New design approaches for flexible architectures and in-memory computing based on memristor technologies

Khaled Alhaj Ali

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Electronique

Par
Khaled ALHAJ ALI

New design approaches for flexible architectures and in-memory computing based on memristor technologies

Thèse présentée et soutenue à Brest, le 13 juillet 2020
Unité de recherche : Lab-STICC, UMR CNRS 6285
Thèse N° : 2020IMTA0197

Rapporteurs avant soutenance :

Ian O'Connor Professeur, Ecole Centrale de Lyon
Lorena Anghel Professeur, Grenoble INP

Composition du Jury :

Président :	Adnan Harb	Professeur, Lebanese International University
Rapporteurs :	Ian O'Connor	Professeur, Ecole Centrale de Lyon
	Lorena Anghel	Professeur, Grenoble INP
Examinateurs :	Rouwaida Kanj	Associate Professor, American University of Beirut
Encadrants :	Mostafa Rizk	Assistant Professor, Lebanese International University
	Jean-Philippe Diguet	Directeur de recherche CNRS, Lab-STICC
Dir. de thèse :	Amer Baghdadi	Professeur, IMT Atlantique
Co-dir. de thèse :	Jalal Jomaah	Professeur, Lebanese University

Invité(s)

	Grégory Di Pendina	Ingénieur de recherche, SPINTEC

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to my supervisors, whom I believe that I was not able do this work without their guidance and support. On the personal side, I believe that the stress of research could not be easy without the support of my family, and friends: First, Thanks to my Parents, whom I believe I owe everything since the first days of my life. Thanks to them for believing in me. Thank you my brother and sisters. Special thanks to my Fiancé, for her support. Finally, for those whom I spent most of my time, for my second family, for my colleagues and friends, thank you for making this journey nice and easy.

Finally, I have tried my best to represent this thesis dissertation as appropriate as possible. I am feeling myself fortunate to finish the thesis on such an interesting topic which is really a great experience.

Résumé long

Le développement récent de nouvelles technologies de mémoires non-volatiles basées sur le concept de memristor a suscité de nombreux efforts pour explorer leur utilisation potentielle dans différents domaines d'application. Ce nouveau type d'éléments nanométriques à deux terminaux présente des propriétés intrinsèques très intéressantes en termes de vitesse de commutation, de densité et de capacité de stockage non-volatile, ainsi que de consommation énergétique. Une grande partie des efforts de recherche menés vise à les exploiter pour établir un système de mémoire unifié et efficace remplaçant les mémoires flash et CMOS actuelles. D'autre part, leur compatibilité pour une intégration avec les technologies CMOS conventionnelles permet de nouvelles idées de conception basée sur une combinaison et une interaction étroite entre mémoire et calcul. Cela introduit de nouvelles opportunités pour concevoir des architectures novatrices, offrant des niveaux sans précédent de densité, de reconfigurabilité et d'efficacité énergétique. Concevoir des architectures qui peuvent s'adapter dynamiquement aux besoins des applications, apporte de grands avantages en termes d'efficacité énergétique et de performances. Une telle flexibilité est nécessaire au niveau du calcul, des interconnexions et de la mémoire. Elle devient une condition préalable importante pour les architectures matérielles utilisées dans une multitude d'applications telles que les communications numériques et le multimédia, où de nouvelles normes et de multiples services apparaissent en permanence, avec des exigences renforcées en termes de performances et d'efficacité énergétique. Cependant, les technologies actuelles sont inefficaces pour la mise en œuvre de systèmes hautement auto-adaptatifs, en raison du coût de la reconfiguration, incluant les délais et la consommation d'énergie. De plus, les accès à la mémoire, qui deviennent prédominants dans de nombreuses applications, constituent un véritable goulot d'étranglement. La nature polyvalente des mémoires permet de trouver des solutions novatrices pour contourner le surcoût d'implémentations matérielles reconfigurables en termes de surface, de temps d'exécution et de consommation d'énergie. Elle ouvre la voie à de nombreuses applications prometteuses à cet égard. Toutefois, pour tirer pleinement parti de cette technologie émergente, de nouveaux paradigmes architecturaux doivent être inventés. L'existence des memristors a ouvert une voie de recherche originale sur les technologies de mémoires non-volatiles, sur les architectures des unités arithmétiques et logiques, ainsi que sur les mémoires et leurs applications. Les caractéristiques des dispositifs memristifs incitent les chercheurs et les concepteurs de circuits à explorer une révision profonde des paradigmes existants de calcul, de stockage et d'accès aux données. En effet, un memristor est un composant à deux terminaux dont la valeur de résistance commute entre deux états de manière permanente (non-volatile) en appliquant une tension avec une polarité, un niveau et une

durée spécifiques. Le concept de memristor a été généralisé aux dispositifs memristifs, car d'autres technologies émergentes de mémoires non-volatiles ont été théoriquement liées aux memristors.

Concepts fondamentaux

Le memristor a été prédit théoriquement par Leon Chua [1] en 1971. Chua a émis l'hypothèse que le memristor, qui est le quatrième dispositif passif, devrait exister et établir une relation entre le flux (ϕ) et la charge (q). La première fabrication d'un dispositif à memristor a été réalisée par le groupe de Williams dans les laboratoires de Hewlett-Packard (HP) [2] en 2008. La structure du dispositif fabriqué est composée d'une couche stoechiométrique (T_iO_2) et d'une couche déficiente en oxygène (T_iO_{2-x}), qui est prise en sandwich entre deux électrodes en platine comme le montre la figure 1(a). Le dispositif nanométrique à deux extrémités obtenu présente une résistance dynamique qui est déterminée par l'intégrale du courant circulant dans le dispositif lui-même. La résistance d'un memristor peut varier entre deux états : l'état de faible résistance (LRS : Low Resistance State) et l'état de haute résistance (HRS : High Resistance State) correspondant respectivement à R_{ON} et R_{OFF}. Les états de résistance R_{ON} et R_{OFF} représentent le niveau logique "0" ou "1". Le memristor a la capacité de conserver sa valeur de résistance même après que la source d'alimentation ait été retirée. La dernière résistance atteinte est naturellement mémorisée sans l'aide d'une source de rafraîchissement de l'état. Cette propriété fait du dispositif memristor un bon candidat pour les prochaines générations de mémoires non-volatiles [3]. En fait, la théorie des memristors est généralisée aux systèmes memristifs. Williams et Chua ont avancé que tous les dispositifs résistifs dotés de mémoire sont classés comme des memristors, quels que soient le matériau du dispositif et les mécanismes physiques de fonctionnement [4][5]. Ils présentent tous une "empreinte digitale" distinctive, qui se caractérise par une boucle d'hystérésis pincée, comme le montre la figure 1.1(b). La boucle d'hystérésis est contenue au premier et troisième quadrant du plan V-I, qui change de forme en fonction de l'amplitude et de la fréquence de la source périodique sinusoïdale de tension/courant d'entrée [4]. Le symbole utilisé pour les memristors est représenté sur la figure 1(c).

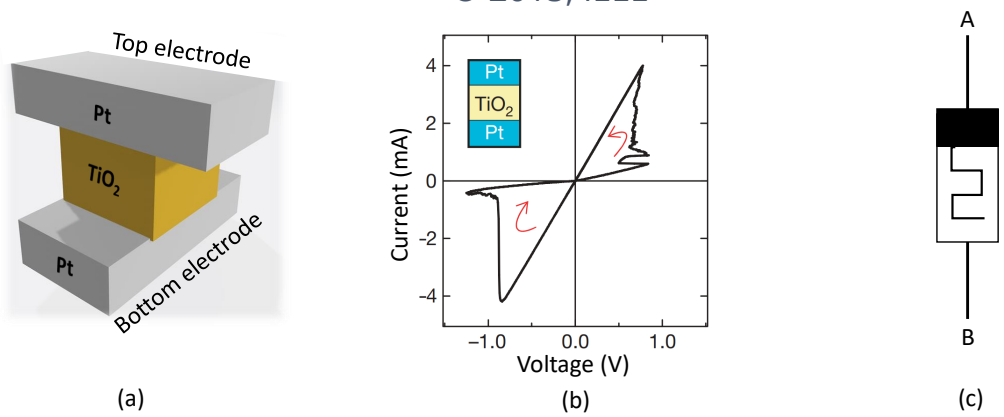


Figure 1: Memristor: (a) Structure, (b) Boucle d'hystérésis [6], (c) Symbole

Objectifs de la thèse et contributions

Dans le contexte mentionné ci-dessus, l'objectif de ce travail de thèse a été d'explorer et d'introduire de nouvelles conceptions basées sur les memristors qui combinent flexibilité et efficacité en proposant des architectures originales qui dépassent les limites des architectures existantes. Notre but est de mener cette exploration et cette étude à différents niveaux, c'est pourquoi nous nous fixons les trois objectifs principaux suivants:

Au niveau des interconnexions

Explorer l'utilisation de dispositifs memristifs pour permettre un haut degré de flexibilité basé sur des interconnexions programmables. Les memristors peuvent être insérés comme des commutateurs reconfigurables au niveau des interconnexions afin d'établir un routage sur la puce.

Au niveau du calcul

Explorer l'utilisation des dispositifs memristifs et leur intégration avec la technologie CMOS pour l'implémentation de fonctions de logique combinatoire. Ces architectures hybrides memristor-CMOS devraient exploiter la forte densité d'intégration des memristors afin d'améliorer les performances des circuits numériques, et en particulier des unités arithmétiques et logiques.

Au niveau de la mémoire

Explorer de nouvelles approches de calcul en mémoire et d'architectures associées qui permettent de combiner efficacement le stockage et le traitement afin de contourner les problèmes liés aux accès mémoire (memory wall) et d'améliorer ainsi l'efficacité

de calcul. De plus, appliquer les techniques et les architectures proposées dans des études de cas d'application réelle afin d'évaluer leurs performances.

Pour atteindre ces objectifs, plusieurs contributions originales ont été proposées dans le cadre de cette thèse de doctorat. Ces contributions peuvent être résumées comme suit :

1. Proposition de la première architecture de transformée de Fourier rapide (FFT) reconfigurable basée sur des memristors, nommée mrFFT. L'architecture originale proposée permet un support efficace de toute combinaison de papillons FFT de radix-2 et radix-3. L'extensibilité est assurée par une topologie de maillage 2D (mesh 2D). La flexibilité est réalisée au niveau des interconnexions, ce qui permet une réutilisation optimisée des ressources matérielles grâce à un routage non-volatil basé sur les memristors.

2. Proposition d'une nouvelle méthodologie de conception pour la mise en œuvre d'une logique combinatoire hybride memristor-CMOS. L'approche s'appuie sur un style de conception logique existant, appelé Memristor Ratioed Logic (MRL). Dans ce contexte, un mapping original de portes logiques MRL sur une structure de crossbar est proposé. L'architecture proposée, appelée X-MRL, combine les attributs de densité et d'extensibilité des crossbars memristifs et la possibilité de leur intégration au-dessus d'une couche de portes logiques CMOS. La conception d'un additionneur complet 1 bit est détaillée sous la forme d'une étude de cas, ainsi que le layout et les résultats de simulation du circuit réalisé avec Cadence Virtuoso et une technologie CMOS 65 nm. La comparaison des résultats avec une implémentation pure CMOS est prometteuse en terme de surface, mais pas en termes de consommation d'énergie en raison des faibles valeurs de résistance des modèles de memristor disponibles.

3. Un nouveau style de conception logique, nommé Memristor Overwrite Logic (MOL), est introduit pour implémenter des opérations logiques sur des structures à base de memristors. Dans l'approche proposée, le memristor se comporte comme un accumulateur logique. Les opérations MOL sont indépendantes des paramètres de la technologie des memristors et tolèrent la variabilité du dispositif memristif. Par conséquent, il est considéré comme éligible pour des applications de haute fiabilité. Il est démontré que le style de conception logique proposé est adapté aux réseaux de crossbars memristifs et peut donc être utilisé pour des applications de calcul en mémoire.

4. Mise en œuvre d'une architecture de mémoire de calcul basée sur MOL. L'architecture

originale proposée, basée sur le couplage de deux crossbars conventionnels, est capable d'effectuer des opérations ET/OU au niveau des bits entre deux mots stockés. L'architecture mémoire MOL proposée peut être configurée simplement entre les modes de stockage (mémoire) et de calcul (traitement). L'architecture mémoire présentée est réalisée en utilisant Cadence Virtuoso avec une technologie CMOS 65nm et un modèle de référence de l'état de l'art pour la technologie STT-MTJ (Spin Transfer Torque - Magnetic Tunnel Junction). Des simulations sont également effectuées pour confirmer l'exactitude du fonctionnement et pour fournir des analyses détaillées des délais et du temps d'exécution, ainsi que de la consommation énergétique.

5. Démonstration de la méthodologie de conception MOL proposée et de la mémoire de calcul associée par le biais de plusieurs études de cas d'application. La première concerne un additionneur complet de N-bits. La séquence des opérations MOL est illustrée. L'addition N-bits est analysée et évaluée en termes de nombre d'étapes de calcul nécessaires et de consommation d'énergie, montrant des améliorations significatives par rapport aux travaux existants dans la littérature.

   La deuxième étude de cas correspond à l'implémentation du calcul du CRC (cyclic redundancy check), couramment utilisé pour la détection des erreurs, qui peut être éventuellement utile pour détecter les erreurs provoquées par la mémoire elle-même. Un CRC de n-bit est effectué dans la mémoire MOL proposée.

   La troisième application porte sur le développement d'une nouvelle architecture dédiée au calcul des réseaux neuronaux profonds (DNN). Cette architecture repose sur des bancs de mémoire MOL répliqués et interconnectés pour effectuer le processus d'accumulation pondérée (multiplication et addition) dans les cellules memristives de stockage.

Structure du manuscrit et résumé des chapitres

Le manuscrit de thèse est composé de quatre chapitres, qui sont résumés dans les paragraphes suivants

Chapitre 1 – Memristor : principes et applications

Ce chapitre présente les fondements scientifiques liés aux contributions développées dans cette thèse. Un aperçu général sur les principes des memristors est fourni, comprenant les définitions de base, le mécanisme de fonctionnement des memristors et leur modélisation.

Le chapitre aborde ensuite l'utilisation des dispositifs memristifs dans trois domaines principaux : stockage, interconnexion et conception logique. Le memristor est d'abord présenté comme un élément de mémorisation émergent qui combine vitesse, densité et non-volatilité. Les avantages par rapport aux mémoires traditionnelles (DRAM, SRAM, NAND flash, ...) sont mis en évidence. L'opportunité d'utiliser les memristors comme éléments de routage au niveau des interconnexions est présentée avec les applications correspondantes dans la littérature. Un grand degré de flexibilité dans les circuits ASIC et FPGA peut être atteint dans ce contexte. Le chapitre aborde ensuite l'utilisation des memristors comme un levier pour de nouvelles générations de conception logique. De nouveaux paradigmes de calcul peuvent être envisagés, comme le calcul en mémoire (in-memory computing), en rupture avec le modèle traditionnel de von Neumann. Dans ce contexte, les schémas existants de conception logique à base de dispositifs memristifs sont passés en revue et une évaluation pour les comparer est proposée à la fin du chapitre.

Chapitre 2 – Architecture FFT reconfigurable basée sur les memristors

Ce chapitre est consacré à la présentation de la conception proposée pour une architecture FFT reconfigurable (mrFFT). Cette première contribution explore l'utilisation de dispositifs memristifs pour mettre en œuvre des interconnexions programmables afin de permettre la conception d'architectures flexibles et une reconfiguration efficace. La FFT est choisie comme étude de cas avec un besoin important en flexibilité en particulier dans le domaine des applications de télécommunications et de traitement de signal. L'idée principale consiste à proposer une architecture de papillon FFT reconfigurable, nommée RBF (reconfigurable butterfly) qui peut être configurée en radix-2 et radix-3 pour supporter des tailles de FFT en puissances de 2 et 3. Plusieurs blocs RBF sont instanciés et disposés dans une topologie de maillage 2D avec des interconnexions à base de memristors servant d'éléments de routage. Trois blocs RBF sont nécessaires pour réaliser un papillon de radix-3 alors qu'un seul est nécessaire pour le radix-2. L'implémentation pipelinée single-path delay feedback (SDF) de la FFT, qui constitue une référence dans la littérature, est considérée. Par conséquent, toute taille de FFT pouvant être factorisée en puissances de 2 et 3 peut être supportée en configurant le nombre correspondant d'étages de pipeline en radix-2 et radix-3 et en connectant de manière appropriée les blocs RBF correspondants. Un exemple d'architecture instanciant 18 blocs RBF est détaillé, avec notamment l'architecture du réseau de routage et le placement des memristors, des multiplieurs complexes, des FIFO et des LUT. Cette architecture de type mrFFT supporte 44 tailles de FFT, y compris les 32 modes de fonctionnement qui sont définis dans la norme 3GPP-

LTE. L'architecture proposée a été évaluée par une estimation analytique du nombre et du pourcentage d'activation des ressources matérielles utilisées. La comparaison avec une référence récente d'architecture flexible de FFT conçue en CMOS montre une réduction significative de plus de 25% du nombre de multiplieurs, d'additionneurs, de FIFO et de multiplexeurs. Elle montre également un meilleur ratio d'activation des multiplieurs et des additionneurs dans l'architecture mrFFT. Cependant, cela se fait au prix de l'intégration de 119 memristors et des blocs de sélection correspondants. Afin d'évaluer avec précision l'efficacité énergétique de l'architecture proposée, des travaux futurs sont proposés afin de mener des simulations mixtes analogiques/numériques.

Chapitre 3 – Conception hybride memristor-CMOS de fonctions de logique combinatoire

Ce chapitre présente la deuxième contribution relative à la proposition d'une nouvelle méthodologie de conception pour l'implémentation d'architectures hybrides memristor-CMOS. L'objectif ici est d'explorer l'utilisation des memristors et leur haute densité d'intégration et compatibilité avec les technologies CMOS pour la conception de circuits combinatoires. L'approche proposée s'appuie sur le schéma de conception logique Memristor Ratioed Logic (MRL). En effet, ce choix est justifié par le fait que la logique basée sur MRL adopte la tension comme variable d'état pour représenter les entrées et les sorties, comme en CMOS. Afin d'exploiter efficacement la densité des memristors qui peuvent être placés au-dessus d'un circuit CMOS, nous proposons d'utiliser la structure extensible des crossbars.

L'approche proposée, appelée X-MRL, permet la réalisation de portes logiques ET/OU à travers des paires de memritors placés d'une manière spécifique, verticalement et horizontalement, sur le crossbar. Le crossbar lui-même est placé au-dessus d'une couche d'inverseurs CMOS. L'approche est illustrée à travers la conception d'un additionneur complet 1-bit. L'architecture conçue nécessite 18 memristors et 9 inverseurs CMOS. Le mapping proposé en suivant l'approche X-MRL ainsi que le layout du circuit réalisé avec Cadence Virtuoso et une technologie CMOS 65 nm sont détaillés. Le modèle de référence VTEAM est utilisé pour les memristors avec des paramètres d'ajustement spécifiques.

L'analyse des délais et du temps d'exécution, la consommation énergétique et la surface sont évaluées et discutées. Comme la surface en X-MRL correspond à celle occupée par les inverseurs CMOS, une réduction significative d'environ 45% est obtenue par rapport à une implémentation pure CMOS de l'additionneur complet. Néanmoins, la valeur de la puissance moyenne consommée est relativement élevée. Cela est dû aux faibles valeurs de R_{ON} et R_{OFF} du modèle de memristors adopté.

Une comparaison détaillée est présentée à la fin du chapitre par rapport à d'autres circuits memristifs d'additionneur complet 1-bit existants, basés sur les approches MRL, MAGIC et IMPLY. La consommation énergétique reste comparable. Toutefois, la comparaison illustre l'avantage clé de l'approche X-MRL concernant le nombre réduit d'étapes de calcul (computational steps). Par conséquent, la combinaison de la consommation énergétique et du temps de calcul au moyen de la métrique Energy.Delay montre une amélioration significative (entre ×5.7 et ×31) par rapport à l'état de l'art.

Chapitre 4 – MOL : Memristor Overwrite Logic pour le calcul en mémoire

Le quatrième chapitre regroupe la présentation de plusieurs contributions liées à l'utilisation de dispositifs memristifs pour le calcul en mémoire (in-memory computing). La première contribution majeure est représentée par l'introduction d'une nouvelle architecture de calcul logique en mémoire, appelé Memristor Overwrite Logic (MOL). Dans MOL, le résultat d'une opération logique de type ET/OU est réécrit dans l'état interne du memristor. Ce dernier agit soit comme accumulateur logique avec son bit précédemment stocké, soit comme opérateur logique entre ses deux terminaux. L'idée est dérivée de la représentation numérique du memristor qui peut être vu comme une expression logique à 3 variables (deux entrées et un état). MOL ressemble à cet égard au schéma de conception logique memristor-based majority (MAJ), mais en surmonte les inconvénients. Plusieurs spécificités marquantes du schéma de conception proposé (MOL) sont illustrées, telles que son opérabilité avec différents dispositifs memristifs et paramètres technologiques, son adaptation à la réalisation d'opérations à l'intérieur de crossbar de memristors, ainsi que la possibilité d'effectuer ses opérations sur un vecteur de bits. L'intégration du MOL dans un crossbar conventionnel est détaillée avec les quatre modes supportés (write, overwrite, read, idle) et les architectures appropriées pour les contrôleurs de périphériques.

La deuxième contribution majeure présentée dans ce chapitre concerne la proposition d'une architecture de mémoire basée sur l'approche MOL pour le calcul en mémoire (MOL-based computational memory). L'architecture proposée, est constituée d'une paire de crossbars memristifs basés sur MOL qui opèrent de manière complémentaire, de leurs drivers et des séquences de contrôle appropriés. Cette structure permet d'effectuer des opérations sur les bits de deux mots stockés, plutôt qu'entre un mot stocké et un mot d'entrée externe, ce qui est une originalité de ce schéma. Le chapitre illustre également comment toute fonction arithmétique générale peut être exécutée en mémoire en la décomposant en une suite d'opérations MOL itératives (micro-instructions).

Pour la validation et l'évaluation des performances, le chapitre présente une étude de

cas détaillée sur la réalisation d'une addition N-bit en mémoire. La procédure de décomposition et de mapping de l'opération arithmétique dans la mémoire de calcul MOL est expliquée, ainsi que la séquence détaillée des opérations. L'implémentation de l'architecture mémoire proposée, avec une largeur de 8 bits, est réalisée en utilisant Cadence Virtuoso avec une technologie CMOS 65nm et un modèle de référence de l'état de l'art pour la technologie STT-MTJ. L'analyse des délais et du temps d'exécution, la robustesse face à la variabilité de la résistance des dispositifs MTJ et la consommation énergétique sont évaluées et discutées. Les comparaisons avec les travaux existants dans ce domaine illustrent clairement les atouts de l'approche proposée pour réduire la latence (nombre réduit d'étapes de calcul) et le nombre de cellules memristives nécessaires. Cependant, ces réductions s'accompagnent avec une plus grande consommation énergétique. Cela s'explique par le fait que l'approche proposée réalise les opérations au niveau vecteur plutôt qu'au niveau bit, ce qui devrait toutefois réduire la complexité de l'unité de contrôle. La tolérance à la variabilité de la résistance est confirmée par des simulations (7% à 21%).

Le chapitre se termine par la présentation de deux autres études de cas d'application pour le calcul en mémoire basé sur l'approche MOL. La première concerne le calcul du CRC (cyclic redundancy check), couramment utilisé pour la détection des erreurs, qui peut être éventuellement utile pour détecter les erreurs provoquées par la mémoire elle-même. La même architecture de mémoire MOL a été réutilisée. La séquence d'exécution en termes d'opérations MOL élémentaires, le nombre requis d'étapes de calcul et les résultats des simulations fonctionnelles sont illustrés et justifiés.

La deuxième étude de cas applicatif porte sur l'utilisation de l'approche MOL de calcul en mémoire pour l'exécution de réseaux de neurones profonds (DNN). La principale contribution concerne ici la mise en œuvre d'une opération optimisée de multiplication-accumulation en mémoire. Afin de réduire le nombre d'étapes de calcul, l'architecture de la mémoire MOL proposée a été légèrement modifiée. La possibilité de réaliser des produits partiels a été incorporée en ajoutant un étage de multiplexeurs-inverseurs pour permettre aux données d'être fournies également au niveau des lignes de la mémoire. Un mapping d'un réseau de neurones simplifié sur la mémoire de calcul est également présenté, avec une estimation de la latence de l'opération de convolution associée.

Conclusion et perspectives

Ce travail de thèse a été consacré à l'exploration du potentiel des technologies émergentes des memristors au niveau des interconnexions, au niveau du calcul et au niveau

de la mémoire. Dans ce contexte, le but était d'explorer et d'introduire de nouvelles approches de conception basées sur les memristors pour combiner flexibilité et efficacité en proposant des architectures originales qui dépassent les limites des architectures existantes. Au niveau des interconnexions, nous avons étudié l'utilisation de dispositifs memristifs pour permettre une grande flexibilité basée sur des réseaux d'interconnexion programmables. Cela a permis de proposer la première architecture de transformée de Fourier rapide reconfigurable basée sur des memristors, nommée mrFFT. Les memristors sont insérés comme des commutateurs reconfigurables au niveau des interconnexions afin d'établir un routage flexible puce. Au niveau du traitement, nous avons exploré l'utilisation de dispositifs memristifs et leur intégration avec les technologies CMOS pour la conception de fonctions logique combinatoire. Ces circuits hybrides memristor-CMOS exploitent la forte densité d'intégration des memristors afin d'améliorer les performances des implémentations numériques, et en particulier des unités arithmétiques et logiques. Au niveau mémoire, une nouvelle approche de calcul en mémoire a été introduite. Dans ce contexte, un nouveau style de conception logique a été proposé, nommé Memristor Overwrite Logic (MOL), associé à une architecture originale de mémoire de calcul. L'approche proposée permet de combiner efficacement le stockage et le traitement afin de contourner les problèmes liés aux accès mémoire et d'améliorer ainsi l'efficacité de calcul. L'approche proposée a été appliquée dans trois études de cas à des fins de validation et d'évaluation des performances. Ces travaux ouvrent la voie à de nombreuses pistes de recherche dans le domaine des architectures de calcul à base de memristors, qui sont détaillées à la fin du manuscrit.

Introduction

The recent development of new non-volatile memory technologies based on the memristor concept has triggered many efforts to explore their potential usage in different application domains. This novel type of two terminal nano-scale elements presents very fast switching characteristics, non-volatile dense storage capacity, and low power consumption [7]. Main part of conducted research efforts aims to exploit them for establishing a unified and efficient memory system replacing current flash and CMOS-based memories [7][8]. On the other hand, the possibility to integrate memristors on top of CMOS logic gates allows for new design ideas based on close combination and interaction between memory and computation. This introduces new opportunities of efficient reconfiguration, high performance, and low power design.

Designing architectures which can adapt dynamically to application needs, brings great advantages in terms of energy efficiency and performances. Such flexibility is required at processing, interconnect, and memory levels. It becomes an important prerequisite for hardware architectures used in multitude applications such as digital communications and multimedia where new standards and multiple services are continuously emerging, with strengthen requirements in terms of performance and energy efficiency. Current technologies are inefficient for highly self-adaptive systems, due to the cost of reconfiguration, including delay and power consumption. Furthermore, memory accesses, which become predominant in many applications, constitute a real bottleneck.

The versatile nature of memristors provides insights for novel solutions to circumvent the overhead of reconfigurable hardware implementations in terms of implementation area, execution time, and power consumption. It paves the way for many promising applications in this regard. However, in order to take full advantages of this emerging technology, new architectural paradigms must be invented. Indeed, although the existence of memristor was theoretically predicted by Leon Chua in 1971 [1], its first physical realization has been only reported in 2008 by a research team at Hewlett-Packard (HP) [2]. This relatively recent event has triggered an orthogonal research path on non-volatile memory technologies, memristive arithmetic logic units, and their applications.

The characteristics of memristive devices are motivating researchers and circuit de-

signers to explore profound revision on the existing paradigms of computation, storage, and access to data. Indeed, a memristor is a two terminal component whose resistance value switches between two states in a permanent way (non-volatile) by applying a voltage with a specific polarity, level, and duration. The concept of memristor has been generalized to memristive devices, as other emerging technologies of non-volatile memories have been theoretically linked to memristors [4].

Objectives of the thesis

In the above mentioned context, the goal of this thesis work is to explore and introduce new memristor-based designs that combine flexibility and efficiency through the proposal of original architectures that break the limits of the existing ones. Our aim is to conduct this exploration and study at different levels, therefore we set the following three main objectives:

At interconnect level

Explore the use of memristive devices to allow high degree of flexibility based on programmable interconnects. Memristors can be inserted as reconfigurable switches at the level of interconnects in order to establish on-chip routing.

At processing level

Explore the use of memristive devices and their integration with CMOS technologies for combinational logic design. Such hybrid memristor-CMOS designs should exploit the high integration density of memristors in order to improve the performance of digital designs, and particularly arithmetic logic units.

At memory level

Explore new in-memory computing approaches and computational memory architectures that allow efficient combination of storage and processing in order to bypass the memory wall problem and thus to improve the computational efficiency. Thereafter, apply the proposed techniques and architectures in real application case studies for the sake of performance evaluation.

Thesis contributions

Towards these objectives, several original contributions have been proposed in the framework of this PhD thesis. These contributions can be summarized as follows:

1. Proposal of the first memristor-based reconfigurable fast Fourier transform (FFT) architecture, namely mrFFT. The proposed original architecture allows an efficient support of any combination of radix-2 and radix-3 butterflies. Scalability is ensured through a 2D mesh topology. Flexibility is realized at the level of interconnects, allowing for optimized hardware reuse through a memristor-based non-volatile routing.

2. Proposal of new design methodology for implementing hybrid memristor-CMOS combinational logic. The approach relies on an existing logic design style, called Memristor Ratioed Logic (MRL). In this context, an original mapping of MRL gates in a crossbar array is proposed. The structure of the proposed MRL-based crossbar design, namely X-MRL, combines the density and scalability attributes of memristive crossbar arrays and the opportunity of their implementation at the top of CMOS layer. The design of a 1-bit full adder is detailed as a case study, together with corresponding layout and simulation results using Cadence Virtuoso toolset and CMOS 65 nm process.

3. A new logic design style, namely Memristor Overwrite Logic (MOL), is introduced for performing logic operations. In the proposed approach, the memristor behaves as logic accumulator. MOL operations are independent from the memristor technology parameters and tolerant against device variability. Therefore, it is considered eligible for applications of high reliability. The proposed logic design style is shown to be adapted to crossbar memory arrays, thus can be employed for in-memory computing applications.

4. Implementation of a MOL-based computational memory architecture. The proposed original architecture, based on coupling two conventional crossbar arrays, is able to perform bitwise AND/OR operations between two stored words. The proposed MOL-memory architecture can be simply configured between storage (memory) and computation (processing) modes. The presented memory architecture is designed using a CMOS 65 nm process and an accurate model of Spin Transfer Torque - Magnetic Tunnel Junction (STT-MTJ) memristive device. Simulations are also conducted to confirm functional correctness and to provide more concrete analyses of delay and power.

5. Demonstration of the proposed MOL design methodology and associated computational memory through several application case studies. The first one concerns an N-bit full addition. The sequence of MOL operations is illustrated in time and space. N-bit addition is analysed and evaluated in terms of required number of

computational steps and energy consumption which show significant improvements over existing works in the literature. Moreover, the well known cyclic redundancy check (CRC) code is taken as another case study. CRC is commonly used in digital data transmission and storage systems to ensure data integrity and detect accidental changes in raw data. An n-bit CRC is performed inside the proposed MOL-memory. A possible application of this approach could be in the detection of faults in the stored raw data inside the memory. Detection can be executed purely inside the memory. The third application case study considers the development of a novel architecture dedicated for deep neural networks (DNN) computation. The architecture relies on interconnected replicated MOL-memory banks to perform the weighted accumulation process (multiplication and addition) within the memory storage cells.

Manuscript Organization

The rest of this manuscript is organized in four chapters as follows:

Chapter 1 introduces the basic concepts related to memristive devices and provides a scientific background related to the presented PhD contributions. It gives an overview on memristor fundamentals, including modeling and operation mechanism. It provides in addition a survey on the related works and applications of memristive devices at the levels of interconnects, logic design and memory.

Chapter 2 is dedicated to the presentation of the proposed design for a reconfigurable FFT architecture (mrFFT). This first contribution explores the use of memristive devices for programmable interconnects. The design is discussed and compared in terms of resources, hardware reuse ratio and configuration time needed.

Chapter 3 presents our contribution related the use of memristive devices for combinational logic design. It introduces our new design methodology for implementing hybrid memristor-CMOS combinational logic, based on Memristor Ratioed Logic (MRL) design style. This consists of a scalable MRL-based crossbar design, namely X-MRL, for the implementation of combinational logic. The design of a full adder based on X-MRL approach is presented and evaluated in terms of energy, delay, and area.

Chapter 4 regroups the presentation of the last three contributions cited above, related to in-memory computing. It starts by presenting the proposed novel logic design style, namely Memristor Overwrite Logic (MOL), and an original implementation

of MOL-based computational memory. MOL design style is analyzed and compared against existing approaches of memristor logic design. Evaluation of the proposed computational memory is conducted by comparing the performance of executing N-bit addition with existing designs in the literature targeting in-memory computing. This chapter also presents the CRC as direct application to the proposed computational memory. Furthermore, the last part of this chapter presents a new design architecture for a computational interconnected memory banks dedicated for DNN applications.

The manuscript ends by summarizing the outcome of this thesis work and by providing several ideas and proposals for future work and further investigations.

List of publications

The results of this thesis work have been disseminated through the following journal and international conference publications.

Journal papers

- Khaled Alhaj Ali, Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet, Jalal Jomaah, Naoya Onizawa, and Takahiro Hanyu "Memristive Computational Memory Using Memristor Overwrite Logic (MOL)", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 11, pp. 2370-2382, Nov. 2020

- Khaled Alhaj Ali, Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet, Jalal Jomaah "X-MRL: MRL Crossbar-Based Design for Combinational Logic Implementation", IEEE Access (In review).

Conference papers

- Khaled Alhaj Ali, M. Rizk, A. Baghdadi, J.-Ph. Diguet, J. Jomaah "Towards memristor-based reconfigurable FFT architecture", in Proceedings of the 29th IEEE International Conference on Microelectronics (ICM), Dec. 2017.

- Khaled Alhaj Ali, M. Rizk, A. Baghdadi, J.-Ph. Diguet, J. Jomaah "Crossbar Memory Architecture Performing Memristor Overwrite Logic", in Proceedings of the 26th IEEE International Conference on Electronics Circuits and Systems (ICECS), Nov. 2019

- Khaled Alhaj Ali, M. Rizk, A. Baghdadi, J.-Ph. Diguet, J. Jomaah “MRL Crossbar-Based Full Adder Design”, in Proceedings of the 26th IEEE International Conference on Electronics Circuits and Systems (ICECS), Nov. 2019.
- Khaled Alhaj Ali, M. Rizk, A. Baghdadi, J.-Ph. Diguet, J. Jomaah “Memristor Overwrite Logic (MOL) for Energy-Efficient In-Memory DNN”, in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), May 2020.

Chapter 1

Memristor: Principals and Applications

THIS chapter introduces the basic concepts related to memristive devices and provides a scientific background related to the presented PhD contributions. The next section provides an overview on memristor fundamentals. This includes basic definitions, the mechanism for memristor operation, and memristor device modeling. Section 1.2 presents a review on emerging non-volatile memory technologies. Section 1.3 discusses the use of memristors at the level of interconnects. Finally, Section 1.4 gives a brief survey on existing memristor-based logic design styles.

1.1 Memristor fundamentals

Memristor was predicted theoretically by Leon Chua [1] in 1971. Chua hypothesized that memristor which is the fourth passive device should exist and hold a relationship between flux (ϕ) and charge (q). The first fabrication of a memristor device was realized by Williams's group at Hewlett-Packard (HP) labs [2] in 2008. The fabricated device structure is comprised of a stoichiometric (T_iO_2) and an oxygen deficient (T_iO_{2-x}) layer, which is sandwiched between two platinum electrodes as depicted in Fig. 1.1(a). The obtained two-terminal nano-device exhibits a dynamic resistance that is determined by the integral of current flowing through the device itself. The resistance of a memristor can vary between two bounds: the low resistance state (LRS) and high resistance state (HRS) corresponding for R_{ON} and R_{OFF} respectively. The resistance states R_{ON} and R_{OFF} represent the logical level '0' or '1'. A memristor has the ability to retain a resistance value even after the power source is removed from the device. The last attained resistance is naturally memorized without the aid of state refreshing source. This property makes the memristor

device a good candidate for the next generations of non-volatile memories (NVM) [3]. In fact, the theory of memristors is generalized to memristive systems. Williams and Chua posited that all resistive devices with memory are classified as memristors, regardless of the device material and physical operating mechanisms [4][5]. They all exhibit a distinctive "fingerprint", which is characterized by a pinched hysteresis loop as shown in Fig. 1.1(b). The hysteresis loop is confined to the first and the third quadrants of the V-I plane, which changes its contour shape according to the amplitude and frequency of the periodic "sinewave-like" input voltage/current source [4]. The corresponding symbol for memristors is shown in Fig. 1.1(c).

Due to their versatile nature, the use of memristive devices has been investigated at the levels of memory, logic design, and interconnects.

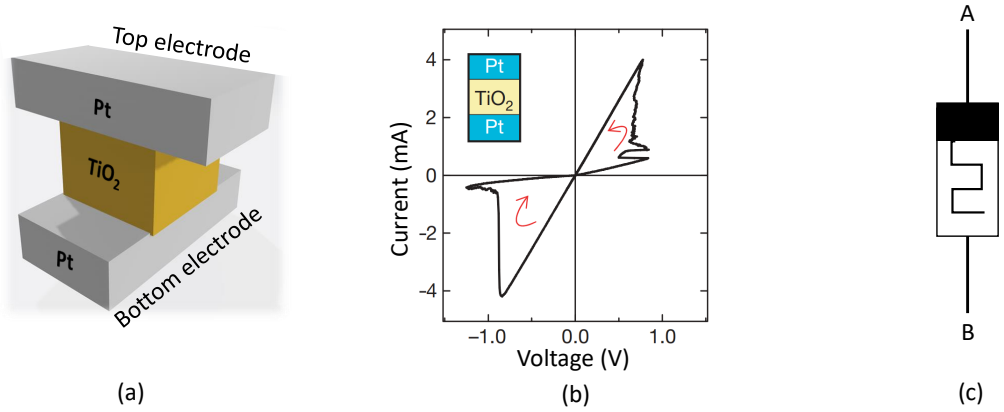


Figure 1.1: Memristor: (a) Structure, (b) Hysteresis loop [6], (c) Symbol

1.1.1 Basic operation

A memristor is a two terminal nano-scale device whose resistance could be modulated between two bounds, the low resistance state (LRS) and the high resistance state (HRS). A large voltage or current bias is able to change the resistance of the device [2]. As illustrated in Fig. 1.2, a positive bias applied on the terminals of the memristor gradually change its resistance toward the HRS state; whereas, a bias in the opposite direction causes its resistance to decrease toward the LRS state. A larger bias typically speeds up the change in the resistance. This change persists after the bias is withdrawn, allowing the state information to be stored. Several memristive devices have been explored in the literature. These devices vary significantly in their properties. Normally, devices with high switching speed, high LRS-HRS margin and high switching endurance are more preferable.

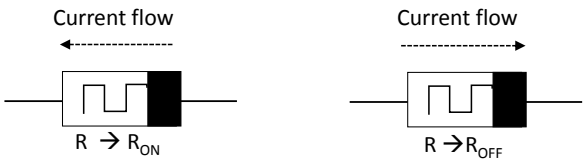


Figure 1.2: Basic operation of a memristor device

1.1.2 Memristor device modeling

HP Labs have described the physical model of memristor as shown in Fig. 1.3. It consists of two layers of TiO_2 sandwiched between platinum contacts [2]. One of the TiO_2 layers has been doped with oxygen vacancies; whereas, the other has been left undoped. As a result, the doped region behaves as semiconductor while the undoped region behaves as an insulator. The width of the doped region $w(t)$ varies between zero and memristor's length

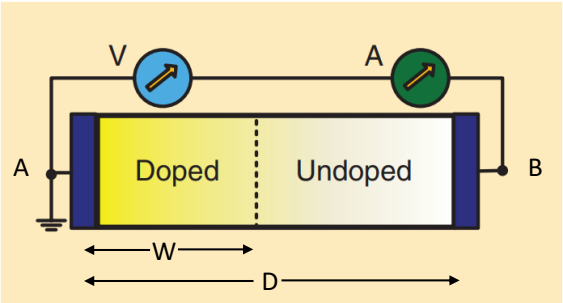


Figure 1.3: TiO_2 memristor model according to [9]

D according to the amount and the direction of the electric charges $q(t)$ moving across the memristor. Hence, applying a certain bias to the memristor leads to the flow of current which in turn changes the value of $w(t)$. Therefore, the virtual boundary separating the doped and undoped regions moves leading to a variation in the memristor total resistance R_{MEM}, which is expressed in (1.1) [2].

$$R_{MEM}(x) = R_{ON}(x) + R_{OFF}(1 - x) \tag{1.1}$$

where $x = \frac{w}{D} \in [0,1]$ and R_{ON} and R_{OFF} are the limiting values of memristor resistance when $w = D$ and $w = 0$ respectively. The speed of the boundary movement between the two ends is called the drift velocity and is represented by the state equation [2]:

$$\frac{dx}{dt} = ki(t) \quad for \quad k = \mu_v \frac{R_{ON}}{D^2} \tag{1.2}$$

where, μ_v is the dopant mobility. Equation (1.2) considers that drift velocity is constant resulting in linear drift model of memristor. However, the experiments which have been

presented in [10] and [11] have proved that the behavior of the implemented memristors is non-linear. To manage the issue of nonlinearity, several models have been proposed in the literature. In [11], the authors have proposed a non-linear dopant drift model as a relation between the current and voltage (I-V) of the memristor. In [12], the drift velocity have been expressed using a window function $f(w)$ in order to model the non-linearity as expressed in (1.3).

$$\frac{dw}{dt} = af(w)V(t)^m \tag{1.3}$$

where a and m are constants, $f(w)$ is the window function and m is an odd integer. The previous presented models are based on the HP physical representation of memristor. In [10], Pickett *et al.* have proposed a more accurate physical model of memristor. A resistor is connected in series with an electron Simmons tunnel barrier [13] instead of connecting two resistors in series, as demonstrated in HP's model. This model exhibits non-linear and asymmetric switching characteristics. Its state equation is expressed in (1.4) [13]:

$$\frac{dx}{dt} = \begin{cases} C_{off}\sinh\left(\frac{i}{i_{off}}\right)\exp\left[-\exp\left(\frac{x-a_{off}}{w_c}-\frac{|i|}{b}\right)-\frac{x}{w_c}\right] \\ C_{on}sinh\left(\frac{i}{i_{on}}\right)\exp\left[-\exp\left(\frac{x-a_{on}}{w_c}-\frac{|i|}{b}\right)-\frac{x}{w_c}\right] \end{cases} \tag{1.4}$$

where the state variable x represents the width of Simmons tunnel barrier, C_{off}, C_{on}, a_{off}, a_{on}, w_c and b are the fitting parameters and i_{off} and i_{on} are current thresholds of the memristor. Obviously, (1.4) shows that Simmons tunnel barrier model is more complicated; hence, it is computationally inefficient. In order to attain a simplified and general model, Kvatinsky et al. [14] have presented the TEAM model, which represents in simpler expressions the same physical model of Simmons tunnel barrier model. Equation (1.5) expresses the state equation representing the TEAM model:

$$\frac{dx}{dt} = \begin{cases} K_{off}\left(\frac{i(t)}{i_{off}}-1\right)^{\alpha_{off}} f_{off}(x), & 0 < i_{off} < i \\ K_{on}\left(\frac{i(t)}{i_{on}}-1\right)^{\alpha_{on}} f_{on}(x), & i < i_{on} < 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{1.5}$$

where i_{on} and i_{off} are current thresholds of the memristor, K_{on}, K_{off}, α_{on} and α_{off} are the fitting parameters and $f_{on}(x)$ and $f_{off}(x)$ are the corresponding window functions of the memristor. However, experimental data acquired from several memristive devices reveals the existence of voltage threshold rather than a current threshold [6]. In [6], the TEAM model has been extended to VTEAM model. Equation (1.6) describes the VTEAM model. It is similar to the expression in (1.5) except for the voltage dependence $v(t)$ and the respective SET and RESET voltage thresholds v_{on} and v_{off}. Moreover, the VTEAM model is considered as a general model since it can be fitted to any other

memristor model [6].

$$\frac{dx}{dt} = \begin{cases} K_{off}\left(\frac{v(t)}{v_{off}} - 1\right)^{\alpha_{off}} f_{off}(x), & 0 < v_{off} < v \\ K_{on}\left(\frac{v(t)}{v_{on}} - 1\right)^{\alpha_{on}} f_{on}(x), & v < v_{on} < 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{1.6}$$

Normally, the window function $f(x)$ is added for a memristor model in order to decelerate the moving boundary of the memristor before reaching its extremities and to guarantee a zero speed exactly when it reaches one of them. In this thesis, we have adopted the VTEAM model to describe the simulated memristor as it provides simple and realistic modeling.

1.2 Memristor as a memory element

1.2.1 Emerging non-volatile memories

Modern systems have to implement additional memory in order to cope with the ever increasing requirement of massive data storage. At the highest abstraction level, memory technologies are classified into volatile and non-volatile. The former need power to retain the stored data; whereas, the latter is capable of retaining it irrespective of the power supply. Non-volatility is measured in terms of the duration of time that data can be retained. The well known static RAM (SRAM) and dynamic RAM (DRAM) belong to the volatile memory category. SRAMs are considered as fast (write/erase time: $40.3ns/0.3ns$) compared with DRAMs (write/erase time: $< 10ns/< 10ns$) [8]. However, the density of SRAM is very low due to its large unit cell area (six transistors: $140F^2$), compared with DRAMs, the unit cell of which is much smaller (one-transistor and one-capacitor: $6F^2$) [7]. Hence, DRAM is used as main memory where the capacity is critical for temporary information storage and processing [15]. SRAMs are usually used in the processors cache where the access time is critical. On the other hand, Non-volatile memories (NVM) are explored. Flash memories such as NAND flash and NOR flash are considered nowadays the most widespread NVM [16]. Flash memories are well optimized, and has a significant commercial presence. Current memory technology roadmap forecasts NAND Flash memory will continue to dominate high-density storage in the short and intermediate terms. However, in long term years, it is expected that NAND flash would face limitations especially when scaling beyond $40nm$ [17].

Several non-conventional NVMs are nowadays being investigated as promising candidates to replace the popular flash memories and potentially even the other conventional

memories such as SRAM and DRAM. For instance, the Spin Transfer Torque Magnetostatic RAM (STT-MRAM) [18], Ferroelectric RAM (FeRAM) [19], phase-change RAM (PCRAM) [20], and resistive RAM (ReRAM) [21] form the category of so-called emerging nonvolatile memories. They are based on two terminal resistive switching elements and belong to the family of memristive devices [4]. These emerging NVMs combine the speed of static random-access memory (SRAM), the density of dynamic random-access memory (DRAM), and the non-volatility of flash memory and so become very attractive as another possibility for future memory hierarchies [8].

Memristive memories are being widely investigated to meet with the requirements of current storage systems. Table 1.1 compares the features of traditional and emerging memory technologies. For instance, RRAM shows promising features in terms of cell size and comparable write time, while it is believed that the low endurance issue of RRAM will be improved in the near future [22]. Moreover, it is expected that these memories will be capable of storing up to 1TB of data on a single chip. Thanks to the ability of "3D-stacking" multiple cells in different configurations in order to save space while still upping the storage limits [23].

Table 1.1: Comparison of conventional and emerging memory technologies [8]

Type	Cell	Feature size	Write time	Endurance
SRAM	Latch	$140F^2$	0.3 ns	$> 3 \times 10^{16}$
DRAM	Stack/trench capacitor	$6F^2$	10 ns	$> 3 \times 10^{16}$
NOR-FLASH	Floating gate	$10F^2$	1 ms	$> 10^5$
NAND-FLASH	Floating gate	$5F^2$	1 ms	$> 10^5$
MRAM	Magnetoresistance	$20F^2$	10 ns	$> 3 \times 10^{16}$
PCRAM	Phase change	$4.8F^2$	20 ns	10^8
FeRAM	Polarization change	$22F^2$	10 ns	10^{14}
ReRAM	Resistance change	$4F^2$	5 ns	$> 10^{10}$

The promising features of memristive NVMs have triggered the research toward developing new paradigms for memory systems. Based on the unique properties of memristive devices, new memory based micro-architectures have been proposed in the literature. Most of the approaches introduces emerging NVM technologies as enablers to the era of memory-intensive computing, which brings interesting opportunities for novel architectural applications.

For instance, the RRAM-based multistate pipeline register (MPR) have been presented in [24]. MPR is different than conventional types of memory, and is used to store multiple data bits, where only a single bit is active and the remaining data bits are idle. The active bit is stored within a CMOS flip flop, while the idle bits are stored within an

RRAM crossbar array co-located with the flip flop. An MPR stores the data of numerous instructions from different threads in the pipeline during execution. MPRs are used to reduce the penalty of a thread switch on a Switch-on-Event multithreading (SoE) processor, enabling a new microarchitecture - Continuous Flow Multithreading (CFMT) [25].

Memristive NVMs are also employed in Field Programmable Gate Array (FPGA) systems. While SRAM-based FPGA suffer from long configuration loading time and excessive leakage power during stand-by, FPGAs using non-volatile memories (NVM) have emerged as a promising alternative [26]. As stated in [27], NVMs eliminate the necessity of loading configuration from off-chip storage, for it preserves the configuration information stored on-chip while powered off. This allows to instantly run the system upon power-up.

1.2.2 Crossbar arrays

At the architectural level, crossbar cell array structure is considered one of the best ways to implement memristive devices [7]. A crossbar array consists of two sets of nano-wires running perpendicular to one another as shown in Fig. 1.4(a). A memristive device is inserted at the intersection of each set of two perpendicular nano wires. Crossbar structure is considered as the optimal topology to implement NVMs such as ReRAMs [7]. It offers several benefits including highest possible device density ($4F^2$), manufacturing simplicity, defect-tolerance [28], scalability and CMOS compatibility. Moreover, crossbars can provide the possibility of having multiple array-layers stacked on top of each other to further augment density and bandwidth similar to the Intel 3D XPoint [29]. A target

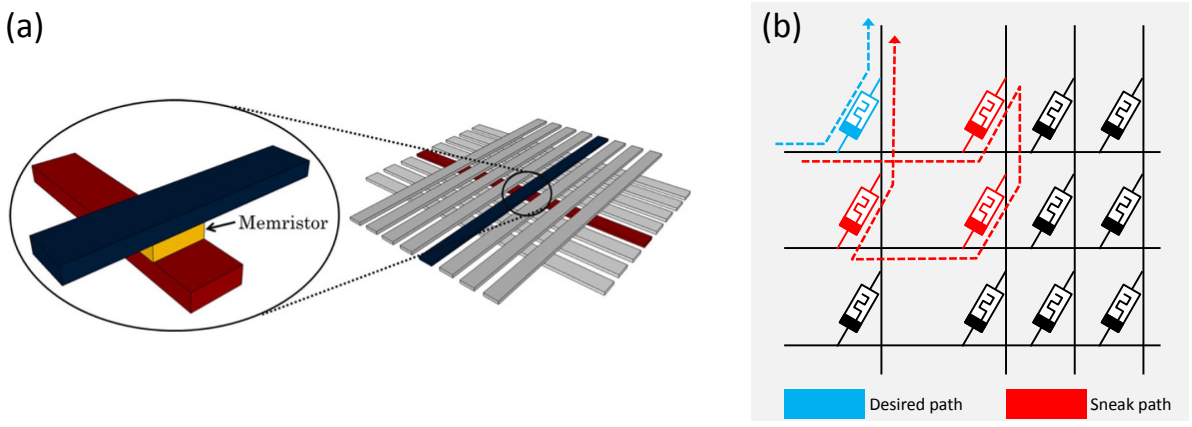


Figure 1.4: Memristive crossbar array: (a) Structure, (b) Sneak path phenomenon

memristive device located at a certain crosspoint is usually configured by biasing its corresponding row (wordline) and column (bitline). The applied bias affects the state of

the selected device by shifting its resistance to the LRS or HRS depending on the applied polarity. However, a typical passive crossbar memory array suffers from large amount of leakage current that propagate through unselected cells as shown in Fig. 1.4(b). These paths, which are known as current sneak paths, lead to several downsides:

i) During the write phase, the unselected cells are exposed to a voltage drop causing a drift in their internal state (state drift phenomena [30]).

ii) During the reading phase, these paths causes a degradation in the sense margin leading to false estimation of the state of the sensed device.

iii) More power is dissipated in the crossbar.

Many solutions have been proposed in the literature to overcome this problem and are classified mainly into 4 classes:

Selectors

Selectors such as transistors or diodes [31][32][15] are connected in series with the memristive cells inside the crossbar. The transistors ensure gating of unselected cells leading to negligible leakage current; whereas, the diodes are used as rectifying devices to isolate the cells being written or read. Fig. 1.5(a) depicts the schematic as well as the structure of transistor type selectors, while Fig. 1.5(b) demonstrates that of diode type selectors.

Bias schemes

Voltages applied to non-accessed wordlines and bitlines are set to fixed values which are different than those applied to accessed ones; examples are the use of 1/3 or 1/2 bias schemes [33] which are demonstrated in Fig. 1.6(a) and Fig. 1.6(b) respectively. Other methods attempts to bypass sneak paths problem using a multistage reading algorithm as the one introduced by the HP Labs team in [34].

Switching-device modifications

In this technique, memristive devices are modified; examples are serially connecting two memristors with opposite polarities, resulting into a "complementary resistive switch-CRS" [35], or employing highly nonlinear memristors to minimize undesired current paths [36].

Information theoretical techniques

Sneak paths are modeled as random error source. These techniques attempts to change the distribution of zeros and ones inside the crossbar array or use encoding schemes and error correcting codes to guarantee sneak-path free readout [37][38].

This is due the fact that sneak paths level is highly dependent on the instantaneous data stored in the crossbar array.

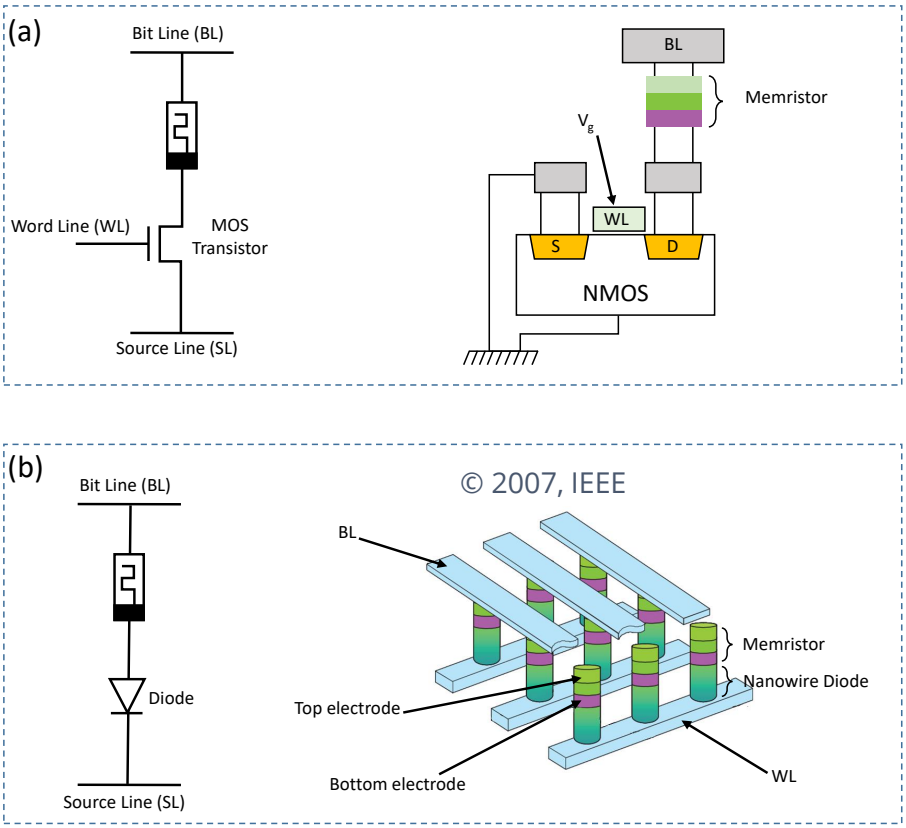


Figure 1.5: Selector devices: (a) Transistor type, (b) Diode type [31].

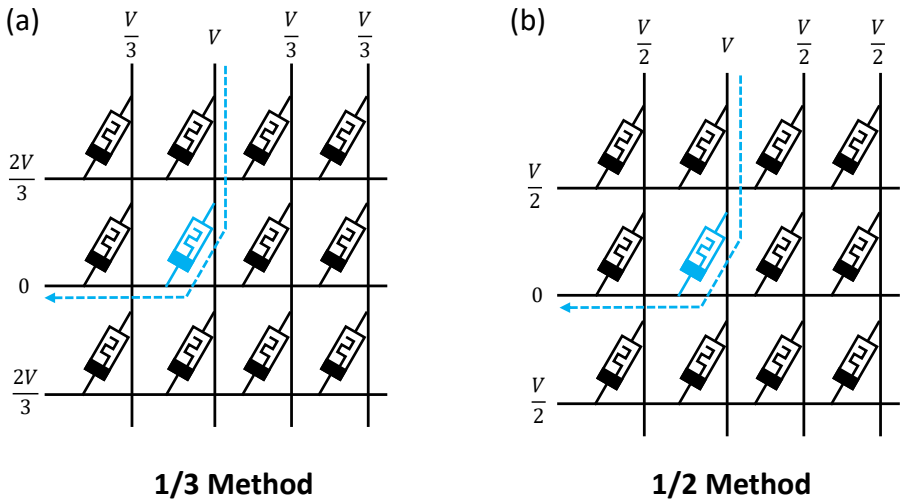


Figure 1.6: Bias schemes: (a) One-third select method, (b) Half select method.

The above solutions contribute to the reduction or removal of the current sneak-paths,

though they still suffer from several limitations. For instance, transistors which are the commonly used selectors, solve sneak paths problem. However, it expands the cell size and makes it hard to achieve cross-point cell size of $4F^2$ and consequently reduces the integration-density. Bias schemes and device modifications normally require complex reading schemes, thus impact both area and performance. Information theoretical techniques are still at their preliminary stages and should be further extended, as their is a need for constructing simple codes which takes complexity into consideration.

1.3 Memristors for reconfigurable interconnects

Before the discovery of the HP memristor (ReRAM device), memristive devices such as MRAM and PcRAM was only employed as storage elements. Since the HP Labs presented the T_iO_2 memristor in 2008, rapid progress on the fabrication of high-quality memristors has been achieved in the past few years [23][39][40]. These memristors are characterized by huge gap between their Low and High resistance state ($R_{OFF}/R_{ON} \simeq 10^6$). This evolution has received significant attention and allowed for employing these devices in new fields. Non-volatile programmable switches are considered as direct applications of these memristors. Memristor-based programmable switches are introduced to achieve efficient and low cost flexibility in application specific integrated circuit (ASIC) designs. Such flexible designs are reconfigured allowing for efficient reuse of resources for different application need. Moreover, memristors are explored by FPGA designers. It is reported that the routing resources in an FPGA (Fig. 1.7) including switch blocks (SBs), connection blocks (CBs) and interconnects can account for up to 70% of the total area, delay, and consumed power [41]. Thus, the improvement of these programmable routing elements in FPGAs is of importance for research and development. In this context, memristors have been presented in the literature [42][43][44][45] as potential alternative to the conventional SRAM-based programmable interconnects in FPGAs.

Generally, memristors are employed to implement non-volatile routing switches by logically connecting/disconnecting wires during configuration mode. Non-volatile routing preserves the configuration information stored on-chip while powered off, allowing the devices to immediately run when it is powered up. On the other hand, leakage power is minimized during stand-by mode, leading the system to work at extremely low-power. Moreover, the opportunity to fabricate memristors on the top of transistor layer in the same die brings significant reduction of the overall area of the system design. These features are of high interest when targeting multi-mode ASIC designs as well as high-end FPGAs. For instance, authors of [42] have proposed an FPGA architecture with

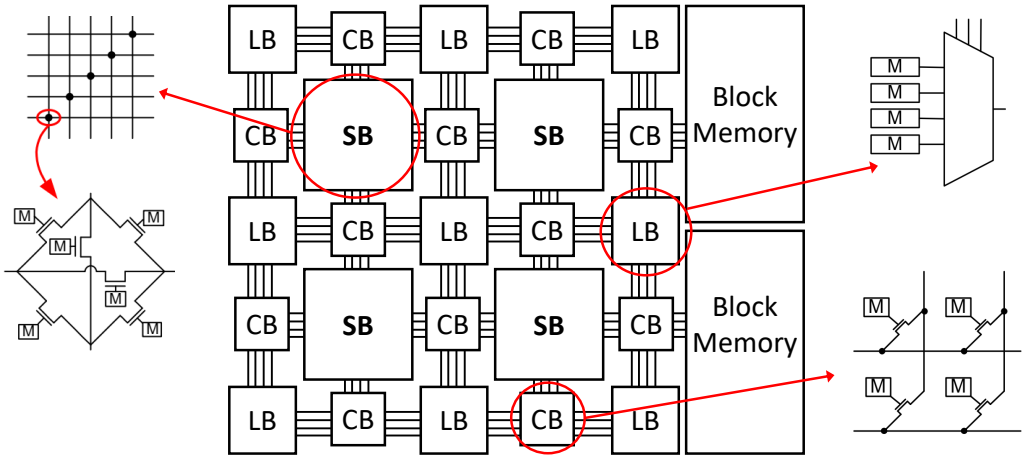


Figure 1.7: Conventional FPGA architecture

memristor-based reconfiguration (mrFPGA). The programmable interconnects of mrFPGA use only memristors and metal wires so that the interconnects can be fabricated over logic blocks, resulting in significant reduction of overall area and interconnect delay. As demonstrated in Fig. 1.8, the area of mrFPGA has been reduced to the total area of the logic blocks only, which takes 10% to 20% of the conventional FPGA area. Several studies [46][47] have demonstrated the use memristive devices as programmable

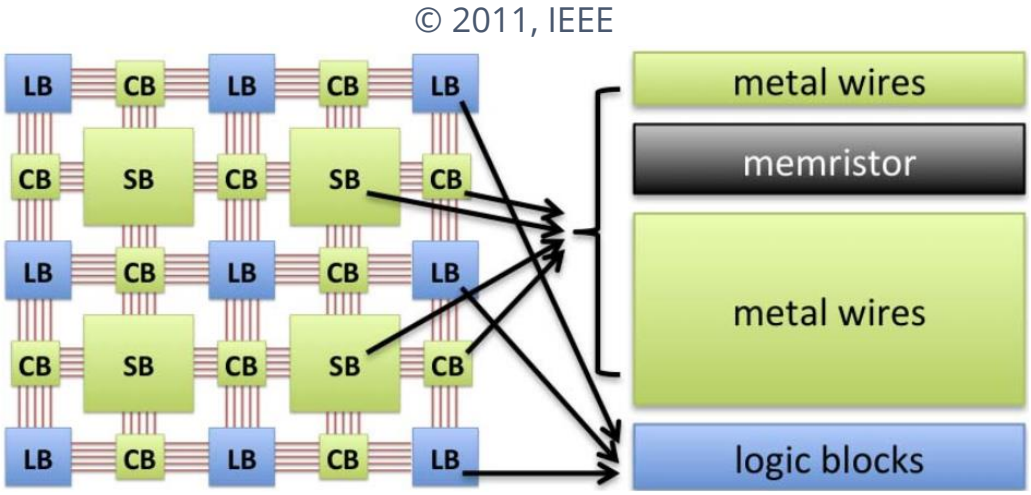


Figure 1.8: Memristor based-programmable routing structure for FPGA proposed in [42]

switches instead of SRAM-based pass transistors in conventional FPGAs. Fig. 1.9(b) shows the typical CMOS routing switch. It consists of a pass transistor controlled by a 6-transistors-SRAM cell. The first memristive routing switch has been proposed in [47]. The switch is made of two programming transistors and one memristor (2T1R). During the configuration stage, programming voltages V_{p1} and V_{p2} are sent through the transistors to set the resistance of the memristor to R_{ON} or R_{OFF}. If it is set to R_{ON}, nodes

"A" and "B" are assumed connected during the operational stage. Otherwise, "A" and "B" are disconnected. During operation stage, pass transistors are isolate programming voltages. The operational voltage $|V_{AB}|$ should be lower than the switching threshold of the memristor to prevent any drift in the state of the switch. This type of memristive switches is simple and can be implemented as junctions in a crossbar structure.

Another memristive switch has been presented in [47]. The switch has the 2T2R structure as shown in Fig. 1.9(c). One programming transistor with a programming voltage V_p is used to configure two complementary memristive devices that are supplied with V_{dd} and GND. Note that $|V_{dd} - V_P|$ is used to program the upper memristor, while V_P is used for the lower one. That is, when one memristor is in the LRS and the other should be in the HRS. During the operational stage, the two junctions work as a voltage divider, with this ratio determining the pass transistor gate voltage. The ratio needs to be large enough to ensure proper functioning.

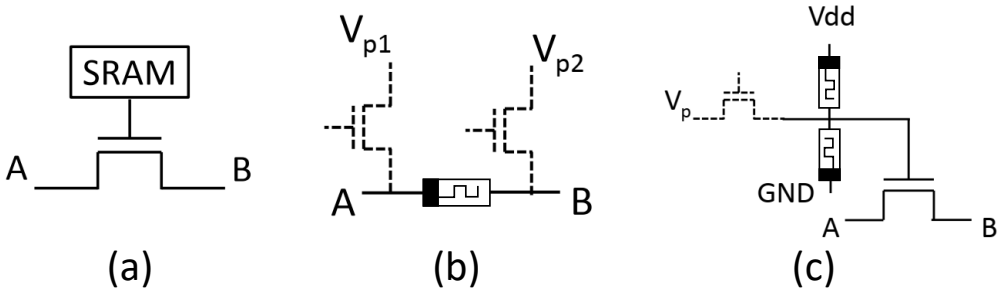


Figure 1.9: Memristive switches: (a) 7T SRAM routing switch, (b) 2T1R routing switch, (c) 2T2R routing switch

On the other hand, memristive switches are also explored as selectors for memory cells. The authors in [48] have presented a 4-memristor switch design which can replace the transistors in the peripheral circuits of NVMs. The proposed design of the switch is used to implement a 4M1M (four memristors per memristor) memory cell. Significant improvements have been demonstrated over the memory cell in [49] at the level of read latency, power dissipation and consecutive correct read number.

1.4 Memristors for logic design

The fast decline of Moore's law is paving the way to explore new set of emerging technology devices [50], as it is difficult to overcome the various physical limitations of the traditional CMOS technology [51]. In this context, the nano-scale size memristor was introduced as a possible alternative candidate as it offers a lot of advantageous features including the capability of executing Boolean logic [52][53][54] in addition to the storage role that we

have discussed earlier. The presence of these two attributes combined has given a great impetus to explore new innovative circuits and systems based on memristors. For instance, one of the main challenges of modern computers nowadays is the memory wall problem which is originated from the mismatch in the performance of processor and memory. There has been a continuous effort to move processing cores closer to where data resides to address the memory wall problem. A memristor-based logic can integrate processing and storage role (in-memory computing), an attribute which can be a promising solution to scale the memory wall. A memristive logic design style is able to compute a certain primitive logic such as AND, OR, NOR, etc. Based on these simple operations, complex arithmetic functions can be executed. Section 1.4.1 is a survey on the different memristive logic design styles introduced in the literature, while section 1.4.2 presents a framework for evaluation of these design styles.

1.4.1 Memristor-based logic design styles

Several memristor-based logic design styles have emerged in the literature. Each has its own capabilities and thus is adapted for a specific type of applications. For instance, Memristor Ratioed Logic (MRL), which has been proposed in [52], integrates memristors and CMOS transistors to implement combinational functional blocks. These blocks are relatively dense compared to those implemented with pure CMOS transistors. The Memristor Threshold Logic has been studied in [55]. The threshold gate has been presented in the domain of bio-inspired information processing. It is considered simple, but still in preliminary stages of fabrication. The Material Implication (IMPLY) [56] and the Memristor Aided logic (MAGIC) [53] are intended for in-memory computing. In these design styles, a memristor serves as a memory element as well as a part of a computational gate inside the memory. Memristors-As-Drivers (MAD) gate has been presented in [57]. MAD has been introduced to overcome the long delays of the IMPLY operations as well as signal degradation and buffering issues in MRL. However, each MAD gate requires a complex driving circuitry; thus, considered not suitable for integration inside a memristive memory. A memristor-based Majority gate (MAJ) has been proposed in [58]. The authors demonstrated that a single memristor is capable of performing a 3-variable majority function. By the aid of additional inversion function (INV), A Boolean expression is represented using majority-inverter graph (MIG). MIGs are then realized sequentially in conventional memristive crossbar arrays. The bipolar resistive switches (BRS) as well as the complementary resistive switches (CRS) logic have been presented in [59]. BRS and CRS behave absolutely identical in terms of logic operation. BRS/CRS operations are special cases of MAJ.

1.4.1.1 IMPLY – Material Implication

In IMPLY, the states of the memristor R_{ON} and R_{OFF} represent logic '0' and '1' respectively. Fig. 1.10(a) shows the implementation of IMPLY gate. It consists of two memristors holding logical states p and q and a reference resistor R_G ($R_{ON} < R_G < R_{OFF}$). The gate is controlled by three voltage levels V_{SET}, V_{COND} and V_{CLEAR}. The initial states p and q represent the inputs to the gate. To perform IMPLY, voltage levels V_{SET} and V_{COND} ($|V_{COND}| < |V_{SET}|$) are applied simultaneously to the ports of the gate. The interaction of the two memristors under the aforementioned voltage leads to the execution of IMPLY according to the truth table shown in Fig. 1.10(a). The value $\overline{p} + q$ is written as an output into memristor of state q. However, a separate mechanism to read the result of the computation and to perform the control the voltages is required. This mechanisms are usually based on the standard CMOS transistors.

The structure of IMPLY allows its integration in a standard memristive crossbar array as illustrated in Fig. 1.10(b), where p and q can represent the states of two memristors located in the same column (or row) within the crossbar. The voltages V_{SET} and V_{COND} are applied to the word lines, while the bit line is connected to a grounded resistor R_G. In fact, IMPLY operation is destructive to the values of p and q. Hence, if the data represented by p is significant, a copy should be performed to reserve a safe version.

This approach is an example of combining memory and logic and thus considered suitable for in-memory computing [56] [60] [61].

1.4.1.2 MAGIC – Memristor Aided Logic

MAGIC is another type of memristive logic [53] where the resistance values represent the logic states. Unlike IMPLY, this logic family makes use of separate memristors to store the input bits (In_1 and In_2) and an additional memristor is used to store the output bit (Out). All basic gates (NOT, AND ,OR, NOR, NAND) can be implemented using the MAGIC design style as illustrated in Fig. 1.11(a). A MAGIC gate operation requires two sequential steps: (i) Initialize the output memristor to either '0' or '1' state. Initialization to '0' corresponds to non-inverting gates like AND/OR, while for inverting gates (NOT/NOR/NAND), it is initialized to '1'. (ii) Apply a suitable voltage V_0 the input port of the gate. The result is written simultaneously into Out.

Though all primitive gates can be implemented using MAGIC design style, only NOR and NOT can be integrated into crossbar arrays. Fig. 1.11(b) presents the implementation of MAGIC-NOR inside a crossbar. Larger Boolean functions can be realized by representing them in terms of NOR and NOT netlists, which are then performed sequentially inside the crossbar array.

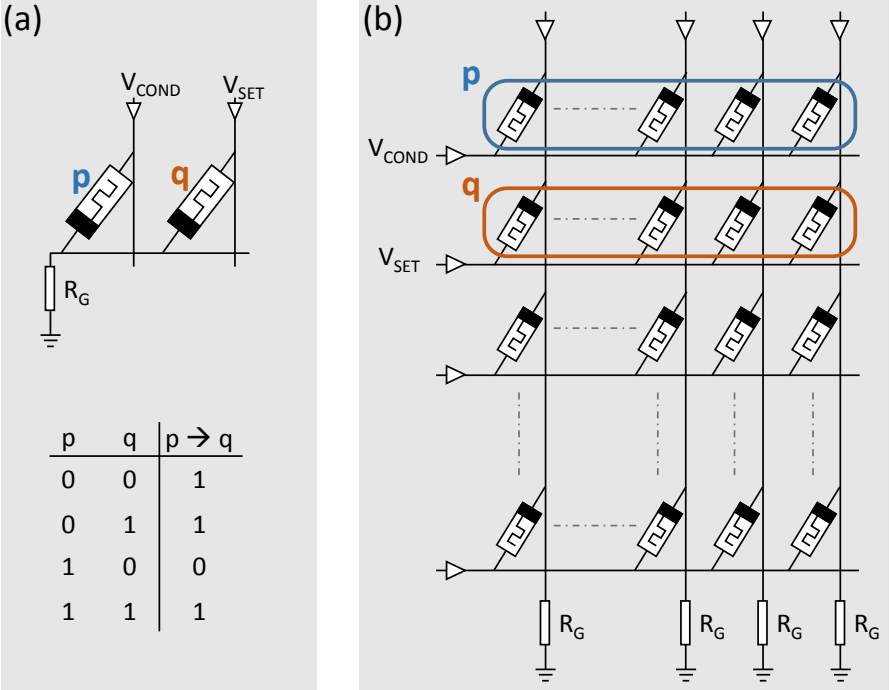


Figure 1.10: IMPLY gate: (a) schematic of a memristor-based IMPLY gate and its corresponding truth table, (b) performing IMPLY in a crossbar array

MAGIC design style suffers from three main drawbacks:

- The previously stored bits In_1 and In_2 are lost,
- The choice of the driving voltage V_0 is highly related to the values of R_{OFF} and R_{ON}
- The written output may be subjected to state drift which may induce errors [53].

1.4.1.3 MRL – Memristor Ratioed Logic

The memristor ratioed logic (MRL) [52] is a typical hybrid CMOS-memristor logic design where the programmable resistance of memristors is exploited in the computation of the Boolean AND and OR functions. MRL opts voltage as the state variable similar to CMOS-based devices. Hence, the computation is accomplished in one single step. This criterion eliminates the drawbacks of the sequential process of IMPLY logic devices. Fig. 1.12 depicts the structures of the MRL AND, NAND, OR, and NOR gates. Both OR and AND gates consist of two anti-serial memristors (i.e. connected serially with opposite polarities); whereas, for NOR and NAND a CMOS inverter is added at the output. Both MRL AND and OR gates react similarly when identical values are set to their input ports (either both inputs are set to logic '1' or '0'). In this case, no current flows through the anti-serial memristors leading to the transfer of the input voltage to

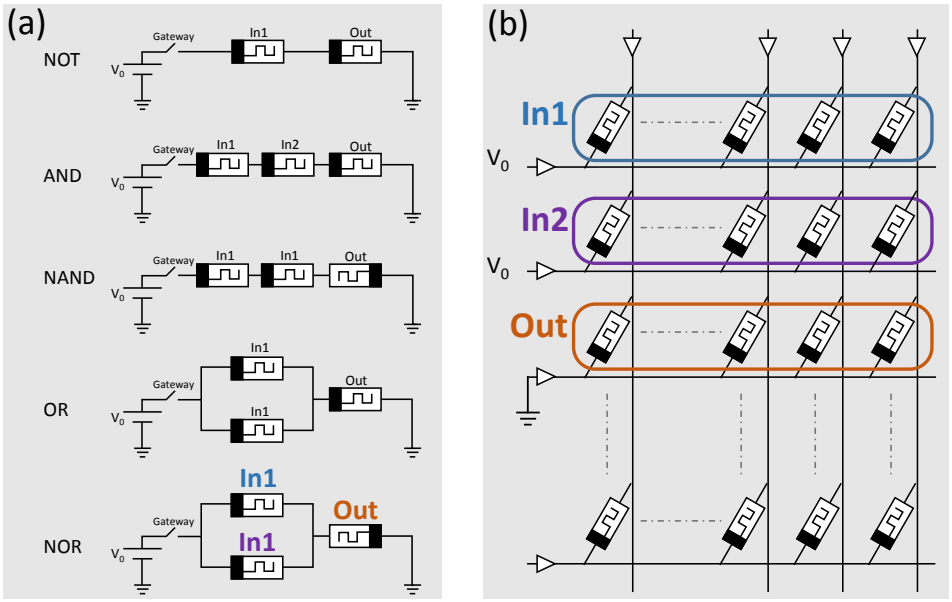


Figure 1.11: Structure of MAGIC NOR gate [53]

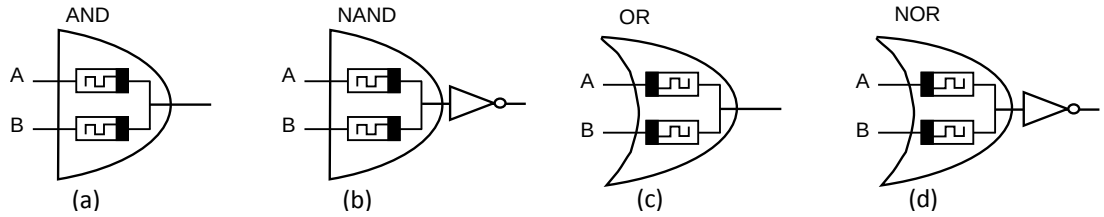


Figure 1.12: Schematic of an MRL (a) AND, (b) NAND, (c) OR, (d) NOR

the output. In the case where different values are set to the input ports (i.e. first port is set to '0' and the second port is set to '1' or vise versa), a current flows from the port with higher potential (logic '1') to the port with lower potential (logic '0'). The resulting potential difference changes the internal state of both memristors in an opposite manner. One memristor tends to attain the R_{ON} state while the other tends to attain the R_{OFF} state. In addition, the connected memristors forms the well-known voltage divider circuit. Assuming $R_{OFF} >> R_{ON}$, (1.7) and (1.8) present the obtained output values V_{out} of MRL OR and AND gates respectively [52].

$$V_{out,OR} = \left(\frac{R_{OFF}}{R_{OFF} + R_{ON}}\right) \times V_{CC} \quad \approx V_{CC} \tag{1.7}$$

$$V_{out,AND} = \left(\frac{R_{ON}}{R_{ON} + R_{OFF}}\right) \times V_{CC} \quad \approx 0 \tag{1.8}$$

Note that the output voltage V_{out} converges to the higher potential (logic '1') in the MRL AND gate and to the lower potential (logic '0') in the MRL OR gate. Fig. 1.13 illustrates the logical operations of the MRL AND gate corresponding to all input combinations. However, cascading several MRL gates leads to a floating output (between logic '0' and

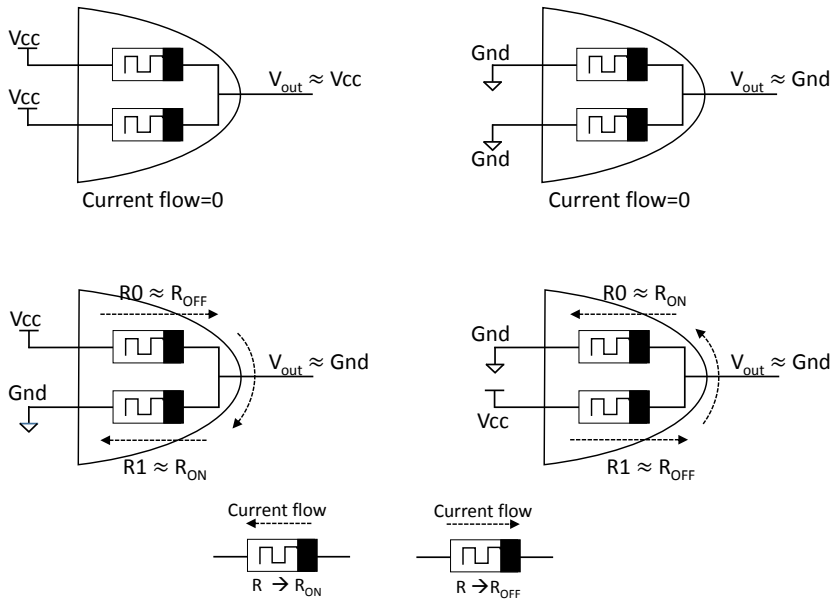


Figure 1.13: Logical operations performed with MRL AND gate

logic '1') due to voltage degradation. Since memristors are passive devices, they cannot amplify signals. Therefore, CMOS inverters can be used as buffers after several stages to restore the attained logical state [52].

Several recent research works presented in the literature exploit the use of MRL to design basic building blocks. In [52], a design dedicated for the universal full adder circuit has been proposed using MRL gates with the aid of CMOS inverters instead of pure CMOS based gates. Although MRL are expected to realize more compact logic design circuits, the switching time of the anti-serial memristors in MRL is substantially slower than the delay of CMOS logic gates (nanoseconds versus picoseconds), MRL gates are still much slower than CMOS logic.

1.4.1.4 MAD – Memristors-As-Drivers

MAD gates, or Memristors-As-Drivers gates has been proposed in [57]. MAD was introduced to overcome the long delays of the IMPLY operations, signal degradation and buffering issues in MRL. Fig. 1.14 presents the implementation of MAD gate which can execute AND operation. The gate is constituted of two input memristors A and B that are connected in series and a separate output memristor AND. Each of the three memristors is attached to a single pull-down resistor R_g. As in IMPLY approach, V_{cond} and V_{set} are used as driving voltages. A controlled switch (can be a MOSFET) is connected in series to the output memristor. It is used to pass or isolate the driving voltage V_{set}. This switch is gated with the voltage sensed at node V_t.

In order to perform the AND operation, the two input operands are assumed to be

initially preloaded into memristors A and B. The output memristor is initialized by logical '0'. V_{cond} and V_{set} are applied simultaneously to the ports of the gate. If the sensed voltage at node V_t is greater than the threshold voltage (V_{apply}) of the switch connected to the output memristor, the switch will close and the voltage V_{set} turns the output memristor into logical '1', otherwise the state of the memristor remains at logical '0'. In fact, the threshold of the switch plays the main role in defining the function of the gate. For the case of MAD-AND, the gate is configured in such a way that the sensed voltage at node V_t exceeds the value V_{apply} only when the two input memristors are in the low resistance state (i.e. logical '1'), resulting in the AND operation.

Other Boolean operations such as OR, NOT, COPY and XOR can be also realized using MAD approach and are presented in Fig. 1.15. In fact, the main challenge in achieving these circuits is to choose an appropriate value of V_{apply} corresponding for each operation. Different value of V_{apply} requires different MOSFET (and pull down resistor) specifications. However, the hardware cannot be easily reconfigured dynamically to alter this value. Moreover, when targeting in-memory computing applications, these gates requires complex driving circuitry which limits its utility in standard crossbar arrays.

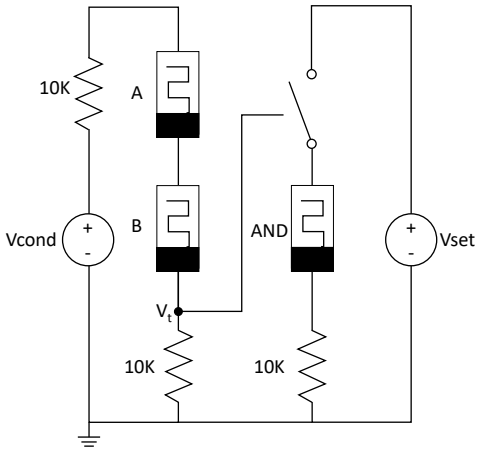


Figure 1.14: MAD AND gate implementation

1.4.1.5 MTL – Memristor Threshold Logic

MTL gate has been proposed in [62]. The gate was presented in the domain of bio-inspired information processing. A generic threshold gate with N-input transfer function is defined as:

$$Y = \begin{cases} 0, & \text{if } \sum_{i=1}^{N} w_i x_i < T \\ 1, & \text{if } \sum_{i=1}^{N} w_i x_i \geq T \end{cases} \tag{1.9}$$

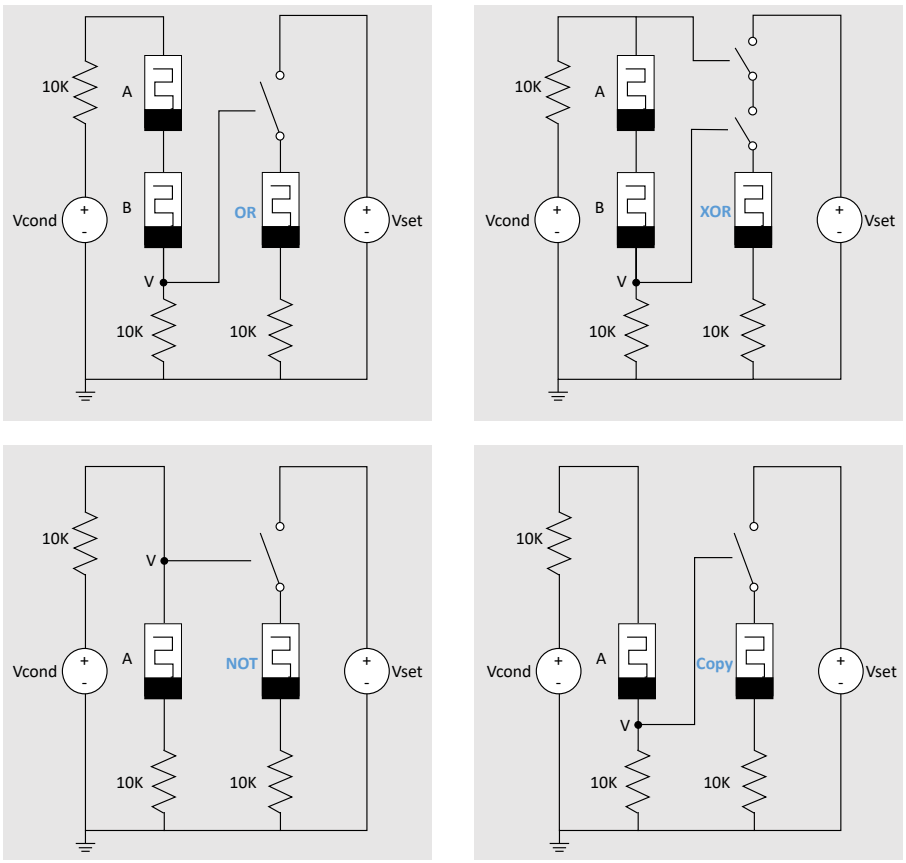


Figure 1.15: MAD OR, XOR, NOT, and Copy gates implementation

where x_i is a Boolean input variable, w_i is an integer weight of the corresponding i-th input, T is a specified threshold and Y is the overall weighted sum of the inputs. Fig. 1.16 shows the implementation of a 3-input MTL gate. The gate uses the configurable conductance of memristors to represent weights at the inputs. The magnitude of the currents flowing through the respective memristors depends on the value of memristance (weight) at these inputs. By the aid of PMOS and NMOS current mirrors, these currents are summed and then compared with a reference current I_{ref} (threshold). If the total current sum exceeds I_{ref}, the output of the MTL gate is the high supply voltage otherwise it is low.

Threshold gates are considered as simple and they do not exhibit the serialization issues of the IMPLY operation or the signal degradation or CMOS issues of the hybrid-CMOS approach. However, threshold gates are very sensitive to state drift which can be a potential problem during operation [62]. MTL is expected to have better performance with memristors which do not drift at low voltage levels.

1.4.1.6 MAJ – Memristor-based Majority

A memristor based majority gate (MAJ), which is proposed in [63] is based on a single memristor operation. In fact, a basic majority function takes on the same value as the

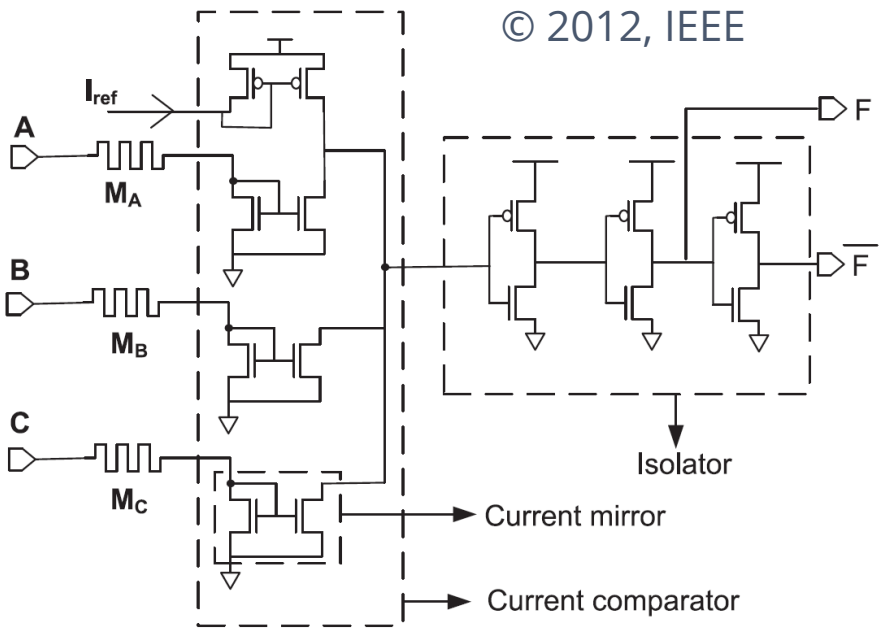


Figure 1.16: A 3-input MTL gate which uses the memristors as weights and I_{ref} as the threshold [62]

majority of its inputs, this is also known as voting logic [58]. A 3-input majority is given as follows:

$$M_3(A,B,C) = AB + AC + BC \tag{1.10}$$

A magority function with the aid of inversion (INV), has been proven to be functionally complete [58]. In other words, magority is capable of constructing other logic operations (Majority inverter graphs (MIGs)) [58][64]. MIGs has been initially introduced as an efficient optimization for Boolean functions that are represented by AND, OR and NOT [58]. Fig. 1.17(a) illustrates how a majority operation is intrinsically realized using a single memristor. Assuming that a memristor initially stores the logical value Z, the top and bottom electrodes are supplied by the logical states A and B respectively. The obtained internal state value is Z', which is expressed in (1.11).

$$Z' = ZA + Z\overline{B} + A\overline{B} = M_3(A,\overline{B},Z) \tag{1.11}$$

As shown in (1.11), Z' follows the 3-variable majority relation between A, $\overline{B}$ and Z. In the same way, MAJ operations are performed to a crossbar array as shown in Fig. 1.17(b). A and B are applied to a memristor through its corresponding row and column. The state of the target memristor changes according to the majority relation. Generally, a Boolean expression is represented using majority-inverter graphs (MIGs), which are then realized sequentially in conventional memristive crossbar arrays.

However, it is worth to mention that the concept of MAJ was introduced earlier in [59]. Authors of [59] have presented the computation with special cases of MAJ that are

realized in bipolar resistive switches (BRS) as well as complementary resistive switches (CRS). More details are discussed in Section 4.3.2.

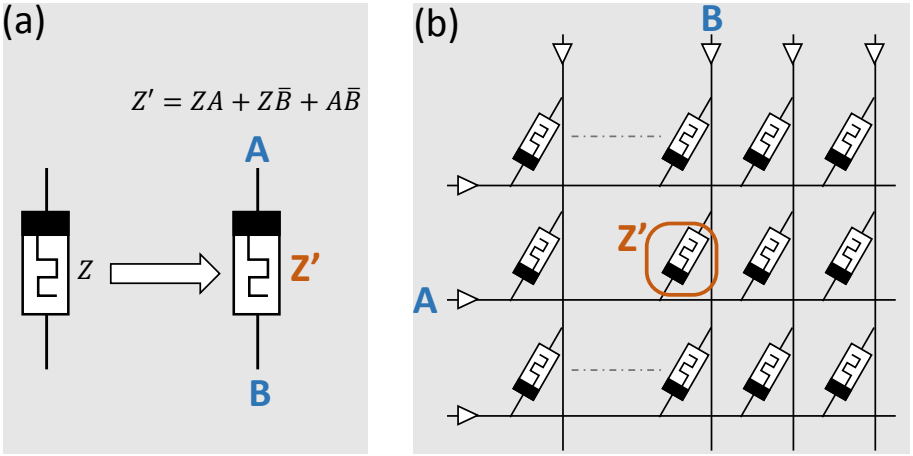


Figure 1.17: Majority gate: (a) Structure, (b) Realization in a crossbar array

1.4.2 Roadmap for evaluation of memristive logic

This section establishes a roadmap for classifying memristive logic families and discusses the fundamental differences of these logic families as compared to conventional CMOS logic systems. Table 1.2 includes some preliminary evaluations for the memristive design styles which are discussed above.

1.4.2.1 Statefulness

A memristive logic has fundamental differences when compared to the conventional CMOS logic. For instance, CMOS logic is based on one state variable (i.e., voltage) to represent data. Thus, input and output data, throughout all intermediate stages are represented only as voltage. This flow is disrupted, in memristive logic since the non-volatile resistance of memristors acts as a primary state variable throughout the execution of the logic. The interaction between resistive states of the memristors and voltages are used to produce different dynamic behaviors that enable the circuit to execute the desired logic operation. This interesting property in memristive logic adds new capabilities and design rules that was not present in traditional von Neumann architectures.

This fundamental difference has manifested in two different types of memristive logic families. The first type involves stateful logic [65], where the resistance is the only state variable for representing inputs, output and intermediate results of computation. In the second type, voltage and resistance are both used for representing variables through out the computation. We denote this type as partially stateful.

The degree of statefulness of a given memristive logic defines its uses and specifically decribes how much a given logic family brings computing closer to storage. *The more a given memristive logic is close to statefulness, the more it is eligible for computing inside memory.* For instance, IMPLY and MAGIC are examples of stateful memristive logic families where inputs and outputs are represented by resistance. Crossbar memory arrays based on IMPLY and MAGIC operations are capable of processing data purely inside the memory and without the need to export them to the outside. Other logic families represents all the inputs as voltage while the output is stored as resistance (e.g., MAJ). In MRL, inputs and outputs are both represented as voltage, which is similar to the conventional CMOS logic, thus can't be employed for in-memory computing.

1.4.2.2 Flexibility

Flexibility is another evaluation metrics of memristive logic design style. A given logic design style is said to be flexible if the same computing elements are used to perform variety of operations [66]. A CMOS logic design is not considered flexible since it has specific functionality which is determined prior to the fabrication process (e.g. ASIC design). IMPLY, MAGIC and MAJ are flexible since the functionality of these design styles can be dynamically chosen during runtime. On the other hand, MRL and MAD are non-flexible since each design yield specific logic operations. Different logic operations requires fundamental change in the circuit design.

1.4.2.3 Crossbar compatibility

The compatibility of memristive logic families with crossbar memory array can also provide a way for evaluation. A memristive logic is said to be compatible with crossbar array when it enables logic computation with the minimal modifications in the conventional topology of the crossbar in addition to the peripheral circuitry. Normally, adding a selector (e.g. MOSFET) to the memristive cell doesn't disturb the crossbar structure. As an example, MAD requires fundamental modifications in the peripheral drivers of the crossbar in order to support logic operations.

1.4.2.4 Reliability

Due to the fact that memristive devices are passive elements, a memristive logic suffers from several reliability issues as compared to conventional CMOS logic. At some stages of computation, a memristive logic suffers form a degradation in the state variable (voltage/resistance), which in turn induces errors and leads to loosing eligibility. In contrast, CMOS logic systems are based on active devices that do not exhibit these issues. For

instance, output memristor in IMPLY, as reported in [67], allows for partial switching leading to an intermediate resistance state. In contrast, MAJ does not exhibit this phenomenon. Another example is MRL where the voltage state shows degradation after several cascading stages. State degradation is usually managed by CMOS-based state refreshing sources. However, this adds overheads (e.g. Area, latency) that should be reconsidered while evaluating the overall performance of such logic systems.

Table 1.2: Preliminary evaluations of memristive logic design styles

Logic family	**Degree of statefulness**	**Flexibility**	**Crossbar compatibility**	**Reliability**
IMPLY	100%	Yes	High	Medium
MAGIC	100%	Yes	High	Medium
MRL	0%	No	Medium	Medium
MAD	100%	No	Low	High
MTL	–	Yes	High	Low
MAJ	33.3%	Yes	High	High

1.5 Summary

In this chapter we have provided a scientific background related to the presented PhD contributions in subsequent chapters. In the beginning of this chapter, the fundamentals of memristor basic operation have been presented. Then, a review on the uses of these devices in three main areas (storage, interconnects and logic design) have been provided. Memristor is introduced first as an emerging NVM element. The main advantages over traditional memory elements (DRAM, SRAM, NAND flash, etc.) have been highlighted. Later, the second potential use of memristor at the level of interconnect has been discussed. Its ability to allow smooth reconfiguration and flexibility has been illustrated with some examples of pre-existing approaches in ASIC as well as FPGA applications. At last, memristor has been introduced as an enabler for the implementation of new generation of logic designs, allowing for new computing paradigms. Various available memristive logic design styles have been discussed. At the end, a roadmap for evaluating these design styles has been proposed.

In summary, memristive devices can prompt new alternative computing architectures for emerging applications in addition to their use as new generations of non-volatile memories. However, many challenges remain open for research and investigation of their efficient use at the level of interconnect, logic design and in-memory computing.

Chapter 2

Memristor Based Reconfigurable FFT Architecture

The design of flexible architectures, which can adapt dynamically to application needs, brings great advantages in terms of energy efficiency and performance. Flexibility is particularly required in digital communication and multimedia applications where new standards and multiple services are continuously emerging. For instance, a well-suited physical layer for emerging and future wireless mobile communications should allow several of its communication parameters to be tuned according to channel conditions instead of being fixed for the worst-case communication scenario [68]. In order to cope with these requirements, flexibility should be enabled at several levels including processing, interconnect, and memory. However, current technologies are still inefficient for such highly adaptive systems due to the cost of reconfiguration in terms of area, delay, and power consumption [41][42].

In this context, the integration of memristors at the interconnect level to enable flexible and efficient configuration of digital architectures is explored. To that end, the Fast Fourier Transform (FFT) has been chosen as application case study since it is used widely in the fiels of digital signal processing and telecommunications. Particularly, FFT is considered recently as an effective tool for spectrum enhancement of orthogonal frequency-division multiplexing (OFDM)-based waveform, which is a central element in the fifth generation new radio (5G-NR) developments [69]. Furthermore, the size of FFT block can vary according to the system parameters and the communication channel conditions. Thus, there is a real need for highly flexible FFT implementations that support multiple configurations.

In this chapter, a memristor-based reconfigurable FFT architecture (mrFFT) is proposed. The architecture can be employed in a reuse and systematic way based on a

proposed reconfigurable butterfly (RBF) that can be configured to support radix-2 and radix-3 kernels. Flexibility is achieved by inserting memristors at the interconnect level leading to programmable memristive nodes.

The rest of the chapter is organized as follows. Section 2.1 introduces the well known FFT algorithm. Thereafter, Section 2.2 discusses classical and reconfigurable pipelined FFT designs. Section 2.3 presents our proposed memristor-based reconfigurable FFT architecture. Comparison of the mrFFT architecture with a recent flexible implementation is conducted in Section 2.4. Section 2.5 discusses the encountered issues related to the mixed-signal simulations. Finally, Section 2.6 concludes the chapter.

2.1 Fast Fourier Transform (FFT)

The Fast Fourier Transform is a reduced form of the Discrete Fourier Transform (DFT). FFT has been developed by Cooley-Tukey [70] as an efficient method to compute DFT. The N-point DFT of an input data sequence $x[n]$ is defined as:

$$X[k] = \sum_{n=1}^{N-1} x[n] e^{-j\frac{2\Pi}{N}nk} \tag{2.1}$$

where N is the size of the input data sequence. The term $e^{-j\frac{2\Pi}{N}nk}$ is the so called twiddle factor (TF) and is usually represented as W_N^{nk} [71]. Computing DFT as represented in (2.1) requires large number of operations $O(N^2)$. However, due to the presence of symmetries in the calculations, significant simplification can be performed leading to decreased complexity. The FFT leverages this symmetry by dividing the DFT into smaller size ones. The results obtained from the shorter DFTs are then combined appropriately. This can reduce the complexity to $O(N \log N)$.

In fact, FFTs are most often designed based on radix-2 kernel. Hence, the size N of the FFT is chosen to be power of 2 ($N = 2^p$ where p is an integer value). In a radix-2 FFT, the DFT is broken down into several DFTs each of size 2 and referred to as butterfly (BF). Fig. 2.1(a) presents the typical flow graph of radix-2 BF. A BF simply takes two inputs (x_0 and x_1) and gives two outputs (X_0 and X_1) according to the following formula

$$\begin{aligned} X_0 &= x_0 + x_1 \\ X_1 &= x_0 - x_1 \end{aligned} \tag{2.2}$$

The data flow graph (DFG) of a 16-point radix-2 FFT is shown in Fig. 2.2. Several approaches [72, 73, 74] have explored the possibility of moving beyond the standard radix-2 in order to support input sizes that are not power of 2. The authors of [73] have

developed the algorithm and the flow graph of radix-3 and radix-5 FFT kernels. Their corresponding flow graphs are shown in Fig. 2.1(b) and Fig. 2.1(c) respectively.

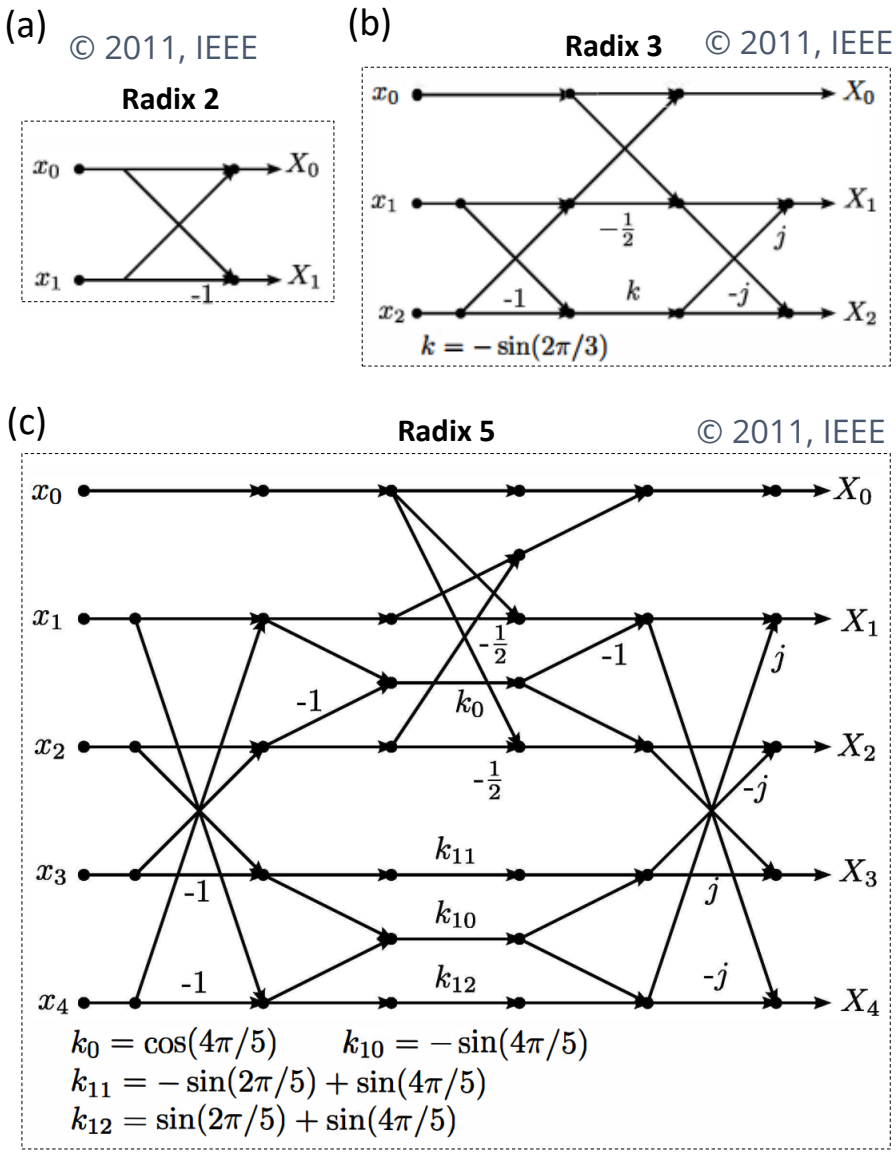


Figure 2.1: Flow graph of (a) radix-2 BF, (b) radix-3 BF and (c) radix-5 BF [73].

2.2 Pipelined FFT architecture designs

2.2.1 Classical FFT architectures

It is possible to implement a fully-parallel FFT flow graph (e.g. 16-point DFG in Fig. 2.2) directly in hardware. However, for large FFTs it may be infeasible, since the area usage would be too large [73]. Hence, FFT implementations are often folded. The throughput

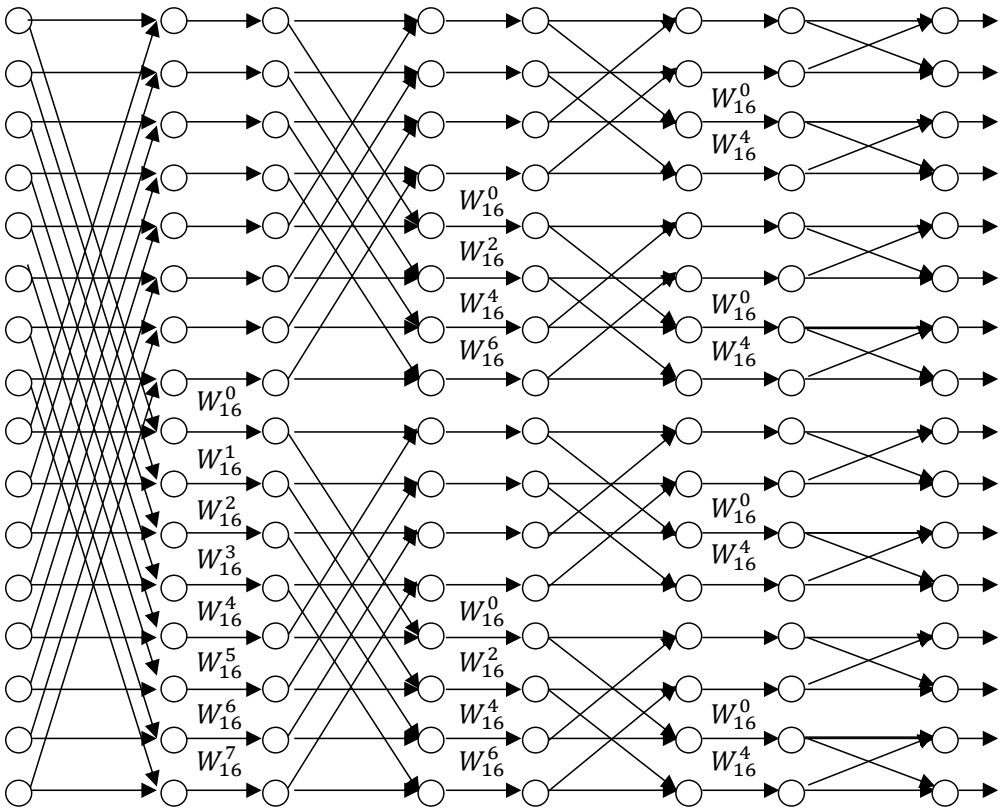


Figure 2.2: Data flow graph of 16-point radix-2 FFT

is traded for a smaller area. A common folded structure is the pipeline structure [71]. Particularly, the single-path delay feedback (SDF) [71] structure is considered a popular design approach to realize a pipelined FFT. Fig. 2.3 presents the architecture of SDF-FFT, which is based on radix-2 kernel. The architecture is a concatenation of several stages. Each pipeline stage includes arithmetic elements (adders) for the realization of the butterfly, First-In-First-Out (FIFO) memory and a complex multiplier. The FIFO at each pipeline stage has a predetermined length, which varies from one stage to another. The role of the FIFOs is to ensure the correct order of the input data sequence and intermediate results.

Each stage of the SDF-FFT works in two rounds. In the first $N/2$ cycles (first round), the BF of the first stage is idle. The input data sequence is directly guided into the FIFO (of size $N/2$) until it is filled. In the next $N/2$ cycles (second round), the BF computes 2-point DFT with the incoming data sequence and the data stored in the FIFO. One of the outputs in this round is streamed to the next stage after being multiplied by the twiddle factor while the other is sent back to the FIFO to be the output in the next run of the first round. The size of the FIFOs in the early stages needs to be larger than those at the end to ensure correct data handling. The first FIFO needs to contain half of the input

data. Thus, its length is $N/2$. On the other hand, the last unit only requires a single data to be stored [73]. Furthermore, stages that are closer to the final one use fewer twiddle factors. In the final stage, all twiddle factors are unity. Hence, the multiplier unit can be removed. The whole architecture has a single input and a single output data streams and generates one sample per iteration.

Besides the basic radix-2 approach, various higher radix pipelined SDF-FFT architectures have been proposed over the past several decades. Radix-4 [75, 76] and radix-8 [77] SDF-FFT have been investigated to expand the design concept. In order to achieve reduced computation complexity, radix-2^2 [78, 79, 80], radix-2^3 [81, 82] and radix-2^4 [83, 84] FFTs have been developed. Note that the radix-2^k ($k > 1$) SDF has a similar DFG to radix-2 SDF. However, the potential benefit of these different algorithms is that some of the complex multipliers only need to use simple coefficients (TFs) [85] or that smaller coefficient memories are required [86]. Furthermore, algorithms are proposed in [73] to apply radix-3 and radix-5 in order to support diverse FFT sizes that are not limited to power of 2.

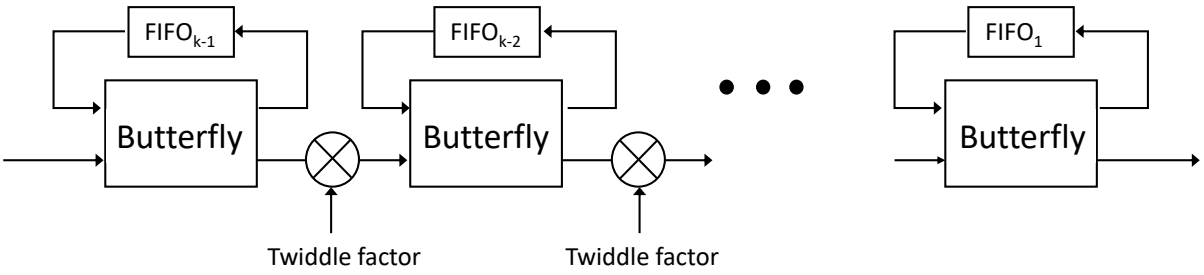


Figure 2.3: The conventional SDF-FFT architecture

2.2.2 Reconfigurable FFT architectures

The conventional SDF-FFT design provides only a single-radix manipulation without any circuit flexibility. Several researches have shown a trend toward multiple-radix processing [87, 88, 89] in order to support diverse FFT sizes. Accordingly, the design of reconfigurable FFTs has been emerged in the literature. The works presented in the literature can be classified into two categories. In the first category, the authors have provided the simplest form of reconfiguration. As illustrated in Fig. 2.4(a), certain portion of the hardware resources are exploited for one radix while other independent parts are reserved for another radices. The overall system allows the concatenation/combination of several processing elements having different radices. However, such approaches usually impose a duplication of hardware resources. Thus, they are considered inefficient in terms of area and energy. In the second category, the authors have made use of the same processing elements for

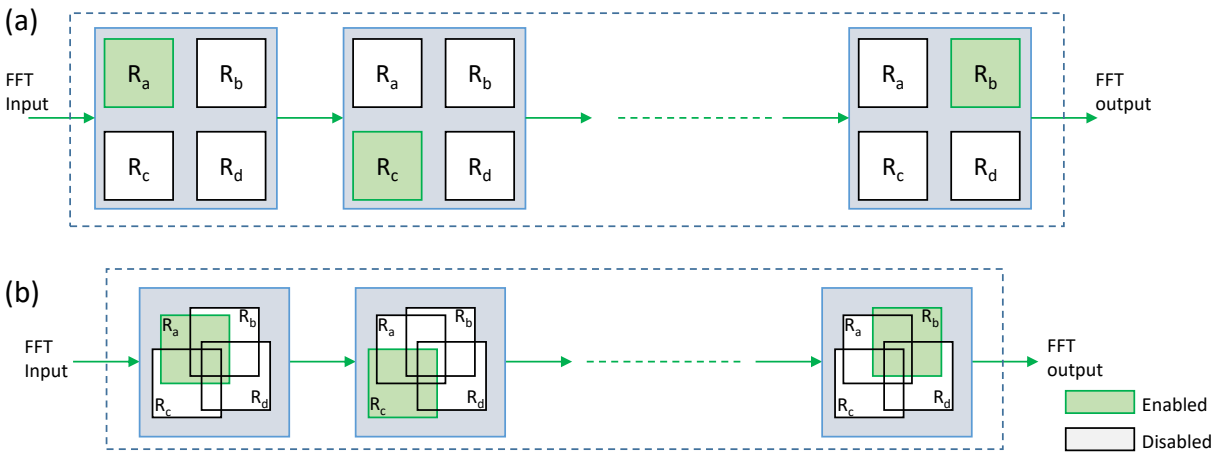


Figure 2.4: The block diagram of the reconfigurable FFT

realizing different radices. As illustrated in Fig. 2.4(b), the hardware resources, which are utilized for executing certain radix, can be reused for executing other radices at different instants of time. These implementations are considered reconfigurable/flexible. Flexibility is usually evaluated according to the achieved hardware reusability ratio, which have to be maximized in order to reduce the utilization area. The authors in [90] have proposed a reconfigurable FFT design that is able to support 48 different configurations with an FFT size up to 2187. As shown in Fig. 2.5, the design is composed of several reconfigurable bricks, which are arbitrarily concatenated to build up a system with reconfigurable FFT size. At each brick stage, flexibility is achieved through a reconfigurable processing element (RC-PE) and reconfigurable FIFO (RC FIFO) memory. RC PE is capable of realizing six types of radices (2, 2^2, 2^3, 3, 3^2 and 2×3), which are the combinations of radix-2 and radix-3. Moreover, the length of RC FIFO at each brick stage can be adapted according to the target FFT size.

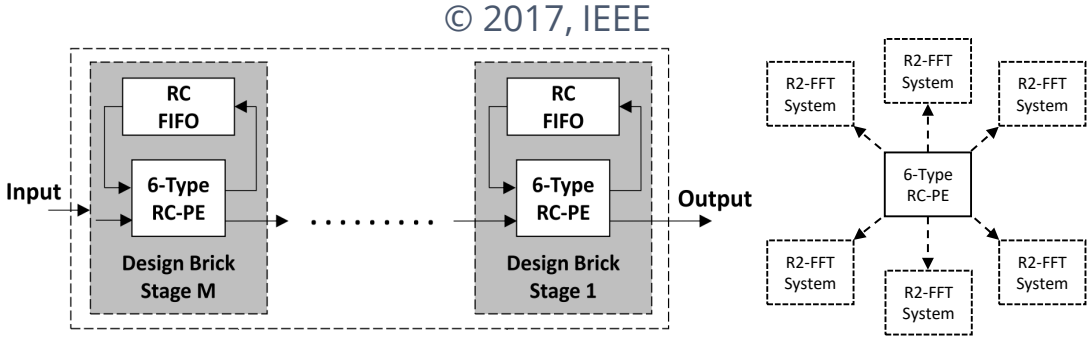


Figure 2.5: Reconfigurable SDF-FFT based on 6T-RC-PE approach [90]

2.3 Proposed mrFFT design

The attained degree of flexibility in the available implementations dedicated for FFT is hampered by the high cost of reconfiguration elements, such as multiplexers, SRAM cells and buffers. These elements constitute the major overhead in terms of area, power consumption and delay. For the sake of realizing efficient flexible FFT design, we have investigated the advantage of exploiting memristive devices as routing switches at the level of interconnects.

As explained in chapter 1, the usage of memristors exceeds storing of configuration to propagate datapath signals replacing CMOS transmission gates and/or Multiplexers. When programmed into LRS, a memristor propagates signals within the datapath leading to the same functionality of transmission gates in ON state. In contrast, when programmed into HRS, a memristor blocks signals in the datapath corresponding to a transmission gate in OFF state.

In order to attain a flexible FFT design, which can be manipulated in a reuse and systematic way, we have proposed a reconfigurable butterfly (RBF), which can be configured to support radix-2 and radix-3 flow graphs based on the algorithms developed in [90]. Based on the RBF design and the flexibility requirements in FFT, a novel memristor-based reconfigurable FFT architecture has been realized.

2.3.1 Reconfigurable butterfly: RBF

The flow graphs of FFT radices are mainly composed of complex adders and constant multipliers. The investigation of the structures of these flow graphs reveals that it is possible to extract similar/common parts. These similarities are exploited toward realizing efficient re-usability of hardware resources leading to efficient flexible designs. The analysis of radix-2 and radix-3 flow graphs shows that a radix-3 flow graph can be split into three similar butterflies and a separate real multiplier ($\times k$) as illustrated in Fig. 2.6(a). Although the three BFs are not exactly identical, they can be merged into a single reconfigurable BF, so called RBF, which is shown in Fig. 2.6(b). An RBF can be configured to execute radix-2 BF or any part of the radix-3 BF. Figure 2.6(c) describes the associated hardware implementation of RBF. It is worth to mention that the added cost for merging the similar parts into a single module (i.e, RBF) is almost negligible and corresponds for two trivial constant multiplications (blocks with orange color in Fig. 2.6(c)). Multiplication by j is realized by simply swapping the real and imaginary parts followed by multiplying the new real part by -1. Moreover, multiplication by 0.5 is achieved by 1-bit shifting to the right, which is simply realized by just discarding the least significant bit

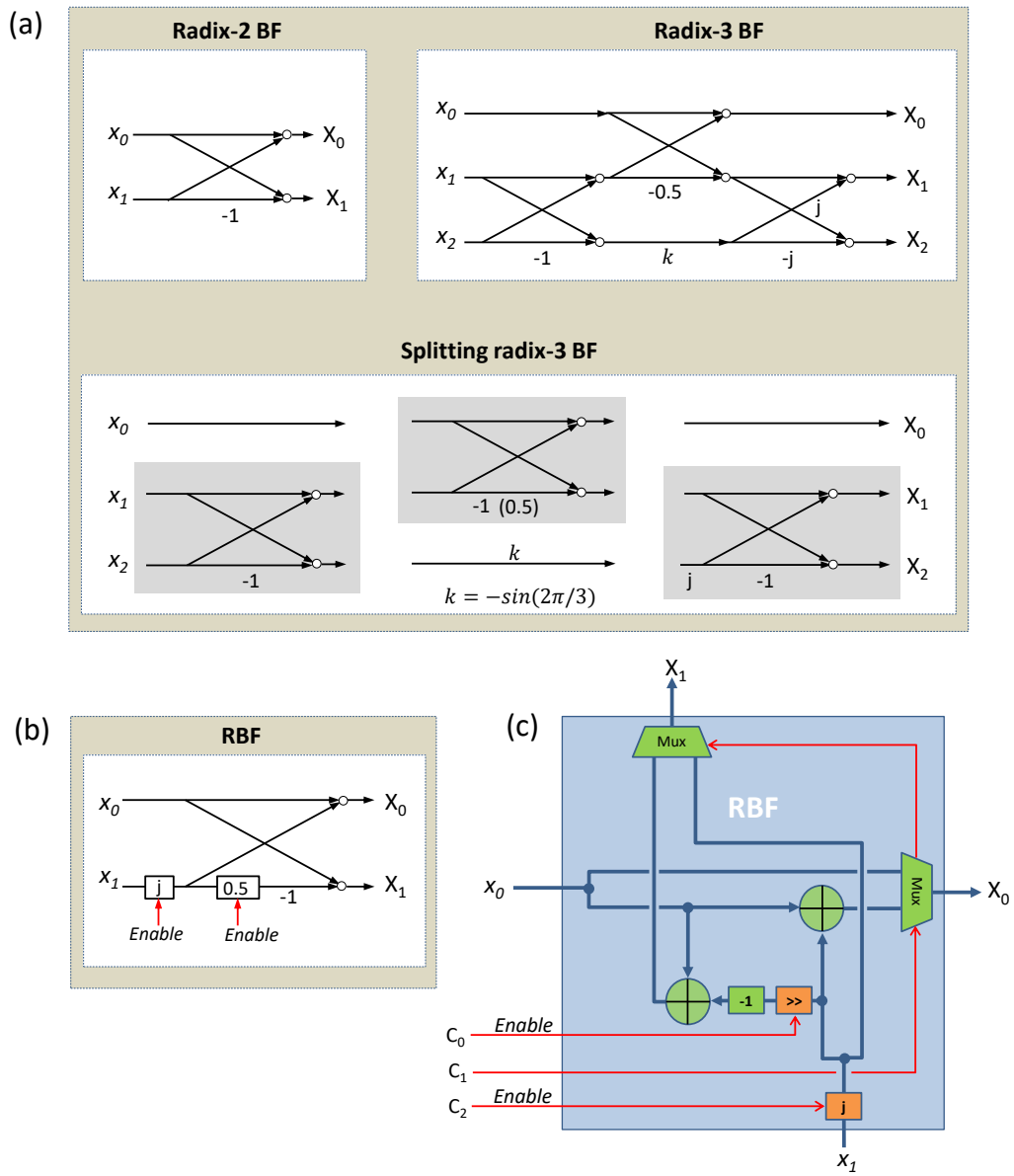


Figure 2.6: The architecture design of the proposed Reconfigurable Butterfly (RBF)

while replicating the most significant bit to fill the vacant position.

2.3.2 mrFFT architecture design

Based on the proposed RBF design, it is possible to implement any flow graph based on radices 2^k, 3^k and their combination (e.g, 2, 2^2, 2^3, 3, 3^2 and 2×3 which are adopted in [90]). Generally, in an SDF-FFT system, the FFT size N determines the number and type (radix) of the pipelined FFT stages to be concatenated. As an example, consider an N-point FFT, where N is equal to $864 = 2^5 3^3$. In fact, N can be factorized in the form $N_{864} = (2^2 2^2 2)(3^2 3)$. Accordingly, an 864-point FFT requires: two stages of radix-2^2, one stage of radix-3^2, one stage of radix-2 and one stage of radix-3. All these stages

can be implemented by routing appropriately the RBFs while feeding the stages with the corresponding set of FIFOs and multipliers. However, considering all possible FFT sizes less than a predefined value N_{MAX} requires a highly flexible design that can freely route the resources.

In order to attain the desired flexibility degree, we target an FFT architecture (mrFFT) with FPGA-like resource arrangement. The proposed mrFFT design is presented in Fig. 2.7. In this design, a group of RBFs are arranged in a 2D mesh topology with memristor-based interconnections serving as routing elements. The corresponding FIFOs and multipliers of mrFFT are well distributed inside the FIFO-Multiplier (FM) block as illustrated in Fig. 2.9. In this work, we define the term memristive node (MN) as a group of m memristors allocated on the diagonal of a crossbar that has square shape as illustrated in Fig. 2.8. The crossbar corresponds to the m-bit horizontal and vertical wires of the datapath. MNs are distributed along the whole design in a regular pattern that is well suited for the routing scheme. A single MN behaves as m-bit switch that can be programmed to logically connect/disconnect:

1. Horizontal wires (HW) with vertical wires (VW).
2. HW or VW with an RBF.
3. HW or VW with the block containing FIFOs and multipliers (FM).

The two selection blocks (SBs) are responsible of changing the states of the memristive nodes during reconfiguration phase. The SBs send the appropriate voltage levels to the terminals of MNs just as writing into a conventional memristive crossbar array. They set a ground voltage V_G at one end of each MN while supplying V^+ or V^- at the other end. A control unit, which is located at the upper right corner of the architecture, control the SBs. It stores the configuration bits corresponding for each FFT configuration/size in two separate lookup tables LUTs. One of the LUTs is responsible for driving the horizontal SB, while the other is simultaneously employed for the vertical SB. On the other hand, the twiddle factors, which are used in the multiplication operations, are stored in a separate LUT inside the FM block.

2.3.3 Supported mrFFT configurations

In order to support radix-2 and radix-3, the FFT size N should be prime factorized $N_{p,q} = 2^p 3^q$ where p and q are integer values and $N_{p,q} \leqslant N_{MAX}$. The number of RBFs needed to support an FFT of size $N_{p,q}$ is $M_{p,q} = p + 3q$. This is due to the fact that three RBFs are needed for each radix-3 flowgraph while only one is needed for radix-2. In

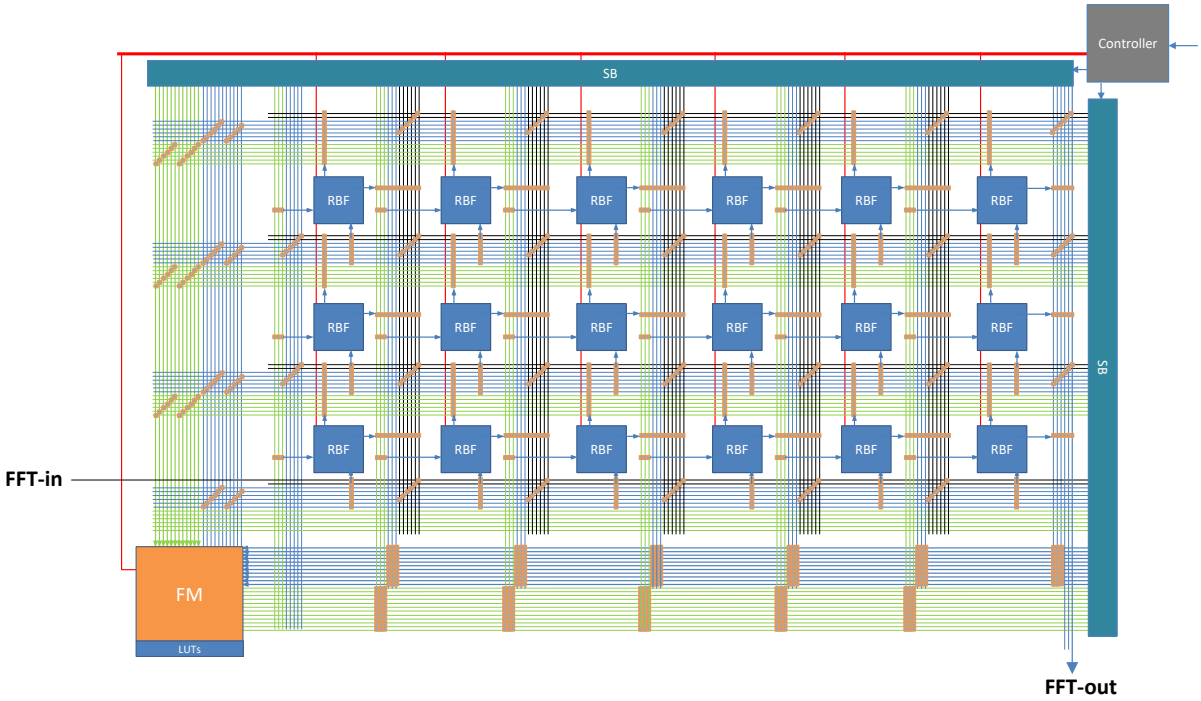


Figure 2.7: The proposed mrFFT architecture

order to support an $N_{p,q}$-point FFT, the values of p and q should satisfy the inequality $p + 3q \leqslant M_{MAX}$ where M_{MAX} represents the number of available RBFs in the mrFFT design. Moreover, the required number and length of FIFOs corresponding to the size $N_{p,q}$ should be less than or equal to the available ones inside the FM block. Besides, the number of available multipliers should suffice the $N_{p,q}$-point FFT that has the maximum number of concatenated stages.

The mrFFT architecture presented in Fig. 2.7 is designed using 18 RBFs (i.e, $M_{MAX} = 18$) and supports up to 44 configurations. Table 2.1 lists a set of FFT sizes which have the form of $N_{p,q}$ within the window $p \leqslant 11$ and $q \leqslant 7$. The mrFFT supports the configurations that are represented in boldface in Table 2.1. On the other hand, the FIFOs reusability has been analyzed over the 44 supported configurations while considering that two FIFOs serves in each radix-3 stage. A set of 12 optimized length FIFOs has been selected and are presented in Table 2.2.

As for the multipliers, it is noticed that at least one complex multiplier is required at the end of each pipelined FFT stage (except for the last stage) whatever the type of the used radix [90]. An additional constant multiplier ($\times k$) is utilized in each radix-3 stage. On the whole, 10 complex multipliers are sufficient for the 44 supported configurations.

In fact, the presented mrFFT design accommodates the 32 operating modes that are defined in 3GPP-LTE standard [87, 90]. However, mrFFT is scalable in size and can be extended. Thus, it is able to support other FFT configurations in case of any future

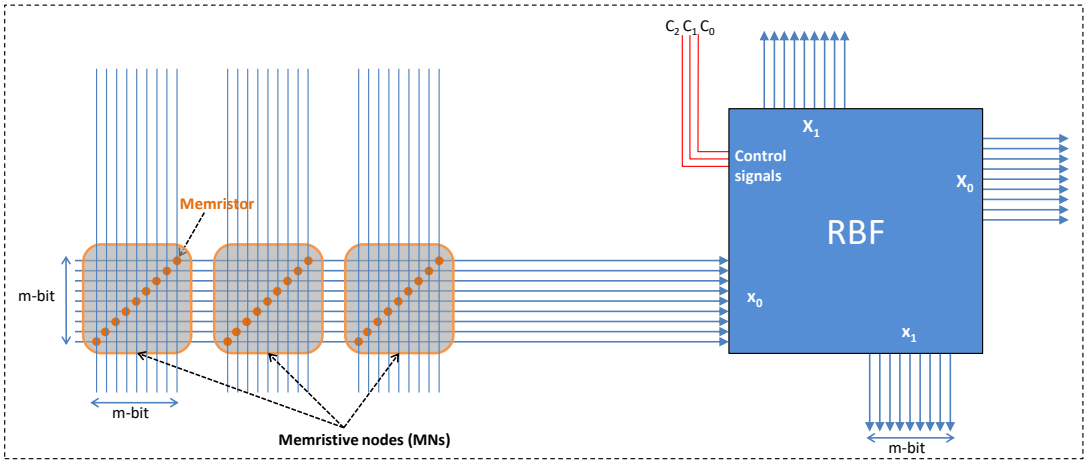


Figure 2.8: The proposed structure of memristive nodes

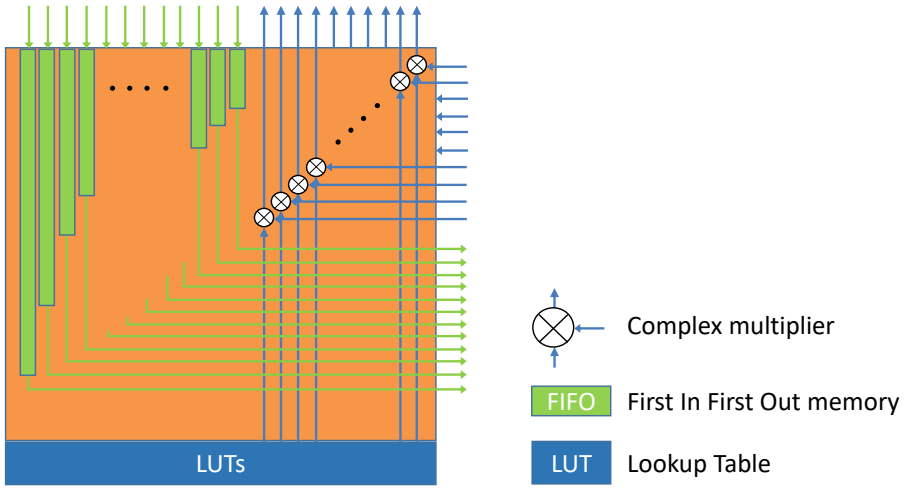


Figure 2.9: The architecture of FM block

standard changes.

Table 2.2: Optimized sizes of the required FIFOs

FIFO Sizes	1024	512	256	128	64	32	16	9	4	3	1	1

2.4 Comparison

In order to determine the relevancy of our proposed mrFFT design, a comparison has been conducted with the recent approach presented in [90]. The amount of utilized resources in our proposed design has been estimated and compared to that in [90]. The comparison results are presented in Table 2.3. The comparison shows that mrFFT reduces significantly the number of the utilized adders (25%), multipliers (37%), 2-to-1 multiplexers (59%),

Table 2.1: The 44 FFT configurations supported by mrFFT with $N = 2^p 3^q$

p \ q	0	1	2	3	4	5	6	7
0	1	**3**	**9**	**27**	**81**	**243**	729	2187
1	**2**	**6**	**18**	**54**	**162**	**486**	1458	4374
2	**4**	**12**	**36**	**108**	**324**	**972**	2916	8748
3	**8**	**24**	**72**	**216**	**648**	**1944**	5832	17496
4	**16**	**48**	**144**	**432**	**1296**	3888	11664	34992
5	**32**	**96**	**288**	**864**	2592	7776	23328	69984
6	**64**	**192**	**576**	**1728**	5184	15552	46656	139968
7	**128**	**384**	**1152**	3456	10368	31104	93312	279936
8	**256**	**768**	2304	6912	20736	62208	186624	559872
9	**512**	**1536**	4608	13824	41472	124416	373248	1119744
10	**1024**	3072	9216	27648	82944	248832	746496	2239488
11	**2048**	6144	18432	55296	165888	497664	1492992	4478976

3-to-1 multiplexers (100%) and FIFOs (25%). On the other hand, the design in [90] has a 55.5% less number of 1-bit shifters. However, these shifters have a trivial implementation as discussed in Section 2.3.1. The attained gains in mrFFT comes at the cost of the adopted routing scheme, which incorporates a set of 119 memristive node in addition to the corresponding selection blocks. In fact, the implementation cost of the SBs can be relatively reduced when shared to other designs targeting the same flexibility requirements (e.g. MN-based routing).

Moreover, the percentage of resource activation has been evaluated in both designs among all supported configurations. The maximum percentage of activation has been presented in Table 2.3. The obtained results show better utilization of adders and multipliers in mrFFT compared to that in [90]. This implies better hardware reusability ratios.

Table 2.3: Analytical comparisons

	Complex multipliers	Complex adders	FIFOs	2MUX1	3MUX1	1-bit shifter	Memristive nodes
# Elements (mrFFT)	10	36	12	36	0	18	119
# Elements ([90])	16	48	16	88	12	8	0
% Resource activation (mrFFT)	100%	94.44%	100%	94.44%	–	27.78%	22.88%
% Resource activation ([90])	81%	67%	–	100%	100%	100%	–

2.5 Limitations in performance evaluation

The evaluation of the devised mrFFT architecture was limited to an analytical estimation of the utilized resources, the percentage of activation and the reusability ratio. Power

consumption and delay analysis could not be evaluated for this first proposed design.

In fact, the digital sub-blocks constituting the proposed mrFFT architecture were described in VHDL and validated separately in ModelSim (Mentor Graphics). Thereafter, we attempted to find a way to include a description which could mimic the memristor device behaviour. These attempts failed as an analog model was mandatory to define the resistive switching behavior of memristive devices.

Available memristor models in the literature are usually described in SPICE or Verilog-A and can therefore be supported by tools such as Virtuoso analog design environment from Cadence. However, these tools can not perform high-level simulation of digital components such as RBF, FM and the control unit of mrFFT. In fact, analog/digital mixed-signal simulation environment is required in this case. We investigated the possibility to use several relevant tools, such as Spectre AMS Designer and Questa ADMS. However, this was not possible due to availability and setup issues.

Therefore, for the rest of this thesis work we targeted low-level simulations, using Cadence Virtuoso, rather than system level. The scope of our contributions has been moved from system to circuit level in order to realize explicit performance evaluation and avoid the aforementioned limitation.

2.6 Summary

In this chapter, a novel memristor-based reconfigurable FFT architecture is proposed. mrFFT is fully pipelined and executes one sample per clock cycle as in the conventional SDF based FFT implementations. mrFFT allows mixing any combination of radix-2 and radix-3 stages based on our proposed reconfigurable butterfly approach. An RBF is optimized as it can execute radix-2 BF or any part of radix-3 BF with the minimal added cost. The proposed scalable mrFFT architecture enables the efficient support of 44 configurations with different FFT sizes including the 32 operating modes that are defined in 3GPP-LTE standard. The RBF blocks are integrated in a 2D mesh topology with memristor-based interconnections and allow for original and optimized hardware reusability. Preliminary comparison with the state-of-the-art work indicates significant reduction in the used resources and improved hardware reusability ratio. In order to evaluate accurately the corresponding energy savings, future work should consider the use of mixed-signal simulation tools.

Chapter 3

Hybrid Memristor-CMOS Design for Logic Computation

MEMRISTOR technology have recently triggered many efforts to extend their usage from memory to computing. Memristor based logic design is an emerging concept targeting efficient computing systems. Several logic families have evolved, each with different attributes. Memristor Ratioed Logic (MRL) has been recently introduced as a hybrid memristor-CMOS logic family. MRL requires efficient design strategy that take into consideration the implementation phase. This chapter presents a novel MRL-based crossbar design namely X-MRL. The proposed structure combines the density and scalability attributes of memristive crossbar arrays and the opportunity of their implementation at the top of CMOS layer. The evaluation of the proposed approach is performed through the design of X-MRL based full adder. The design is presented with the layout and the corresponding simulation results using Cadence Virtuoso toolset.

The next section presents our motivation to investigate hybrid memristor-CMOS logic design based on MRL. Section 3.2 illustrates the proposed X-MRL approach for realizing Boolean computation. Section 3.3 presents the proposed design for a full adder based on X-MRL approach. The layout of the obtained X-MRL design is illustrated in Section 3.4. Section 3.5 provides the obtained simulation results along with the performance analysis. Comparison with previous published designs is presented in Section 3.6. Finally, Section 3.7 concludes the chapter.

3.1 Motivation for hybrid memristor-CMOS design

At the device level, the complexity of further scaling down the conventional CMOS technology in order to keep pace with the Moore's prediction has encountered major challenges

[91][92]. The advent of memristor-CMOS process that combines CMOS devices with the nano-scale size memristors have provided new opportunity to reduce the utilization of silicon area. This paves the way for innovating new circuits that are almost removed from the more established design domains. However, yet CMOS devices which are considered active, cannot be totally replaced by the passive memristive devices. Thus, the integration of CMOS and memristors is essential to the development of memristor technology. To this end, hybrid configurations have been proposed that make use of the advantages of CMOS while utilizing the high density of memristors. On the other side, out of the available memristor-based logic design styles (including the proposed MOL), the Memristor Ratioed Logic (MRL) is the only design style that meets the conventional CMOS in terms of the adopted state variable. As discussed in chapter 1, both MRL and CMOS uses the voltage as the only state variable for representing inputs and outputs throughout all intermediate stages. Thus, MRL is very qualified for the integration in the current CMOS designs and even later can dominate.

In fact, MRL is a hybrid memristor-CMOS logic family. The goal behind MRL is to implement conventional combinational logic circuits which are the building blocks of digital systems. The main idea is to replace as much as possible transistors with nano-scale size memristors, while keeping the same role of the intended digital architecture. Several works have utilized MRL design style to implement various digital architectures [52] [93] [94]. The integration of memristors and CMOS devices in MRL still lacks a consistent way for arranging memristors at the top of CMOS layer. The integration should be realized in such a way that efficiently exploits the promising characteristics of memristive devices such as density and scalability.

3.2 X-MRL design procedure

It is well known that any Boolean function could be written in the form of the sum of products (SoP). Accordingly, it can be implemented using MRL-AND and MRL-OR with the aid of CMOS inverters. In order to clarify easily the proposed method, Fig. 3.1 illustrates the design and implementation of the simple function $F = AB + AC$. Fig. 3.1(a) shows that the function F is implemented using two MRL-AND and one MRL-OR. Fig. 3.1(b) depicts the schematic layout, which illustrates the equivalent mapping of the function onto a crossbar structure that we call X-MRL. The vertical pairs of memristors corresponding to MRL-AND generate an output which drives the input of the horizontal pair that represents MRL-OR. Fig. 3.1(c) presents a 3D view of the resulting crossbar structure. Fig. 3.1(d) is another simplified representation of the obtained crossbar. The

same procedure could be performed to implement other Boolean functions. Although the obtained array is a combination of AND and OR gates, the positive poles of the allocated memristors rely on the same planar side, which is considered as an advantage at the level of their fabrication. The placement of the crossbar at the top of CMOS inverters is demonstrated in section 3.4

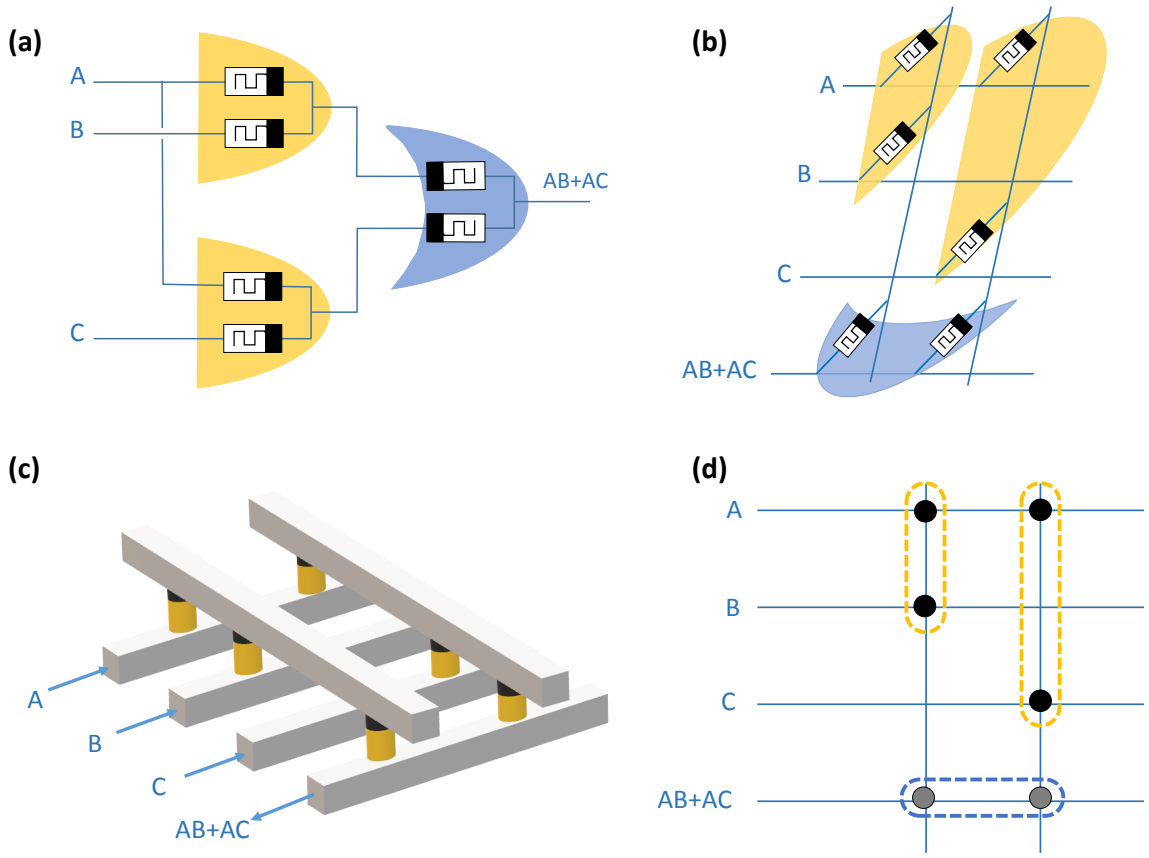


Figure 3.1: Example of an MRL logic function performed using X-MRL

3.3 X-MRL based full adder

This section presents as an example the design of the 1-bit full adder using X-MRL design technique. Equations (4.4) and (4.5) present the expressions of the 1-bit full adder in the SoP format.

$$S = A \oplus B \oplus C_{in} = C_{in}(\overline{\overline{A}B + A\overline{B}}) + \overline{C_{in}}(\overline{A}B + A\overline{B}) \tag{3.1}$$

$$C_{out} = AB + BC_{in} + AC_{in} \tag{3.2}$$

where A and B are the inputs, C_{in} is the input carry, S is the 1-bit adder output and C_{out} is the output carry. Fig. 3.2(a) presents the direct form of an MRL based 1-bit full adder. Fig. 3.2(b) presents the proposed circuit design of the 1-bit full adder using MRL-based crossbar structure. The design requires 18 memristors, which are distributed

among vertical and horizontal wires, in addition to nine CMOS inverters. In the figure, the black vertical pairs of memristors represent the AND gates while the gray horizontal pairs represent the OR gates (as illustrated in Fig. 3.1). The CMOS inverters are responsible for either inverting (NOT operation) and/or performing signal restoration for the logical state of the signal after several cascading stages.

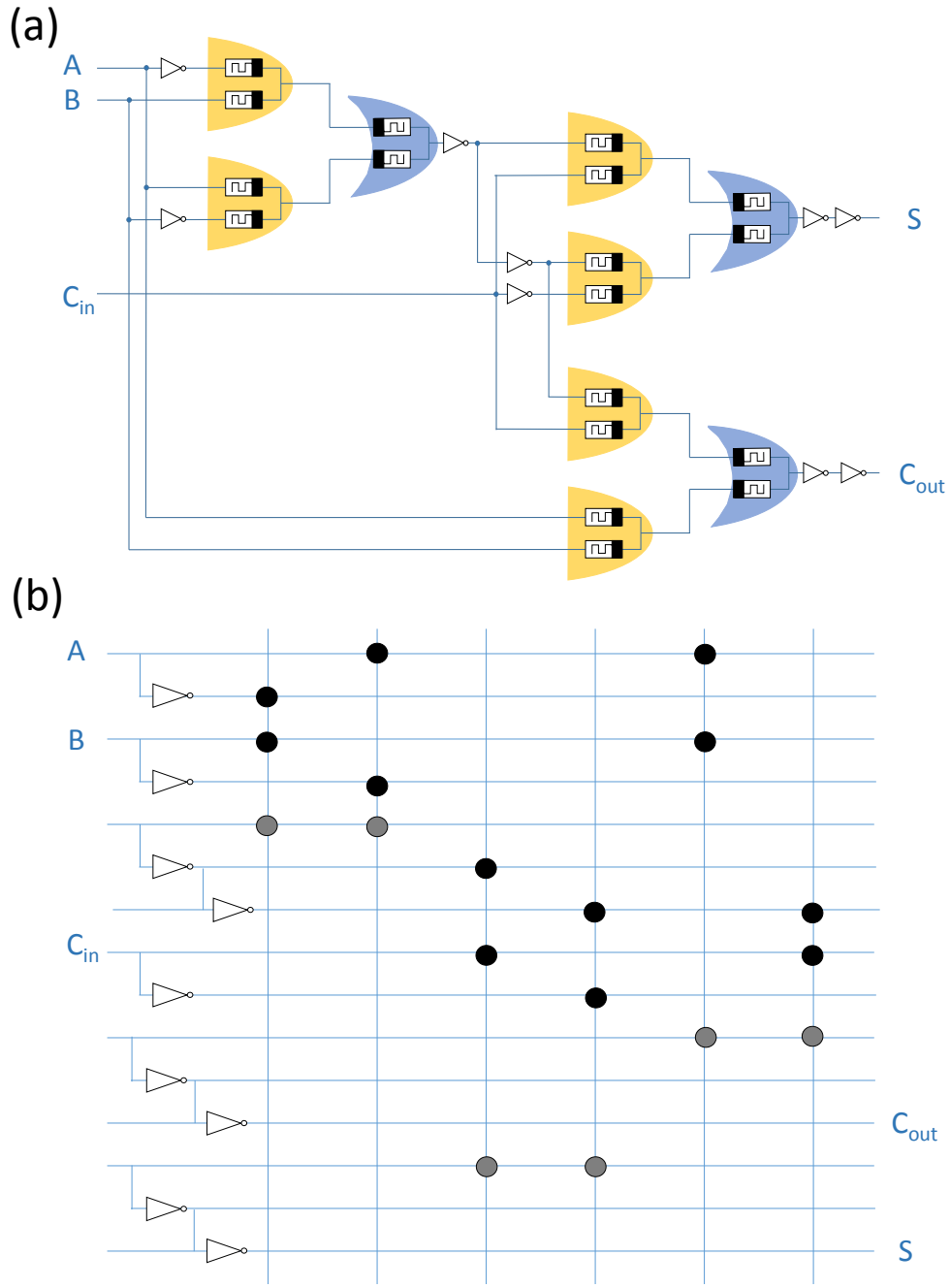


Figure 3.2: 1-bit Full Adder based on the proposed X-MRL structure

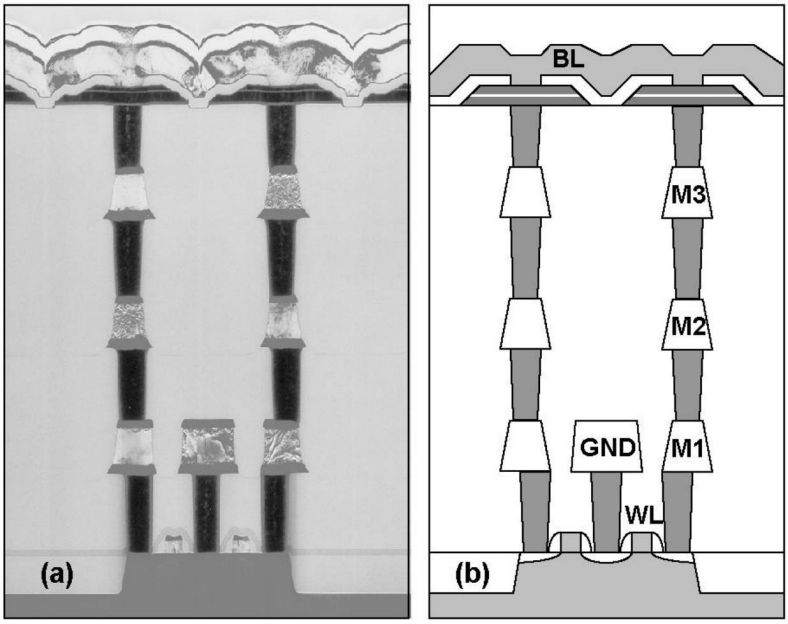


Figure 3.3: Memristor layer at the top of VIAs [95]: (a) a TEM image. (b) a schematic view.

3.4 Layout

The circuit of the full adder is composed of a memristor crossbar layer in addition to few inverters. Fig. 3.4 presents the layout of the circuit using Cadence Virtuoso tool. In this layout, the positions of the allocated memristors are assigned virtually due to the lack of their definition in Cadence library. The layout is mainly composed of three layers. The first layer is the polysilicon layer which is dedicated to the connection of the gates of NMOS and PMOS transistors. This layer is presented in red color in the figure. The second and third layers, which are so-called Metal1 and Metal2 and presented in the figure in violet and blue colors respectively, are dedicated to the wiring. In order to attain the desired crossbar structure, horizontal wires are constructed in the Metal2 layer; whereas for the vertical wires, the connections, which are already utilized for the implementation of the required CMOS inverters, are reused to complete the crossbar structure. However, the hight of the utilized memristors is too short (around $10nm$ [23]) to allow to link horizontal and vertical wires through two different layers. Therefore, these links are achieved through vertical interconnect accesses (VIAs) as demonstrated in [95]. Fig. 3.3 is a schematic view and cross-sectional transmission electron microscopy (TEM) image of memristor integrated with CMOS in the same die [95]. The allocated memristors in our proposed layout are implemented at the top of the VIAs just under Metal2 layer. Accordingly, the CMOS inverters occupies the total utilized area and the additional Metal2 layer have to

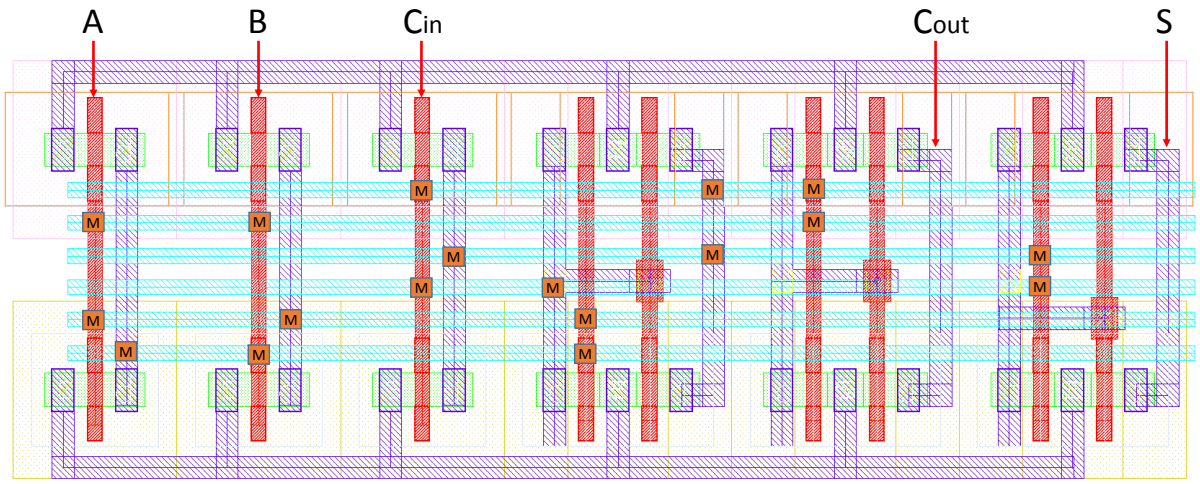


Figure 3.4: Proposed layout for the hybrid memristor-CMOS 1-bit full adder based on X-MRL design technique

be reserved for memristors. Hence, in such hybrid memristor-CMOS architectures, the size of implementation area depends on the number of the required inverters.

3.5 Simulation and performance analysis

3.5.1 Memristor model fitting

The VTEAM model is adopted in order to realize a realistic modeling of practical memristors. Table 3.1 provides the experimental data of various available memristors with their respective properties. Among these memristors, the HfO_x memristor which has been reported in [40] has the properties which suit the MRL gates. The device is characterized by low switching delay $300\,ps$ at low operating voltage $1.4V$. These characteristics make this memristor eligible to be implemented in the same die with the current CMOS devices. An important work have been done to determine the VTEAM model parameters that fit with the physical parameters of HfO_x, which have been described in [40]. Table 3.2 shows the determined VTEAM model parameters. The model parameters are chosen to produce switching delay of $300\,ps$ for a voltage pulse of $1.4V$ as reported in [40]. Fig. 3.5 shows the switching behavior of the memristor corresponding for SET and RESET pulses. The device is assumed to be totally switched when the boundary position w reaches either 1% or 99% of the total length D of the memristor, corresponding for SET ($V_{set} = 1.4V$) and RESET ($V_{reset} = -1.4V$) operations respectively. The boundary conditions of the memristor is managed by a Biolek window function. The mathematical function of the Biolek window [96], which is described in (3.3), provides continuous and smooth transition of

Table 3.1: Practical memristor devices

Material	R_{ON} (ohm)	R_{OFF} (ohm)	R_{OFF}/R_{ON}	Switching speed	Voltage range	Reference
TiO_{2-x}	-	-	>300	1 ns	-1.5V to +1.5V	[23]
FTJ	1.6×10^5	4.6×10^7	>200	10 ns	-5.6V to +4.2V	[97]
HfO_2	1.2×10^2	10^5	10^3	<1 ns	<1.5V	[39]
HfO_x	<10 k	>100 k	>100	**300 ps**	**<1.4V**	[40]
TMO	-	100 k	-	10 ns to 100 ns	3V	[98]
HfO_2	2×10^3	2×10^5	100	-	-1.5V to +1V	[99]
$TiN/TiO_x/HfO_x/TiN$	1 k	>1 M	>1000	5 ns	-1.5V to +1.5V	[100]

Table 3.2: VTEAM fitting parameters for physical device in [40]

Parameter	Value	Parameter	Value
R_{ON}	1 $k\Omega$	p	2
R_{OFF}	200 $k\Omega$	α_{on}	3
D	3 nm	α_{off}	3
K_{on}	-0.0162 m/s	V_{on}	0.16 V
K_{off}	0.0162 m/s	V_{off}	-0.16 V
x_{on}	0 nm	x_{off}	3 nm

boundary when reaching one of the extremities of the memristor.

$$f(x) = 1 - (x - stp(-i(t)))^{2p} \tag{3.3}$$

where $stp(.)$ represents a unit step function, and p is a positive integer. Low values of p lead to smooth transition of the boundary of the memristor when reaching its extremities, whereas high values lead to sharp transitions.

3.5.2 Performance analysis

Transient simulation has been conducted for the proposed design of the X-MRL-based full adder in Cadence Virtuoso environment. The CMOS 65 nm technology at standard 1.2V has been adopted. Fig. 3.6 shows all the possible combinations at the inputs A, B and C_{in} in addition to the corresponding outputs S and C_{out}. The performance is analyzed below for the proposed design.

3.5.2.1 Timing analysis

Fig. 3.7 presents the definition of rising time (T_r) and the time delay (T_d). Accordingly, the conducted simulation of the proposed design shows that these extracted parameters

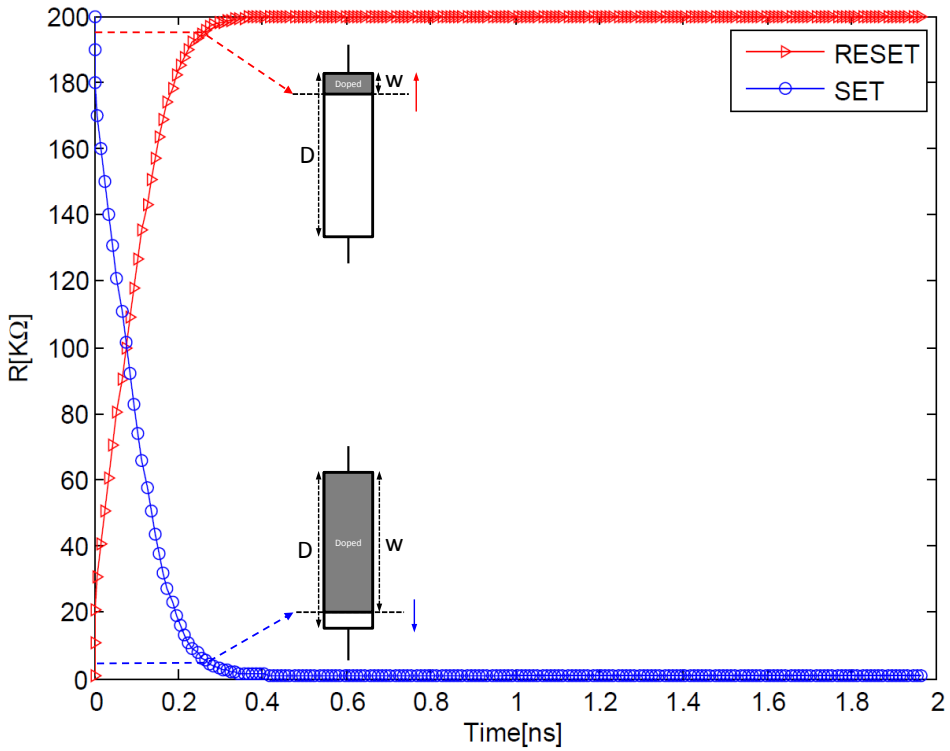


Figure 3.5: Memristor switching time for $V_{set} = 1.4V$ and $V_{reset} = -1.4V$ according to the device in [40]

(T_r and T_d) change among different value combinations of A, B and C_{in}. The maximum recorded values are as follows: $T_r = 82\ ps$, $T_d = 1.2\ ns$, and $T_f = 586\ ps$ where T_f is the falling time. These values are considered for the worst case performance. The conducted simulation shows that the values T_r, T_f and T_d are affected by the switching speed of the memristor which in turn can be controlled by K_{on} and K_{off}. On the other hand, slowing down the switching speed of the memristors increases the glitches. Fig. 3.8 shows the glitches appearance when reducing K_{on} and K_{off} levels to $-0.01\ m/s$ and $0.01\ m/s$ respectively. Particularly, the high resistance state (R_{OFF}) of the memristors has a direct effect on the value of T_d which increases when increasing the value of R_{OFF}. Moreover, it is noticed that increasing R_{OFF} acts as a filter for the glitches. Therefore, the total delay is directly affected by the memristor physical properties.

3.5.2.2 Energy consumption

Fig. 3.6(b) shows the total instantaneous power $p_T(t)$ consumed by the proposed design of the full adder. The peak values in $p_T(t)$ refer to the dynamic power consumption. The lower bound in $p_T(t)$, which is formed after the end of each transition, corresponds to the static power. A slight difference appears between the levels of the static power recorded

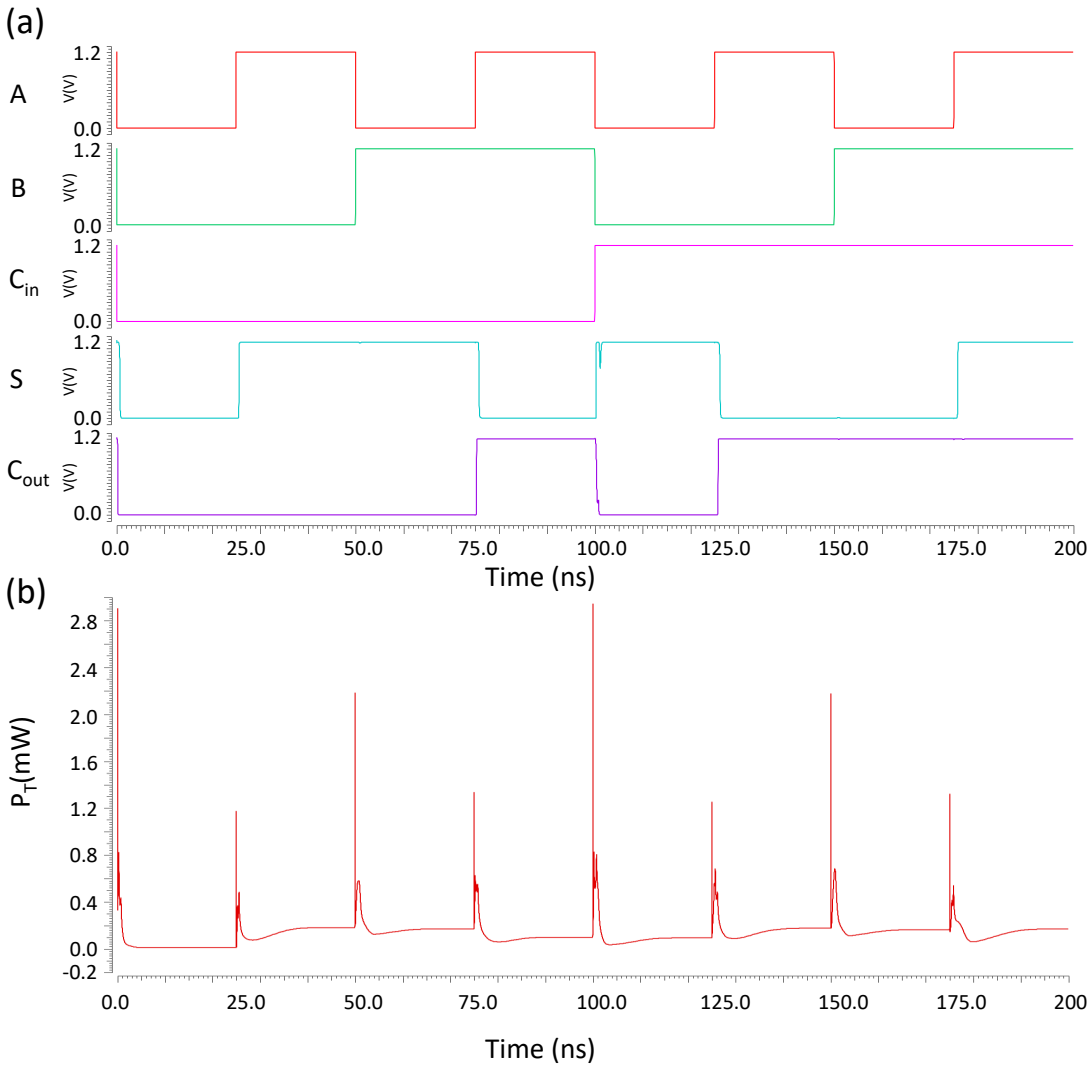


Figure 3.6: Transient response of the proposed full adder for the input signals A, B and C_{in}

after each transition. This difference is due to the change in the equivalent resistance state of the cascaded memristors for a new combination of the input signals A, B and C_{in} which in turn leads to a different level of current leakage. The average power consumed in the proposed design of the full adder is $279.5\,\mu W$. This value is evaluated at frequency $f = 200\,MHz$ which is near the max possible frequency at the inputs of the full adder with hybrid structure. Higher values of R_{OFF} have to be used in order to achieve hybrid architectures with low power consumption.

3.5.2.3 Utilized area

In fact, a single memristor has an area in the order of $4F^2$ [23], where F is the minimum feature size. Hence, memristors are implemented at the top of CMOS due to their nano-

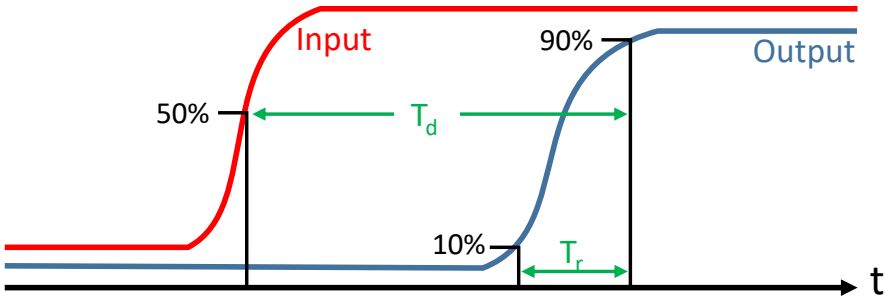


Figure 3.7: Definition of the rise time T_r and delay T_d

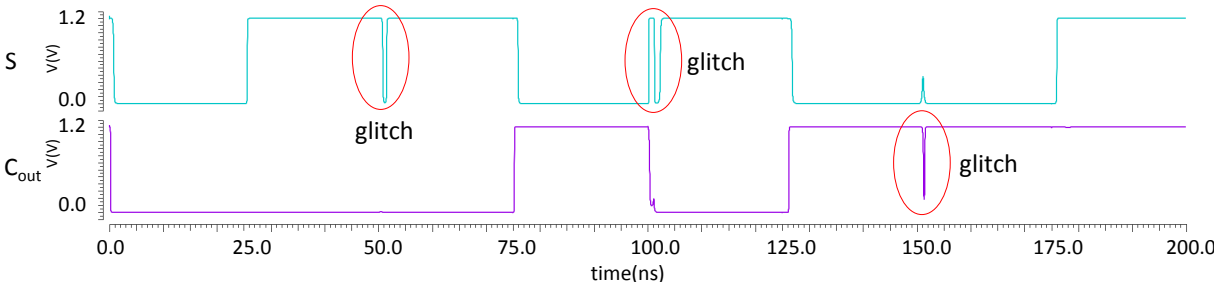


Figure 3.8: Glitches appearance when slowing down the switching speed of the memristor. Parameters in Table 3.2 are adopted except for $K_{on} = -0.01\ m/s$ and $K_{off} = 0.01\ m/s$

scale and compatibility at the level of fabrication. Thus, the allocated memristors in the proposed X-MRL design do not add any overhead in terms of implementation area. The total required area refers to that occupied by CMOS devices only, which depends on the number of inverters as discussed in Section 3.4. Fig. 3.4 presents the proposed layout. The total area of the X-MRL design is 8.16 μm^2 compared to 14.78 μm^2 which is utilized in the case of pure CMOS implementation, leading to 44.79% area saving.

3.6 Comparison

The proposed hybrid memristor-CMOS based full adder has been compared with previous published designs, which are dedicated to the 1-bit full adder. Note that related works in the literature lack an estimation about the utilized area for their proposed designs. Moreover, in order to achieve a fair comparison in terms of energy consumption, the energy is evaluated per 1 addition operation. The time period for an addition operation in our proposed full adder design is set to be the minimum possible time (i.e. max frequency). This subsection presents the comparison summary which is also shown in Table 3.3.

In [93] an optimized implementation of an MRL based 1-bit full adder is proposed. The authors have developed an algorithm which searches for the best form of the boolean

functions of the sum (S) and carry (C). The desired form should lead to an implementation with the minimum possible number of CMOS inverters. The inverters positions are allocated in such a way that removes signal degradation. The proposed circuit design of the full adder in [93] has less number of memristors as well as CMOS transistors by 11.1% and 33.3% respectively compared to our proposed design. However, the obtained logic function in [93] is not in the form of SoP. Thus, it is not possible to allocate memristors in a crossbar structure. This leads to more wiring at the fabrication stage which in turn increases the implementation area dramatically. As for energy consumption, the values reported in [93] are in the normalized form; hence, they can not be used for comparison.

In [101], a design for a 1-bit full adder has been proposed based on memristor MAGIC-NOR and NOT gates. A crossbar structure has been adopted and several optimization techniques have been used to minimize the number of rows and columns of the crossbar as well as the number of computational steps. It has been shown that a compromise exists between the size of crossbar and the needed number of steps to perform a full addition. A minimum size of 3×3 crossbar (i.e. 9 memristors) with a total latency of 35 computational steps is achieved. In contrast, our proposed design uses 18 memristors distributed among crossbar structure in addition to 9 CMOS inverters. The output is evaluated in 1 computational step. Concerning energy consumption, the proposed design in [101] consumes 0.3 pJ to achieve a 1-bit full addition process; whereas, our proposed design consumes 0.69 pJ.

In [102], an N-bit addition has been performed using MAGIC operations (i.e. NOR and NOT gates). Several approaches are presented by the authors for realizing logic within crossbars. The best among these approaches in terms of latency corresponds to $10N + 3$ computational steps which leads to 13 clock cycle for the case of 1-bit full adder. However, $13N - 3$ memristors are reserved (i.e. 10 memristors for N=1) to accomplish the 1-bit addition process. For the purpose of minimizing the number of reserved memristors inside the crossbar, an area optimized crossbar structure is also proposed in [102]. Only 5 memristors are utilized however $15N$ (i.e. 15 for N=1) computational steps are required to achieve the 1-bit full addition. As a result, our proposed design, which requires 1 computational step, outperforms the designs presented in [102] in terms of latency. Regarding the energy consumption, all the proposed approaches in [102] have almost same energy dissipation which is about 3.16 pJ for the case of N=1. Hence, the proposed design in [102] consume 4.5 times more energy than our proposed design.

In [103], an N-bit ripple carry adder (RCA) circuit in a memristor crossbar structure has been presented. The MAGIC design style has been used to implement the logic gates. By considering N=1 which is the case of 1-bit addition, the proposed crossbar

Table 3.3: Comparison with previous approaches

Reference	**#Memristors**	**#CMOS transistors**	**Energy**	**#steps**	**Step delay**	**Energy.Delay**
(This work)	18	18	0.69 pJ	1	2.5 ns	1.72 $pJ.ns$
MRL [93]	16	12	-	1	-	-
MAGIC [101]	9	Peripheral drivers	0.3 pJ	35	1.89 ns	19.84 $pJ.ns$
MAGIC (Optimized #steps) [102]	10	Peripheral drivers	3.16 pJ	13	1.3 ns	53.40 $pJ.ns$
MAGIC (Area optimized) [102]	5	Peripheral drivers	3.16 pJ	15	1.3 ns	61.62 $pJ.ns$
MAGIC [103]	15	Peripheral drivers	0.68 pJ	13	1.12 ns	9.94 $pJ.ns$
MAGIC (Naive mapping) [104]	15	Peripheral drivers	0.68 pJ	12	1.43 ns	11.66 $pJ.ns$
MAGIC (Compact mapping)[104]	24	Peripheral drivers	0.89 pJ	16	1.43 ns	20.36 $pJ.ns$
IMPLY [105]	6	Peripheral drivers	-	23	5 ns	-

MAGIC based design requires 15 memristors and can perform the addition operation in 13 clock cycles. Compared to our proposed design, the adder design in [103] needs 13 times more clock cycles to perform addition operation while it requires 3 less memristors to be implemented. On the other hand, our design consumes 1.01 times more energy than the proposed design in [103].

In [104], logic operations has been realized by two methods using MAGIC. The first method corresponds to a naive mapping. It maps the NOR/NOT netlist into a single row of the crossbar. For the case of 1-bit full addition, 12 NOT/NOR sequential operations are required on a total number of 15 memristors. The overall energy consumption is estimated as 0.68 pJ. The second method corresponds to the compact mapping. In this method NOR/NOT MAGIC operations are performed on rows and columns of a crossbar to realize logic functions. A 1-bit full addition process is performed on an 8×3 crossbar structure (i.e. 24 memristors) and requires 16 computational steps. The overall energy consumption is evaluated as 0.89 pJ. Compared to our design, the naive mapping and the compact mapping consume 1.01 times less and 1.28 times more energy respectively.

In [105], the authors have proposed a 1-bit full adder, which has been designed using IMPLY logic. The proposed design needs 23 computational steps to perform the addition. The 1-bit full adder proposed in [105] requires 6 memristors, which is 33.3% of the memristors utilized in our design. However, IMPLY logic design approach adopts 3 different voltage levels (V_{COND}, V_{SET} and V_{CLEAR}). So, an additional circuitry such as analog multiplexers should be added to drive the allocated memristors. This induces an overhead in terms of the total utilized area when compared to our proposed design. Note that the energy consumption is not considered by the authors. Table 3.3 summarizes the above presented comparison results. The table illustrates the key advantage of the

proposed approach regarding the reduced number of computational steps with respect to other existing designs. The energy consumption remains comparable. The $Energy.Delay$ metric is used for a global direct evaluation. This metric combines both delay and energy consumption. As shown in the table, our proposed design outperforms all existing related ones. The improvement in $Energy.Delay$ is between ×5.7 and ×31.

On the other hand, for the works that have adopted MAGIC and IMPLY in [101] [102] [105] [103] and [104], the initialization and the evaluation of the rows and columns of the memristive crossbar require a separate CMOS controller. Moreover, a conversion mechanism is required in these designs. This mechanism includes a sensing amplifier which converts the resulting stored bits from the resistance state to the voltage state [52]. These additional peripheral drivers result in additional overheads in area and power consumption.

3.7 Summary

In this paper, an MRL-based crossbar design namely X-MRL is proposed. X-MRL approach is dedicated for the implementation of combinational logic. The design methodology of X-MRL efficiently integrates memristors with CMOS devices to improve density and scalability. Using X-MRL, Boolean function are represented using pairs of memristors mapped efficiently into crossbar structure. The obtained memristive crossbar is stacked at the top of CMOS layer. For evaluation purposes, we design hybrid memristor-CMOS full adder based on X-MRL approach. Based on realistic memristor parameters model and CMOS 65 nm process, the design is simulated in Cadence Virtuoso environment. The obtained layout of the full adder demonstrates a 44.79% area reduction compared to that implemented with pure CMOS technology. Moreover, the *Energy.Delay* metric is used for comparison. It reveals an improvement between ×5.7 and ×31 with respect to the available literature.

Chapter 4

MOL – Memristor Overwrite Logic for In-Memory Computing

This chapter regroups the last three contributions of the thesis related to in-memory computing. It introduces a novel logic design style, namely Memristor Overwrite Logic (MOL), associated with an original MOL-based computational memory. MOL combines the simplicity of MAGIC/IMPLY techniques and the accuracy of MAJ. Moreover, MOL can operate with different memristive device technologies and allows for significant reduction in the number of required memristors and computational steps.

The chapter is organized as follows. It starts by motivating the need for in-memory computing paradigm and by reviewing related existing efforts that investigate the use of memristive devices. Then, Section 4.2 illustrates the limitations that accompanies existing memristor based logic design styles. Section 4.3 presents our proposed MOL approach. Section 4.4 discusses the integration of MOL into the conventional memory configurations. Section 4.5 presents our proposed configurable MOL-based computational memory architecture. The proposed design and its configuration methodology are demonstrated by a case study of N-bit full addition in Section 4.6. Simulations and performance analysis are illustrated in Section 4.7. Comparison with relevant implementations in the literature is presented in Section 4.8. In order to further illustrate the potential of the proposed approach, two other application case studies are presented in Section 4.9, related to in-memory computing of CRC and DNN. Finally, Section 4.10 provides a short summary of the chapter.

4.1 Memristive devices for in-memory computing

The aggressive growth in the size of processed data in addition to the increasing numbers of processing cores have lead to intensive data traffic between memory and processing cores. In-memory computing have been introduced to overcome the memory wall problem. Instead of sending large amount of data to the processing cores, part of the tasks are computed inside the memory. This reduces the memory accesses bottleneck and can significantly improve performance. Computing within memristive memories are motivated by the unique properties of memristors and their versatile nature. In this context, several recent contributions have been proposed to enable computation within memristive memory arrays and can be classified in two categories.

The first category involves using the memristor as single-level cell (SLC) [56, 106, 102, 103, 101, 107, 108]. The second category includes work that uses the memristor as multi-level cell (MLC) or analog cell [109, 110, 111]. MLC-based computing is promising when targeting applications with intensive multiply-accumulate operations, such as convolutional neural networks (CNN) [111]. However, a number of challenges remain in terms of manufacturability and computational accuracy regarding device variability, pattern-dependent current leakage and the area overhead of peripheral circuits [112]. Major semiconductor foundries have not included MLC technology in their development roadmaps in the near future [111].

In contrast, SLC cells have a larger readout margin that makes them tolerant against process variation and resistance drift effects. Based on SLCs, different logic design styles have been introduced together with different realizations on memristive crossbar arrays. IMPLY [56] and MAGIC [53] have been introduced to enable in-memory logic operations. Although promising results are demonstrated, IMPLY and MAGIC techniques still impose specific technology and design constraints. For instance, in order to attain acceptable performance in these techniques, the ratio R_{OFF}/R_{ON} of the adopted memristive devices should be relatively high. Moreover, authors of [67] have reported that IMPLY does not ensure binary resistance switching of memristors in some cases.

More recently, other in-memory computing techniques have emerged as alternatives. Among these, the memristor-based majority (MAJ) [63] has been introduced to overcome the aforementioned limitations. However, other downsides arise at the architectural level. MAJ design style is relatively complex in terms of peripheral circuits as well as excessive in-out data movement which in turn impacts latency. Detailed analysis of these limitations is provided in the next section.

4.2 Limitations of existing logic design styles

As illustrated in chapter 1, several memristive design styles have been widely explored in various fields. In this work, our target application concerns in-memory computing. Therefore, for the rest of this chapter, some logic design styles are excluded such as MRL, MAD and MTL. The rest design styles including MAGIC, IMPLY, MAJ and CRS are considered. Limitations that accompanies these design styles are discussed below.

Authors of [102][61, 113, 104, 114] have presented several approaches where logic functions are broken down into several MAGIC or IMPLY operations. These operations are then performed sequentially inside memristive crossbar arrays. However, these approaches have several design constraints:

- The analysis in [67] reveals that IMPLY cannot achieve the full resistance switching of the output memristor in case both input memristors of the IMPLY gate are in the R_{OFF} resistance state. Hence, the corresponding state of the output memristor is not fully digital.
- Output memristors in IMPLY and MAGIC may be subjected to state drift [56][102].
- The performance of these design styles is highly dependent on the technology of the adopted memristive device (e.g. requirement of memristive devices with high R_{OFF}/R_{ON} ratio) [56][102].
- The corresponding basis functions provided by IMPLY and MAGIC are not diverse enough to allow fast logic mapping with minimum computational cycles.

MAJ-based logic design has been recently explored by several authors [63][107]. MAJ relies on a digital representation of memristors, so the limitations faced in IMPLY and MAGIC can be overcome. However, at the architecture level, other downsides arise:

- In-memory computing architectures based on MAJ, which are available in the literature, require additional load operations, which read data bits outside the memory. This induces the overheads in terms of total critical path, number of cycles and the complexity of the dedicated control unit.
- Architectures based on MAJ involve significant modifications in the peripheral circuitry of the memory. The write operations are performed on bit-lines (BLs) as well as word-lines (WLs) instead of BLs only.

These limitations hold also for CRS logic design approach [59] as it can be considered as a special case of MAJ.

4.3 MOL logic design

In this section we introduce a new memristor-based logic design style namely Memristor Overwrite Logic (MOL). MOL approach is highly adapted for computing within memristive crossbar arrays and avoids the limitations encountered by pre-existing logic design styles.

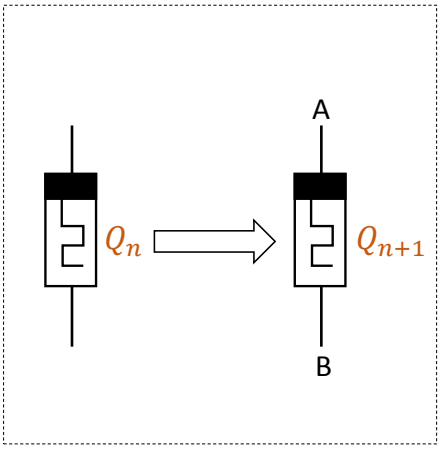
4.3.1 Digital representation of memristive devices

The nonvolatile internal resistance state of memristor could be changed according to the magnitude and duration of the applied bias across its terminals [23]. However, a non sufficient magnitude or duration leads to an intermediate resistance state R where $R_{ON} < R < R_{OFF}$. In this case, the state of the memristor can not be considered as binary, which in turn leads to more sophisticated modeling of the internal state of memristive devices in the analog domain. However, in a digital design, we could think about the memristor as a two-state element where its resistance $R \in \{R_{ON}, R_{OFF}\}$ and ignoring any other intermediate states if we succeed to guarantee a sufficient magnitude and duration of the bias across its terminals. Based on this understanding, the internal state of a memristor is defined in the digital domain. Let Q_n be the current internal state of a memristor while Q_{n+1} is the next state after applying a new external bias represented by A and B as shown in Fig. 4.1(a). Hence, Q_{n+1} will a be function of the logical states at the terminals A and B and the previous internal state Q_n. By considering all the possible combinations of A, B and Q_n as shown in Fig. 4.1(b), the state equation of a memristive device is expressed as follows:

$$Q_{n+1} = Q_n A + Q_n \overline{B} + A\overline{B} = M_3(A, \overline{B}, Q_n) \tag{4.1}$$

where M_3 represents the 3-variable majority function, which is defined in [58]. This expression demonstrates that a majority function is an intrinsic feature of memristive devices [63]. Based on the Boolean expression presented in (4.1), the finite state machine (FSM) of a memristive device is demonstrated in Fig. 4.1(c). Accordingly, the equivalent latch circuit of a memristive device can be implemented as shown in Fig. 4.2 where Q is the internal state of the memristor. To translate the Boolean value of Q into a resistance between the terminals of the memristor, an analog multiplexer is added. It selects either one of the two resistors, which resistances are R_{ON} or R_{OFF} where $Q = 0$ and $Q = 1$ are mapped to R_{ON} and R_{OFF} respectively. Note that this schematic is valid and useful from the digital perspective, so it cannot be used for simulation in the analog domain.

(a)



(b)

Q_n	A	B	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(c)

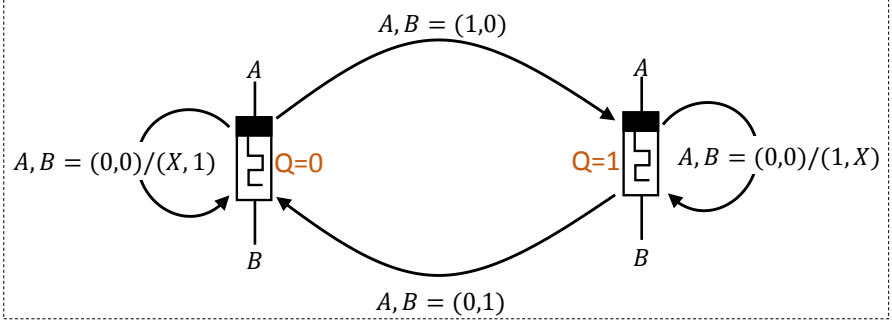


Figure 4.1: Memristor: (a) internal state after applying external bias represented by A and B; (b) truth table; (c) finite state machine (FSM)

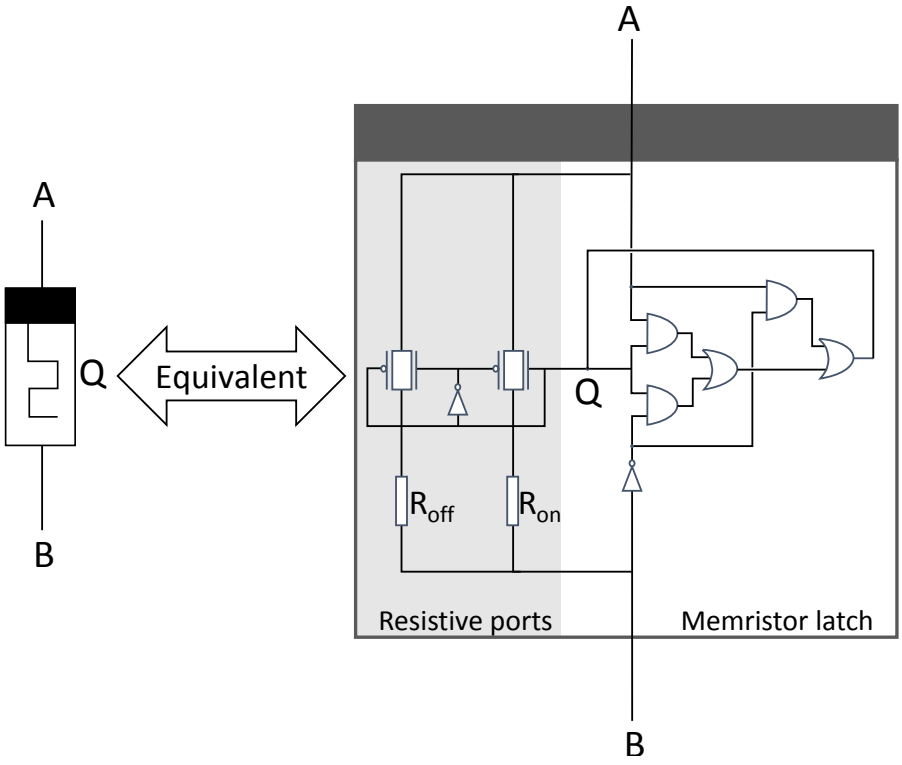


Figure 4.2: Equivalent latch circuit of memristor with binary resistive ports

4.3.2 MOL logic procedure

The state representation of memristor expressed in (4.1) clarifies its computational capability and simplifies its integration in the digital domain. Six possible cases can be derived from (4.1) and are listed in (4.2). Fig. 4.3 is an illustration of these cases. They are split into two groups. The first group includes the cases from 1 to 4, which correspond to MOL. In these 4 cases, a memristor acts as logic accumulator. The previously stored bit Q_n is subjected to OR/AND with the new input $A/\overline{B}$ while the other terminal of the memristor is set to logic "0" or logic "1" depending on the desired function. The obtained output is simultaneously saved in the form of new internal state Q_{n+1}. The remaining cases (i.e. 5 and 6) are achieved by initializing the memristor to a known state (logic "0" or logic "1"). The inputs A and B are sent to the memristor ports simultaneously. The output is saved as the new internal state (Q_{n+1}) of the memristor. In fact, these two cases correspond to CRS logic operations that are explored in the literature [108][63][59].

Although MOL operations are special cases of the 3-variable majority, working with MOL is much simpler. MOL highly resembles the conventional write operation. One end of each memristor is reserved for the input operands, while the other end is employed for selection. In contrast, MAJ employs both terminals of the memristor for the input operands. This makes MOL more adapted to crossbar memory arrays.

$$Q_{n+1} = \begin{cases} Q_n + A\,, & B = 0, \quad case:1 \quad (MOL) \\ Q_n A\,, & B = 1, \quad case:2 \quad (MOL) \\ Q_n + \overline{B}\,, & A = 0, \quad case:3 \quad (MOL) \\ Q_n \overline{B}\,, & A = 1, \quad case:4 \quad (MOL) \\ A\overline{B}\,, & Q_n = 0, \quad case:5 \quad (CRS) \\ A + \overline{B}\,, & Q_n = 1, \quad case:6 \quad (CRS) \end{cases} \tag{4.2}$$

The same concept applies to a vector of bits. Fig. 4.4 illustrates that two consecutive steps are enough for achieving MOL operations on an N-bit vector. In step 1, which is presented in Fig. 4.4(a), the input vector $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ is written into the N memristors by mapping logic "0" and logic "1" to the normalized voltage levels $-1V$ and $1V$ respectively while the common horizontal line is set to $0V$. At the end of this step, the resulting state of a given memristor M_k is $Q_k = I_k$. In step 2, the same N memristors are overwritten with the input vector $A = [A_{N-1}\ A_{N-2}...\ A_1\ A_0]$. However, the input voltage level on the common horizontal line is set to $0V$ or $1V$ depending on the desired operation. For the case of MOL-OR (Fig. 4.4(b)), B is set to $0V$ and the result, which is stored in a given memristor M_k, is $Q'_k = A_k + I_k$. For the case of MOL-AND (Fig. 4.4(c)), B is set to $1V$ and the result, which is stored in a given memristor M_k, is $Q'_k = A_k I_k$.

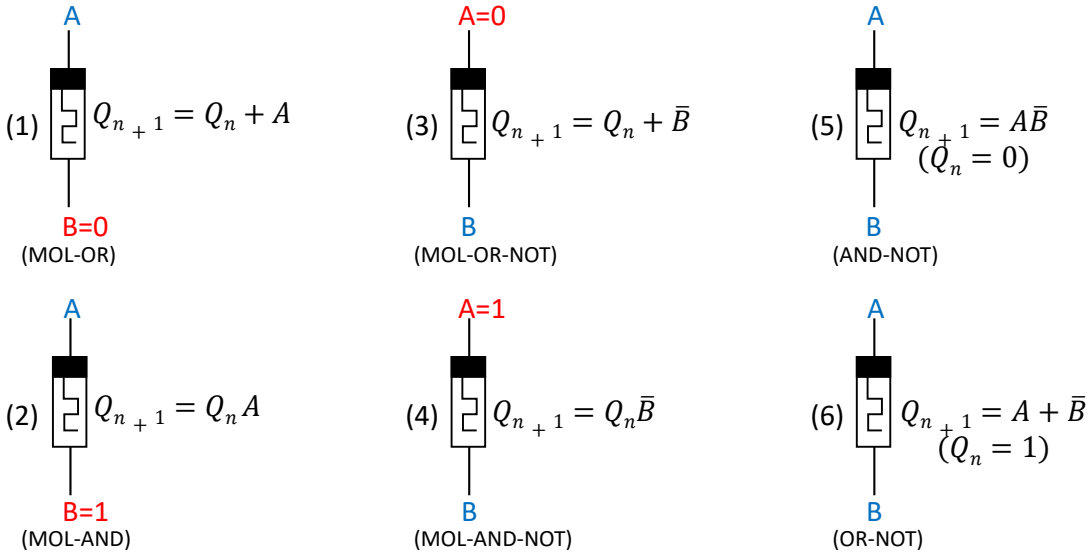


Figure 4.3: Six possible logic cases performed by a memristor

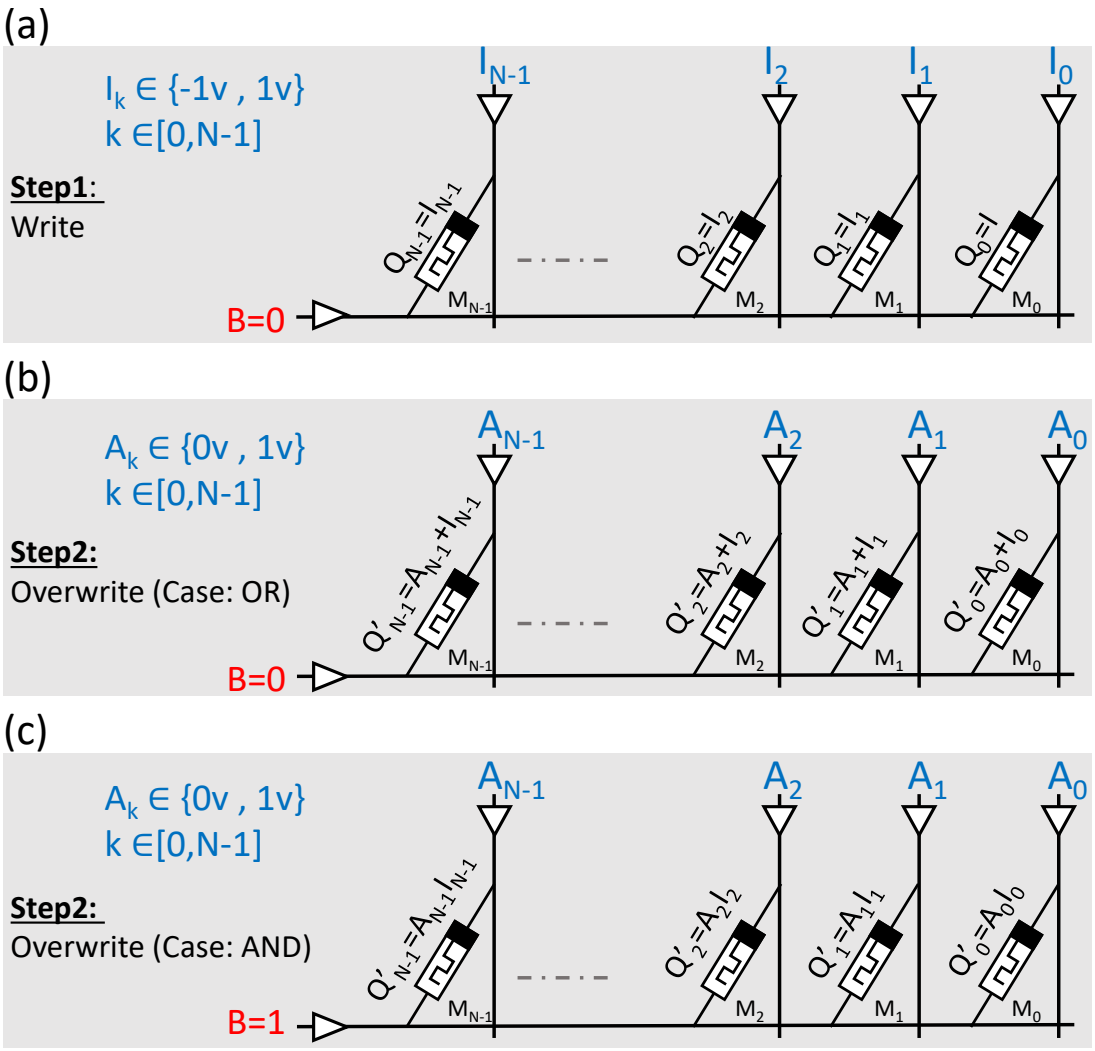


Figure 4.4: Performing MOL on a vector of bits; (a) writing N-bits into memritsors; (b) overwrite step to perform MOL-OR; (c) overwrite step to perform MOL-AND

4.3.3 Performing MOL inside memristive crossbars

The proposed MOL can be performed in memristive crossbar arrays. The input data bits to the crossbar can be either written or combined logically with the currently stored bits inside the crossbar. This can be simply achieved by choosing the appropriate normalized voltage levels for representing the arriving bits (i.e. -1/1 for write and 0/1 for MOL). Fig. 4.5(a) illustrates that a single or multiple rows of the crossbar could be selected for either MOL-OR or MOL-AND operations with the incoming data bits $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ being applied on the columns. Similarly, Fig. 4.5(b) shows that a single or multiple columns of the crossbar could be selected for either MOL-OR-NOT or MOL-AND-NOT operations with the incoming data bits of the vector I applied on the rows.

4.4 Realization of MOL in 1M/1T1M crossbars

Crossbars constitute the core element of emerging memristive memories (e.g. RRAMs and MRAMs). Integrating MOL with crossbar-memory architectures can lead to promising enhancements and provides additional computational capabilities to these memories. However, this imposes updating memory peripheral drivers to cope with MOL operations in addition to its main storage function. Fig. 4.6(a) presents the proposed memory architecture which is capable of performing MOL. As illustrated in Section 4.3, write and overwrite operations could be performed along the rows as well as the columns of the crossbar. However, in a conventional memory architecture, the flow of the incoming data bits is along bit-lines only while the word-lines are reserved for addressing. Thus, MOL operation, which is similar to a write operation, could be only performed along BLs. In this case, MOL-OR and MOL-AND are the only supported logic operations in the proposed memory architecture. The architecture shown in Fig. 4.6(a) can be configured in four different modes:

1- Write mode: The input N-bit vector $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ is first mapped via bit-line driver (BLD) into the normalized voltage levels of $-1V$ and $1V$ corresponding for logic "0" and "1" respectively. Fig. 4.7(a) presents the schematic of BLD at the transistor level. The respective voltage levels ($-1V$ and $1V$) are then provided to the BLs of the memristive crossbar through the Isolation Block (ISO), which acts in this mode as a connecting switch. Fig. 4.7(b) illustrates the internal structure of ISO. Simultaneously, the enabled addressing decoder selects a single WL. The selected WL is supplied with a voltage V_{SEL}, which is already shared to the input of each transmission gate corresponding to every WL. The shared voltage V_{SEL} is set to the normalized voltage level of $0V$. The unselected WLs remain floating in the high impedance state (Z).

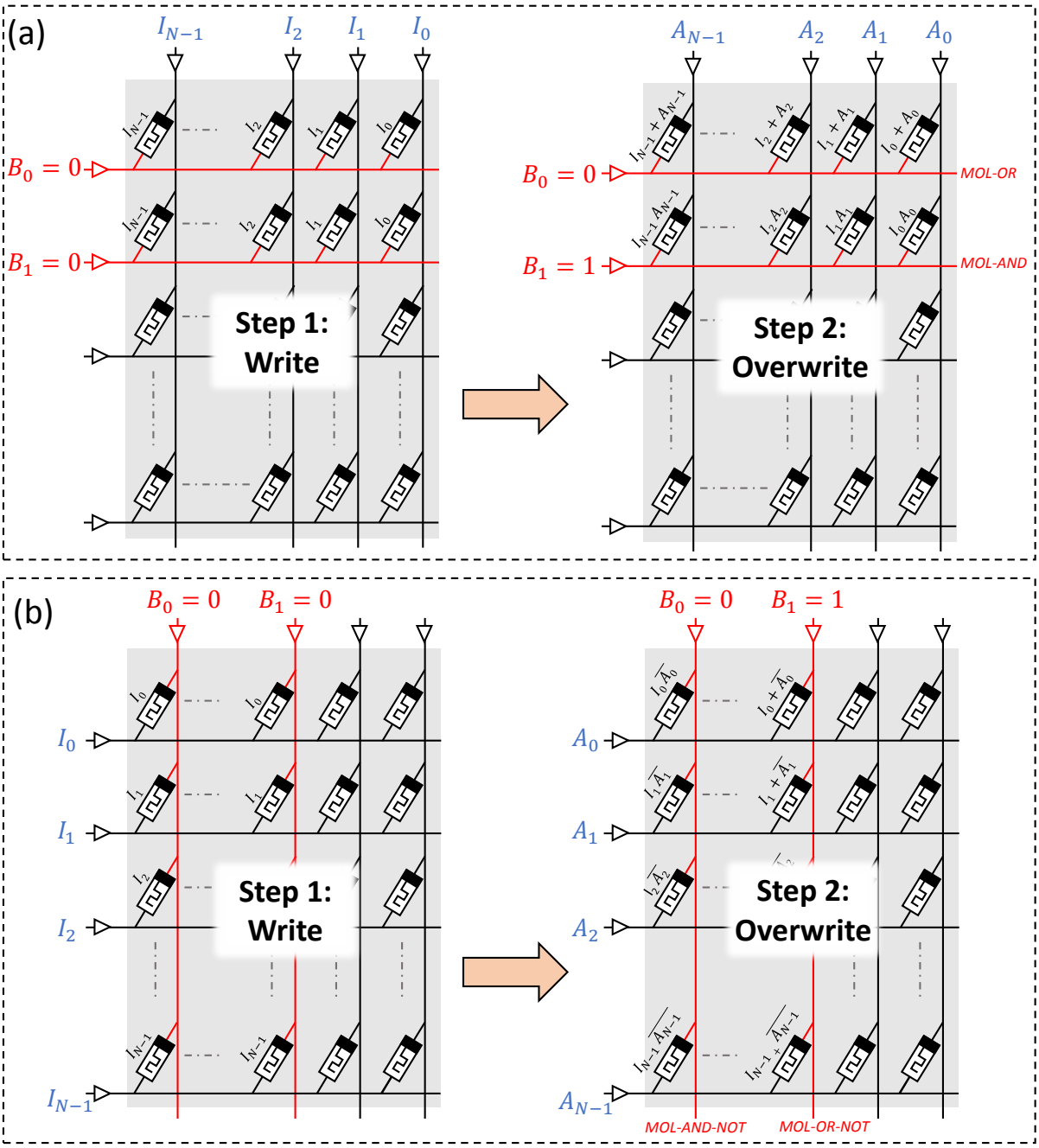


Figure 4.5: MOL inside memristive crossbar: (a) MOL-OR or/and MOL-AND; (b) MOL-OR-NOT or/and MOL-AND-NOT

2- Overwrite mode: In this mode, the function of the memory is switched to perform MOL among its memristive crossbar. As stated above, both MOL-OR and MOL-AND have to be supported. For the case of MOL-OR, the input data bits are mapped to the normalized voltage levels of $0V$ and $1V$ corresponding for logic "0" and logic "1" respectively. The addressing decoder performs its normal selection function for a single WL. ISO is kept at the connecting state. The level of V_{SEL} is also set to $0V$ as in the case

of write mode. The resulting bits of the MOL-OR operation are simultaneously stored in the selected WL. MOL-AND is performed similarly but V_{SEL} is switched to the high voltage level (i.e. $V_{SEL} = 1V$).

3- Read mode: In this mode, a single WL is selected to sense the corresponding states of its allocated memristors individually. BLD is isolated using the ISO block, which acts in this case as an open switch. The selection voltage V_{SEL} is set to $0.5V$ (normalized). The sensing current generated through each memristor have to guarantee a stability of its internal state (no state drift). A sensing amplifier (SA) circuitry, whose architecture is illustrated in Fig. 4.7(c), is used to measure the voltages across the reference resistors of respective resistances R. R is chosen to be the mid value between R_{ON} and R_{OFF} (i.e. $R = (R_{ON} + R_{OFF})/2$). By considering $R_{ON} < R_{OFF}$, the voltage across a reference resistor, which is in series to the sensed memristor, would be either in the neighborhood of $0V$ or $0.5V$. Depending on the state of the sensed memristor, the three cascaded inverters magnifies this difference leading to $-1V$ or $1V$ to the output.

4- Idle mode: In this mode, the memory is not active. The memristive crossbar is totally isolated to preserve its internal state. The IB block is in the isolation mode. Hence, all BLs are in the high impedance state (Z). Moreover, the address decoder is disabled. Thus, none of the WLs is selected, keeping them in the Z state. The architecture presented in Fig. 4.6(a) adopts the 1-memristor (1M) configuration for the structure of the crossbar. In other words, each cell consists of one memristor which connects the vertical and horizontal nano-wires of the crossbar. However, the 1M crossbar configuration suffers from the sneak paths phenomenon [30]. Sneak paths correspond to current paths through unselected cells in a memristive array. These undesired paths lead in some cases to a drift in the state of unselected memristive cells during write or overwrite operations. Moreover, it gives false estimation about the real logical state of a given selected memristor during reading mode. This phenomenon degrades the overall memory performance. Several efforts have been devoted in the literature to overcome sneak path phenomenon [30] [38] [115]. All proposed methods are limited to a certain crossbar size. Thus, increasing the size of the memristive crossbar beyond a certain limit will eventually lead to the sneak paths. A possible solution to stop these paths is to use a selector in series with each allocated memristive cell. This solution induces overheads in terms of the total utilized area of the memory which in turn loses the ultra high density attained in the 1M case. In [116], a transistor is used as a selector. Thus, each cell inside the memory consists of one transistor in series with one memristive device (1T1M). The obtained crossbar architecture for the 1T1M configuration is considered as sneak-path free. Fig. 4.6(b) presents our proposed 1T1M memory architecture with added MOL capabilities. The

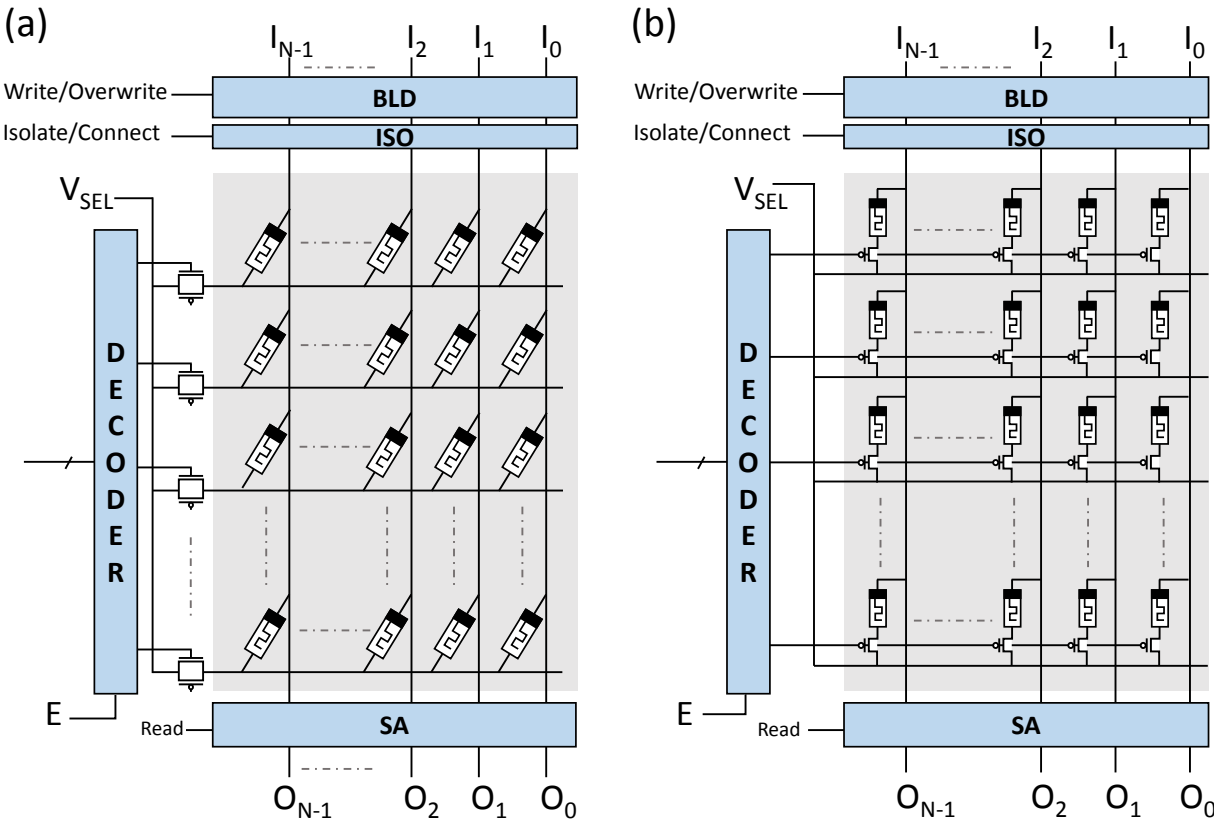


Figure 4.6: Memory architecture performing MOL: (a) 1M configuration; (b) 1T1M configuration

WL transmission gates, that have been used in the 1M case are no longer used in the case, of 1T1M memory architecture. Normally, each transmission gate is equivalent to two MOSFETs. Thus, for an $N \times M$ memristive crossbar array, additional $NM - 2N$ MOSFETs are used in the 1T1M architecture compared to that in the 1M case. The obtained 1T1M architecture has the same four control modes previously introduced for the 1M case.

4.5 MOL-based Computational memory

In this section, a MOL-based computational memory architecture is introduced. The architecture is able to perform MOL operations between two stored word lines. The original architecture, which is formed of two interconnected MOL memory blocks, works in a complementary manner.

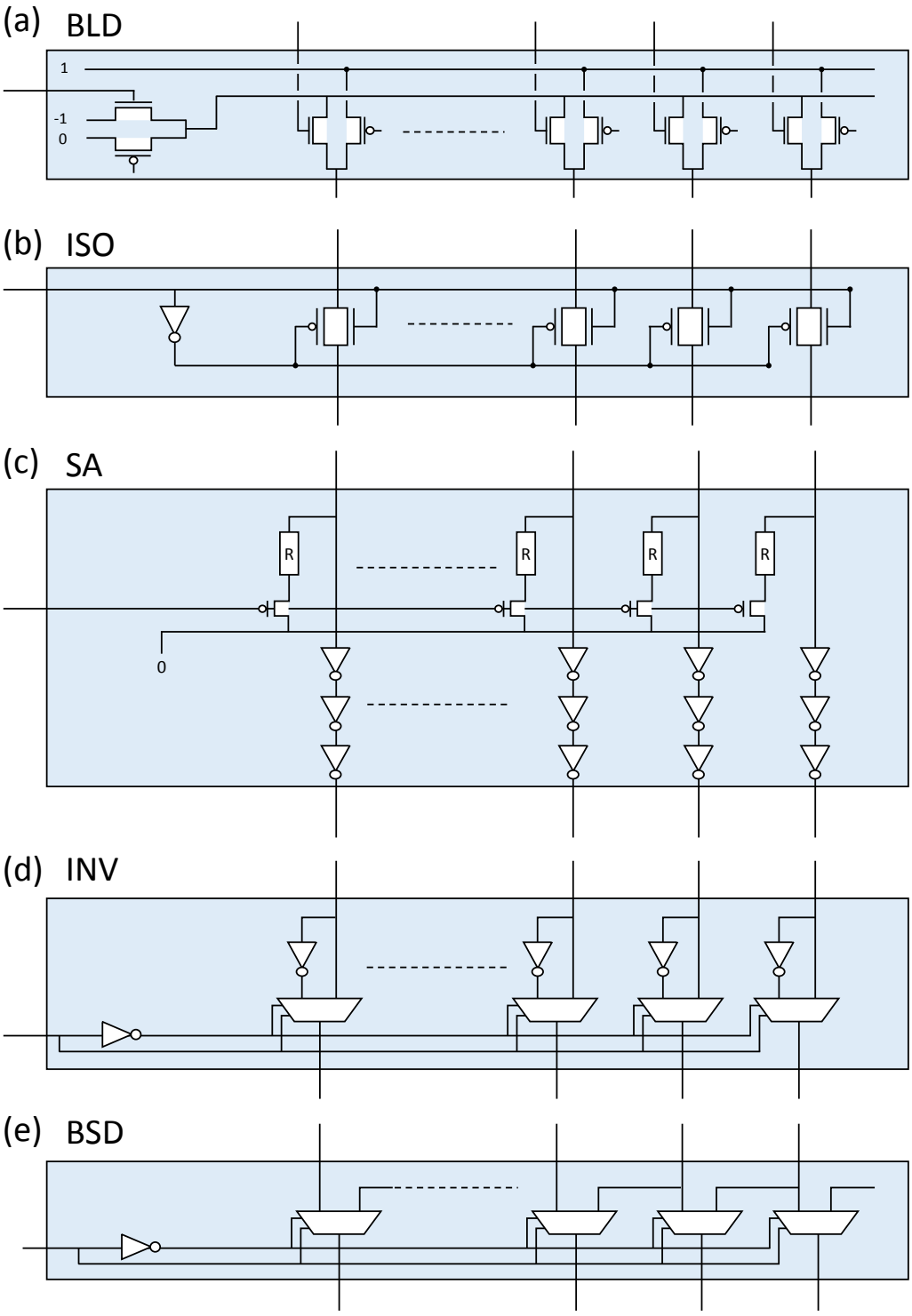


Figure 4.7: Drivers architectures for the proposed MOL-memory approach

4.5.1 Architecture

The proposed MOL-memory architectures, which are presented in section 4.4, act as logic accumulators for the newly arriving bits. In other words, computation in such memory is restricted for logic accumulation. Accordingly, performing general Boolean functions

in this memory requires an additional process to load the stored data bits outside the memory. These additional load operations are at odds with the concept of computation inside the memory. To overcome these load operations, we propose the use of two coupled MOL memories (MOL-memory-A and MOL-memory-B), that work in complementary manner. At each time step, one of these memories acts as source of input data-bits of the second memory. The second memory performs MOL with the previously stored bits in its memristive crossbar. Fig. 4.8 illustrates our proposed computational-memory architecture. The architectures of MOL-memory-A and MOL-memory-B are identical. A controlled inverting driver (INV) is added after the sensing stages of the two memories. The function of this driver is to achieve a complete logic, as the OR and AND logic operations supported by the memories are not universal. So, additional NOT operation is needed to allow the description of any Boolean function. The architecture of INV is illustrated in Fig. 4.7(d). A 1-bit barrel shift driver (BSD) is added to enable bit-level operations in addition to vector-level operations. The BSD is responsible for ensuring switchable connections between the two memory blocks. It can be reconfigured either to pass the data bits or to shift them on the fly with no need for an additional cycle. The architecture of the BSD is presented in Fig. 4.7(e). The proposed MOL-memory architecture presented in Fig. 4.8 is capable of performing numerous operations including logic computation and storage. Table 4.1 lists the most important (not all) operations that could be achieved. For each listed operation, a set of appropriate commands are simultaneously sent to the blocks constituting the architecture. A single operation requires one computational step. As an example, the case 19 in Table 4.1 corresponds to the arithmetic operation expressed in (4.3)

$$M_B(n) = M_B(n) \;\; AND \;\; \overline{M_A(m)} \tag{4.3}$$

where $M_A(m)$ and $M_B(n)$, are the bit-vectors located at the addresses m and n corresponding for MOL-memory-A and MOL-memory-B respectively. For this case, MOL-memory-A is set to the read mode. It reads the bit-vector $M_A(m)$, which undergoes a bitwise inversion through INV block. The 1-bit shifter is disabled. Simultaneously, MOL-memory-B, is set to the overwrite mode to perform MOL-AND with the vector $M_B(n)$. The result of the bitwise logic operation replaces the previous vector $M_B(n)$. The process is performed during one computational step.

4.5.2 Performing general arithmetic tasks

Generally, an arithmetic function (e.g. addition, subtraction, compare, etc.) could be expressed in Boolean form. Accordingly, breaking the Boolean form into several MOL

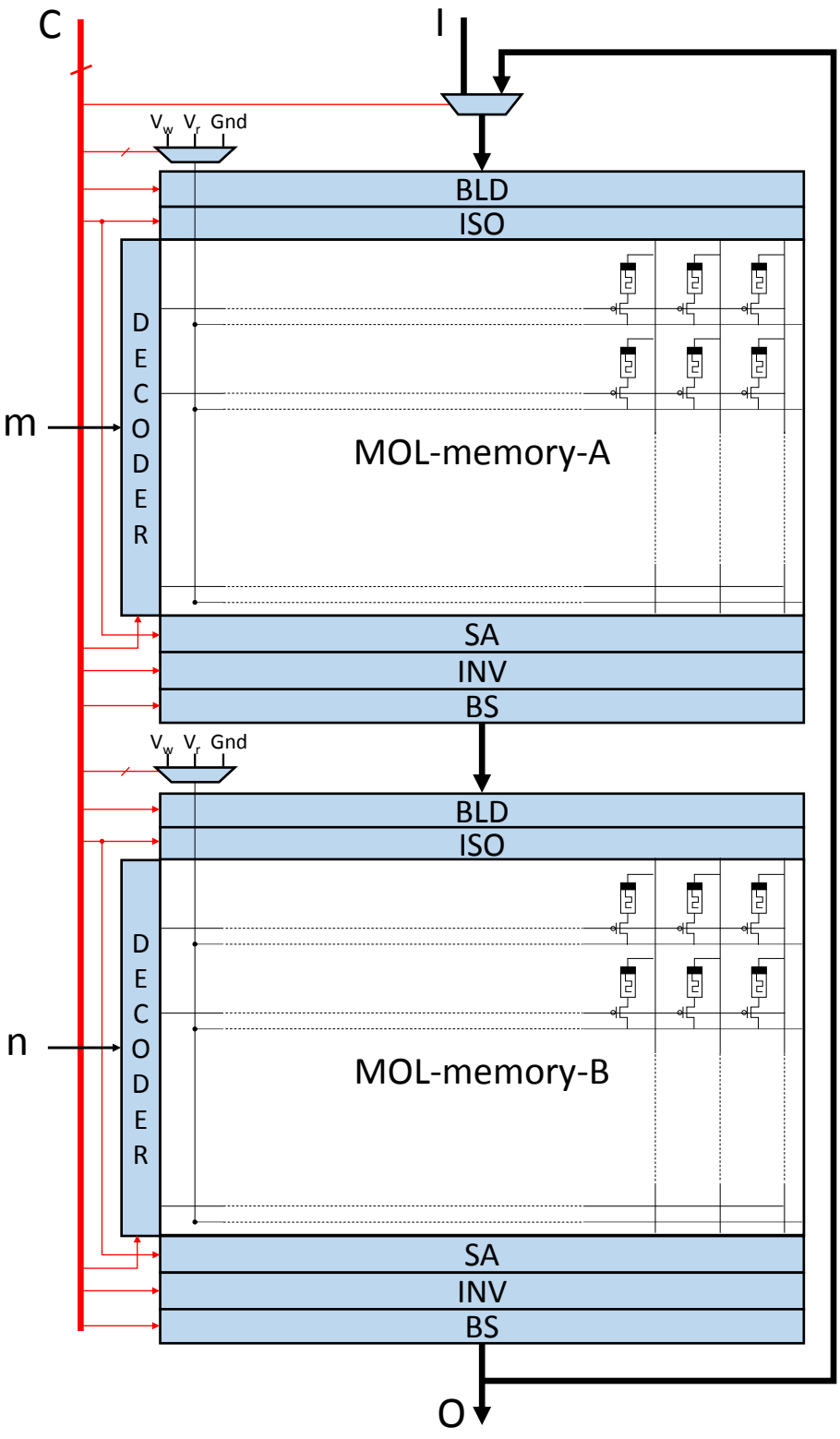


Figure 4.8: Computational Memory Architecture

operations allows its execution inside the proposed computational memory. Thus, the execution of an arbitrary Boolean function requires several computational steps so MOL operations are executed iteratively to finalize the desired arithmetic task. For this pur-

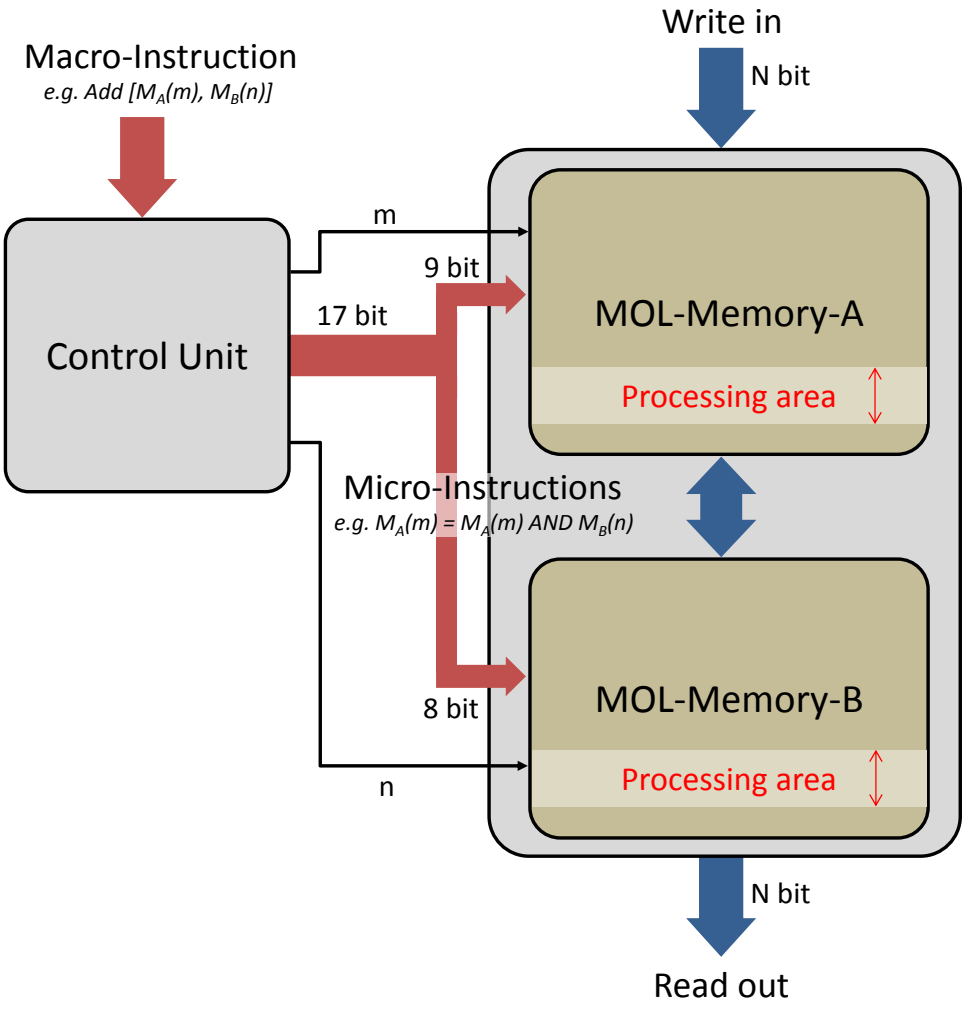


Figure 4.9: Architecture diagram of MOL-based computational memory with its dedicated control unit

pose, an external controller, which arranges these iterative operations is needed. Fig. 4.9 shows the block diagram which illustrates the general structure of the memory and the controller. When the controller receives an instruction from the processor, it decides the role of the memory whether for storage or computation. Specifically, for the case of computation, the controller breaks the received macro-instruction into several iterative micro-instructions, which can be performed by the proposed memory. In our case, micro-instructions correspond to the set of operations listed in Table 4.1. A processing area should be reserved in each of MOL-memory-A and MOL-memory-B in the proposed computational memory. The area could be dynamically changed according to the need (such as the number of required tasks). Moreover, the location of the processing area could be also changed periodically. The reason for location change is to attain better endurance for the memristive memory cells that are subjected to continuous stress.

Table 4.1: Encoding table

		110 101 011 011 1 0				AND Read Write OR Pass Invert						AND Read Write OR Pass Invert			Select-out Select-in
			Enable Disable	Write Overwrite	Isolate Connect				Enable Disable	Write Overwrite	Isolate Connect				
Operation	**Binary**	**Micro-Instruction**	$\mathbf{E_A}$	$\mathbf{BLD_A}$	$\mathbf{ISO_A}$	$\mathbf{Mode_A}$	$\mathbf{INV_A}$	$\mathbf{BSD_A}$	$\mathbf{E_B}$	$\mathbf{BLD_B}$	$\mathbf{ISO_B}$	$\mathbf{Mode_B}$	$\mathbf{INV_B}$	$\mathbf{BSD_B}$	**Sel**
0	00000	$M_A(m) = I$	1	1	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
1	00001	$M_B(n) = I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
2	00010	$O = M_A(m)$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	1	1	x
3	00011	$O = M_B(m)$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	0	1	x
4	00100	$O = \overline{M_A(m)}$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	0	1	x
5	00101	$O = \overline{M_B(m)}$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	1	1	x
6	00110	$M_A(m) = M_B(n)$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
7	00111	$M_B(n) = M_A(m)$	1	x	0	1 0 1	1	1	1	1	1	0 1 1	x	1	x
8	01000	$M_A(m) = \overline{M_B(n)}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
9	01001	$M_B(n) = \overline{M_A(m)}$	1	x	0	1 0 1	0	1	1	1	1	0 1 1	x	1	x
10	01010	$M_A(m) = M_A(m)\ AND\ I$	1	0	1	1 1 0	1	1	0	x	0	1 1 1	x	1	0
11	01011	$M_B(n) = M_B(n)\ AND\ I$	0	0	1	0 1 1	0	1	1	1	1	1 1 0	1	1	0
12	01100	$M_A(m) = M_B(n)\ OR\ I$	1	0	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
13	01101	$M_B(n) = M_B(m)\ OR\ I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
14	01110	$M_A(m) = M_A(m)\ AND\ M_B(n)$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	1	1
15	01111	$M_B(n) = M_B(n)\ AND\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	1 1 0	x	1	x
16	10000	$M_A(m) = M_A(n)\ OR\ M_B(n)$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
17	10001	$M_B(n) = M_B(m)\ OR\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	0 1 1	x	1	x
18	10010	$M_A(m) = M_A(m)\ AND\ \overline{M_B(n)}$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	1	1	1
19	10011	$M_B(n) = M_B(n)\ AND\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	1 1 0	x	1	x
20	10100	$M_A(m) = M_A(m)\ OR\ \overline{M_B(n)}$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
21	10101	$M_B(n) = M_B(m)\ OR\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	0 1 1	x	1	x
22	10110	$M_A(m) = M_B(n) << 1$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
23	10111	$M_B(n) = M_A(m) << 1$	1	x	0	1 0 1	1	0	1	1	1	0 1 1	x	1	x
24	11000	$M_A(m) = M_A(m)\ AND\ [M_B(n) << 1]$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	0	1
25	11001	$M_B(n) = M_B(n)\ AND\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	1 1 0	x	1	x
26	11010	$M_A(m) = M_A(n)\ OR\ [M_B(n) << 1]$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	0	1
27	11011	$M_B(n) = M_B(m)\ OR\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	0 1 1	x	1	x
28	11100	$M_A(m) = \overline{M_B(n) << 1}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
29	11101	$M_B(n) = \overline{M_A(m) << 1}$	1	x	0	1 0 1	0	0	1	1	1	0 1 1	x	1	x

4.5.3 Towards an efficiency-improved computing

Usually, the execution on a processor of a certain arithmetic task, that requires loading data bits outside the memory, introduces long latency. This is either caused by movement of data between various levels of the memory-hierarchy and the processing unit as in conventional von Neumann model or even long critical path in the case of more advanced methods such as near-memory computing. These movements constitute the major challenge in terms of energy efficiency and computing latency and is usually denoted as the memory wall. In-memory computing has been developed to overcome the memory wall by avoiding the long latency originated from these data movements. For our proposed MOL-based computational memory, the execution of an arithmetic task requires several successive MOL-operations. The high bandwidth of in-memory computing allows to minimize the step period of each operation. However, when the complexity of arithmetic tasks scales up, the corresponding number of MOL-operations becomes large which again causes considerable latency and energy consumption. For these tasks, conventional computing could be more efficient. Accordingly, in this work we consider that in-memory computing is a complementary approach to the conventional von Neumann model. In

other words, arithmetic tasks should be decided whether to be executed in classical way (outside memory) or based on MOL in order to improve computational efficiency.

4.6 MOL based in memory N-bit full addition

In this section, an N-bit full addition is considered as a case study to evaluate the functionality of our proposed computational memory architecture.

4.6.1 Proposed iterative N-bit full addition process dedicated for computational MOL-memory

Generally, full adder is the basic digital building block for several computational operations (i.e. addition, subtraction and multiplication). Thus, implementing a full addition process inside the memory is the first step toward in-memory computing. Equations (4.4) and (4.5) present the well known expressions of the 1-bit full addition.

$$S = A \oplus B \oplus C_{in} \tag{4.4}$$

$$C_{out} = AB + BC_{in} + AC_{in} \tag{4.5}$$

where A and B are the inputs, C_{in} is the input carry value, S is the 1-bit adder output and C_{out} is the output carry. The operator $\oplus$ corresponds to the boolean XOR. Assume that all the inputs are initially stored in the memory. The boolean functions of S and C_{out} are written in the form of sum of products (SoP), so that their expressions could be mapped into the proposed computational memory using sequential MOL operations. The inputs of a given MOL operation should be aligned on the same columns (ie. same bit-lines) in the memory, otherwise, a pre-shifting process is required to align the corresponding inputs. Accordingly, the number of steps required to achieve the computation of S and C_{out} is affected by the relative positions of the input A, B and C_{in} inside the memory. In order to minimize the number of computational steps as well as reserve the minimum possible processing area, a dedicated N-bit addition process is proposed. The process uses a specific sequence of each operation listed in Table 4.1. Consider the two N-bit vectors A^N and B^N. The addition of A^N and B^N leads to the vector sum S^{N+1}. Normally, the additional 1-bit in S^{N+1} is reserved for the expected overflow in the addition process. We propose to follow the procedure illustrated in Algorithm 1 to achieve a vector level addition of A^N and B^N:

- Stage 1: The vector sum S_0 which is of length $N + 1$ is initialized by the bitwise XOR of A^N and B^N. Similarly, the vector carry C_0 of length $N + 1$ is initialized by

the bitwise AND of A^N and B^N. The expressions of S_0 and C_0 are presented in (4.6) and (4.7) respectively.

$$S_0 = A \oplus B \tag{4.6}$$

$$C_0 = AB \tag{4.7}$$

- Stage 2: Each time, a new vector sum S_{i+1} and vector carry C_{i+1} are created based on their previous values S_i and C_i respectively. Equation (4.8) and (4.9) demonstrate the respective expressions of S_{i+1} and C_{i+1}. This process is repeated $N-1$ times.

$$S_{i+1} = S_i \oplus (C_i << 1) \tag{4.8}$$

$$C_{i+1} = S_i\,(C_i << 1) \tag{4.9}$$

The operator "<< 1" stands for the 1-bit shift to the left. At the end of this iterative process, the final obtained vector S_{N-1} corresponds to the sum of A^N and B^N while C_{N-1} will be a zero vector.

Algorithm 1 N-bit addition dedicated for computation inside MOL-memory

```
1: procedure ADD(A, B)                          ▷ A and B are N-bit vectors
2:     S_0 ← A ⊕ B
3:     C_0 ← AB
4:     for i ← 0 to N − 2 do
5:         S_{i+1} ← S_i ⊕ (C_i << 1)
6:         C_{i+1} ← S_i(C_i << 1)
7:     end for
8:     return S_{N−1}                           ▷ The sum of A and B
9: end procedure
```

4.6.2 In-memory N-bit full addition procedure

The proposed iterative N-bit addition process can be mapped into the computational MOL-memory using the operations listed in Table 4.1. Fig. 4.10 shows a space-time representation of the N-bit full addition process, which is realized within MOL-memory-A and MOL-memory-B. For each computational step, the new contents of the memories are listed in a new single column in Fig. 4.10. Assume the case where the two vectors A^N and B^N, that are subjected to addition, are initially stored inside MOL-memory-A at the addresses m_1 and m_2 respectively. Additional two word-lines have to be reserved inside MOL-memory-B to attain the addition of A^N and B^N. The two stages that are presented in section 4.6.1 are realized as follows:

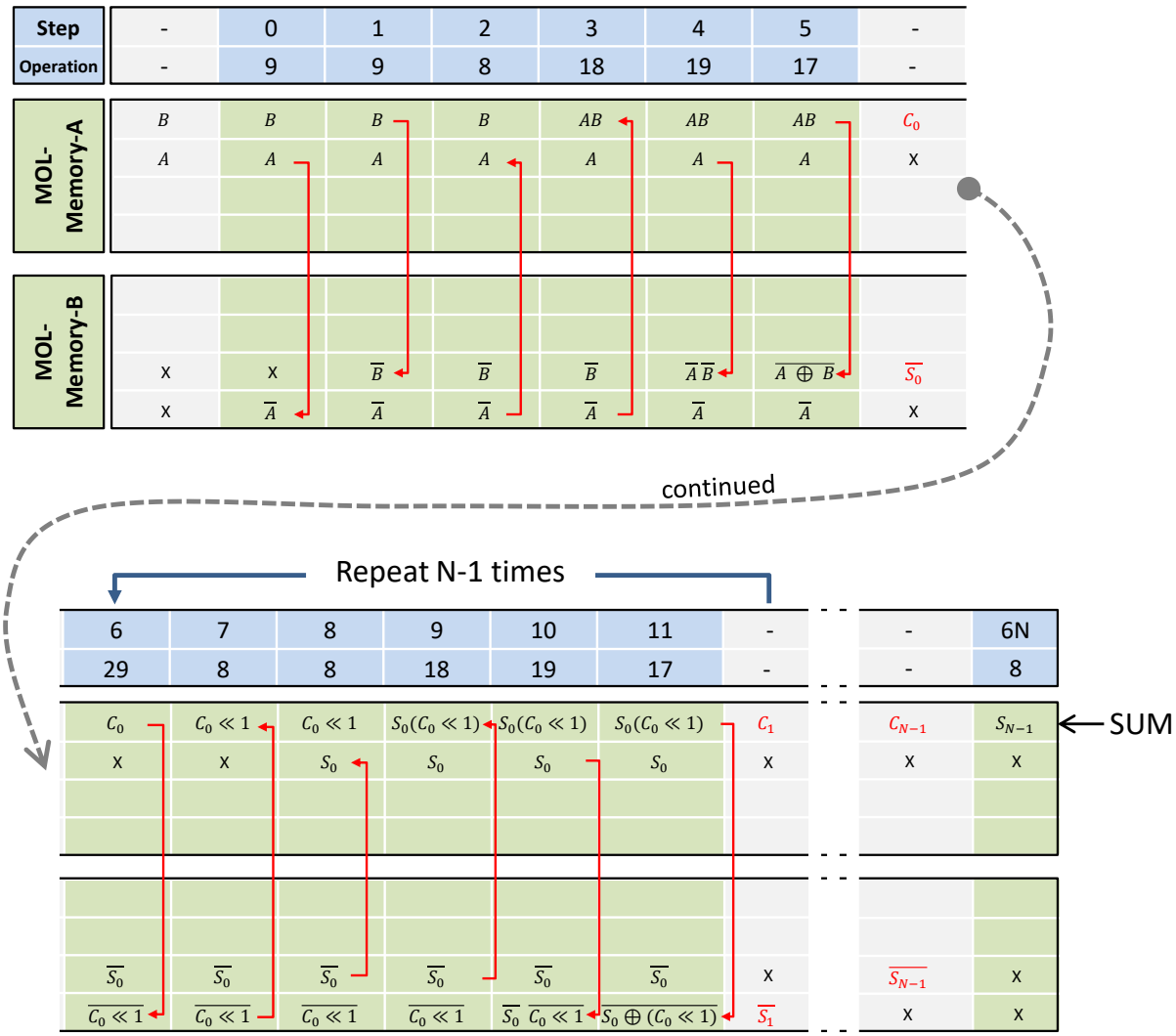


Figure 4.10: Operations sequence for an in-memory N-bit addition process using MOL-memory

- Stage 1: Corresponds to the steps between 0 and 5 using the six micro-operations that have the sequence order shown in Fig. 4.10. At the end of this stage, the bitwise AND of A and B (i.e. $C_0 = AB$) is stored in MOL-memory-A while the XNOR of A and B (i.e. $\overline{S_0} = \overline{A \oplus B}$) is stored in MOL-memory-B.

- Stage 2: In this stage, the steps between 6 and 11 are repeated N-1 times. Their corresponding micro-instructions have the sequence order shown in Fig. 4.10. Each time the initial vector C_i is shifted to the left by one bit and the resulting vector undergoes bitwise AND with the initial vector S_i. The obtained result is referred as C_{i+1}, which expression is presented in (4.9). Simultaneously, the shifted version of C_i undergoes bitwise XNOR with the initial vector S_i to obtain the new vector sum S_{i+1}. At the end of this process, the vector $\overline{S_{N-1}}$ is stored in MOL-memory-B. Thus,

an additional step is required to make a bitwise inversion of the obtained vector. The resulting vector S_{N-1}, which represents the N-bit addition of the vectors A and B, is stored in MOL-memory-A.

4.6.3 Space-time analysis of the N-bit addition process

The total number of computational steps required to complete the N-bit addition is $6N+1$ steps as shown in Fig. 4.10. The total number of memristors reserved for the execution of the N-bit addition is $4N$ memristors corresponding to four rows of the MOL-memory architecture. These rows include the initial locations of A and B, although the initial bits of the vectors A and B are lost. However, in some cases, the destruction of the input vectors is undesired, especially when these inputs are required for another computational tasks. In order to avoid this case, pre-copy operations of the two input vectors A and B could be performed to reserve safe versions of these vectors. Thus, two additional computational steps are required for this case and the new total number of computational steps becomes $6N + 3$. The considered operation sequence in Fig. 4.10 corresponds to the case where A and B are both located in MOL-memory-A. However, another two cases should be considered also: (i) If A and B belong to different MOL-memories, one additional pre-copy operation could be performed to drag the input vector contained in MOL-memory-B to MOL-memory-A. (ii) If A and B are both contained in MOL-memory-B, two additional pre-copy operations are needed to drag them to MOL-memory-A. These pre-copy operations are performed to maintain the same operation sequence, which is presented in Fig. 4.10. Pre-copy operations can be avoided with different sequences (one for each case, with common parts).

4.7 Simulation and performance analysis

In this section, we study the performance of the proposed computational memory architecture which is implemented using a realistic model of Magnetic Tunnel Junction (MTJ) device and a CMOS 65nm technology node. The study includes timing analysis, energy consumption and robustness against device variability.

4.7.1 Adopted memristive device

Several memristive devices have been explored in the literature. In fact, MOL technique could apply to all types of bipolar memristive devices holding two resistance states R_{ON} and R_{OFF}. Among these devices, memristors such as HfO_x [40] and TiO_2 [23] ex-

hibit promising characteristics with their high switching speed (sub-ns) and their high R_{OFF}/R_{ON} ratio (> 100). However, current memristor technologies suffer from endurance limitations. Although several efforts have been carried out to enhance endurance [40], the allowed number of switchings per memristor is still limited in the range of 10^6 to 10^{12} for the best case. This value is relatively low for targeting intensive computations inside memristive crossbars. The Spin Transfer Torque Magnetic Memory (STT-MRAM) [117], which have been redescribed in terms of memristive systems [118], is considered as one of the most promising nonvolatile memories (NVM). STT-MRAM is eligible for high reliability applications [119] due to its high endurance (> 10^{15}) [23]. As illustrated in Fig. 4.11, an MTJ cell is mainly composed of two ferromagnetic layers sandwiching an ultra-thin tunnel barrier. The resistance of the MTJ cell depends on the relative orientation of magnetization in the free and reference layers. The low resistance state (logic '0') of the MTJ corresponds to the parallel configuration (P) with resistance R_P, while its high resistance state (logic '1') is reached in the case of anti-parallel configuration (AP) with resistance R_{AP}. The magnitude of the applied current I must exceed a critical value noted as I_{C0} to allow switching. In contrast to memristors, MTJs are characterized by

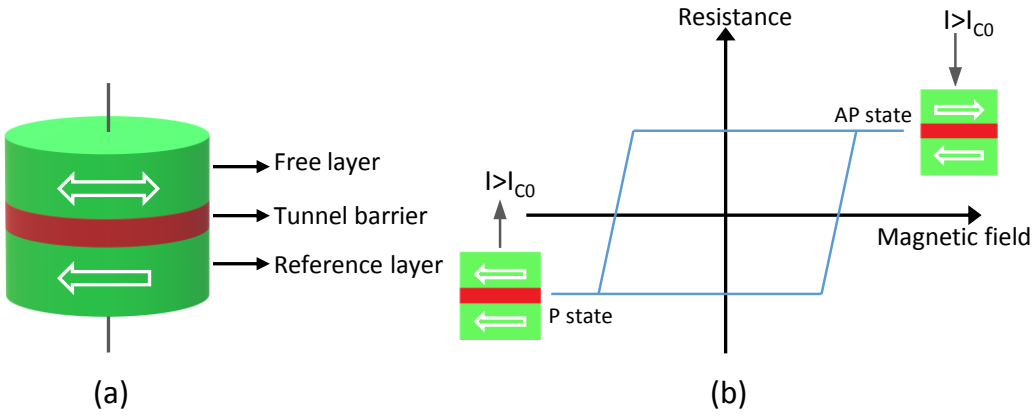


Figure 4.11: Typical MTJ: (a) Core structure, (b) Resistance variation

a relatively low margin between R_P and R_{AP}. The corresponding margin is commonly evaluated as the Tunnel Magnetoresistance (TMR) ratio, whose expression is presented in (4.10):

$$TMR = \frac{\Delta R}{R_P} = \frac{R_{AP} - R_P}{R_P} \tag{4.10}$$

However, such a low margin has no effect on switching an MTJ cell but on the corresponding sensing mechanism of the state of this cell. This requires a more complicated sensing driver to estimate and decide the corresponding state of a given selected row inside the memory. In this work, we have used MTJs with perpendicular magnetic anisotropy (PMA). The adopted PMA MTJ is formed of CoFeB/MgO/CoFeB layers. The physical

model describing the static, dynamic and stochastic behaviors of the STT PMA MTJ is presented in [120] and [121]. In order to fit with experimental results in the literature, the technology parameters corresponding to the material composition are kept at their default values. Other parameters which depend on the designers' choice are presented in Table 4.2 with their corresponding values. Fig. 4.12 shows the switching behavior of an MTJ device when it is fed with a square signal of amplitude $1.2V$. τ_{AP-P} and τ_{P-AP} correspond to the switching delays from AP to P state and the reverse case respectively. In fact, switching delay varies according to the applied voltage level. Fig. 4.13 illustrates the variation of τ_{AP-P} and τ_{P-AP} with respect to the applied voltage level. The graph indicates that switching delay decreases with the increase of the voltage while switching from P to AP state is faster than the reverse operation (i.e. $\tau_{AP-P} < \tau_{P-AP}$).

The choice of the memristive device type is not constrained by a specified MOL requirement. It can be observed from the mechanism of MOL technique that it involves direct access to the terminals of memristive devices which highly resembles conventional write operation. During the operation of MOL, the potential difference between the terminals of the memristive device always attains a binary level. Accordingly, MOL can be implemented in a wide range of memristive memories without specifying particular device features. In contrast, the structure of pre-existing logic design styles either establishes a series connection of a resistor (e.g. IMPLY) or series connection of the memristive devices (e.g. MAGIC) for normal operation. This undoubtedly prevents direct access to the memristive device terminals and consequently imposes specific device constraints, such as the requirement of sufficient HRS/LRS ratio and/or operated with thresholds type devices only.

Table 4.2: Adopted variables and parameters for PMA MTJ device

Parameter	**Value**	**Description**
t_{ox}	$0.85\,nm$	Thickness of oxide barrier
$TMR(0)$	70%	TMR ratio with 0 stress voltage
$Area$	$\pi \times 20\,nm \times 20\,nm$	MTJ surface
t_{sl}	$1.3\,nm$	Thickness of free layer

4.7.2 Performance analysis

Transient simulation has been conducted for the proposed design of the MOL-memory architecture. Based on the adopted STT PMA MTJ device and the CMOS $65nm$ process,

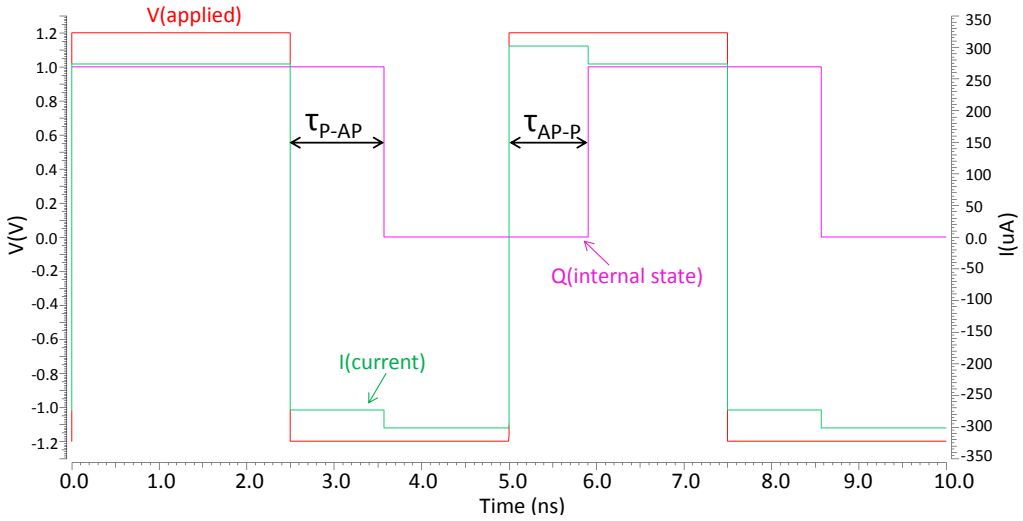


Figure 4.12: Switching behavior of MTJ device when fed with square signal

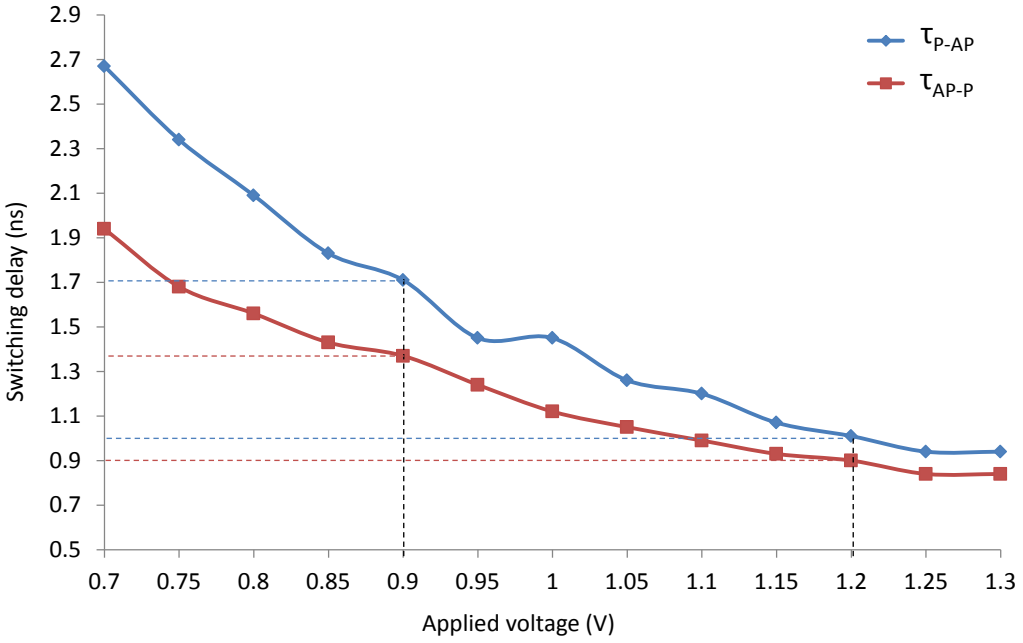


Figure 4.13: Switching delay of an MTJ cell as function of applied voltage level

simulations have been carried out using Cadence Virtuoso toolset. In order to evaluate the performance of the architecture, the N-bit addition process described in Section 4.6 is performed. The size of the crossbar is chosen to be 8×8 for MOL-memory-A as well as MOL-memory-B. The size N is chosen to be 8 bits for both numbers A and B. The corresponding operating voltage is set to $1.2V$ for logic '1' and $-1.2V$ for logic '0'. Based on the obtained transient results, total latency is evaluated as well as the total energy consumption. As an example, Fig. 4.14 presents the corresponding internal states of the 4 word-lines that are reserved for the 8-bit addition process, which is performed on the two arbitrary vectors A=[01011011] and B=[00111111]. The control signals of the

MOL-memory architecture follow the operation sequence presented in of Fig. 4.10.

4.7.2.1 Timing analysis

The first two steps correspond to the initialization of vectors A and B inside MOL-memory-A. The corresponding sum S=[10011010] is evaluated after $6N+1$ computational steps which is equal to 49 for $N = 8$. In fact, the max delay is noticed to be $\tau_{Max} = 1.7\,ns$ which is greater than the max switching delay of MTJ devices operating at $1.2V$. This is due to the voltage drop noticed along CMOS drivers. The actual voltage supplied to MTJ devices is $0.9V$ (could be interpreted from Fig. 4.13). This significant voltage drop (25%) is due to the adoption of low values of R_P and R_{AP}. Moreover, the width W of MOSFETs has a direct effect on the voltage drop percentage. This voltage drop could be mitigated by increasing W, but this induces overheads on the total area of CMOS drivers.

Therefore, the duration (T) of each computational step must be greater than τ_{Max}. The variability in τ_{Max} due to the stochastic switching behavior of MTJs should also be considered. Thus, an additional guard interval (τ_g) is introduced to guarantee the switching of the MTJs. The resulting step duration for the proposed MOL-memory architecture is $T = \tau_{Max} + \tau_g = 1.7 + \tau_g$. We set τ_g at $100\,ps$ which corresponds to 6% of τ_{Max}, so the duration T is equal to $1.8\,ns$. The minimum time required for finalizing the addition operation (neglecting the 2 initialization steps) is evaluated as $49 \times 1.8\,ns = 88.2\,ns$.

4.7.2.2 Robustness against resistance variability

Due to the limit of the manufacturing technology, the actual thickness of oxide layer and free layer of MTJ devices cannot be fixed at a constant value. They typically vary in a small range, but can lead to a relatively important variation in the values of LRS and HRS of MTJ. Therefore, we have examined the effect of MTJ resistance variability on the performance of our proposed MOL-based computational memory architecture. Simulations are conducted by performing the 8-bit addition. The adopted MTJ parameters TMR, t_{sl} and t_{ox} are kept as presented in Table 4.2 while subjecting them to a random process. The parameters are chosen to follow either uniform or Gaussian distribution. In Gaussian distribution, no error has been detected even when reaching a variation percentage of 21% for TMR, t_{sl} and t_{ox}. As for uniform distribution, the tolerated variation reaches 7%. This demonstrates the robustness of the proposed design against the resistance variability of MTJ devices.

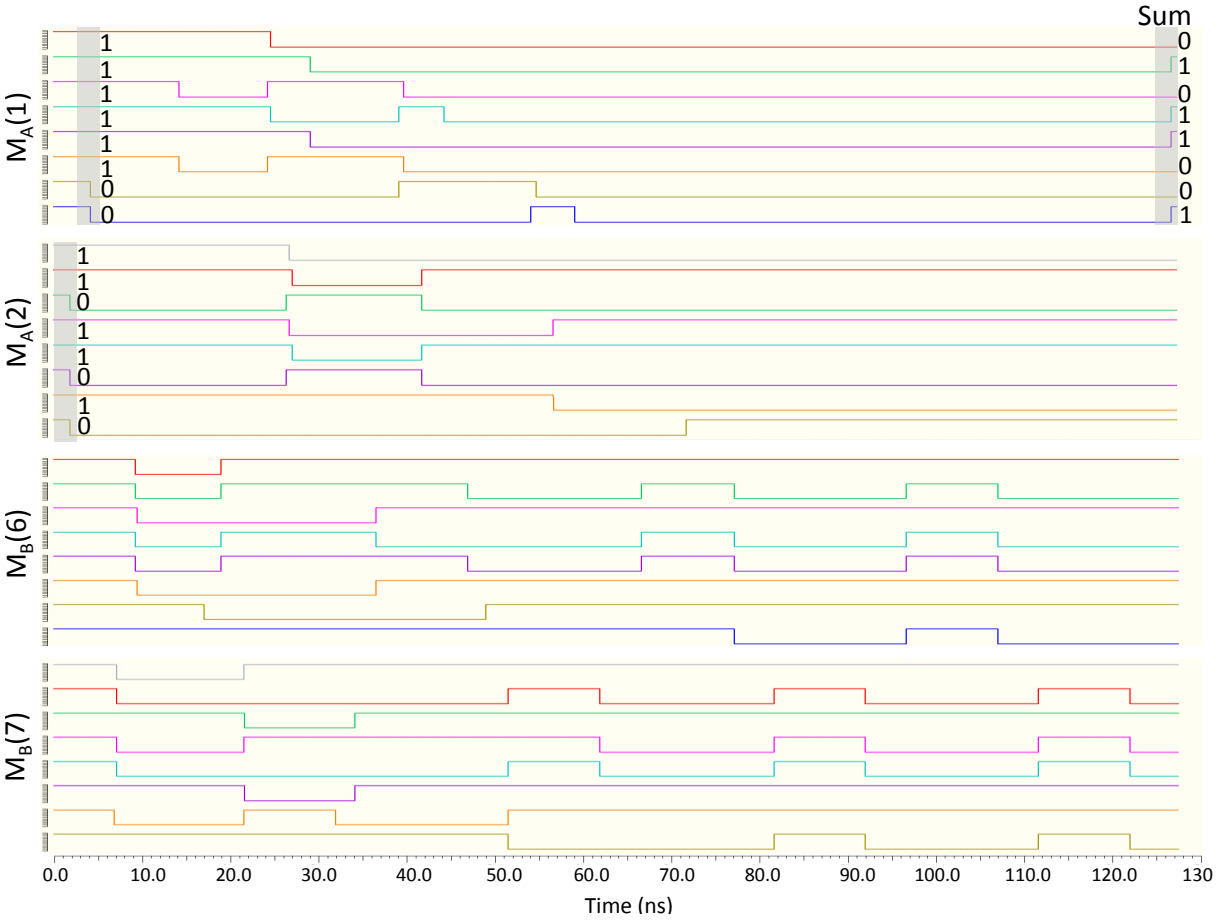


Figure 4.14: Transient simulation for the in-memory 8-bit addition process

4.7.2.3 Energy estimation

Energy consumption differs according to the operation: read, write or performing computation. In this section we will focus on the energy consumed by the memristive crossbar of the MOL-memory architecture neglecting the consumed energy by the peripheral drivers.

(i) Write-energy: Consider a single MTJ device located inside MOL-memory architecture. The energy consumed when a single bit is written into this MTJ device mainly depends on its previous resistance state (R_P or R_{AP}) and its final one. Hence, the 4 cases for write-energy are considered in (4.11).

$$\begin{aligned} E_{w_{0/0}} &= \frac{{V_w}^2}{R'_{AP}} T \\ E_{w_{0/1}} &= \frac{{V_w}^2}{R'_{AP}} \tau_{_{AP-P}} + \frac{{V_w}^2}{R'_P} (T - \tau_{_{AP-P}}) \\ E_{w_{1/0}} &= \frac{{V_w}^2}{R'_P} \tau_{_{P-AP}} + \frac{{V_w}^2}{R'_{AP}} (T - \tau_{_{P-AP}}) \\ E_{w_{1/1}} &= \frac{{V_w}^2}{R'_P} T \end{aligned} \tag{4.11}$$

where $E_{w_{i/j}}$ corresponds to the write-energy needed to put the MTJ device in state $i \in \{0,1\}$, after it was in the previous state $j \in \{0,1\}$; V_w and T are the write voltage and write duration respectively; R'_{AP} and R'_P represent the resistance states of 1T1M cell. $R'_{AP} = R_{AP} + R_{MOS}$ and $R'_P = R_P + R_{MOS}$. Generally, the values of i and j are not deterministic, but the 4 cases presented in (4.11) are considered as equiprobable, since there is no pre-knowledge about the data bits inside the memory as well as the bits that would be written. Thus, the average write-energy is estimated as the average sum of the 4 write-energy cases as presented in (4.12).

$$E_w = \frac{1}{4}\sum_{i,j} E_{w_{i/j}} = \frac{{V_w}^2}{R'_{AP}}\left(\frac{T}{2} - \frac{\Delta\tau}{4}\right) + \frac{{V_w}^2}{R'_P}\left(\frac{T}{2} + \frac{\Delta\tau}{4}\right) \tag{4.12}$$

where $\Delta\tau = \tau_{P-AP} - \tau_{AP-P}$. Assuming that the term $\frac{\Delta\tau}{4}$ is almost negligible compared to $\frac{T}{2}$, the overall expression in (4.12) is simplified in (4.13).

$$E_w \approx \frac{{V_w}^2}{2R_w}T \quad \text{with} \quad R_w = \frac{R'_P R'_{AP}}{R'_P + R'_{AP}} \tag{4.13}$$

R_w represents the equivalent resistance of two MTJs having opposite states and connected in parallel.

(ii) Read-energy: Reading a single MTJ device requires a sensing voltage V_r and a reference resistor R_{Ref} connected in series with a MOSFET. The total resistance of this 1T1R cell is R'_{Ref}. The corresponding state of the sensed MTJ device is assumed to be stable. The two possible cases for read-energy are presented in (4.14).

$$\begin{aligned} E_{r_0} &= \frac{{V_r}^2}{R'_{AP} + R'_{Ref}}T \\ E_{r_1} &= \frac{{V_r}^2}{R'_P + R'_{Ref}}T \end{aligned} \tag{4.14}$$

where E_{r_0} and E_{r_1} represents the required energy consumption for sensing AP and P states respectively during a period T. The corresponding average read-energy is expressed in (4.15).

$$E_r = \frac{{V_r}^2}{2R_r}T \quad \text{with} \quad R_r = \frac{(R'_P + R'_{Ref})(R'_{AP} + R'_{Ref})}{(R'_P + R'_{Ref}) + (R'_{AP} + R'_{Ref})} \tag{4.15}$$

(iii) Computation-energy: Computational operations that are performed inside MOL-memory architecture are classified into MOL or copy operations. Table 4.3 summarizes the energy consumed by each type of operation. Using the specifications of the adopted MTJ which are listed in Table 4.4, the average energy consumed by a MOL operation could be expressed as $E_{MOL} = E_w/2 + E_r = 0.196\,pj$ whereas that consumed by a copy operation is calculated as $E_{COPY} = E_w + E_r = 0.333\,pj$.

Normally, computation inside MOL-memory architecture is performed on N bits simultaneously. For the N-bit addition process which is performed within $6N+1$ cycles, $3N$ cycles corresponds to MOL operations while $3N+1$ cycles corresponds to copy operations. Thus, the overall consumed energy (E_T) could be expressed as in (4.16).

$$E_T = (3N)(N\,E_{MOL}) + (3N+1)(N\,E_{COPY}) \tag{4.16}$$

By substituting the corresponding values of E_{MOL} and E_{COPY} presented in Table 4.3, the expression of the total energy becomes $E_T = 1.587N^2 + 0.333N$. Specifically, for the 8-bit addition process, E_T is equal to 104.2 pJ. The value of the energy consumption extracted by simulation is 124.43 pJ.

Table 4.3: Energy consumed by a computational operation

In1	**In2**	**MOL-AND**	**MOL-OR**	**Copy**
0	0	$E_{w_{0/0}} + E_{r_0}$	E_{r_0}	$E_{w_{0/0}} + E_{r_0}$
0	1	E_{r_0}	$E_{w_{1/0}} + E_{r_0}$	$E_{w_{1/0}} + E_{r_0}$
1	0	$E_{w_{0/1}} + E_{r_1}$	E_{r_1}	$E_{w_{0/1}} + E_{r_1}$
1	1	E_{r_1}	$E_{w_{1/1}} + E_{r_1}$	$E_{w_{1/1}} + E_{r_1}$

Table 4.4: Specifications

Specification	**Value**
R_{AP}	$6\,K$
R_P	$3.97\,K$
R_{MOS}	$0.5\,K$
R_{Ref}	$4.8\,K$
V_w	$0.588\,V$
V_r	$0.9\,V$
τ_{AP-P}	$1.4\,ns$
τ_{P-AP}	$1.7\,ns$
T	$1.8\,ns$

4.8 Comparison

In this section, the proposed MOL-memory architecture has been compared with recently published relevant designs (listed in Table 4.5) targeting in-memory computing. The comparison has been carried out based on the performance of N-bit addition in terms of latency, energy consumption and utilized area. Note that the considered area incorporates only the memristors involved in the computation regardless of the size of the crossbar.

4.8.1 MOL vs IMPLY and MAGIC

- Except for the parallel approach in [56], our proposed design, which uses only $6N+1$ steps to perform addition, outperforms all IMPLY and MAGIC based designs listed in Table 4.5 in terms of number of computational steps. In fact, [56] uses the parallel approach which is intended to increase the level of parallelism in computation. However this approach requires significant modifications in the crossbar structure by adding connections between its rows. This leads to an increased area compared to the conventional crossbar structure.

- The step delay in our proposed design is $1.8\,ns$. Although the designs presented in [102], [104] and [103] adopt memristive devices that provide better step delay ($1.12\,ns$ to $1.43\,ns$), the total latency in our proposed design is still the minimum ($10.8N + 1.8\,ns$). The best case achieved with the competitor designs is recorded in [102] with $13N + 3.9\,ns$ (i.e. ~20% more latency).

- In the proposed design, $4N$ memristors participate in the execution of the N-bit addition. This number ranges from $11N - 1$ to $24N$ for the majority of the designs based on MAGIC, so our proposed design exhibits ×1.75 to ×5 area reduction. On the other hand, the IMPLY based serial approach [56], MAGIC based area optimized design [102] and the design presented in [106] use a fixed number of memristors to perform addition operation. In other words, the required number of memristors is independent of the size N of the addition operation. This area optimization comes at the cost of high number of computational steps (×2.5 to ×18.8).

- The average energy consumed in pJ for the memristive crossbar in our design is $1.5867N^2 + 0.333N$. This quadratic expression indicates a significant energy consumption in the order of $\times N$ as compared to the linear energy expressions for the other designs listed in the table. The reason for this energy gap is that for each step the same bitwise operation is performed on the whole word-line (size N). However, the other approaches from the literature perform 1 bit operation in each step.

Although our methodology induces overheads on the total energy consumption, working on the vector level rather than bit level greatly simplifies the corresponding control unit and reduces its complexity.

4.8.2 MOL vs MAJ and CRS

- Logic representation using MIGs has experimentally shown promising results in logic optimization [122]. Memristive devices can efficiently execute the intrinsic resistive MAJ operation. The authors of [63][107] present a programmable in-memory computing system namely Programmable Logic-in-Memory (PLiM). The instruction set for the PLiM architecture is based on the MAJ operation. As investigated in [107], the number of required memristors for the addition is $\sim 2N$, which is equal to 50% of that in our approach. However, the execution of an N-bit addition inside PLiM requires $15N$ cycles for the best case, which is $\times 2.5$ the number of cycles required in our proposed design. This high number of computational steps is related to the repeated read out operations of intermediate results, which impacts in addition the step delay and energy consumption (not evaluated in [107]).

- The number of computational steps achieved in [108], which uses the CRS approach, is less than that of our proposed computational memory. However, other parameters such as the step delay which is not investigated by the authors is expected to be greater. This is due to the fact that the presented architecture, based on two separated memory blocks, uses an intermediate control unit which reads data bits from one memory block and redistribute them along BLs and WLs of the other memory block. This process increases significantly the overall critical path and consequently the step delay. The number of memristive cells required in [108] is also less than that in our proposed design. However, it is clear that based on this approach, the reserved area corresponds to a fixed location inside the memory, as the input bits cannot be shared to all WLs especially for large memory sizes. This affects the endurance of memristive cells participating in the computation which are subjected to continuous stress.

As explained in Section 4.5.2, the proposed memristive computational memory is able to perform any general arithmetic function by breaking it into a netlist of iterative MOL operations. As MOL is based on the primitive AND/OR operations, the ABC tool [123], which has been employed for existing logic design styles [104][114], could be also leveraged in order to realize the synthesis task. This will be considered in our future work.

Table 4.5: Comparison of different logic families for N-bit addition in terms of area, latency and energy consumption

Reference	**Method**	**# Steps**	**Step delay**	**Latency (ns)**	**Area (# memristive cells)**	**Energy (pJ)**
(This work)	MOL	$6N + 1$	$1.8ns$	$10.8N + 1.8$	$4N$	$1.587N^2 + 0.333N$
[56]	IMPLY Serial	$29N$	-	-	2	$\sim 9.5N$
[56]	IMPLY Parallel	$5N + 18$	-	-	$6N - 1$	$\sim 9.5N$
[106]	IMPLY	$89N$	-	-	4	-
[102]	MAGIC Area optimized	$15N$	$1.3ns$	$19.5N$	5	$\sim 3.365N$
[102]	MAGIC Latency optimized	$12N + 1$	$1.3ns$	$15.6N + 1.3$	$11N - 1$	$\sim 3.365N$
[102]	MAGIC Transpose I	$15N + 1$	$1.3ns$	$19.5N + 1.3$	$22N - 3$	$\sim 6.53N$
[102]	MAGIC Transpose II	$10N + 3$	$1.3ns$	$13N + 3.9$	$13N - 3$	$\sim 4.72N$
[103]	MAGIC	$12N + 1$	$1.12ns$	$13.44N + 1.12$	$14N + 1$	$0.684N$
[104]	MAGIC (Naive mapping)	$12N$	$1.43ns$	$17.6N$	$15N$	$0.684N$
[104]	MAGIC (Compact mapping)	$16N$	$1.43ns$	$22.8N$	$24N$	$0.894N$
[101]	MAGIC	$20N + 15$	$1.89ns$	$37.8N + 28.35$	12	$0.3N$
[107]	MAJ (Naive)	$\sim 22N$	-	-	$\sim 4N$	-
[107]	MAJ (MIG rewriting)	$\sim 16N$	-	-	$\sim 3N$	-
[107]	MAJ (Rewriting and compilation)	$\sim 15N$	-	-	$\sim 2N$	-
[108]	CRS (PC-Adder)	$2N + 4$	-	-	$2N + 1$	-
[108]	CRS (TC-Adder)	$4N + 5$	-	-	$N + 2$	-

4.9 Applications of MOL

As discussed earlier in this chapter, MOL is highly eligible for in-memory computing. Thus, various applications which suffers from the intensive data transfers (read/write) can employ MOL technique to bypass (or at minimum reduce) the memory wall problem.

In order to further illustrate the potential of the proposed MOL approach, we investigated in this section the possibility of using it in two different application domains. First, we introduce the implementation of cyclic redundancy check (CRC) algorithm inside our proposed computational memory. CRC is a well known error-detecting code, commonly used to ensure data integrity in digital communications and storage devices. Then, we explore the use of MOL in the field of neural networks. We present an architecture design that targets deep neural networks (DNN) computation. The design is capable of performing the weighted accumulation process, which is considered the main cause of data traffic in DNN systems.

4.9.1 In-memory CRC computing

4.9.1.1 Cyclic redundancy check

Cyclic redundancy check is an error-detecting code based on the theory of cyclic error-correcting codes [124]. It is commonly used in digital data transmission and storage systems to ensure data integrity and detect accidental changes in raw data [125].

The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection, was first proposed by W. Wesley Peterson in 1961 [124]. Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors.

As illustrated in Figure 4.15, a CRC-enabled device calculates a short, fixed-length binary sequence, known as the check value or CRC, for each block of data to be sent or stored. The CRC is equal to the remainder of a polynomial division of a data block. The calculated CRC is then appended to the data, forming a codeword.

When a codeword is received, the CRC can be checked simply by performing the polynomial division on the received bit-stream and comparing the remainder (also called syndrome) with an expected residue constant (usually zero). Specification of a CRC code requires the definition of a so-called generator polynomial, which becomes to the divisor of this polynomial division, If the remainder doesn't match the expected residue, then it can be considered that the data block contains errors.

In case of detecting an error, the device may take corrective action or requesting that the data block be sent again. Otherwise, the data is assumed to be error-free (although,

with some small probability, it may contain undetected errors).

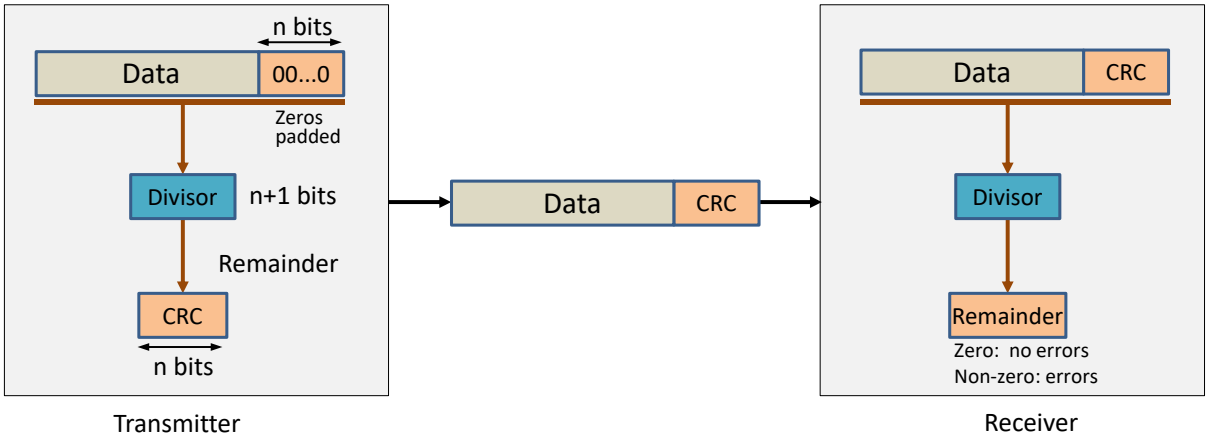


Figure 4.15: Cyclic redundancy check

4.9.1.2 CRC computation

To compute an n-bit binary CRC code, a polynomial division is performed at the transmitter side using the basic XOR and shift operations. The basic idea is illustrated in Figure 4.16 through a simple example with a CRC length n= 3 and a generator polynomial (divisor) equal to "1011" (length n+1). The CRC algorithm acts as follows [124]:

1. Align the bits representing the input in a row.
2. The input bits are right padded by zeros of length n bits, which corresponds to the length of the CRC code.
3. Position the (n+1) bit pattern representing the divisor underneath the left-hand end of the row.
4. Perform a bitwise XOR of the polynomial divisor with the bits above it.
5. Repeat the process on the remainder of the previous step. However, the divisor is now shifted one bit to the right. If the leftmost bit of the remainder is zero, the divisor shifts over to align with the next 1. The process is repeated until the divisor reaches the right-hand end of the input row.
6. The final result obtained is the n bits at the right-hand end of the row. These n bits are the remainder of the division, and corresponds to the value of the CRC.

The validity of a received message can easily be verified by performing the above calculation again, this time with the CRC value right padded instead of zeros. The remainder should equal zero if there are no detectable errors.

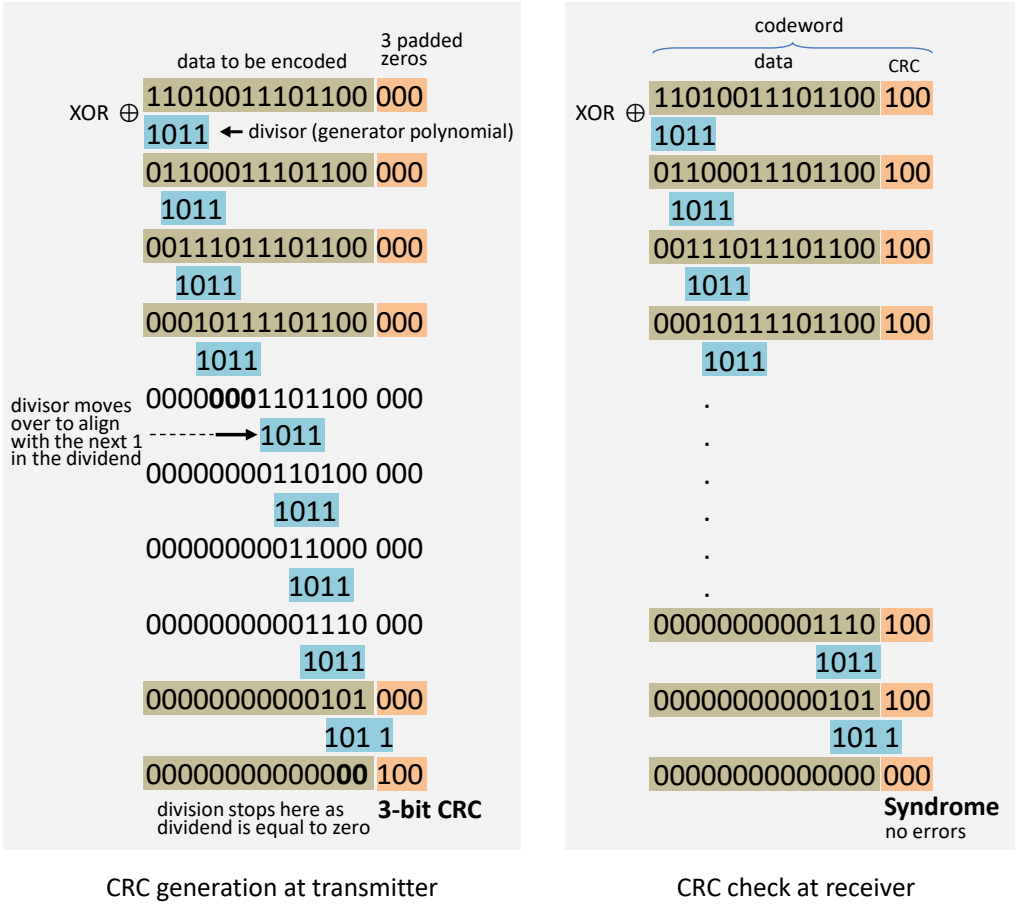


Figure 4.16: Example of a cyclic redundancy check generation

4.9.1.3 MOL-based in-memory CRC computation

As discussed earlier, the algorithm for performing CRC is based on iterative bitwise XOR and shift operations. Hence, CRC computation can be simply achieved inside the memory as it doesn't involve complex arithmetic functions. Specifically, our proposed MOL-based computational memory is highly adapted for such bitwise operations:

1. Resources: In fact, the ability to implement CRC algorithm inside the memory for some kind of applications can eliminate the need for a dedicated separate logic unit, as CRC can be trivially realized inside our proposed computational memory.

2. Flexibility: Due to the ability to store variant types of generator polynomials (divisor), in-memory CRC computation can simply supports various standards and specifications (e.g. CRC-16, CRC-32, CRC-64 [124]). The maximum supported CRC specification depends on the width of the memory.

3. Detection of memory faults: The implementation of error-detection codes such as CRC purely inside the memory can be of great interest as it can be employed to

detect memory errors occurring inside the memory itself. Such errors can even be caused by defected cells (e.g. at the level of fabrication [126][127]) in the crossbar array or due to the state drift of cells which becomes significant after a predetermined period of memory usage (or number of read/write operations).

4.9.1.4 Simulation results

Simulations are carried out on the proposed computational memory architecture to show the realization of CRC computation inside the memory. The device parameters of the memory including MTJ and CMOS are kept as they are presented in Section 4.7. The width of the memory (N) is also kept at 8 bits for simplicity. Out of the available 8-bit width, 5 bits are reserved for data and the rest (3 bits) are reserved for the generated CRC.

The memory is initialized by the data bits vector (A) as well as the generator polynomial bits (P). A 4-bit generator polynomial is used in this case. Appropriate sequence of micro-instructions are sent to the memory. This sequence has been derived from the CRC computation steps described above. The micro-instructions follow the space-time diagram presented in Fig. 4.17. It requires $6N-1$ steps to execute the CRC generation. At the transmitter side, an additional step is required to append the CRC to the initial data vector. Hence, the final codeword is obtained in $6N$ steps. Fig. 4.18 shows a transient simulation of the codeword generation inside the memory. Arbitrary data and generator polynomial are used in this example.

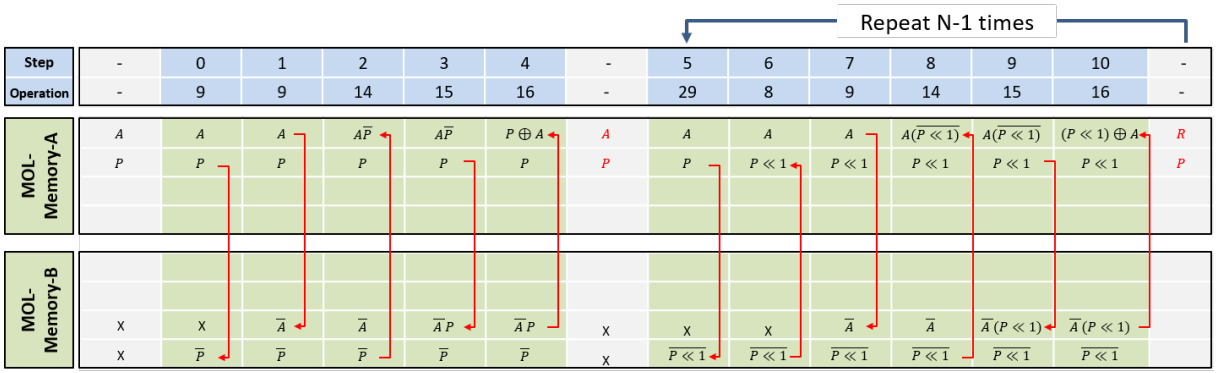


Figure 4.17: Operations sequence for in-memory CRC computation

Similarly, for the validation of the correctness of the data, the whole codeword is divided by the generator polynomial using the same sequence of micro-instructions. As shown in Fig. 4.19, the resulting syndrome is equal to zero, which indicates that the codeword is error free.

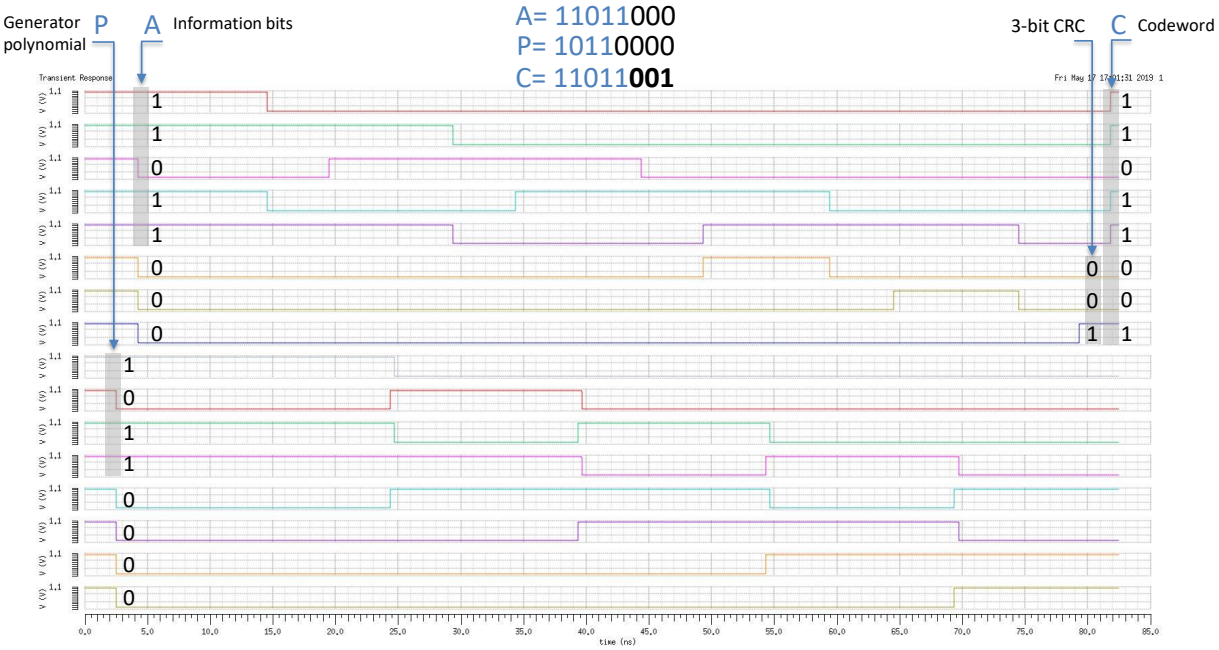


Figure 4.18: Cyclic redundancy check at the transmitter side

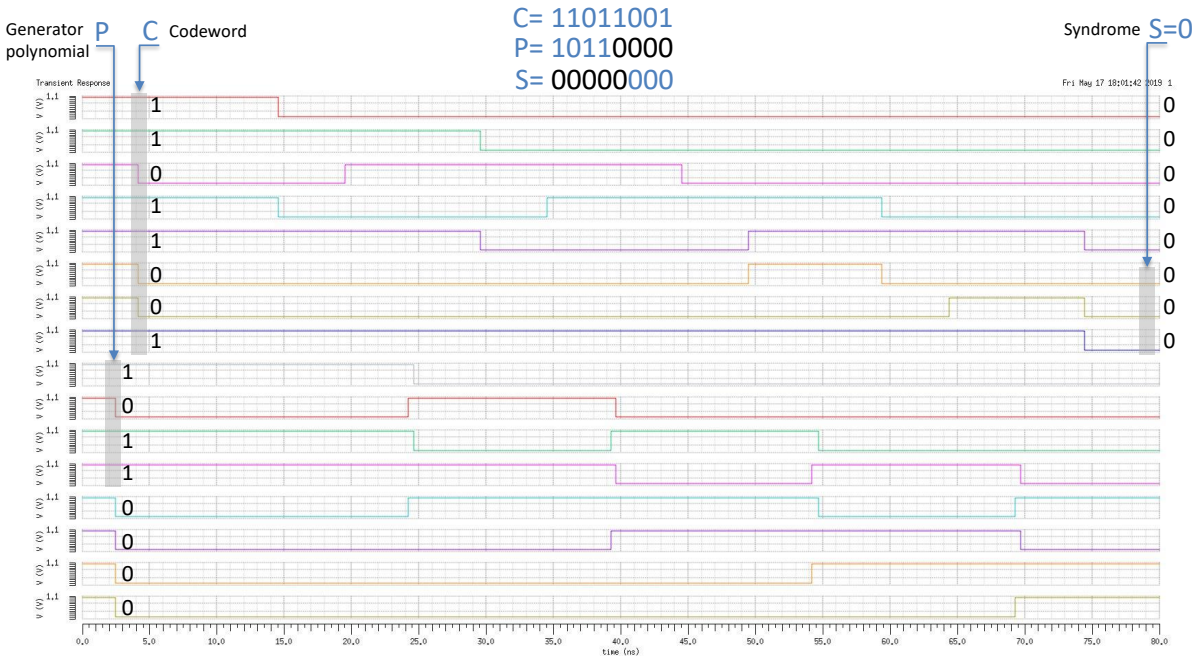


Figure 4.19: Cyclic redundancy check at the receiver side

4.9.2 In-memory DNN computing

Emerging and future IoT devices are expected to analyse raw data by running machine learning algorithms such as neural networks [128] [129]. Acquired data will be typically sent to the cloud for processing. However, this doesn't guarantee the required real-time response. Edge computing aims to get rid of the network latency by processing data locally.

Yet, running an intensive workloads such as deep neural networks on traditional cores (CPUs and GPUs) results in slow processing speed and high energy consumption. This is due to the intensive data movement between memory and processing cores. Although several recent works tried to accelerate processing through dedicated parallel hardware designs, data movement cost is still a critical technical challenge [130]. In this work, we aim to bypass the memory wall by employing our proposed in-memory computing technique for DNN computation. In this context, we propose a novel programmable architecture design for in-memory deep neural networks (DNN). The original architecture allows to execute in-memory addition and multiplication operations associated with the execution of a DNN weighted accumulation process.

4.9.2.1 Optimized multiply & accumulate process inside memory

The key operation of neural networks is the multiply-accumulate (MAC), which is responsible for the execution of weighted accumulation process. In fact, it is possible to realize MAC operations inside our proposed computational memory architecture based on iterative addition operations. However, this is practically inefficient, as it imposes significant number of computational steps which constitute a bottleneck. This is due the dependency of the total number of steps on the size N of the added operands (i.e. a single addition requires $6N + 1$ steps) inside memory. In order to realize reasonable MAC operations, we optimize the multiplication process inside the memory. An in-memory multiplication is achieved in two consecutive phases: (i) Phase 1 corresponds to the one-step partial product which is illustrated below and (ii) Phase 2 corresponds to the addition of the obtained partial product vectors inside the memory.

(i) Partial products: Fig. 4.20(b) illustrates the use of case 5 (demonstrated in section 4.3.2) inside crossbar array in order to realize partial product in two consequtive steps. The first step corresponds to initializing all the memristive cells in the crossbar to the logic zero. In the second step, an input vector A and an inverted vector B are fed to the columns and rows of the crossbar respectively. This combination results in a partial product of the two input vectors. The result is achieved in a single computational step.

In order to allow our proposed computational memory to perform this product, the data bits need to access the wordlines of the crossbar array. Therefore, we add an inverting multiplexer for each sub-array to allow the data to be supplied at the word-lines of the memory. It is worth to mention that an inverting multiplexer involves less resources as compared to classical multiplexer. The multiplexer can be configured to either pass the address or the data bits that would undergo partial product. Fig. 4.21 presents the modified sub-array designed in the 1M as well as 1T1M configuration model. A

diagram of the proposed computational memory (CMEM) is shown in Fig. 4.22. The diagram illustrates the structure of CMEM which can be configured between storage and computation including the capability of performing partial products.

(ii) Addition of the partial products: Normally, an $N \times M$ multiplication requires addition of M partial products, each of size N bits, to generate a $(N + M)$-bit products. Knowing that each addition operation inside CMEM requires $6N + 1$ steps, the total number of cycles to obtain the final product is $(M - 1)(6N + 1)$.

In order to reduce the required number of steps in the addition of the partial products, we use the method of carry save adder (CSA) to add multiple numbers together in tree structure. CSA provides a 3:2 operands reduction, with a fixed latency irrespective of the size N of the operands. Fig. 4.23 represents a diagram showing the addition of M operands using tree-like CSA blocks. A single 3:2 addition inside CMEM requires a latency of 13 steps. The last stage is a classical 2:1 adder and requires $6N + 1$ steps. Therefore, the overall latency required to add M operands is estimated as $C = 13(M - 2) + 6N + 1$ steps. By including the first two cycles required to obtain the partial products, a $N \times M$ multiplications inside the memory takes $C + 2$ (i.e. $13M + 6N - 23$) steps in total.

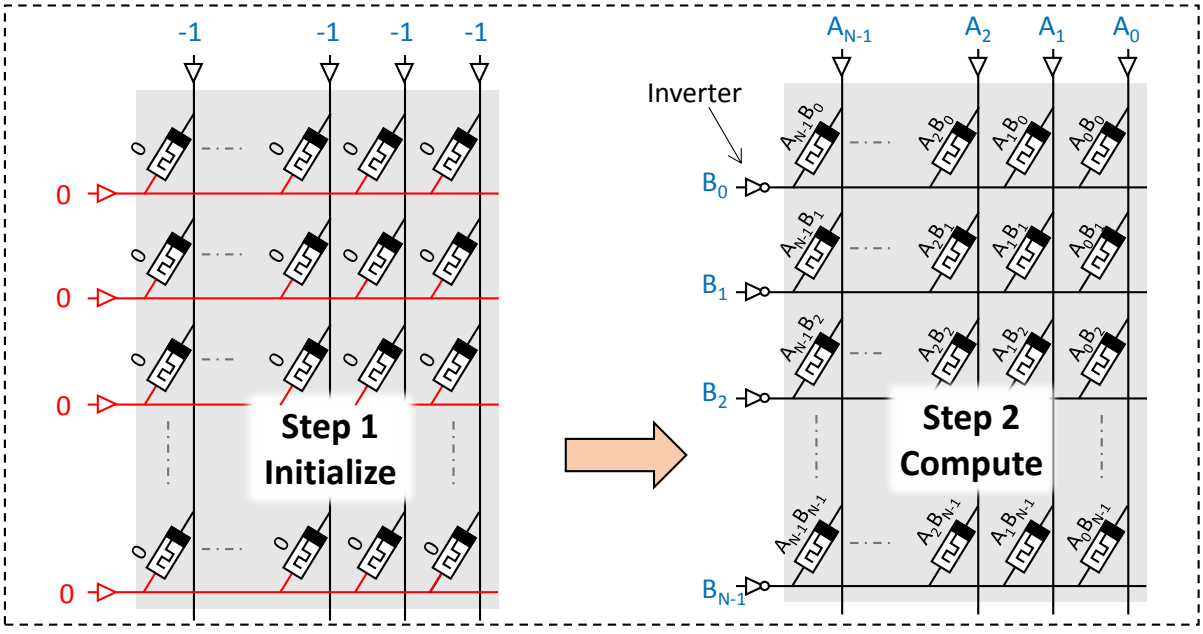


Figure 4.20: Realization the partial products of inside memristive crossbar array.

4.9.2.2 CMEM-based DNN architecture

Deep neural network is a popular category of machine learning algorithms. Generally, it is presented as a network of interconnected neurons, containing an input layer, an output layer and one or more hidden layers. Fig. 4.24(a) presents an example of neural network

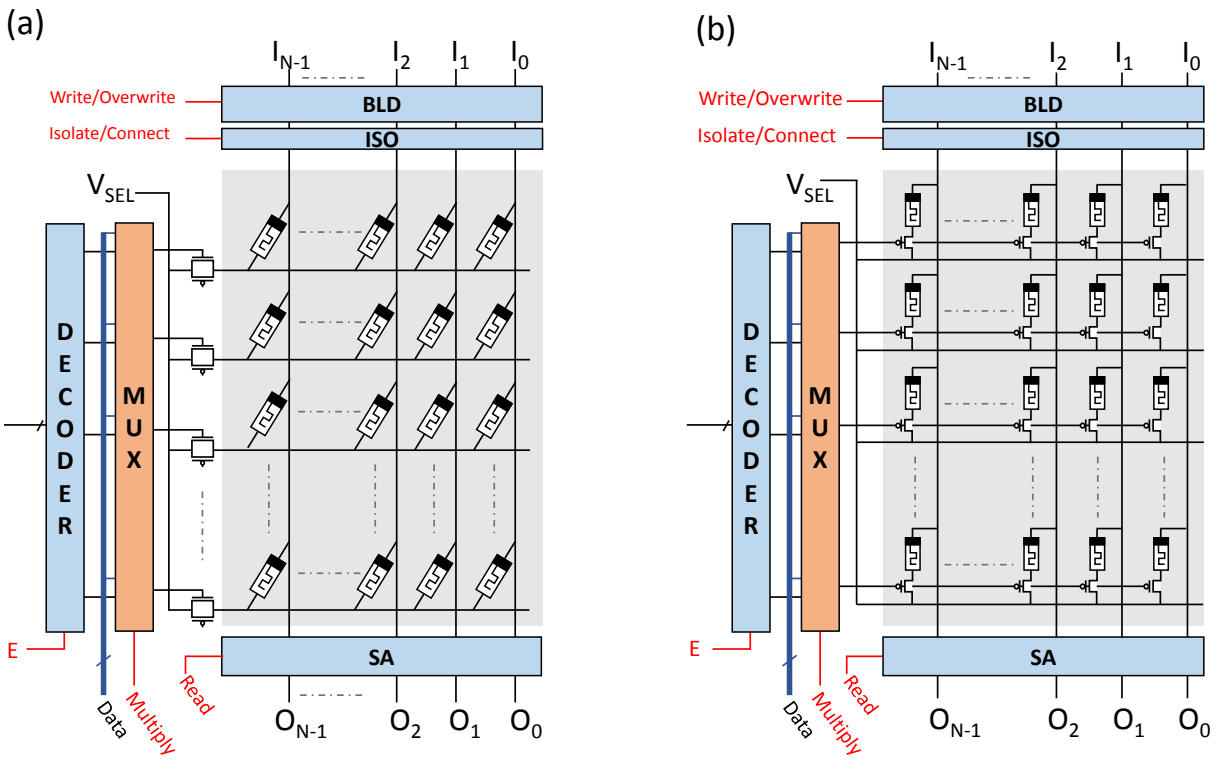


Figure 4.21: MOL memory architecture: (a) 1M model and (b) 1T1M model.

with an input layer of M neurons, an output layer of three neurons and no hidden layers for simplicity. As illustrated in the figure, each output neuron executes a weighted accumulation of the input vector. Fig. 4.24(b) presents the multiply-accumulate (MAC) operation in a matrix form. The total number of weighted accumulations is proportional to the size of the network, i.e. size of input vectors and number of hidden layers. For large DNN, this represents a major challenge as it implies intensive data movement between memory and processing cores. In fact, the cost of the multiply-accumulate operation and read/write memory accesses becomes considerable in terms of time and energy consumption [131]. In order to reduce this cost, we investigated the use of CMEM to perform in-memory multiply-accumulate operations.

By considering the example of the network presented in Fig. 4.24, a CMEM block is in charge of computing the output of a single neuron in the network layer. Thus the number of required CMEM blocks is equal to the number of neurons in that layer. Fig. 4.25 presents the proposed architecture design for in-memory DNN computation, illustrated for the simplified neural network of Fig. 4.24. It is composed of interconnected CMEM blocks that executes in parallel. Parallel execution is possible, since neurons in the same layer have no dependency to each other. For instance, the convolution of the input vector $[X_i]$ with the weight matrix $[A_i\ B_i\ C_i]$ is performed as follows. The weight

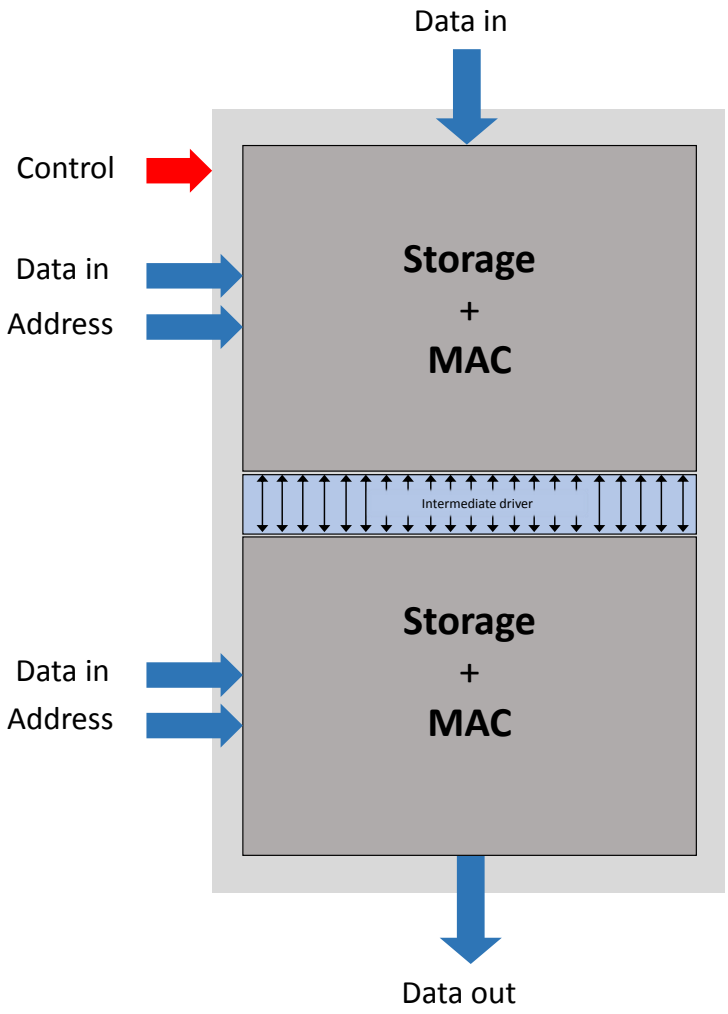


Figure 4.22: Simplified diagram of the MOL-based computational memory (CMEM)

matrix $[A_i\ B_i\ C_i]$ is supplied from the classical write path of the memory blocks while the input data vector $[X_i]$ is shared to all blocks along their rows. The outputs Y_A, Y_B and Y_C are executed simultaneously. The estimated total latency for the convolution operation is equal to $(C + 15)K + 6(N + M) + 1$ cycles.

For multi-layered networks such as DNN, CMEM blocks that are employed for the execution of a single layer takes over the execution of other layers. Thus, reusing the same hardware resources for different layers regardless the depth the network.

Moreover, the interconnected CMEM blocks share the same address decoder, MUX block and control signals. Therefore, the proposed design efficiently utilizes the peripheral circuits by sharing them to all sub-memory blocks on one hand, and between storage and computation operations on the other hand. This implies significant reduction in area overhead and simplifies the control. In fact, our design is based on binary memristive devices only.

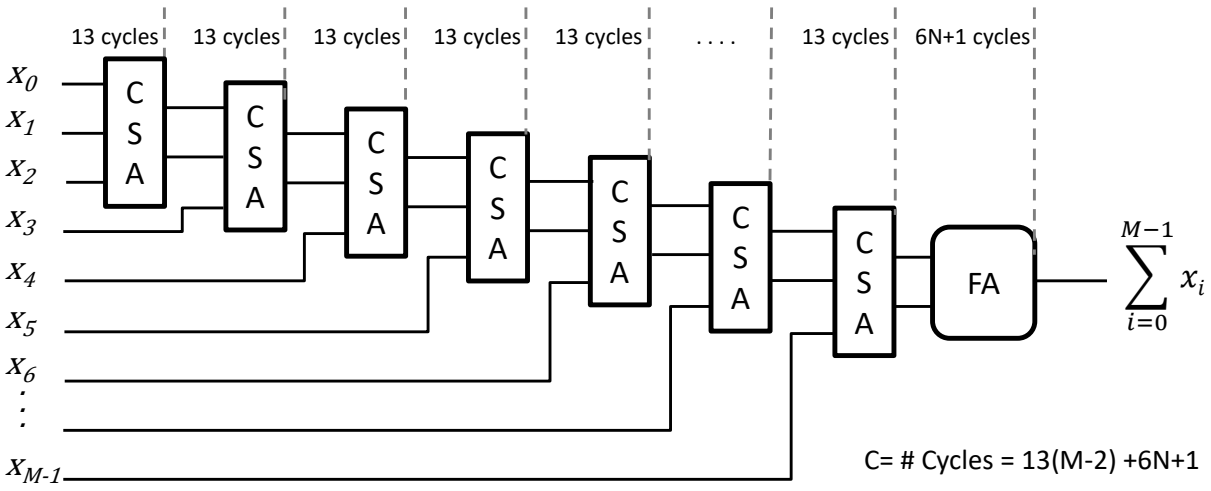


Figure 4.23: Addition of M operands using tree-like CSA blocks.

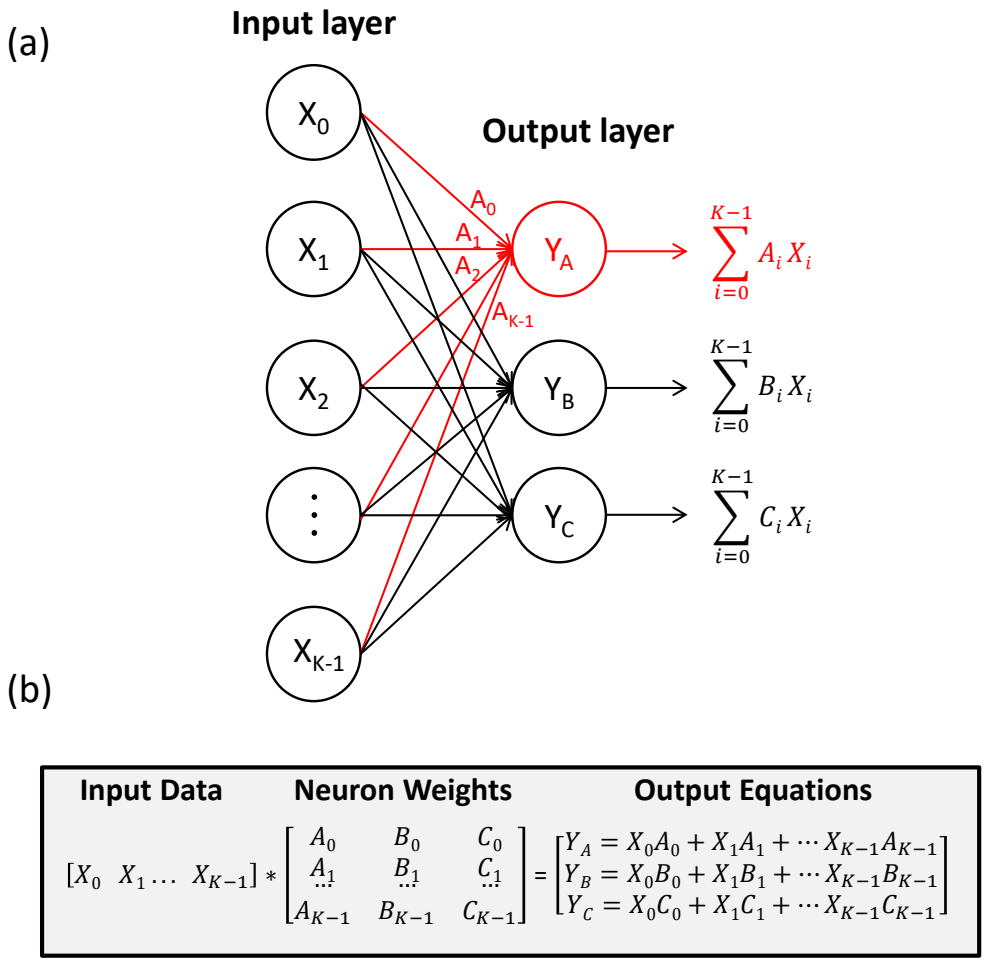


Figure 4.24: Example of a simplified neural network: (a) Network diagram (b) Matrix form representation.

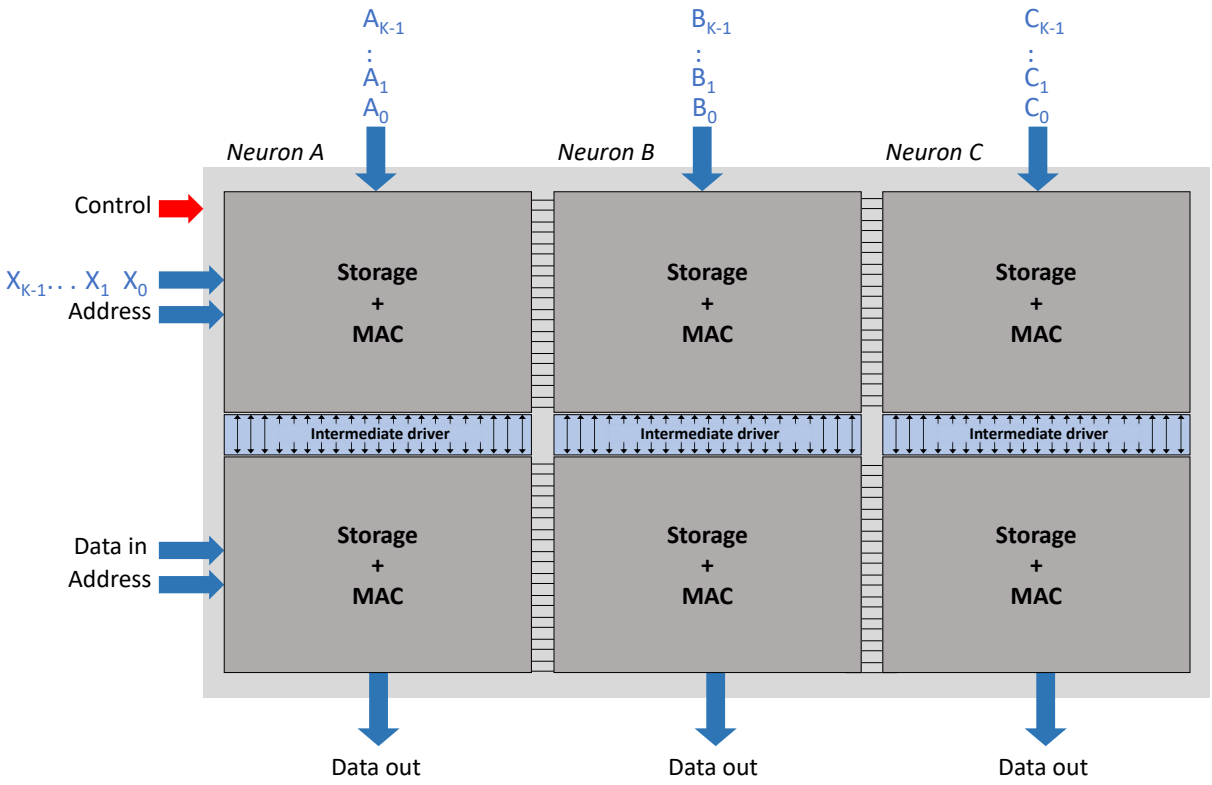


Figure 4.25: Proposed design for in-memory DNN computation, illustrated for the simplified neural network of Fig. 4.24.

4.10 Summary

In this chapter, the MOL design style is introduced together with an original architecture for MOL-based computational memory. This novel logic design style is inspired from a digital representation of memristors. Unlike existing approaches, MOL can operate with different memristor technologies, regardless the LRS-HRS margin and with linear as well as threshold-type memristive devices. Furthermore, the proposed original computational memory architecture, with appropriate drivers and control sequences, allows the execution of numerous logic operations, at bit or vector-level, in one or two computational steps at most. In order to illustrate the benefits of the proposed approach and to evaluate its performances, the implementation of an N-bit full addition using the proposed MOL-based computational memory has been detailed. The design is simulated in Cadence Virtuoso environment using CMOS 65nm process and realistic model parameters for STT-PMA-MTJ device. Comparisons with existing recent approaches demonstrate significant reductions in terms of latency and area.

The last section of the chapter we investigate the use of MOL approach in two applications. The CRC have been presented as a direct application that can be realized in our proposed MOL-based computational memory. The benefits of using CRC inside

memory have been discussed. The second application targets deep neural networks. we proposed a novel architecture design for realizing DNN in-memory. The architecture is composed of interconnected computational memory blocks namely CMEMs. An optimized in-memory addition and multiplication operations associated with the execution of the DNN weighted accumulation process is illustrated as a relevant case study. The proposed design addresses the inefficiency of moving data between memory and processing cores which is time and energy consuming. The design efficiently utilizes peripheral driving circuits which are shared between all memory blocks.

Conclusions and future work

Conclusions

THIS thesis work has been devoted to explore the potential of emerging memristor technologies for flexible interconnections, logic design, and in-memory computing. The manuscript started with a comprehensive state-of-the-art review on the basic fundamentals of memristor technology and the recent progress in various fields involving memristor designs. First, memristors have been introduced as storage elements that combine several advantageous features including high speed, high density, and non-volatility. Thereafter, the opportunity of using memristors as routing elements at the interconnect level have been discussed with the corresponding relevant applications in the literature. Last, we have illustrated the ability to use memristors to design memristive logic gates which are considered as enablers for new computational paradigms, such as in-memory computing, different from the traditional von Neumann models. Several existing memristive logic design styles have been reviewed in this context, ending with a roadmap for evaluation purpose. Challenges faced in these fields have been highlighted.

In this context, several original contributions have been proposed and are summarized below:

1. First, we have explored the feasibility of using memristors for realizing high degree of flexibility at the level of interconnects. We chose FFT as an application case study, as it is a popular component in telecommunications. Moreover, the flexibility in FFT is highly desired as the size of FFT block varies according to the parameters of the communication system. This requires supporting a wide range of FFT sizes that cannot be manipulated in a single radix FFT. A memristor-based reconfigurable FFT architecture (mrFFT) is proposed. mrFFT can be manipulated in a reuse and systematic way based on a proposed reconfigurable butterfly (RBF) that can be configured to support radix-2 and radix-3 kernels. The proposed design allows for optimized hardware reuse through a memristor-based non-volatile routing scheme, which is based on a programmable memristive nodes approach. The

proposed architecture is scalable and supports 44 configurations with different FFT sizes including the 32 operating modes that are defined in 3GPP-LTE standard. As compared to the state-of-the-art, mrFFT significantly reduces the utilized hardware resources and increases the percentage of active resources during execution.

2. Another contribution is introduced concerning the proposal of new design methodology for integrating memristors and CMOS devices in order to realize combinational logic with reduced area. The approach relies on the existing logic design style namely Memristor Ratioed Logic (MRL). The proposed integration method (X-MRL) efficiently arranges memristors in crossbar array stacked at the top of the CMOS layer. X-MRL structure combines the density and scalability attributes of memristive crossbar arrays. Based on X-MRL, the circuit design of a 1-bit full adder has been presented together with its corresponding layout. The conducted simulations demonstrate 44.79% area reduction compared to pure CMOS implementation of full adder. However, a large gap still exists in terms of power consumption due to the low resistance values of available memristor models. Power consumption is expected to be improved awaiting newly developed memristors having low leakage current (high resistance values)

3. Finally, three major contributions have been proposed related to the use of memristive devices for in-memory computing. An original logic design style, referred to as Memristor Overwrite Logic, has been introduced. MOL operates with different memristor technologies in contrast to the pre-existing logic design styles, particularly MAGIC and IMPLY, that require sufficient HRS/LRS ratio and are operated with threshold type devices only. In addition, we have shown that MOL is highly eligible to be integrated inside crossbar arrays, as it is an intrinsic feature of memristive devices. Based on MOL approach, an original computational memory architecture has been proposed. The architecture is able to execute bitwise AND/OR operations within its storage cells and can be simply configured between storage and computation modes. An N-bit addition is considered a case study for performance evaluation. The proposed computational memory design is verified with Cadence Virtuoso toolset based on CMOS 65nm process and accurate model parameters for STT-PMA-MTJ device. Comparisons with recent relevant contributions in the literature demonstrate significant improvements in terms of latency and area. Moreover, simulation results indicate that MOL has a relatively high tolerance against resistance variability (7% to 21%) that usually arises due to defects in fabrication. This robustness was revealed despite the low TMR value (i.e low HRS/LRS ratio) of the adopted MTJ device. In order to show the eligibility of the aforementioned

approach, we have investigated its suitability in real application use cases. An N-bit CRC algorithm has been implemented inside the proposed computational memory. In-memory CRC computation is of great interest as it can be employed to detect accidental errors caused by the memory itself. This eliminates the need for a dedicated separate logic unit.

Afterwards, we explored the use of MOL approach for DNN applications. A novel design for realizing in-memory DNN computation has been developed. The proposed design involves computing addition and multiplication operations associated with the execution of the DNN weighted accumulation process. The objective is to bypass the memory wall issue by performing these operations purely within the storage cells. The proposed architecture relies on interconnected replicated computational memory banks referred to as CMEM that share efficiently the driving circuitries.

Future work

In this thesis, novel and efficient memristor-based designs have been developed. The presented contributions confirm the potential for high flexibility, performance improvement, and complexity reduction with respect to traditional computing systems. Yet, many other research efforts could be conducted on this emerging topic. Here we propose several ideas and proposals for future work and further investigations.

Regarding the use of memristive devices at the level of interconnects to design flexible architectures, we believe that these devices can bring the desired high degree of flexibility with minimal added resources. The concept of non-volatile routing scheme is also promising in terms of energy savings. Future work could consider the use of mixed-signal simulation environment to integrate both analog and digital parts of the proposed mrFFT architecture and to conduct the corresponding performance analysis.

Another key point that could be investigated in this context is the drift in memristor state during the operation phase of the design. A low operation voltage is recommended in order to avoid such phenomenon. However, the adopted voltage level should be within the acceptable range of CMOS technology. Moreover, a memristor with relatively high threshold voltage is highly desirable as it widens the margin between programming and operation voltage levels. This prevents the drift in memristance during the operation phase.

Regarding the use of memristive devices for logic design, further research efforts at the device level are of great importance. Memristive devices are still being actively

explored and developed using variety of materials and deposition techniques. Thus, there is a scope for improving the device characteristics. It is expected to increase the ON-OFF resistance ratio of memristive devices, while reducing their switching delay and leakage currents. Accordingly, in addition to the area saving achieved using the proposed X-MRL technique, adopting such future memristive devices has the potential to greatly reduce power and delay.

Regarding the use of memristive devices for in-memory computing, the presented MOL design style offers several benefits over pre-existing ones in terms of time and area. The structure of the proposed computational memory demonstrates the optimal choice for such logic design. Future work in addition to possible extensions in this direction could be established at the circuit level, device level, and physical topology.

- First, the controlled intermediate driver is the only gate for all signals that are exchanged between the two coupled memory blocks during execution. Optimizing this driver can be of great advantage. For instance, the controlled inversion block (INV) can be dispensed. Another alternative efficient inversion can be established during the readout using the SA block just by triggering the read bias (i.e. 1 instead of 0). This should save area and power, besides reducing the critical path which impacts the step delay in each MOL operation.

- The effect of lowering the operating voltage level less than 1.2 V deserves investigation. It seems that reducing the voltage level (within the acceptable range of the technology process) decreases the leakage current in MTJ devices and thus reduces power consumption. However, it increases the switching delay of these devices which in turn increases step delay and consequently the total consumed energy. This trade-off could be explored in order to devise optimal operating point for the proposed design.

- The HRS and LRS values also affect the energy and delay of each computational step. Increasing their values should definitely improves power consumption. However, tuning these values should fit the real physical characteristics specified by the technology provider. This involves all memristive devices including resistive, magnetoresistive, and phase change devices.

- The layout implementation of the design is of high importance. A realistic layout model for the STT-PMA-MTJ device should be adopted in order to perform post-layout simulation. This allows for the estimation of the exact occupied area, power consumption, and delay. A potential drawback that might arise in the lay-

out implementation is the long pathway of the wires that span from an output of MOL-memory-B to an input of MOL-memory-A (see Fig. 4.8). In fact, these wires induce significant delay in the datapath, which in turn shrinks the attained large bandwidth of the in-memory operations and degrades the overall performance. This problem becomes more critical when scaling the memory for larger sizes. A possible solution is to fold the two memory arrays at the top of each other based on 3D-stacking technique [29]. This solution should totally eliminate the need for these wires and lead to more compact design with improved performance.

- Design of the computational memory controller unit and the associated micro-operations synthesis tool. The idea is to propose an efficient solution to automate the generation of MOL micro-operations by mapping sequences of arithmetic operations on the proposed computational memory. The sequence of these micro-operations will configure the memory drivers and select the operation modes for in-memory computation. This allows to map complex arithmetic tasks and simplifies the integration of MOL-memory architecture in large systems.

As general thoughts, it is currently erroneous to think that memristor technology would completely replace CMOS. As discussed earlier, memristors are passive elements. Therefore, CMOS devices are still indispensable for their operation. Hence, investigating hybrid designs is the only way to leverage their promising characteristics. Furthermore, in-memory computing techniques should not be considered as complete replacement to the conventional von Neumann model. In fact, for some tasks, in-memory operations may have comparatively lower performance. Instead, working on integrating a computational memory as an accelerator accompanying processor cores would be of great interest. Tasks should be then decided whether to be executed in conventional processors or directly computed inside the memory.

Moreover, improved memristor models are highly desirable and critical to broaden their use. These models should be accurate, available and easy-to-fit with experimental data. Furthermore, high level simulation tools dedicated for memristor design could be established. Such tools should allow for rapid simulation of large-scale systems. This would allow students, researchers, and industrials to explore memristor behaviors and applications at a more rapid and progressive rate, leading to further creative ideas and faster developments of memristor-based designs.

Lastly, the work presented in this dissertation motivates further exploration of full system memristor implementations. The work described shows that even limited by the current

state of memristor technology, these devices yield strong results. Modern architectures should be re-envisioned from the top level down in order to apply the findings presented in this thesis. The key outcome from this work can be exploited to design entirely novel systems that break the limits of traditional von Neuman model.

Bibliography

[1] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, p. 80, 2008.

[3] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: opportunities and challenges," in *proc. of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2011, pp. 325–334.

[4] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[5] H. Kim, M. P. Sah, and S. P. Adhikari, "Pinched hysteresis loops is the fingerprint of memristive devices," *arXiv preprint arXiv:1202.2437*, 2012.

[6] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[7] D. S. Jeong, R. Thomas, R. Katiyar, J. Scott, H. Kohlstedt, A. Petraru, and C. S. Hwang, "Emerging memories: resistive switching mechanisms and current status," *Reports on progress in physics*, vol. 75, no. 7, p. 076502, 2012.

[8] J. S. Meena, S. M. Sze, U. Chand, and T.-Y. Tseng, "Overview of emerging non-volatile memory technologies," *Nanoscale research letters*, vol. 9, no. 1, p. 526, 2014.

[9] S. P. Mohanty, "Memristor: from basics to deployment," *IEEE Potentials*, vol. 32, no. 3, pp. 34–39, 2013.

[10] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, pp. 074–508, 2009.

[11] J. J. Yang, M. D. Pickett, X. Li, D. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnology*, vol. 3, no. 7, p. 429, 2008.

[12] E. Lehtonen and M. Laiho, "CNN using memristors for neighborhood connections," in *proc. of the International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, 2010, pp. 1–4.

[13] J. G. Simmons, "Generalized formula for the electric tunnel effect between similar electrodes separated by a thin insulating film," *Journal of Applied Physics*, vol. 34, no. 6, pp. 1793–1803, 1963.

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, 2013.

[15] S. Natarajan, S. Chung, L. Paris, and A. Keshavarzi, "Searching for the dream embedded memory," *IEEE Solid-state circuits magazine*, vol. 1, no. 3, pp. 34–44, 2009.

[16] Y. Nishi and B. Magyari-Kope, *Advances in non-volatile memory and storage technology*. Woodhead Publishing, 2019.

[17] K. Kim and J. Choi, "Future outlook of NAND flash technology for 40nm node and beyond," in *21st IEEE Non-Volatile Semiconductor Memory Workshop*, 2006, pp. 9–11.

[18] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, p. 165209, 2007.

[19] J. F. Scott and C. A. P. De Araujo, "Ferroelectric memories," *Science*, vol. 246, no. 4936, pp. 1400–1405, 1989.

[20] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *proc. of the 36th annual international symposium on Computer architecture*, 2009, pp. 2–13.

[21] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[22] J. Nickel, "Memristor materials engineering: From flash replacement towards a universal memory," in *proc. of the IEEE IEDM Advanced Memory Technology Workshop*, 2011, pp. 1–3.

[23] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[24] R. Patel, S. Kvatinsky, E. G. Friedman, and A. Kolodny, "Multistate register based on resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1750–1759, 2014.

[25] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based multithreading," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 41–44, 2013.

[26] K. J. Han, N. Chan, S. Kim, B. Leung, V. Hecht, B. Cronquist, D. Shum, A. Tilke, L. Pescini, M. Stiftinger *et al.*, "A novel flash-based FPGA technology with deep trench isolation," in *22nd IEEE Non-Volatile Semiconductor Memory Workshop*, 2007, pp. 32–33.

[27] Y. Chen, J. Zhao, and Y. Xie, "3D-NonFAR: three-dimensional non-volatile FPGA architecture using phase change memory," in *proc. of the 16th ACM/IEEE international symposium on Low power electronics and design*, 2010, pp. 55–60.

[28] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, no. 6, pp. 1165–1172, 2005.

[29] J. Handy, "Understanding the intel/micron 3d xpoint memory," *Proc. SDC*, 2015.

[30] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[31] Y. Zhang, S. Kim, J. P. McVittie, H. Jagannathan, J. B. Ratchford, C. E. Chidsey, Y. Nishi, and H.-S. P. Wong, "An integrated phase change memory cell with ge nanowire diode for cross-point memory," in *IEEE Symposium on VLSI Technology*, 2007, pp. 98–99.

[32] M.-J. Lee, Y. Park, B.-S. Kang, S.-E. Ahn, C. Lee, K. Kim, W. Xianyu, G. Stefanovich, J.-H. Lee, S.-J. Chung *et al.*, "2-stack 1D-1R cross-point structure with oxide diodes as switch elements for high density resistance RAM applications," in *IEEE International Electron Devices Meeting*, 2007, pp. 771–774.

[33] Y.-C. Chen, C. Chen, C. Chen, J. Yu, S. Wu, S. Lung, R. Liu, and C.-Y. Lu, "An access-transistor-free (0T/1R) non-volatile resistance random access memory (RRAM) using a novel threshold switching, self-rectifying chalcogenide device," in *IEEE International Electron Devices Meeting*, 2003, pp. 37–4.

[34] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, no. 42, p. 425204, 2009.

[35] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature materials*, vol. 9, no. 5, pp. 403–406, 2010.

[36] Y.-C. Chen, C.-C. Lin, S.-T. Hu, C.-Y. Lin, B. Fowler, and J. Lee, "A novel resistive switching identification method through relaxation characteristics for sneak-path-constrained selectorless rram application," *Scientific reports*, vol. 9, no. 1, pp. 1–6, 2019.

[37] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Information-theoretic sneak-path mitigation in memristor crossbar arrays," *IEEE Transactions on Information Theory*, vol. 62, no. 9, pp. 4801–4813, 2016.

[38] Y. Cassuto, S. Kvatinsky *et al.*, "Sneak-path constraints in memristor crossbar arrays," in *proc. of the IEEE International Symposium on Information Theory*, 2013, pp. 156–160.

[39] J. Kang, P. Huang, B. Gao, H. Li, Z. Chen, Y. Zhao, C. Liu, L. Liu, and X. Liu, "Design and application of oxide-based resistive switching devices for novel computing architectures," *IEEE Journal of the Electron Devices Society*, vol. 4, no. 5, pp. 307–313, 2016.

[40] H. Lee, Y. Chen, P. Chen, P. Gu, Y. Hsu, S. Wang, W. Liu, C. Tsai, S. Sheu, P. Chiang *et al.*, "Evidence and solution of over-reset problem for HfO_x based resistive memory with sub-ns switching speed and high endurance," in *proc. of the IEEE International Electron Devices Meeting*, 2010, pp. 19–7.

[41] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[42] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *proc. of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2011, pp. 1–8.

[43] R. Hasan and T. M. Taha, "Memristor crossbar based programmable interconnects," in *IEEE Computer Society Annual Symposium on VLSI*, 2014, pp. 94–99.

[44] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 216–229, 2007.

[45] C. Dong, D. Chen, S. Haruehanroengra, and W. Wang, "3-D nFPGA: A reconfigurable architecture for 3-d cmos/nanomaterial hybrid digital circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 11, pp. 2489–2501, 2007.

[46] P.-E. Gaillardon, M. H. Ben-Jamaa, G. B. Beneventi, F. Clermidy, and L. Perniola, "Emerging memory technologies for reconfigurable routing in FPGA architecture," in *17th IEEE International Conference on Electronics, Circuits and Systems*, 2010, pp. 62–65.

[47] S. Tanachutiwat, M. Liu, and W. Wang, "FPGA based on integration of CMOS and RRAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 11, pp. 2023–2032, 2010.

[48] Y. Zhang, Y. Shen, X. Wang, and L. Cao, "A novel design for memristor-based logic switch and crossbar circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 5, pp. 1402–1411, 2015.

[49] P. Junsangsri and F. Lombardi, "Design of a hybrid memory cell using memristance and ambipolarity," *IEEE Transactions on Nanotechnology*, vol. 12, no. 1, pp. 71–80, 2012.

[50] J. M. Shalf and R. Leland, "Computing beyond moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, 2015.

[51] K. J. Kuhn, "Considerations for ultimate cmos scaling," *IEEE transactions on Electron Devices*, vol. 59, no. 7, pp. 1813–1828, 2012.

[52] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL–memristor ratioed logic," in *proc. of the International Workshop on Cellular Nanoscale Networks and their Applications*, 2012, pp. 1–6.

[53] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC–Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[54] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-based IMPLY logic design procedure," in *proc. of the IEEE International Conference on Computer Design (ICCD)*, 2011, pp. 142–147.

[55] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115–119, 2013.

[56] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.

[57] L. Guckert and E. E. Swartzlander, "MAD gates–memristor logic design using driver circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 171–175, 2016.

[58] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2015.

[59] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[60] K. Bickerstaff and E. E. Swartzlander, "Memristor-based arithmetic," in *proc. of the Conference Record of the Asilomar Conference on Signals, Systems and Computers*, 2010, pp. 1173–1177.

[61] K. C. Rahman, D. Hammerstrom, Y. Li, H. Castagnaro, and M. A. Perkowski, "Methodology and design of a massively parallel memristive stateful IMPLY logic-based reconfigurable architecture," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 675–686, 2016.

[62] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "An energy-efficient memristive threshold logic circuit," *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 474–487, 2012.

[63] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[64] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 948–953.

[65] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[66] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive logic: A framework for evaluation and comparison," in *27th IEEE International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[67] X. Fang and Y. Tang, "Circuit analysis of the memristive stateful implication gate," *Electronics Letters*, vol. 49, no. 20, pp. 1282–1283, 2013.

[68] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5g be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[69] J. Yli-Kaakinen, T. Levanen, M. Renfors, M. Valkama, and K. Pajukoski, "FFT-domain signal processing for spectrally-enhanced CP-OFDM waveforms in 5G new radio," in *proc. of the 52nd IEEE Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1049–1056.

[70] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[71] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *proc. of the 10th International Parallel Processing Symposium (IPPS)*, 1996, pp. 766–770.

[72] E. Dubois and A. Venetsanopoulos, "A new algorithm for the radix-3 FFT," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 3, pp. 222–225, 1978.

[73] J. Löfgren and P. Nilsson, "On hardware implementation of radix 3 and radix 5 FFT kernels for LTE systems," in *proc. of the NORCHIP*, 2011, pp. 1–4.

[74] S. Goedecker, "Fast radix 2, 3, 4, and 5 kernels for fast fourier transformations on computers with overlapping multiply–add instructions," *SIAM Journal on Scientific Computing*, vol. 18, no. 6, pp. 1605–1611, 1997.

[75] E. E. Swartzlander, W. K. Young, and S. J. Joseph, "A radix 4 delay commutator for fast fourier transform processor implementation," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 5, pp. 702–709, 1984.

[76] M. Hasan, T. Arslan, and J. S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 fft processor for wireless lan applications," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 1, pp. 128–134, 2003.

[77] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A dynamic scaling fft processor for dvb-t applications," *IEEE Journal of solid-state circuits*, vol. 39, no. 11, pp. 2005–2013, 2004.

[78] I.-C. Park, W. Son, and J.-H. Kim, "Twiddle factor transformation for pipelined fft processing," in *proc. of the 25th International Conference on Computer Design*, 2007, pp. 1–6.

[79] G. Bi and G. Li, "Pipelined structure based on radix-2 2 fft algorithm," in *proc. of the 6th IEEE Conference on Industrial Electronics and Applications*, 2011, pp. 2530–2533.

[80] N. H. Cuong, N. T. Lam, and N. D. Minh, "Multiplier-less based architecture for variable-length fft hardware implementation," in *proc. of the Fourth International Conference on Communications and Electronics (ICCE)*. IEEE, 2012, pp. 489–494.

[81] S. He and M. Torkelson, "Designing pipeline fft processor for ofdm (de) modulation," in *proc. of the International Symposium on Signals, Systems, and Electronics*. IEEE, 1998, pp. 257–262.

[82] S. Bouguezel, M. O. Ahmad, and M. S. Swamy, "New radix-(2/spl times/2/spl times/2)/(4/spl times/4/spl times/4) and radix-(2/spl times/2/spl times/2)/(8/spl times/8/spl times/8) DIF FFT algorithms for 3-D DFT," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 2, pp. 306–315, 2006.

[83] M. Shin and H. Lee, "A high-speed four-parallel radix-2 4 FFT/IFFT processor for UWB applications," in *proc. of the IEEE International Symposium on Circuits and Systems*, 2008, pp. 960–963.

[84] J. Lee and H. Lee, "A high-speed two-parallel radix-24 FFT/IFFT processor for MB-OFDM UWB systems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 91, no. 4, pp. 1206–1211, 2008.

[85] M. Garrido, F. Qureshi, and O. Gustafsson, "Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 7, pp. 2002–2012, 2014.

[86] H.-Y. Lee and I.-C. Park, "Balanced binary-tree decomposition for area-efficient pipelined fft processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 4, pp. 889–900, 2007.

[87] 3GPP. (2017) Evolved universal terrestrial radio access (E-UTRA); physical channels and modulation. [Online]. Available: https://www.3gpp.org/

[88] C. Yu and M.-H. Yen, "Area-efficient 128-to 2048/1536-point pipeline FFT processor for LTE and mobile WiMAX systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1793–1800, 2014.

[89] J. Chen, J. Hu, S. Lee, and G. E. Sobelman, "Hardware efficient mixed radix-25/16/9 FFT for LTE systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 221–229, 2014.

[90] X.-Y. Shih, Y.-Q. Liu, and H.-R. Chou, "48-Mode Reconfigurable Design of SDF FFT Hardware Architecture Using Radix-3 2 and Radix-2 3 Design Approaches," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 6, pp. 1456–1467, 2017.

[91] K. Seshan, "Limits and hurdles to continued cmos scaling," in *Handbook of Thin Film Deposition*. Elsevier, 2018, pp. 19–41.

[92] H. Iwai, "Cmos technology after reaching the scale limit," in *proc. of the 8th International Workshop on Junction Technology (IWJT'08)*. IEEE, 2008, pp. 1–2.

[93] B. Liu, Y. Wang, Z. You, Y. Han, and X. Li, "A signal degradation reduction method for memristor ratioed logic (MRL) gates," *IEICE Electronics Express*, vol. 12, no. 8, pp. 1–6, 2015.

[94] J. Chowdhury, K. Das, and K. Rout, "Implementation of 24T memristor based adder architecture with improved performance," *International Journal of Electrical, Electronics and Data Communication*, vol. 3, no. 6, 2015.

[95] K. Tsunoda, K. Kinoshita, H. Noshiro, Y. Yamazaki, T. Iizuka, Y. Ito, A. Takahashi, A. Okano, Y. Sato, T. Fukano *et al.*, "Low power and high speed switching of Ti-doped NiO ReRAM under the unipolar voltage source of less than 3V," in *proc. of the IEEE International Electron Devices Meeting (IEDM)*, 2007, pp. 767–770.

[96] Z. Biolek, D. Biolek, and V. Biolkova, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, 2009.

[97] A. Chanthbouala, V. Garcia, R. O. Cherifi, K. Bouzehouane, S. Fusil, X. Moya, S. Xavier, H. Yamada, C. Deranlot, N. D. Mathur *et al.*, "A ferroelectric memristor," *Nature Materials*, vol. 11, no. 10, p. 860, 2012.

[98] I. Baek, M. Lee, S. Seo, M. Lee, D. Seo, D.-S. Suh, J. Park, S. Park, H. Kim, I. Yoo *et al.*, "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *proc. of the IEEE International Electron Devices Meeting (IEDM)*, 2004, pp. 587–590.

[99] T. Diokh, E. Le-Roux, S. Jeannot, M. Gros-Jean, P. Candelier, J. Nodin, V. Jousseaume, L. Perniola, H. Grampeix, T. Cabout *et al.*, "Investigation of the impact of the oxide thickness and reset conditions on disturb in HfO_2-RRAM integrated in a 65***nm*** CMOS technology," in *proc. of the IEEE International Reliability Physics Symposium (IRPS)*, 2013, pp. 5E–4.

[100] H. Lee, P. Chen, T. Wu, Y. Chen, C. Wang, P. Tzeng, C. Lin, F. Chen, C. Lien, and M.-J. Tsai, "Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO_2 based RRAM," in *proc. of the IEEE International Electron Devices Meeting (IEDM)*, 2008, pp. 1–4.

[101] P. L. Thangkhiew, R. Gharpinde, P. V. Chowdhary, K. Datta, and I. Sengupta, "Area efficient implementation of ripple carry adder using memristor crossbar arrays," in *proc. of the International Design & Test Symposium (IDT)*, 2016, pp. 142–147.

[102] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (MAGIC)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[103] P. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, and I. Sengupta, "Efficient implementation of adder circuits in memristive crossbar array," in *proc. of the IEEE Region 10 Conference (TENCON)*, 2017, pp. 207–212.

[104] R. Gharpinde, P. L. Thangkhiew, K. Datta, and I. Sengupta, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 355–366, 2018.

[105] M. Teimoory, A. Amirsoleimani, J. Shamsi, A. Ahmadi, S. Alirezaee, and M. Ahmadi, "Optimized implementation of memristor-based full adder by material implication logic," in *proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2014, pp. 562–565.

[106] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *proc. of the IEEE International Symposium on Nanoscale Architectures*, 2009, pp. 33–36.

[107] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, *Logic Synthesis for Majority Based In-Memory Computing, Chapter in Advances in memristors, memristive devices and systems.* Springer, 2017.

[108] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[109] C.-X. Xue *et al.*, "A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *proc. of the IEEE international Solid-State Circuits Conference-(ISSCC)*, 2019.

[110] W.-H. Chen, W.-J. Lin *et al.*, "A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *proc. of the IEEE international Electron Devices Meeting (IEDM)*, 2017.

[111] C.-X. Xue *et al.*, "A 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *proc. of the IEEE international Solid-State Circuits Conference-(ISSCC)*, 2020.

[112] W.-H. Chen *et al.*, "CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors," *Nature Electronics*, vol. 2, no. 9, pp. 420–428, 2019.

[113] R. B. Hur and S. Kvatinsky, "Memristive memory processing unit (MPU) controller for in-memory processing," in *proc. of the IEEE International Conference on the Science of Electrical Engineering (ICSEE)*, 2016, pp. 1–5.

[114] P. L. Thangkhiew, R. Gharpinde, and K. Datta, "Efficient mapping of Boolean functions to memristor crossbar using MAGIC NOR gates," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–11, 2018.

[115] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *proc. of the IEEE*, vol. 100, no. 6, pp. 2021–2032, 2012.

[116] Y. Huai *et al.*, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[117] S. Ikeda, K. Miura, H. Yamamoto, K. Mizunuma, H. Gan, M. Endo, S. Kanai, J. Hayakawa, F. Matsukura, and H. Ohno, "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature materials*, vol. 9, no. 9, p. 721, 2010.

[118] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE electron device letters*, vol. 30, no. 3, pp. 294–297, 2009.

[119] T. Hanyu, T. Endoh, D. Suzuki, H. Koike, Y. Ma, N. Onizawa, M. Natsui, S. Ikeda, and H. Ohno, "Standby-power-free integrated circuits using MTJ-based VLSI computing," *proc. of the IEEE*, vol. 104, no. 10, pp. 1844–1863, 2016.

[120] Y. Wang, Y. Zhang, E. Deng, J.-O. Klein, L. A. Naviner, and W. Zhao, "Compact model of magnetic tunnel junction with stochastic spin transfer torque switching for reliability analyses," *Microelectronics Reliability*, vol. 54, no. 9-10, pp. 1774–1778, 2014.

[121] Y. Wang, H. Cai, L. A. Naviner, Y. Zhang, J.-O. Klein, and W. Zhao, "Compact thermal modeling of spin transfer torque magnetic tunnel junction," *Microelectronics Reliability*, vol. 55, no. 9-10, pp. 1649–1653, 2015.

[122] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *proc. of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[123] Berkeley Logic Synthesis and Verification Group. (2005). ABC – a system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/~alanmi/abc/

[124] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *proc. of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[125] S. Sheng-Ju, "Implementation of cyclic redundancy check in data communication," in *proc. of the international Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2015, pp. 529–531.

[126] W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Computation-oriented fault-tolerance schemes for rram computing systems," in *proc. of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 794–799.

[127] A. Chaudhuri, B. Yan, Y. Chen, and K. Chakrabarty, "Hardware fault tolerance for binary rram crossbars," in *proc. of the IEEE International Test Conference (ITC)*, 2019, pp. 1–10.

[128] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[129] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "Looknn: Neural network with no multiplication," in *proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1775–1780.

[130] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[131] L. Fick and D. Fick, "Introduction to compute-in-memory," in *IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2019, pp. 1–65.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Nouvelles approches de conception d'architectures flexibles et de calcul en mémoire basées sur les technologies des memristors

Mots clés : Memristor; calcul en mémoire; crossbar; conception numérique; mémoire non-volatile

Résumé : Le développement récent de nouvelles technologies de mémoires non-volatiles basées sur le concept de memristor a suscité de nombreux efforts pour explorer leur utilisation potentielle dans différents domaines d'application. Les propriétés uniques de ces dispositifs memristifs et leur compatibilité pour une intégration avec les technologies CMOS conventionnelles permettent de nouveaux paradigmes de conception d'architecture, offrant des niveaux sans précédent de densité, de reconfigurabilité et d'efficacité énergétique. Dans ce contexte, le but de ce travail de thèse était d'explorer et d'introduire de nouvelles approches de conception basées sur les memristors pour combiner flexibilité et efficacité en proposant des architectures originales qui dépassent les limites des architectures existantes. Cette exploration et cette étude ont été menées à trois niveaux : interconnexion, traitement et mémoire. Au niveau des interconnexions, nous avons étudié l'utilisation de dispositifs memristifs pour permettre une grande flexibilité basée sur des réseaux d'interconnexion programmables. Cela a permis de proposer la première architecture de transformée de Fourier rapide reconfigurable basée sur des memristors, nommée mrFFT. Les memristors sont insérés comme des commutateurs reconfigurables au niveau des interconnexions afin d'établir un routage flexible puce. Au niveau du traitement, nous avons exploré l'utilisation de dispositifs memristifs et leur intégration avec les technologies CMOS pour la conception de fonctions logique combinatoire. Ces circuits hybrides memristor-CMOS exploitent la forte densité d'intégration des memristors afin d'améliorer les performances des implémentations numériques, et en particulier des unités arithmétiques et logiques. Au niveau mémoire, une nouvelle approche de calcul en mémoire a été introduite. Dans ce contexte, un nouveau style de conception logique a été proposé, nommé Memristor Overwrite Logic (MOL), associé à une architecture originale de mémoire de calcul. L'approche proposée permet de combiner efficacement le stockage et le traitement afin de contourner les problèmes liés aux accès mémoire et d'améliorer ainsi l'efficacité de calcul. L'approche proposée a été appliquée dans trois études de cas à des fins de validation et d'évaluation des performances.

Title: New design approaches for flexible architectures and in-memory computing based on memristor technologies

Keywords: Memristor; In-memory computation; Crossbar array; Logic design; Non-volatile memory

Abstract: The recent development of new non-volatile memory technologies based on the memristor concept has triggered many research efforts to explore their potential usage in different application domains. The distinctive features of memristive devices and their suitability for CMOS integration are expected to lead for novel architecture design paradigms enabling unprecedented levels of energy efficiency, density, and reconfigurability. In this context, the goal of this thesis work was to explore and introduce new memristor-based designs that combine flexibility and efficiency through the proposal of original architectures that break the limits of the existing ones. This exploration and study have been conducted at three levels: interconnect, processing, and memory levels. At interconnect level, we have explored the use of memristive devices to allow high degree of flexibility based on programmable interconnects. This allows to propose the first memristor-based reconfigurable fast Fourier transform architecture, namely mrFFT. Memristors are inserted as reconfigurable switches at the level of interconnects in order to establish flexible on-chip routing. At processing level, we have explored the use of memristive devices and their integration with CMOS technologies for combinational logic design. Such hybrid memristor-CMOS designs exploit the high integration density of memristors in order to improve the performance of digital designs, and particularly arithmetic logic units. At memory level, we have explored new in-memory computing approaches and proposed a novel logic design style, namely Memristor Overwrite Logic (MOL), associated with an original MOL-based computational memory. The proposed approach allows efficient combination of storage and processing in order to bypass the memory wall problem and thus to improve the computational efficiency. The proposed approach has been applied in three real application case studies for the sake of validation and performance evaluation.