistes de l'apprentissage", *Thèse de doctorat* de l'université Pierre et Marie Curie, Orsay, 1987.

[Dan02] : G. Danese, F. Leoporati, and S. Ramat. "A parallel neural processor for real-time applications". J *IEEE Micro - MICRO,* vol. 22, pp. 20-31, 2002.

[Dar89] : T.R. Darmala, P.K. Bhagat, "Fault Tolerance of neural Networks". *Proc. Of Energy and Information Technology in the Southeast Conference*, Colombia, SC, 1989.

[Dash]: NC3001 TOTEM Digital Processor for Neural Networks, data sheet in: http://icwic.cn/icwic/data/pdf/cd/cd010/80570.pdf.

[Diet07] : Dietrich, S., M. Angerbauer, M. Ivanov, et al, "A Nonvolatile 2-Mbit CBRAM Memory Core Featuring Advanced Read and Program Control". *IEEE J. Sol.-State Circ.*, vol. 42, pp. 839, 2007.

[Dimi08]S. Dimitrios, I. Papaefstathiou, M. G.H. Katevenis, "Building an FoC Using Large, Buffered Crossbar Cores". *IEEE Designe and Test of computer*, vol 25, pp 538-548, 2008.

[Dior96]: DIORIO Chris, HASLER Paul, MINCH Bradley A, MEAD Carver A., "A single-transistor silicon synapse". *IEEE transactions on electron devices*, vol 43, n°11. 1996.

[Dior02] : C. Diorio, D. Hsu, and M. Figueroa, "Adaptive CMOS: From biological inspiration to systems on-a-chip". *Proceedings of the IEEE*, vol. 90, pp. 245–357, 2002.

[Dongx]: Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., Chen, Y. "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement". *Proceedings of the 45th annual Design Automation Conference*, 2008.

[Dongy] : Dong, Y., G. Yu, M. C. McAlpine, W. Lu, and C. M. Lieber, "Si/a-Si Core/Shell Nanowires as Nonvolatile Crossbar Switches". *Nano Lett.* vol. 8, pp 386, 2008.

[Dris09] : Driscoll, T., H.-T. Kim, B. G. Chae, M. Di Ventra, and D. N. Basov, "Phase-transition driven memristive system". *Appl. Phys. Lett.*, vol. 95, pp. 043503. 2009.

[Dur01] : Lisa J K Durbeck et Nicolas J Macias, "The Cell Matrix: an architecture for nanocomputing". *Nanotechnology*, vol. 12, pp. 217-230, 2001.

[Duyg11] : Duygu Kuzum, Rakesh G. D. Jeyasingh, Byoungil Lee, and H.-S. Philip Wong "Nanoelectronic Programmable Synapses Based on Phase Change Materials for Brain-Inspired Computing". *Nano lett.* Article ASAP, 2011.

[Emm91] : M.D Emmerson, R.I Damper, A.J.G Hey, C Upstill, "Fault tolerance and Redundancy of Neural Nets for the Classification of Accoustic Data". *Proc. Of the IEEE ICASSP'91*, pp. 1053-1056. 1991.

[Emm92] : Emmerson M.D. et Damper R.I., "Relations betweenFault Tolerance and Internal Representations for Multi-Layer Perceptrons". *Proc. of the IEEE ICASSP'92*, pp. 281-284, 1992.

[Eike10] : Eike Linn, Roland Rosezin, Carsten Kügeler & Rainer Waser « Complementary resistive switches for passive nanocrossbar memories ». Nature Materials, vol. 9, pp. 403–406, 2010.

[Fact]: http://facets.kip.uni-heidelberg.de/

[Fost02] : Sarah Elisabeth Fost, Arun F. Rodridgues, Andrew W. Janiszewski, Randal T. Rausch, Peter M. Kogge, " Memory in motion: A Study of Storage Structures in QCA", *1st workshop on Non-Silicon computation*, held in conjunction with 8th Int. Symp. on High Performance Computer Architecture, Boston, MS. Février. 3, 2002.

[Froe02] : Froemke, R. C., and Y. Dan, "Spike-timing-dependent synaptic modification induced by natural spike trains". *Nature,* vol. 416, pp. 433-8, 2002.

[Gar03] : Gary H. Bernstein, "Quantum-dot Cellular Automata: Computing by Field polarization". *Design Automation Conference*, vol. 1, pp. 268-273 2003.

[Gil07] : N.E. Gilbert and M.N. Kozicki. "An embeddable multilevel-cell solid electrolyte memory array". IEEE Journal *Solid-St Circ*, vol. 42, pp. 1383-1391, 2007.

[Girau06] : B. Girau, "FPNA : Applications and implementations. In A. Omondi and J. Rajapakse, editors, FPGA Implementations of Neural Networks". *Springer*, 2006.

[Gold01] : S. C. Goldstein, M. Budiu, "Nanofabrics : Spatial computing using molecular electronics". *In : 28th International Symposium on Computer Architecture*, pp. 178.191.2001.

[Gold03] : Mahim Mishra and Seth Copen Goldstein, "Defect tolerance at the end of the roadmap". In *International Test Conference (ITC)*, 2003.

[Gor10] : A. Gorchetchnikov et al., "General Form of Learning Algorithms for Neuromorphic Hardware Implementation". *BMC Neuroscience*, vol. 11 (supp. 1), 2010,

[Gree01] : J.E. Green, J.W. Choi, A. Boukai, Y. Bunimovich et al, "A 160-kilobit molecular electronic memory patterned at 1011 bits per square centimeter". Nature, vol. 445, pp. 414-417, 2007.

[Gros01] : S. Grossberg and J. Williamson, "A Neural Model of How Interlaminar Connections of Visual Cortex Develop into Adult Circuits that Carry Out Perceptual Grouping and Learning". *Cerebral Cortex*, vol. 11, no. 1, pp. 37-58, 2001.

[Gros07] : G. Grossi and F. Pedersini. "FPGA implementation of an adaptive stochastic neural Model". In *Artificial neural networks*-ICANN, vol 4668 of LNCS. Springer, 2007.

[Gsch96]: M. Gschwind, V. Salapura, and O. Maisch berger. "A generic building block for Hopfield neural networks with on-chip learning". In Proc. *ISCAS,* Vol supplement (May 12-15), pp 49-52, 1996.

[Hack09] : Gergel-Hackett, N., B. Hamadani, B. Dunlap, J. Suehle, C. Richter, C. Hacker, and D. Gundlach, "A Flexible Solution-Processed Memristor". IEEE El. Dev. Lett. 30, 706, 2009.

[Ham08]: D. Hammerstrom. "Biologically Inspired Architectures". Rainer Waser, Wiley-Series: Nanotechnology, 2008.

[Ham90] Dan Hammerstrom. "A vlsi architecture for highperformance, low-cost, on-chip learning". In *Proceedings of Internationnal Join Conference on Neural Network*, pp 537 – 544, 1990.

[Han03] : Han J and Jonker P, "A defect- and fault tolerant architecture for nanocomputers". *Nanotechnology,* vol. 14, pp. 224–30, 2003.

[Han04] : Han J., "Thesis "Fault-tolerant architectures for nanoelectronic and quantum devices", *thèse de doctorat*, Delft University of Technology, Netherlands, 2004.

[Han05] : Han J., Jonker P., "Toward hardware redundant, fault tolerant logic for Nanoeletronics". *IEEE Design Test of computer*, vol.22, pp. 328-339, 2005.

[Har09] **:** N.Z. Haron, S. Hamdioui, S. D. Cotofana, "Emerging Non-CMOS Nanoelectronic Devices - What Are They? ". *IEEE-NEMS'09* Shenzhen, China. pp.63-68, 2009.

[Hasl95] : P. Hasler, C. Diorio, B. A. Minch, and C.A. Mead, "Single transistor learning synapses". *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge, MA, pp. 817-824, 1995.

[Hasl06] : P. Hasler, E. Farquhar and C. Gordon, "Building Large Networks of Biological Neurons". 28th Annual International Conference of the *IEEE Engineering in Medicine and Biology Society*, EMBS '06, vol. Supplement, pp. 6548-6551, 2006.

[Haw04] : Hawkins, J., and Blakeslee, S., *On Intelligence*, Times Books, New York, 2004.

[Heat98] : Heath J.R., et al., "A defect-tolerant architecture for nanotechnology". *Science*, vol.280, pp. 1716–1731, 1998.

[Heb50] : D. Hebb, "The Organization of Behavior". Journal of Clinical Psychology. *Journal of Clinical Psychology*, vol 6, pp 307, 1950.

[Hick62] : T. W. Hickmott "Low-Frequency Negative Resistance in Thin Anodic Oxide Films ". *J. Appl. Phys*, vol. 33, pp. 2669, 1962.

[Holl89] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network with 10240 'floating gate' synapses". *Proceeding of the International Joint Conference on Neural Networks*, Washington, D.C., vol. 2, pp. 191-196, 1989.

[Hogg06] : T. Hogg and G. Snider, "Defect-tolerant adder circuits with nanoscale crossbars". *IEEE Transactions on Nanotechnology*, vol. 5, pp. 97–100, 2006.

[Huan04] : Jing Huang, Mehdi B. Tahoori, and Fabrizio Lombardi, "On the Defect Tolerance of Nano-Scale Txo-Dimensional Crossbars". *Defect and Fault-Tolerance in VLSI Systems*, IEEE International Symposium d on, pp 96-104, 2004.

[Huan06] : Jing Huang, M. Momenzadeh, F. Lombardi, "Defect Tolerance of QCA Tiles". *Proceedings of the Design Automation & Test in Europe Conference*, vol. 1, pp.168, 2006.

[Pujo94] : H. Pujol, J.O. Klein, E. Belhaire, P. Garda, "Ra : An analog Neurocomputer for the synchronous Boltzmann machine". MicroNeuro94, 26-28, Torino, Italie. pp. 449--455, 1994.

[Hyn07] : K.M Hynna and K. Boahen "Silicon neurons that burst when primed". IEEE International Symposium on Circuits and Systems, *ISCAS*, pp. 3363-3366, 2007.

[Igel05] : C. Igel, M. Toussaint, and W. Weishui, Trends And Applications in Constructive Approximation. Detlef H. Mache and Jozsef Szabados and Marcel G. de Bruin, 2005, ch. Rprop using the natural gradient compared to Levenberg - Marquardt.

[Int1] : "Intel® Core™ i7 Processor Extreme Edition – Overview." [Online]. Available: http://www.intel.com/products/processor/corei7EE/index.htm.

[Int2] : "Intel Museum – The Intel 4004." [Online]. Available: http://www.intel.com/about/companyinfo/museum/exhibits/4004/index.htm

[ITRS07] : International Technology Roadmap for Semiconductor, http://www.itrs.net

[Jeon09] : Jeong, D. S., H. Schroeder, and R. Waser, «Mechanism for bipolar switching in a $Pt/TiO_2/Pt$ resistive switching cell», *Phys Rev B*, vol. 79, pp 195317, 2009.

[Jo08]: Jo, S. H., and Wei Lu, "CMOS Compatible Nanoscale Nonvolatile Resistance Switching Memory». *Nano Lett*, vol, 8, pp. 392, 2008.

[Jo09]: Jo S.H et al, «High-Density Crossbar Arrays Based on a Si Memristive System », *Nano Letter,* Vol 9, pp. 870-874, 2009.

[Jo10]: Jo, S. H., T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, « Nanoscale Memristor Device as Synapse in Neuromorphic Systems ». *Nano Lett*, vol. 10, pp. 1297, 2010.

[JoB07] : J.O. Klein et E. Belhaire « Synapse nano-électrique et procédé d'apprentissage d'une telle synapse ». Déposé à l'INPI sous le n° 07 05532, le 27 juillet 2007.

[Jok09] : Jacques-Olivier Klein, « Impact des technologies sur les architectures de calcul ». Habilitation à Diriger des Recherches (*HDR*), spécialité physique, Université de paris sud, *IEF*, 2009.

[Jok09b] : Klein J.-O., Belhaire E. « TEACHING METHOD FOR A NEURONAL NANO- BLOCK ». *Brevet*, dépôt n° PCT/FR2009/050943 du 20/5/2009.

[Jung04] : Jung, G. Y. et al. « Fabrication of a 34 x 34 crossbar structure at 50 nm half-pitch by UV-based nanoimprint lithography». *Nano Lett*, vol. 4, pp. 1225–1229, 2004.

[Kar89] : Karpovsky M.G, Nagvajara P., « Design of Self-Diagnostic Boards by Signature Analysis ». IEEE Trans. On Industrial Electronic, vol. 36, pp. 241-254, 1989.

[Ken97] : J. Kennedy and J. Austin. «A parallel architecture for binary neural networks». In Proc. MicroNeuro, pp 225–231, 1997.

[Ker94]: P. Kerlirzin, "Etude de la robustesse des réseaux multicouches", Thèse de doctorat de L'université Paris sud, Orsay, 1994.

[Klir95] : Klir, G. J., and B. Yuan ; « Theory and Applications (Prentice Hall) », 1st Fuzzy Sets and Fuzzy Logic edition.1995.

[Klr07] : K.L. Rice, C.N. Vutsinas, and T.M. Taha. «A preliminary investigation of a neocortex

model implementation on the cray XD1». In Proc. SC - SuperComputing, 2007.

[Koho90] Kohonen (T.). "Unsupervised Learning Algorithms. Neural Networks : Biological Computers or Electronic Brains". *Springer Verlag*, pp. 29-36, 1990.

[Koz05] : Kozicki, M., M. Park, and M. Mitkova, "Nanoscale memory elements based on solid-state electrolytes". *IEEE Trans. Nanotechn.*, vol. 4, pp. 331, 2005.

[Kry09] : Kryder, M. H., Kim, C. S. « After Hard Drives - What Comes Next? ». *IEEE Transactions on Magnetics*, vol. 45, pp. 3406 - 3413, 2009.

[Kuek01] : P. J. Kuekes, R. S. Williams, and J. R. Heath, «Molecular-wire crossbar interconnect » (*MWCI*) for signal routing and communications, US Patent 6,314,019, 2001.

[Kuek05] : P. Kuekes, W. Robinett, G. Seroussi and R. S. Williams, «Defect-Tolerant Interconnect to Nanoelectronic Circuits: Internally Redundant Demultiplexers Based on Error-Correcting Codes » *Inst. Phys. Nanotechnology*, issue. 16, pp. 869–882, 2005.

[Kuek05b] : P. Kuekes, D. R. Stewart, and R. S. Williams, « The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits». *J. Appl. Phys.*, vol. 97, pp. 034301-034301-5, 2005.

[kund05] : Kund, M. Beitel, G. Pinnow, C.-U. Rohr, T. et al, « Conductive bridging RAM (CBRAM): an emerging non-volatile memory technology scalable to sub 20nm». *Electron Devices Meeting*, pp. 754 – 757, 2005.

[Lai08] : Lai, S. K., «Flash memories: successes and challenges»; *IBM J. Res. Dev.*, vol. 52, pp. 529-535, 2008.

[Lai10] : Lai, Q., L. Zhang, Z. Li, W. F. Stickle, R. S. Williams, and Y. Chen, « Ionic/Electronic Hybrid Materials Integrated in a Synaptic Transistor with Signal Processing and Learning Functions ». *Advanced Materials*, vol. 22, pp. 2448, 2010.

[Lee05] : Lee, Y., Lee, J., Kim, Y., and Ayers, J., A "Low Power CMOS AdaptiveElectronic Central Pattern Generator Design ". *48th Midwest Symposium on Circuits and Systems*, vol. 2, pp 1350- 1353, 2005.

[Lee07] : Lee, D., D. jun Seong, I. Jo, F. Xiang, R. Dong, S. Oh, and H. Hwang, " Resistance switching of copper doped MoO_x films for nonvolatile memory applications". *Appl. Phys. Lett.*, vol. 90, pp. 122104, 2007.

[Lee09] : Lee, B., Ipek, E., Mutlu, O., Burger, D. "Architecting phase change memory as a scalable DRAM alternative". In International Symposium on Computer Architecture (*ISCA*), 2009.

[Leht09] : Lehtonen, E., and M. Laiho, "Stateful implication logic with memristors". In Proceedings of the International Symposium on Nanoscale Architectures (*NANOARCH'09*), issue. 30, pp. 33, 2009.

[Leht10] : Lehtonen, E., J. H. Poikonen, and M. Laiho, "Two memristors suffice to compute all boolean functions", *Elect. Lett.,* vol. 46, pp 239, 2010.

[Lent93] : C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Berstein, "Quantum cellular automata". *Nanotechnology*, 4 :49–57, 1993.

[Lent97]: C. Lent. "A device architecture for computing with quantum dots". *Proceedings of the IEEE*, Vol. 85, No. 4 pp. 541-557, 1997.

[Levy83] : Levy W. B., Steward O. "Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus". *Neuroscienc,* vol 8, pp. 791–797, 1983.

[Li04] : Li, Chao; Fan, Wendy; Lei, Bo; Zhang, Daihua, "Multilevel memory based on molecular devices". *Applied Physics Letters*, vol. 84, pp. 1949 – 1951, 2004.

[Likha03] : Likharev K., MAYR, A., MUCKRA, I. and TÜREL, Ö. "CrossNets - High-performance neuromorphic architectures for CMOL circuits". *Ann Ny Acad Sci,* vol. 1006, pp. 146-163, 2003.

[Likha04] : Konstantin K. Likharev. "CMOL : A new concept for nanoelectronics". 12th Int. Symp. Nanostructures : *Physics and Technology*, June 2004.

[Likha05] : Likharev, K. K., and D. B. Strukov, "in Introducing Molecular Electronics", edited by G. F. G. Cuniberti and K. Richter (Springer), volume 657, pp. 447-477, 2005.

[Linco03] : P. Lincoln A. DeHon and J. Savage. "Stochastic assembly of sublithographic nanoscale interfaces". *IEEE Transactions on Nanotechnology*, 2(3) :165–174, 2003.

[Lind96] : C. S. Lindsey, Th. Lindblad, "Experience with the Reactive Tabu Search as Implemented in the Totem Chip", *proceedings of the ICNN'96* in Washington D.C, 1996.

[Masa94] : ITO Masao; "La plasticité des synapses". *La Recherche*, vol 25, pp. 778-785, 1994.

[Mats94] : Matsuoka K. et Kawamoto M., "A Neural Net for Blind Separation of Nonstationary Signal Sources". *Proc. of the IEEE ICNN'94*, pp. 221-227. 1994.

[Mbud01]: M. Budiu S. C. Goldstein. "Nanofabrics : Spatial computing using molecular electronics". Proceedings of the 28th Annual International Symposium on Computer Architecture, pp. 1-12, 2001.

[McC43]: W.S. McCulloch and W. Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity". *Bulletin of Math. BioPhy.*, vol, 5, pp 115-133, 1943.

[Mead89] : C. A. Mead, "Analog VLSI and Neural Systems", Addison-Wesley, Reading, MA, 1989.

[MiHe08] : Michel Haiyu He, "Contribution à l'étude de l'impact des nanotechnologies sur les Architectures : Apprentissage d'inspiration neuronale de fonctions logiques pour circuits programmables ". *Thèse de doctorat.* Université paris sud, Institut d'électronique fondamentale (IEF), 2008.

[Min69] : L.M. Minsky and A.S. Papert. *Perceptrons: An introduction to computation geometry.* 1988.

[Moor88] : Moore W.R., "Conventional Fault-Tolerance and Neural Computers". *Neural Computers*, pp. 29-37. Springer Verlag, 1988.

[Myo11] : Myoung-Jae Lee, Chang Bum Lee et al, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures". *Nature Material*, vol. 10, pp. 625-630, 2011.

[Naka08]: M. Nakayama et al., "Spin transfer switching in TbCoFe/CoFeB/ MgO/ CoFeB/TbCoFe magnetic tunnel junctions with perpendicular magnetic anisotropy", *Journal of Applied Physic*, vol. 103, 07A710, 2008.

[Neu37]: A V.Neumann ; "On computable numbers, with an application to the entscheidungs problem". *Proc. London mathematical Soc.*, 43 :230–265, 1937.

[Neu56] : Neuman J.V.: "Probabilistic logics and the synthesis of reliable organisms from unreliable

Components". *Princeton University Press*, Princeton N.J, pp.43-98, 1956.

[Niko02] : Nikolic K, Sadek A and Forshaw M. "Fault tolerant techniques for nanocomputers" *Nanotechnology,* vol 13, pp 357–62, 2002.

[Orlo02] A. O. Orlov, I. Amlani, R. Kummamuru, R. Rajagopal, G. Toth, J.Timler, C. S. Lent, G. H. Bernstein, and G. L. Snider. "Power gain in quantum-dot cellular automata latch ". *Appl. Phys. Lett.*, vol 81, pp. 1332–1334, 2002.

[Liao11] : Liao, S.-Y. Retrouvey, J.-M. Agnus, G. Zhao, W. Maneux, C. Fregonese, S. Zimmer, T. Chabi, D. Filoramo, A. Derycke, V. Gamrat, C. Klein, J.-O. "Design and Modeling of a Neuro-Inspired Learning Circuit Using Nanotube-Based Memory Device". *IEEE Transactions on Circuits and Systems I: Regular Papers, à paraître.*

[Liao11b] : Si-Yu LIAO "Caracterisation electrique et électro-optique de transistor a base de nanotube de carbone en vue de leur modelisation compacte ". *Thèse de doctorat* à l'université Bordeaux I, 2011.

[Pers08]: Pershin, Y. V., and M. Di Ventra "Spin memristive systems: Spin memory effects in semiconductor spintronics". *Phys. Rev. B,* vol. 78, pp. 113309, 2008.

[Pers09] : Y. V. Pershin, S. La Fontaine and M. Di Ventra, "Memristive model of amoeba's learning» Phys. Rev. E, vol. 80, pp. 021926, 2009.

[Pers10] : Pershin, Y.V., Di Ventra, M., "Neuromorphic, Digital and Quantum Computation with Memory Circuit Elements ". *arXiv*:1009.6025v1, 2010.

[Phat92] : D.S. Phatak and I. Koren, "Fault Tolerance of Feedforward Neural Nets for Classification Tasks". Proceedings of International Joint Conference on Neural Nets (*IJCNN*), vol. 2, pp. 386-391, 1992.

[Pint00] : Pinto, R. D, et al., "Synchronous behavior of two coupled electronic neurons". *Phys Rev E.*, vol. 62, pp. 2644-56, 2000.

[Ram93] : U. Ramacher et al. " Multiprocessor and Memory Architecture of the Neurocomputer synapse-1". *J. Neural Syst.* vol. 4, pp. 333-336, 1993.

[Rao06] : Wenjing Rao, A. Orailaglu, and R. Karri, "Nanofabric totpologie and reconfiguration algorithms to support dynamically adaptive fault tolerance", *VLSI Test symposium, 2006.* Proceeding. 24th IEEE, pp. 214-221, 2006.

[Raou08]: Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y.-C., Shelby, R. M., Salinga, M., "Phase-Change Random Access Memory: A Scalable Technology". *IBM J. Res. & Dev.*, vol. 52, pp. 465-479, 2008.

[Raou09] : Simone Raoux, " Phase Change Materials ". *Annual. Review. Material.* Research., vol. 39, pp. 25-48, 2009.

[Ros62]: F. Rosenblatt. "Principles of Neurodynamics:Perceptrons and Theory of Brain Mechanism". *Spartan*, 1962.

[Ros96] : M. Rossmann, B. Hesse, K. Goser, A. B¨uhlmeier, and G. Manteuffel. "Implementation of a biologically inspired neuron-model in FPGA". *In Proc. MicroNeuro*, pages 322– 329, 1996.

[Ried94] : M. Riedmiller, "Rprop - description and implementation details". Institut fur Logik, Komplexit¨at und Deduktions syteme - University of Karlsruhe, Tech. Rep., 1994.

[Rum86]: Rumelhart (D.E.), Hinton (G.E.) et Williams (R.J.). "Learning Representations by Back-Propagation Errors". *Nature*, vol. 323, pp. 533-536, 1986.

[Ruc02] : U. Ruckert, "ULSI architectures for artificial neural networks". *IEEE Micro*, vol.22, no. 3, pp. 10–19, May 2002.

[Sad04] : Sadek A S, Nikolic K and Forshaw M. , "Parallel information and computation with restitution for noise-tolerant nanoscale logic networks". *Nanotechnology,* vol. 15, pp. 192–210, 2004.

[Saka93]: S. Sakaue, T. Kohda, H. Yamamoto, S. Maruno, Y. Shimeki, "Reduction of Required Precision bits for Back-propagation Applied to Pattern Recongnition". *IEEE Transaction on Neural Network*, vol. 4, pp. 270-275, 1993.

[Sala94] : V. Salapura, M. Gschwind, and O. Maisch berger. "A fast FPGA implementation of a general purpose neuron". *In Proc. FPL*, 1994.

[Saig04] : S. Saighi. "Circuits et systèmes de modélisation analogique de réseaux de neurones biologiques : application au développement d'outils pour les neurosciences computationnelles". *Thèse de doctorat*, University Bordeaux 1– n 2891,2004.

[She04] : Shepherd, G. "Introduction to Synaptic Circuits," *in The Synaptic Organization of the Brain*, edited by Gordon Shepherd, 5th edition, Oxford University Press, 2004.

[Shi03] : Shi, R. Z. and Horiuchi, T., "A Summating, Exponentially-Decaying CMOS Synapse for Spiking Neural Systems". Neural Information Processing Systems Foundation *NIPS,* 2003.

[Shim07] : Shima, H., F. Takano, H. Akinaga, Y. Tamai, I. H. Inoue, and H. Takagi, "Resistance switching in the metal deficient-type oxides: NiO and CoO". *Appl. Phys. Lett.*, vol. 91, pp. 012901, 2007.

[SiVe67] : J. G. Simmons and R. R. Verderber "New Conduction and Reversible Memory Phenomena in Thin Insulating Films". *Proc. R. Soc. A*, vol. 301, pp. 77-102, 1967.

[Sma06] : F. Smach, M. Atri, J. Miteran, and M. Abid. "Design of a neural networks classifier for face detection". *J. of Computer Science*, vol 2, pp 257-260, 2006.

[Snid08] : Snider, G.S. "Spike-timing-dependent learning in memristive nanodevices". Nanoscale Architectures, 2008. *NANOARCH 2008*. IEEE International Symposium on, pp. 85 – 92, 2008.

[Snid11] : Greg Snider, "Instar and outstar learning with memristive nanodevices". *Nanotechnology* 22 015201, 2011.

[Stan03]: M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architectur". *Proc. Of the IEEE*, 91:1940–1957, 2003.

[Stra11] : Strachan JP, Strukov DB, Borghetti J, Yang JJ, Medeiros-Ribeiro G, Williams RS, "Switching location of a Bipolar Memristor. Chemical, Thermal and Structural Mapping". *Nanotechnology*, vol. 22, pp. 254001, 2011.

[Stru05] : D. B. Strukov and K. K. Likharev, "CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-terminal Nanodevices". *Nanotechnology*, vol. 16, pp. 888-900, Apr. 2005.

[Stru08] ; Strukov, D. B., G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found". *Nature*, vol. 453, pp 80-83, 2008.

[Stru07] : D. B. Strukov and K. K. Likharev, "Reconfigurable Hybrid CMOS/Nanodevice Circuits for Image Processing". *IEEE Trans. on Nanotechnology*, vol. 6, pp. 696-710, 2007.

[Stru09] : D. B. Strukov, J. L. Borghetti, and R. S. Williams, " Coupled Ionic and Electronic Transport Model of Thin-Film Semiconductor Memristive Behavior". *Small*, vol. 5, pp. 1058-1063, 2009.

[Stru09a] : Strukov, D. B., and R. S. Williams, "Exponential ionic drift: fast switching and low volatility of thin-film memristors ". *Appl. Phys. A Mater. Sci. Process.*, vol. 94, pp. 515, 2009.

[Stru09b] : Strukov, D. B., and R. S. Williams "Four-dimensional address topology for circuits with stacked multilayer crossbar arrays". *Proc. Nat. Ac. Sci.* vol. 106, pp 20155, 2009.

[Taho05] : M.B. Tahoori. "Defect, yield, and design in sublithographic nano-electronics". Defect and Fault Tolerance in VLSI System. *20th IEEE International Symposium on*, pp 3-11, 2005.

[Tana89] : Tanaka H., "Study of a High Reliable System against Electric Noise and Element Failures". *Proc. of the International Symposium on Noise and Clutter Rejection in Radars and Imaging Sensors,* pp. 415-420,1989.

[Taci10] : Taciano Perez "Non-Volatile Memory : Emerging Technologies And Their Impacts on Memory Systems", Technical report N°60, Porto Alegre, 2010.

[Taur97a] : *Y. Taur*, et al., "*CMOS* scaling into nanometer regime", *Proc. IEEE*, vol. *85*, pp. 486 - 504, 1997.

[Taur97b] : Y. Taur, E.J Nowak, "CMOS Devices below 0.1pm : How Hight Will Performance Go ?", *International Electron Device Meeting IEDM*, pp 215-218, 1997.

[Tehr08] : Mohammad Tehranipoor, Reza M.P. Rad, "Defect Tolerance for Nanoscale Crossbar-Based Devices". *IEEE Design and Test of Computers*, vol. 25, pp. 549-559, 2008.

[Tchen06] : T. C. Chen, "Overcoming Research Challenges for CMOS Scaling: Industry Directions". *Proceedings of 8th International Conference on Solid-State and Integrated Circuit Technology*, pp. 4–7, 2006.

[Tim02]: J. Timler and C. S. Lent. Dissipation and gain in quantum-dot cellular automata. J. *Appl. Phys.*, 91 :823–831, 2002.

[Torr06]: Cesar Torres-Huitzil and Bernard Girau. "Massively distributed implementation of a spiking neural network for image segmentation on FPGA". Neural Information Processing Letters and Reviews, special issue on *Bio-inspired Models and Hardware*, vol 10, pp 105-114, 2006.

[Tour03] : James M. Tour, Long Cheng, David P. Nackashi et al, "Nanocell electronic memories". *J. Am. Chem. Soc*, vol 125 (43), pp 13279–13283, 2003.

[Turel03]: Ö. Türel, I.Muckra, and K. Likharev. " Possible nanoelectronic implementation of neuromorphic networks". Proceedings of the 2003 International Joint *Conference on Neural Networks*, pages 365–370, 2003.

[Turel04] : Ö. Türel, Lee J-H, Ma X and Likharev K, "Neuromorphic architectures for nanoelectronic circuits", Int. J. Circuit Theory Appl. 32 277–302, 2004.

[Turel05] : Türel et al. "Architectures for nanoelectronic implementation of artificial neural networks: new results". *Neurocomputing*, vol. 64, pp. 271-283, 2005.

[Vent09]: Di Ventra, M.; Pershin, Y.V.; Chua, L.O., "Circuit Elements With Memory: Memristors, Memcapacitors, and Meminductors". *Proceedings of the IEEE*, vol. 97, pp 1717-1724, 2009.

[Volk05] :Volkovskii, A., et al., "Analog electronic model of the lobster pyloriccentral pattern generator," *J. Phys.: Conf. Ser.* vol. 23, pp. 47, 2005

[Xiao09] : Xiaobin Wang; Yiran Chen; Haiwen Xi; Hai Li; Dimitrov, D.; "Spintronic Memristor Through Spin-Torque-Induced Magnetization Motion", *IEEE Electron device lett*, vol. 30, pp. 294 – 297, 2009.

[Yan08]: Yang, J. J., M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices". *Nat. Nanotechnol*, vol. 3, pp 429, 2008.

[Wang07]: Z. Wang and K. Chakrabarty, "Built-in Self-test and Defect Tolerance in Molecular Electronics-based Nanofabrics". I. Electron. Test, vol 23, pp 145-161, 2007.

[Wang10] : F. Z. Wang, N. Helian, S. Wu, M.-G. Lim, Y. Guo, M. A. Parker, "Delayed Switching in Memristors and Memristive Systems". *IEEE Electron Device Letters*, vol.31, pp.755-757, 2010.

[Wid60] : Widrow, B., & Hoff, M. E. "Adaptive switching circuits". IRE WESCON Convention Record, pp. 96-104. 1960.

[Wid90]: B. Widrow and M.A. Lehr. "30 years of adaptive neural networks". In *Proc. IEEE*, vol. 78, pp. 1415-1442, 1990.

[Wij08] : J.H.B.Wijekoon and P.Dudek, «Compact Silicon Neuron Circuit with Spiking and Bursting Behaviour », *Neural Networks*, Vol 21, N 2-3, pp 524-534, 2008.

[Wuj09] : Wu J, Richard L, McCreery, « Solid-State Electrochemistry in Molecule/TiO2 Molecular Heterojunctions as the Basis of the TiO2 ». Memristor"*Journal of The Electrochemical Society*, vol 156 pp 29-37, 2009.

[Wut07] : Wuttig, M., and N. Yamada, « Phase-change materials for rewriteable data storage », *Nature Materials*, vol. 6, pp. 824, 2007.

[Zha06] : Zhaoning Yu, Wei Wu, Gun-Young Junz, D L Olynick and R Stanley Williams et al, « Fabrication of 30 nm pitch imprint moulds by frequency doubling for nanowire arrays », *Nanotechnology,* vol. 17, pp. 4956, 2006.

[Zhon03] : Z. Zhong, D. Wang, Y. Cui, M. W. Bockrath, and C. M. Lieber, « Nanowire Crossbar Arrays as Address Decoders for Integrated Nanosystems ». *Science.*, vol. 302, pp. 1377-1379, 2003.

[Zhou09] : Zhou, P., Zhao, B., Yang, J., Zhang, Y. « A Durable and Energy E_cient Main Memory Using Phase Change Memory Technology ». *The 36th International Symposium on Computer Architecture*, pp. 14-23, June 2009.

[Zhu94] : Zhu et al. « Autocompensated capacitive circuit for stochastic neural networks ». *Electron Lett,* vol. 30, pp. 330-331, 1994.

[Ws09] : Zhao W., Belhaire E., Chappert C., Mazoyer p. « Power and Area Optimization for Run-Time Reconfiguration System On Programmable Chip Based on Magnetic Random Access Memory ». *IEEE transactions on magnetics*, vol. 45, pp. 776, 2009.

[Ws10] : W. S. Zhao, G. Agnus, V. Derycke, A. Filoramo, J.-P. Bourgoin, and C. Gamrat, «Nanotube devices based crossbar architecture: toward neuromorphic computing ». *Nanotechnology*, vol. 21, pp. 175202, 2010.

[Ws11] : W.S. Zhao, J. Duval, J.-O. Klein, and C. Chappert, "A compact model for magnetic tunnel junction (MTJ) switched by thermally assisted Spin transfer torque (TAS + STT)", *Nanoscale Research Letters*, vol. 6, pp. 368, 2011.