Circuit Design, Architecture and CAD for RRAM-based FPGAs

THÈSE N^{O} 8084 (2017)

PRÉSENTÉE LE 24 NOVEMBRE 2017
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DES SYSTÈMES INTÉGRÉS (IC/STI)
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Xifan TANG

acceptée sur proposition du jury:

Prof. A. P. Burg, président du jury
Prof. G. De Micheli, Dr P.-E. J. M. Gaillardon, directeurs de thèse
Prof. M. Huebner, rapporteur
Dr J. Ryckaert, rapporteur
Prof. P. Ienne, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

We’re paratroopers, Lieutenant.
We’re supposed to be surrounded.
— Richard Winters

To my parents and grandparents…

Acknowledgements

It is an amazing experience to spend six years in EPFL pursuing my master and PhD degrees. It is my great honor to have Prof. Giovanni De Micheli and Prof. Pierre-Emmanuel Gaillardon supervising my doctoral researches. Without their insights and tremendous support on both technical works and scientific writings, this work may not be possible. Their serious attitudes on scientific researches drive me to improve my works to the most. In addition, their sincere advices on personal development also inspire me greatly.

I am also grateful to my scientific collaborators: Prof. Paolo Ienne, Dr. Mathias Soeken, Prof. Zhufei Chu, Prof. Vasilis F. Pavlidis, Dr. Jian Zhang, Dr. Hu Xu, Edouard Giacomin, Kim Gain, Dr. Grace Zgheib, Dr. Ana Petkovska and Maxime Thammasack for their advices and important contributions to technical work. In particular, I really appreciate the technical contributions from Prof. Zhufei Chu, Dr. Jian Zhang and Edouard Giacomin. Their works indeed have added remarkable value to my research outcomes.

I should also express my deepest appreciation to Prof. Lingli Wang, who showed me the world of FPGA and taught me good habits at the beginning of my academic career. His encouragement solids my motivation in pursuing a PhD degree.

I would like to express my appreciation to my colleagues in Integrated Systems Laboratory, especially Mme. Christina Govoni for helping me with all the administrative work. I should also express my appreciation to IT manager, Rodolphe Buret, for his hard work in maintaining powerful computers and servers. I thank Dr. Jian Zhang, Prof. Zhufei Chu and Dr. Hu Xu for the collaboration work broadening my vision and knowledge. I am glad to have Winston Jason Haaswijk and Eleonora Testa as my office mate, for sharing happiness and sadness during work hours.

I would like to thank my family: my mother Weiqian Tang, my father Jianhua Zhang, my grandparents Yongming Tang and Jinzhu Chen for supporting me unconditionally all the time. It is their spiritually supports that give me the infinite courage and determination to crash any difficulties during my PhD.

Last but not least, I would like to thank Dr. Jian Zhang, Dr. Hao Zhuang, Dr. Tian Guo, Bin Jin, Yujie Wu, Jun Ma, Dr. Hezhi Zhang, Dechao Sun and all of my friends, who let me enjoy the life in Switzerland and the happy time we spent together.

Lausanne, August 2017 Xifan Tang

Abstract

Field Programmable Gate Arrays (FPGAs) have been indispensable components of embedded systems and datacenter infrastructures. However, energy efficiency of FPGAs has become a hard barrier preventing their expansion to more application contexts, due to two physical limitations: (1) The massive usage of routing multiplexers causes delay and power overheads as compared to ASICs. To reduce their power consumption, FPGAs have to operate at low supply voltage but sacrifice performance because the transistors drive degrade when working voltage decreases. (2) Using volatile memory technology forces FPGAs to lose configurations when powered off and to be reconfigured at each power on.

Resistive Random Access Memories (RRAMs) have strong potentials in overcoming the physical limitations of conventional FPGAs. First of all, RRAMs grant FPGAs non-volatility, enabling FPGAs to be "Normally powered *off*, Instantly powered *on*". Second, by combining functionality of memory and pass-gate logic in one unique device, RRAMs can greatly reduce area and delay of routing elements. Third, when RRAMs are embedded into datpaths, the performance of circuits can be independent from their working voltage, beyond the limitations of CMOS circuits. However, researches and development of RRAM-based FPGAs are in their infancy. Most of area and performance predictions were achieved without solid circuit-level simulations and sophisticated *Computer Aided Design* (CAD) tools, causing the predicted improvements to be less convincing.

In this thesis, we present high-performance and low-power RRAM-based FPGAs from transistor-level circuit designs to architecture-level optimizations and CAD tools, using theoretical analysis, industrial electrical simulators and novel CAD tools. We believe that this is the first systematic study in the field, covering:

From a circuit design perspective, we propose efficient RRAM-based programming circuits and routing multiplexers through both theoretical analysis and electrical simulations. The proposed 4T(ransitor)1R(RAM) programming structure demonstrates significant improvements in programming current, when compared to most popular 2T1R programming structure. 4T1R-based routing multiplexer designs are proposed by considering various physical design parasitics, such as intrinsic capacitance of RRAMs and wells doping organization. The proposed 4T1R-based multiplexers outperform best CMOS implementations significantly in area, delay and power at both nominal and near-V_t regime.

From a CAD perspective, we develop a generic FPGA architecture exploration tool, FPGA-SPICE, modeling a full FPGA fabric with SPICE and Verilog netlists. FPGA-SPICE provides different levels of testbenches and techniques to split large SPICE netlists, in order to obtain better trade-off between simulation time and accuracy. FPGA-SPICE can capture area and power characteristics of SRAM-based and RRAM-based FPGAs more accurately than the currently best analytical models.

From an architecture perspective, we propose architecture-level optimizations for RRAM-based FPGAs and quantify their minimum requirements for RRAM devices. Compared to the best SRAM-based FPGAs, an optimized RRAM-based FPGA architecture brings significant reduction in area, delay and power respectively. In particular, RRAM-based FPGAs operating in the near-V_t regime demonstrate a 5× power improvement without delay overhead as compared to optimized SRAM-based FPGA operating at nominal working voltage.

Key words: Resistive Memory, Field Programmable Gate Array, Circuit Design, Programming Structure, Multiplexer, Physical Design, Computer-Aided Design

Résumé

Les Réseaux de Portes Programmables in Situ (*Field Programmable Gate Arrays* - FPGA) sont des composants indispensables aux systèmes embarqués et aux infrastructures de systèmes de données. Cependant, l'efficacité énergétique des FPGA est devenue une barrière empêchant leur expansion à de nouveaux contextes d'applications, du fait de deux limitations physiques : (1) L'utilisation massive de multiplexeurs de routage engendre une augmentation des délais et de la consommation énergétique par rapport aux ASICs. Afin de réduire leur consommation d'énergie, les FPGAs peuvent fonctionner à faible tension d'alimentation mais cela engendre une perte de performances car les transistors se dégradent lorsque la tension de fonctionnement diminue. (2) L'utilisation d'une technologie de mémoire volatile oblige les FPGA à reconfigurer leurs informations de configurations à chaque mise sous tension.

Les mémoires résistives (*Resistive Random-Access Memory* - RRAM) ont de forts potentiels pour surmonter les limitations physiques des FPGA conventionnels. Premièrement, les RRAMs permettent aux FPGA d'être non-volatiles, leur permettant ainsi de ne pas perdre leur configuration lors de la mise hors tension et d'être instantanément opérationnels lors de la mise sous tension. Deuxièmement, en combinant la fonctionnalité de la mémoire et de la logique des portes de transmission dans un seul et même composant, les RRAM peuvent considérablement réduire l'aire et le délai des éléments de routage. Troisièmement, lorsque les RRAM sont intégrées dans les chemins d'accès, les performances des circuits peuvent devenir indépendante de la tension de fonctionnement, bien au-delà des limites des circuits CMOS. Cependant, les recherches et le développement des FPGA basés sur des RRAMs en sont à leurs débuts. La plupart des prédictions en termes d'aire et de délai ont été réalisées sans simulations approfondies au niveau du circuit et sans outil de Conception *Assistée par Ordinateur* (CAO), rendant incertaines les prédictions de performances.

Dans cette thèse, nous proposons des FPGA haute performance et faible consommation, basés sur RRAMs au travers de l'étude des circuits au niveau du transistor jusqu'aux optimisations architecturales et la création d'outils CAO spécifiques, et en utilisant l'analyse théorique, les simulateurs électriques industriels et les nouveaux outils de CAO. Nous sommes convaincus que c'est la première étude du domaine couvrant :

Du point de vue de la conception de circuits, nous proposons des circuits de programmation efficaces basés sur des RRAMs et des multiplexeurs de routage évalués à la fois à travers des analyses théoriques et des simulations électriques. La structure de programmation 4T(ransitor) 1R(RAM) proposée démontre des améliorations significatives en termes de courant de programmation, par rapport à la structure de programmation 2T1R la plus populaire.

Des multiplexeurs de routage basés sur les structures 4T1R sont proposés en considérant divers facteurs parasites tels que la capacité intrinsèque des RRAMs et l'arrangement des zones de dopage substrat. Les multiplexeurs basés sur les 4T1R surpassent les implémentations CMOS de manière significative en termes d'aire, délai et de consommation énergétique, en régime nominal et en régime proche de la tension de seuil.
Du point de vue de la CAO, nous développons un outil générique d'exploration d'architectures de FPGAs, FPGA-SPICE, capable d'exporter le modèle SPICE ou verilog d'un FPGA complet. FPGA-SPICE fournit différents niveaux de banc d'essais et des techniques pour diviser les larges représentations SPICE afin d'obtenir les meilleurs compromis en termes de temps de simulation et précision. FPGA-SPICE peut capturer les caractéristiques des FPGA basées sur SRAM et RRAM en termes d'aire et de consommation plus précisément que les meilleurs modèles analytiques actuels.
Du point de vue de l'architecture, nous proposons des optimisations au niveau de l'architecture pour les FPGA basés sur des RRAMs et quantifions les spécifications minimales pour les RRAMs. Par rapport aux meilleurs FPGAs basés sur des SRAM, une architecture FPGA optimisée basée sur des RRAMs apporte de grandes améliorations en termes d'aire, de délai et de consommation. En particulier, les FPGAs basées sur des RRAMs fonctionnant en régime proche de la tension de seuil démontrent une consommation énergétique 5 fois inferieur sans délais supplémentaires par rapport aux FPGAs optimisés utilisant des SRAMs et fonctionnant à la tension de travail nominale.

Mots clefs : Mémoire Résistive, Réseaux de Portes Programmables in Situ, Conception de Circuits, Structures de Programmation, Multiplexeur, Conception Physique, Conception Assistée par Ordinateur

Contents

Acknowledgements **i**

Abstract (English/Français/Deutsch) **iii**

List of figures **xi**

List of tables **xvii**

1 Introduction **1**
- 1.1 Overview of RRAMs 2
- 1.2 Advantages and Challenges for FPGAs 4
- 1.3 Opportunities in RRAM-based FPGAs 5
- 1.4 Contributions and Organization 6

2 Background and Previous Works **11**
- 2.1 RRAM Technology 11
  - 2.1.1 Resistive Characteristics 13
  - 2.1.2 Capacitive Modeling 15
  - 2.1.3 Trade-off between R_{LRS} and C_P 16
  - 2.1.4 Co-Integration with CMOS Technology and Scaling Trends 16
  - 2.1.5 Process Variations 18
  - 2.1.6 Material Engineering for Application Requirements 19
- 2.2 Conventional FPGA Architectures 20
  - 2.2.1 Classical Architectures 20
  - 2.2.2 Architectural Enhancements 25
  - 2.2.3 Circuit Designs in FPGAs 30
  - 2.2.4 Memory Technologies for FPGAs 34
- 2.3 Previous works about RRAM-based Circuit Designs and FPGA Architectures 38
  - 2.3.1 Programming Structures 38
  - 2.3.2 Non-Volatile Flip-Flop and SRAM 41
  - 2.3.3 Multiplexer and Crossbar Designs 41
  - 2.3.4 RRAM-based FPGA Architectures 43
- 2.4 FPGA Architecture Exploration Tool and Power Modeling Technique 46
  - 2.4.1 FPGA EDA flow 46

2.4.2 Probability-based Power Estimation Techniques 47
2.5 Summary 52

3 RRAM-based Circuit Designs 53

Part 1: RRAM-based Programming Structures

3.1 Experimental Methodology 54
3.2 Limitations of 2T1R Programming Structure 54
3.2.1 2T1R Circuit Structure 55
3.2.2 I-V Characteristics of 2T1R Structure 56
3.2.3 Physical Design Difficulties 58
3.2.4 Area Estimation 59
3.2.5 Electrical Simulations 60
3.2.6 Discussion About Limitations 61
3.3 2TG1R Programming Structure 62
3.3.1 2TG1R Circuit Structure 62
3.3.2 Area Estimation 63
3.3.3 Electrical Simulations 65
3.3.4 Summary: Advantages and Limitations 66
3.4 4T1R Programming Structure 67
3.4.1 4T1R Circuit Structure 68
3.4.2 Theoretical Analysis on I-V Characteristics 69
3.4.3 Current Density Boosting Methodologies 71
3.4.4 Area Estimation 73
3.4.5 Benefits of 4T1R structures 74
3.4.6 Summary on the 4T1R programming structures 76
3.4.7 Discussion 77

Part 2: RRAM-based Multiplexer Designs

3.5 Basic 4T1R-based Multiplexer 80
3.5.1 Multiplexer Structure and Programming Strategy 80
3.5.2 Limitations from a Physical Design Perspective 82
3.6 Improved 4T1R-based Multiplexer 83
3.6.1 One-level Multiplexer Structure 83
3.6.2 Physical Design Advantages 85
3.6.3 Two-level and Tree-like multiplexer Structure 86
3.6.4 Sharing deep N-Well between multiplexers 88
3.6.5 Constraints on the Programming Voltage V_{prog} 89
3.6.6 Analytical Comparison between 4T1R multiplexers 92
3.7 Optimal Physical Design Parameters 93
3.7.1 RC modeling of General 4T1R-based multiplexers 93
3.7.2 Physical Position of RRAMs 96
3.7.3 Programming Transistor Sizing Technique 97
3.8 Experimental Results 99

3.8.1 Experimental Methodology . . . 99
3.8.2 Transient Analysis . . . 100
3.8.3 Best W_{prog} for RRAM-based Multiplexers . . . 100
3.8.4 Optimal RRAM Location . . . 104
3.8.5 Area Comparison . . . 105
3.8.6 Delay Improvements . . . 106
3.8.7 Energy and Power Benefits . . . 110
3.8.8 Area-Delay and Power-Delay Products Analysis . . . 110
3.9 Impact of Process Variations of RRAMs . . . 110
3.9.1 Impact of Variations on C_P . . . 111
3.9.2 Impact of Variations on V_{set} . . . 112
3.9.3 Impact of Variations on V_{reset} . . . 113
3.10 Summary . . . 114

4 Simulation-based Architecture Exploration Tool 117
4.1 Principles . . . 118
4.1.1 SPICE Modeling . . . 119
4.1.2 Verilog Modeling . . . 119
4.2 Extended Architecture Description Language . . . 121
4.2.1 Transistor-level Module Declaration . . . 121
4.2.2 Physical Structure Modeling . . . 124
4.2.3 Configuration Circuitry . . . 126
4.3 Transistor-level Circuit Netlist Generation . . . 126
4.3.1 Inverters/Buffers . . . 127
4.3.2 Pass-gate Logic . . . 128
4.3.3 SRAM . . . 128
4.3.4 Scan-chain Flip-Flop . . . 129
4.3.5 IO Circuits . . . 130
4.3.6 Multiplexers . . . 130
4.3.7 Look-Up Tables . . . 134
4.3.8 Channel Wire . . . 135
4.4 Netlist Partitioning Strategies . . . 137
4.4.1 Voltage Stimuli and Loads Extraction . . . 138
4.4.2 Parasitic Activity Estimation . . . 138
4.5 Experimental Results . . . 139
4.5.1 Methodology . . . 139
4.5.2 Functional Verification . . . 140
4.5.3 Studies on Runtime, Memory Usage and Accuracy . . . 141
4.5.4 Power Breakdowns . . . 143
4.5.5 Accuracy Examination vs. VersaPower . . . 144
4.5.6 Area Characteristics of SRAM-based FPGAs . . . 144
4.6 Summary . . . 147

5 RRAM-based FPGA Architectures 149
5.1 General Vision . . . 150
5.1.1 Choice of Non-volatile Modules . . . 150
5.1.2 Configuration Circuits . . . 152
5.1.3 Experimental Methodology . . . 153
5.1.4 Area Characteristics . . . 154
5.1.5 Power Characteristics . . . 157
5.1.6 Overall Performance . . . 164
5.2 Architecture-level Optimizations . . . 164
5.2.1 Experimental Methodology . . . 165
5.2.2 Unified Connection Block . . . 166
5.2.3 Increase Capacity of SB MUXes . . . 172
5.2.4 Smaller Best Length Wire < 4 . . . 174
5.2.5 RRAM-based FPGAs vs. SRAM-based FPGAs . . . 177
5.3 Summary . . . 177

6 Conclusion and Future Work 181
6.1 Summary of Contributions . . . 181
6.2 Future Work . . . 185

A An appendix 187
A.1 Examples of FPGA-SPICE Architecture Modeling . . . 187

Bibliography 214

Curriculum Vitae 215

List of Figures

1.1 A RRAM Device (a) sandwiched structure and (b) I-V Characteristics: V_{set} and I_{set} converts part of metal oxide to low-resistance state. 3

1.2 Power consumption of (a) a SRAM-based FPGA and (b) a RRAM-based FPGA. . 5

1.3 Use SRAM + transistors or RRAMs to propagate and block datapath signals. . . 6

2.1 (a) RRAM in pristine state; (b) RRAM in *Low Resistance State* (LRS); (c) RRAM in *High Resistance State* (HRS). 12

2.2 I-V characteristic of (a) a URS RRAM; (b) a BRS RRAM. 12

2.3 (a) Size of filaments inside a RRAM achieved by $I_{set,min}$; (b) Size of filaments inside a RRAM achieved by $I_{set,max}$; (c) I-V characteristics of a RRAM with Bipolar Resistive Switching 14

2.4 Alternative integrations: (a) Natively combine with source/drain or gate of transistors; (b) Locate between metal layers. 17

2.5 Impact of cell area on R_{HRS} and R_{LRS} [Courtesy by [1]]. 18

2.6 Generic FPGA Architecture. 21

2.7 Detailed CLB Architecture. 22

2.8 Bi-directional global routing architecture. 23

2.9 Bi-directional global routing architecture featured by (a) $L = 1$; (b) $L = 2$. 25

2.10 Tile-based FPGA Architecture. 27

2.11 Tile and enhanced CLB architecture. 28

2.12 Uni-directional global routing architecture. 29

2.13 A uni-directional routing track featured by $L = 2$. 29

2.14 (a) Symbol of a N-input routing multiplexer; (b) One-level implementation [2, 3]. 31

2.15 Alternative routing multiplexer design topologies: (a) two-level; (b) tree-like [2, 3]. 31

2.16 Look-Up Table (LUT): (a) principle internal structure; (b) transistor-level design of a 2-input LUT [4]. 33

2.17 Transistor-level design of a master-slave D-type *Flip-Flop* with asynchronous set and reset [4]. 34

2.18 (a) 6-Transistor SRAM design [4]; (b) Configuration circuits for SRAM arrays. . . 35

2.19 Scan-Chain Flip-Flop (SCFF) design and associated configuration circuits [5, 6] 36

2.20 (a) Embedded Flash Process (Courtesy by [7]); (b) Erasing operation of a Flash transistor (Courtesy by [7]); (c) Programming operation of a Flash transistor (Courtesy by [7]). 37

2.21 (a) A transmission gate controlled by a SRAM; (b) Equivalent Flash-based programmable switch. (Courtesy by [7]) 37
2.22 Three most commonly used programming structures: (a) 1T(ransistor)1R(RAM), (b) 1T(ransistor)2R(RAM) and (c) 2T(ransistor)1R(RAM). 39
2.23 A non-volatile master-slave Flip-Flop design [5, 6]. 42
2.24 A non-volatile SRAM design [5, 6]. 43
2.25 Early designs of 2T1R-based multiplexers: (a) A N-input onelevel structure [9]; (b) An illustrative example of two-level and tree-like 4:1 structure [10]. 44
2.26 Early RRAM-based FPGA architectures (a)LUTs embedded with 2T1R programming structures; (b)SRAMs are replaced by 2T1R programming structures. . . . 45
2.27 Classical EDA flow for FPGA architecture exploration purpose. 47
2.28 Examples of signals for switching activity modeling. 48
2.29 Dynamic power modelling: (a) an CMOS inverter with a load capacitance C_L; (b) Equivalent RC model; (c) Input transition from low to high voltage level. 50

3.1 System-level implementations exploiting the 2T1R programming structure: (a) scan chain [8]; (b) memory bank [9]. 55
3.2 A 2T1R programming structure extracted from system-level implementations in Fig. 3.1 57
3.3 I-V characteristics of the 2T1R structure. 58
3.4 (a) Asymmetric bulk management of the 2T1R structure; (b) Symmetric bulk management of the 2T1R structure; (c) Single well application of layout; (d) Triple well application of layout. 59
3.5 Transient analysis on voltages and current in the 2T1R structure during a set process ($W_{prog} = 5$, $V_{prog} = 3.0V$, $W_{inv} = 20$, 1 $W_{prog} = 320nm$). 61
3.6 V_{DS1} and V_{DS2} in 2T1R structure under diverse V_{prog} ($W_{inv} = 20$) 62
3.7 V_{DS1} and V_{DS2} in 2T1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$) 63
3.8 (a) I_{ds} in 2T1R structure under diverse V_{prog} ($W_{inv} = 20$); (b) I_{ds} in 2T1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$) 64
3.9 A 2TG1R programming structure extracted from system-level implementations in Fig. 3.1 65
3.10 V_{DS1} and V_{DS2} in 2TG1R structure under diverse V_{prog} ($W_{inv} = 20$); 66
3.11 V_{DS1} and V_{DS2} in 2TG1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$) 67
3.12 (a) The proposed 4T1R structure (b) Extracted 4T1R structure in a **set** process . 68
3.13 I-V characteristics of the 4T1R structure: (a) V_{set}=V_{reset}; (b) $V_{set} < V_{reset}$ or $I_{set} < I_{reset}$; (c) $V_{set} > V_{reset}$ or $I_{set} > I_{reset}$. 70
3.14 I-V characteristics of the 4T1R structure during set process when: (a) Boosting W_{prog}; (b) Boosting V_{prog}. 71
3.15 Comparison on V_{DS} of programming transistors under diverse W_{prog} and V_{prog} in 2T1R, TG-based 2T1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$) . 75

3.16 Comparison on I_{ds} in 2T1R, 2TG1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$) . . . 76
3.17 Comparison on driving current per minimum transistor width under diverse W_{prog} and V_{prog} between 2T1R, TG-based 2T1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$) . . . 77
3.18 Comparison on area-delay product of 2TG1R and 4T1R structures ($W_{inv} = 20$). 78
3.19 Comparison on power-delay product of 2TG1R and 4T1R structures ($W_{inv} = 20$). 78
3.20 Comparison on R_{LRS} in 2TG1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$) . . . 79
3.21 Circuit design and well arrangement of a naive $N:1$ one-level 4T1R-based multiplexer . . . 81
3.22 Improved one-level N-input 4T1R-based multiplexer: (a) operating mode ($V_{DD,well} = V_{DD}$, $GND_{well} = GND$); (b) set process ($V_{DD,well} = -V_{prog} + 2V_{DD}$, $GND_{well} = -V_{prog} + V_{DD}$); (c) reset process ($V_{DD,well} = V_{prog}$, $GND_{well} = V_{prog} - V_{DD}$; . . 84
3.23 Cross-section of the layout of 4T1R multiplexers: (a) naive design; (b) improved design. . . . 87
3.24 Schematic of a robust two-level N-input 4T1R-based multiplexer. . . . 88
3.25 Schematic of a robust tree-like N-input 4T1R-based multiplexer. . . . 89
3.26 Cascading two N-input one-level 4T1R-based multiplexers: share Deep N-Wells efficiently. . . . 90
3.27 Cross-section of the layout of a 4T1R programming structure: (a) during reset process; (b) during set process. . . . 91
3.28 (a) Critical path of a general RRAM-based multiplexer; (b) General critical path of RRAM-based multiplexer; (c) Equivalent RC model. . . . 94
3.29 Relation between x_i and delay of a RRAM-based multiplexer. . . . 97
3.30 Relation between W_{prog} and delay of a RRAM-based multiplexer. . . . 98
3.31 Transient analysis of a 2-input 4T1R-based multiplexer in Fig. 3.22(a): (a) signal waveforms of programming phase; (b) signal waveforms of operation. . . . 101
3.32 Impact of W_{prog} on the delay of 50-input improved 4T1R-based multiplexers ($x = L$). . . . 102
3.33 Two case studies on the best W_{prog} of improved 4T1R-based multiplexers ($x = L$): (a) impact of the multiplexing structures when $V_{DD} = 0.9V$ (b) impact of V_{DD}. . 103
3.34 Delay comparison of improved 4T1R-based multiplexers featured by $x = 0$ and $x = L$. . . . 104
3.35 Layout of 16-input multiplexers: (a) CMOS two-level structure; and (b) 4T1R-based two-level structure. . . . 105
3.36 Delay comparison between CMOS and 4T1R-based multiplexers: (a) delay improvements of one-level, two-level and tree-like structures ($V_{DD} = 0.7V$); (b) delay efficiency of one-level structure at near V_t regime. . . . 107
3.37 Power comparison between CMOS and 4T1R-based multiplexers: (a) energy improvements of one-level, two-level and tree-like structures ($V_{DD} = 0.7V$); (b) power reduction of one-level structure at near V_t regime. . . . 108

3.38 Comparison between CMOS multiplexers and 4T1R-based multiplexers: (a) Area-Delay Product; (b) Power-Delay Product. 109
3.39 Impact of parasitic capacitance of RRAM C_P on the delay of one-level 4T1R-based multiplexers ($V_{DD} = 0.9V$). 111
3.40 R_{HRS} degradation when $V_{set} = \{0.4, 0.6V, 0.8V\} < V_{DD} = 0.9V$. 112
3.41 (a) R_{LRS} degradation when $V_{reset} = 0.3V$ over $1k$ operating cycles; (b) Voltage across a RRAM in LRS (V_A and V_C in Fig. 3.22(a)) during operation; and (c) R_{LRS} degradation when $V_{reset} = 0.3V$ in a switching cycle. 113

4.1 FPGA-SPICE EDA flow for SPICE modeling purpose. 118
4.2 Ilustration of the full-chip-level testbenches. 120
4.3 Ilustration of the grid-level testbenches. 121
4.4 Ilustration of the component-level testbenches. 122
4.5 FPGA-SPICE EDA flow for synthesizable Verilog purpose. 123
4.6 An I/O pad: (a) VPR abstract-level modeling, and (b) actual physical design. . . 125
4.7 Transistor-level circuit design of (a) an inverter and (b) a tapered buffer. 127
4.8 Transistor-level circuit design of (a) a global routing multiplexer, (b) a local routing multiplexer, and (c) the internal tree-like structure. 131
4.9 Transistor-level circuit design of a 4T1R-based multiplexer. 133
4.10 An example of the transistor-level design of a LUT 135
4.11 (a) A length-2 unidirectional wire (highlighted in red) within FPGA routing architecture; (b) Corresponding RC modeling of segments 136
4.12 Ilustration of the voltage stimuli generation and load extraction techniques. (a) BLE multiplexer with its architectural context; (b) extracted testbench. 137
4.13 An example for parasitic nets estimation. 138
4.14 An illustration of the waveforms for functional verification purpose. 141
4.15 Waveforms of a sample circuit: inverter, achieved by ModelSim simulation: (a) full waveform with configuration phase highlighted in red rectangle and operation phase highlighted in blue rectangle; (b) an example of a programming clock cycle; (c) an example of a operating clock cycle. 142
4.16 Power breakdown results of the considered FPGA architecture between FPGA-SPICE and VersaPower averaged over the MCNC big20 benchmark suite for 22nm, 45nm and 180nm technology nodes. 145
4.17 Full-chip layouts of 40nm SRAM-based FPGAs with CLB array size 5×5, a channel width of 300. 146
4.18 Area breakdown of SRAM-based FPGAs which are configured by (a) BL/WL decoders, and (b) scan-chain flip-flops. 147

5.1 Memory access organization in SRAM-based FPGA: SRAMs are placed in an array and SRAMs in the same column/row share the same BL/WL. 150
5.2 Memory access organization in RRAM-based FPGA: RRAMs belonging to the same multiplexer/NV SRAM are placed in the same column and share BL/WL. 151

5.3 Full-chip layouts of 40nm SRAM-based and RRAM-based FPGAs with CLB array size 5 × 5. 155
5.4 Area breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA. 155
5.5 Full-chip area comparison between SRAM-based and RRAM-based FPGAs by sweeping channel widths from 50 to 300. 156
5.6 Standard cell area comparison between SRAM-based and RRAM-based FPGAs by sweeping channel widths from 50 to 300. 156
5.7 Leakage paths of N-input multiplexers: (a) SRAM-based (b)RRAM-based 159
5.8 Impact of R_{HRS} on the average static power of a 2-input 4T1R-based multiplexer 160
5.9 Impact of R_{HRS} on the average static power of a 2-input 4T1R-based multiplexer with tapered buffer at output 161
5.10 Normalized power consumption of SRAM-based and RRAM-based architectures with different R_{HRS} 162
5.11 Static power breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA. . 163
5.12 Dynamic power breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA.163
5.13 Area, delay and energy comparison between SRAM-based and RRAM-based FPGAs operating at nominal and near-V_t regime. 164
5.14 Classical interconnection from routing tracks to LUT inputs. 167
5.15 Proposed interconnection from routing tracks to LUT inputs. 168
5.16 An illustrative example of the proposed routing architecture($K = 6$) with $F_{c,in} = 0.33$ and $F_s = 6$. 169
5.17 Normalized average area, delay, power and channel width of baseline and proposed architecture by sweeping $F_{c,in}$: (a) SRAM-based architectures; (b) RRAM-based architectures. 170
5.18 Tile area comparison between a traditional FPGA architecture and the proposed RRAM FPGA architecture for different channel width W. 171
5.19 (a) Driver multiplexer and fan-outs of a Length-L wire; (b) Equivalent RC model of a Length-L wire. 172
5.20 Normalized average area, delay, power and channel width of baseline and proposed architectures by sweeping F_s: (a) SRAM-based architectures; (b) RRAM-based architectures. 173
5.21 Normalized average area, delay, power and channel width of baseline and proposed architectures by sweeping L: (a) SRAM-based architectures; (b) RRAM-based architectures. 176
5.22 Normalized average area, delay, energy and channel width of baseline and proposed architectures: (a) baseline SRAM-based architectures; (b) baseline RRAM-based architectures; (c) proposed RRAM-based architectures 178
5.23 Normalized average area, delay, power, channel width, ADP and PDP of classical SRAM-based and proposed RRAM-based architectures. 178

List of Tables

2.1 Bipolar RRAMs with different metal oxide materials 19
2.2 FPGA Architecture Parameters 24
2.3 Analytical comparison between CMOS one-level, two-level and tree-like multiplexers 32
2.4 Static probability and transition density of the signals in Fig. 2.28. 48

3.1 Voltages arrangements for operation, set and reset examples in Fig. 3.22(a)(b)(c) 85
3.2 Analytical comparison on area, delay and switching energy of N-input 4T1R-based multiplexers. 92

4.1 Comparison of runtime, memory usage and total power of full-chip/grid/component-level testbenches for 22nm, 45nm and 180nm technology nodes in the case of the MCNC big20 benchmark *s298*. 143
4.2 Comparison of accuracy by modules in full-chip/grid/component-level testbenches for 22nm, 45nm and 180nm technology nodes in the case of the MCNC benchmark big20 *s298*. 144

5.1 Resistance of leakage paths of the 4T1R-based multiplexer in 5.7(b) whose starting point is **p3** and ending points are **n4**, **n5**, **n6** and **n7** 159
5.2 Delay comparison between SRAM-based and RRAM-based routing multiplexers. 165

6.1 Summary of Contributions in Differnt Research Fields. 182

1 Introduction

Strong demand from the *Internet of Things* (IoT) have fueled researches on high-performance and energy-efficient computer-based systems [10, 11, 12]. We confront challenges from two-pronged ecosystems in IoT: low-power mobile devices and cloud services. The mobile devices are supposed to stay active for a long period with a limited battery life. For these devices, energy-efficiency is the most critical factor due to a tight power budget. Cloud services are actually provided by datacenters, aiming at processing huge amount of data from mobile devices or other sources. For datacenters, high-performance computing is a more important metric than energy efficiency since they are supposed to deal with abundant data while being power supplied through the grid.

Since invented in 1984, *Field Programmable Gate Arrays* (FPGAs) have demonstrated themselves not only as an alternative implementation media of *Application Specific Integrated Circuits* (ASICs) but also as an indispensable component of embedded systems and datacenter infrastructures [13, 14], growing to a $ 4.5 billion per year industry [15, 16]. The programmability and large I/O bandwidth of FPGAs brings significant advantages in realizing energy-efficient and high-throughput applications, e.g., deep learning network [17]. Meanwhile, programmability and I/O bandwidth cost general FPGA implementations 20× bigger area, 4× longer delay and 12× higher power consumption, when compared to ASICs [18]. Such overheads prohibit FPGAs from massive deployment in ultra-low-power embedded systems.

Resistive Random Access Memories (RRAMs) [1, 19], a member of the emerging *Non-Volatile Memories* (NVM) family [20], have become a promising candidate in displacing conventional memory technologies of FPGAs, such as SRAM [21] and Flash [7]. Potentials of RRAMs have been investigated in many fields, i.e., memory storage [22], neuromorphic computing [23], hardware security [24] and FPGAs [25, 9, 26, 27, 28]. In particular, RRAM-based FPGAs are predicted to improve area, delay and power in addition to non-volatility, thus being an effective component for IoT applications. Still, researches and development of RRAM-based FPGAs are in their infancy. Circuit simulations focus on functional verification and employ analytical RRAM models. Area and performance predictions are achieved without fully considering physical design issues, e.g., the parasitic effects of RRAMs and their associated transistors.

Additionally, the efficiency of RRAM-based circuit topologies has not been carefully examined. Lacking solid circuit-level studies, FPGA architecture explorations based on RRAMs would be less meaningful. Moreover, current FPGA architecture exploration tools provide limited supports in accurate power analysis, especially for emerging memory technologies. It is entirely possible that the predicted improvements of RRAM-based FPGAs are counteracted when the parasitic effects are considered and accurate power analysis are conducted. Therefore, it is necessary to examine the concept with realistic device modelling, circuit designs under physical design considerations and accurate architecture-level simulations.

In this thesis, we present RRAM-based FPGAs from transistor-level circuit designs to architecture-level optimizations and fast prototyping techniques. We validate their high-performance and low-power advantages over *Static Random Access Memory* (SRAM)-based FPGAs with theoretical analysis, industrial electrical simulators and novel *Electrical Design Automation* (EDA) tools. We believe that this is the first systematic study about RRAM-based essential circuit designs and FPGA architectures. To motivate our work, the rest of this chaper is organized as follows. Section 1.1 provides a brief overview about RRAM technology and explains their outstanding features to be exploited in circuit designs and FPGAs. Section 1.2 is devoted to analyzing the advantages of SRAM-based FPGAs and their bottlenecks in low-power applications. Section 1.3 introduces the opportunities of RRAM-based FPGAs in overcoming the limitations of their SRAM-based counterparts. Section 1.4 lists the major contributions of this thesis and the approaches to achieve them.

1.1 Overview of RRAMs

Since their popularization in 2004 [29], *Resistive Random Access Memories* (RRAMs) are expected to trigger revolutionary changes in many applications. In terms of functionality, a RRAM can be simply regarded as a non-volatile configurable resistor, which can hold information when powered down. A RRAM device exhibits resistive switching between *High Resistance State* (HRS) and *Low Resistance State* (LRS) thanks to forming and rupturing the conductive filaments in its metal oxide, as illustrated in Fig. 1.1(a). By applying a proper combination of programming voltage and programming current between electrodes, resistance states can be switched, following the I-V curve in Fig. 1.1(b).

The non-volatile property of RRAMs have attracted interest in replacing SRAMs, *Dynamic Random-Access Memories* (DRAMs) and even Flash RAMs in computer systems. Compared to volatile memories, e.g., SRAMs and DRAMs, using RRAMs can save reconfiguration time and energy when the entire system wakes up from sleep modes, appealing to IoT and mobile applications. Different from Flash memory, RRAMs are compatible with *Back-End-of-Line* (BEoL) fabrication and hence are envisioned to be stacked on the top of the transistors, reducing fabrication cost and improving footprint of whole system. Besides, BEoL compatibility allows memories to be close to the computing logic, significantly reducing the access time to memories.

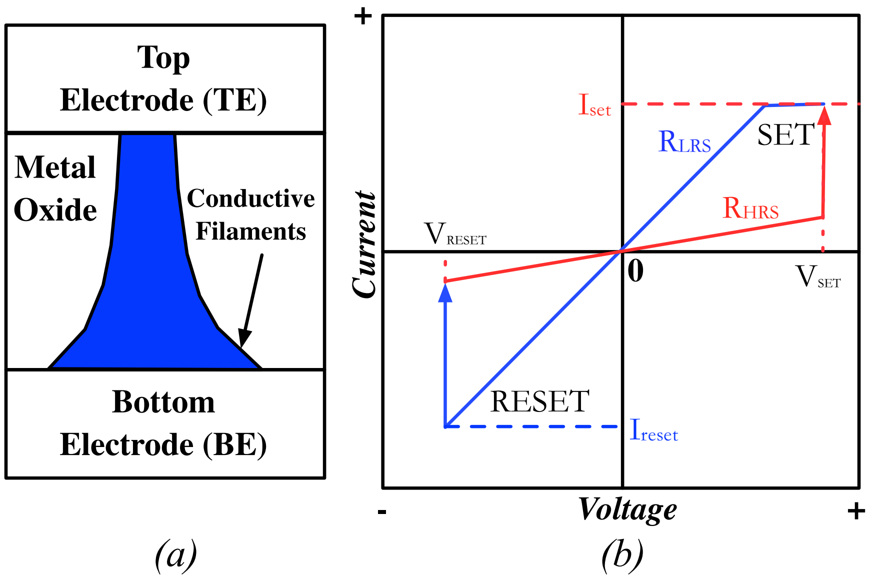


Figure 1.1 – A RRAM Device (a) sandwiched structure and (b) I-V Characteristics: V_{set} and I_{set} converts part of metal oxide to low-resistance state.

The configurable resistive property of RRAMs have been catalyst of research in In-Memory Computing [30, 31, 32], Neuromorphic Computing [33, 34] and *Physical Unclonable Function* (PUF) [35, 36]. The HRS and LRS can represent '0' and '1' in boolean logic, similar to the *on* and *off* states of a transistor. Hence, the two resistance states can be exploited to realize digital circuits, replacing transistors [37, 30, 38]. Interestingly, even a RRAM-based memory array is capable of implementing logic gates such as majority gate by properly connecting RRAMs [30, 31]. Such capability is called In-Memory Computing, which enables simple computing tasks to be shifted from CPUs to memories. Since long memory access time becomes a major bottleneck in accelerating modern CPU-based systems, such computing paradigm provides a promising solution. More than boolean logic, RRAMs can also realize multi-value logic thanks to its tunable resistance. By adjusting programming current, RRAMs can achieve resistance between HRS and LRS, which is a unique advantage of RRAMs over other NVM technologies, such as *Magnetoresistive Random Access Memories* (MRAMs) [39] and *Phase-Change Random Access Memories* (PCRAMs) [40]. Such resistive characteristic allow RRAMs to model the states of a neuron in human brain, which is the basis of Neuromorphic Computing. Furthermore, the stochasticity in resistive switching mechanism leads to that resistance of RRAMs is different from cycle to cycle [1]. As a result, RRAMs can be employed in PUF designs as the key to encrypt hardware designs.

In particular, the programmable resistance, non-volatility and BEoL features are attractive to FPGAs, where 90% of area is consumed by volatile memory cells and programmable routing elements. More issues about RRAM-based FPGAs will be discussed in Section 1.3.

1.2 Advantages and Challenges for FPGAs

Thanks to their rich programmable resources, FPGAs can implement *any* circuits by appropriately configuring memory cells and thus have two benefits over other implementations, e.g., ASICs :

(1) Low *Non-Recurring Engineering* (NRE) costs. In addition to design efforts, fabricating an ASIC chip requires heavy NRE fees from silicon manufacturer (for example, > $1 million for 14nm FinFET technology), covering the cost of making lithography masks, wafer-level packaging and building testing platforms. With FPGAs, not only NRE costs but also design efforts can be saved since implementing circuits only involves programming existing silicon.

(2) Fast time-to-market. Full fabrication of an ASIC chip typically requires more than 6 weeks while a FPGA can be instantly programmed and deployed in a system. To make things worse, more iterations on designing and fabrication are needed if any problems are detected in the first manufacturing. Short production cycles is compelling nowadays as competition in consumer electronics becomes fierce.

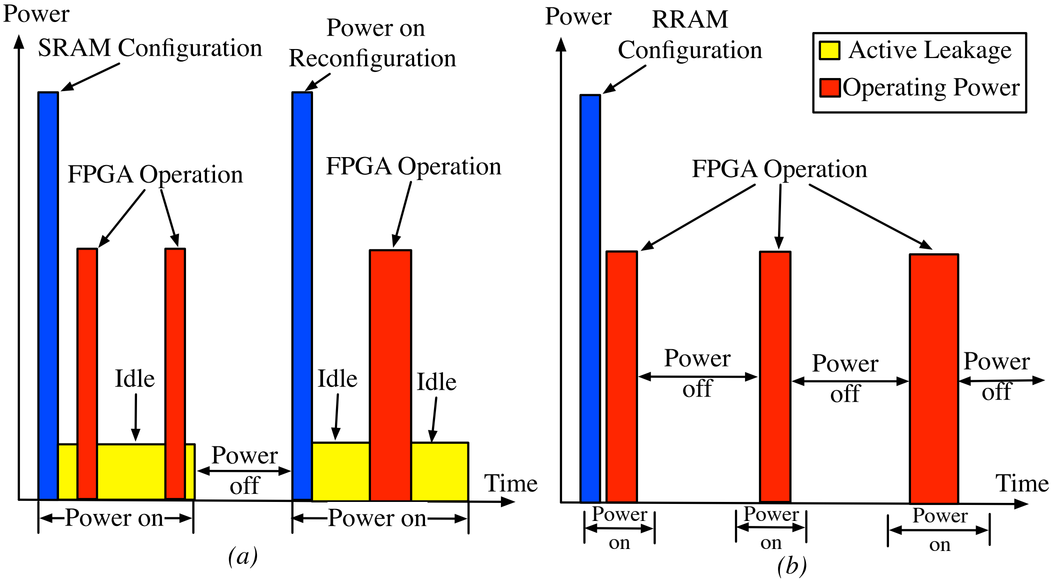
Therefore, once introduced, FPGAs gain popularity in low volume applications where ASIC manufacturing cost is extremely high. Recent years witness FPGA's expansion in medium or even high volume applications, i.e., co-processors, thanks to their programmable and parallel nature. FPGAs can efficiently parallelize algorithms that are hard for *Central Processing Unit* (CPU) + *Graphic Processing Unit* (GPU) platforms, such as machine learning and video encoding/decoding. An representative example is Microsoft's Bing Search Engine, which employs CPU + FPGA platforms and achieves 40× speed-up [13, 14].

Despite their success, FPGAs are facing challenges from their physical limitations generally preventing them to embrace the IoT era. Programmable routing multiplexers in FPGAs have higher resistance and capacitance than metal wires and also drive more fanouts to guarantee routability, consuming more area and reducing circuit speed. Intensive usage of routing multiplexers introduces more signal activities, causing significant power overhead. To reduce power consumption, FPGAs have to operate at low supply voltage but sacrifice performance because speed of transistors have to degrade when working voltage decreases [41, 42, 43]. Using volatile memory technology, i.e., SRAMs, forces FPGAs to lose configurations when powered down and to be reconfigured at each power on. Such drawback leads to embarrassment in using FPGA-based embedded systems, as illustrated in Fig. 1.2(a): Power-off has to pay additional reconfiguration time and energy next time wake up. Otherwise, power-on burns more power and reduce battery cycle. To continue the success in future, it is worthwhile to advance FPGA technology by overcoming these physical limitations.

1.3 Opportunities in RRAM-based FPGAs

RRAM-based technology can bring three fundamental advancements to FPGA architectures, meeting the low-power demands of IoT:

(1) Non-volatility of RRAMs allows FPGAs to be frequently switched on and off without the additional reconfiguration time and energy, as depicted in Fig. 1.2(b). When powered down, RRAM-based FPGAs can hold configurations and consume zero leakage power. Such "Normally *off*, Instantly *on*" property can be achieved by simply replacing SRAMs with RRAMs [25].



(a) *(b)*

Figure 1.2 – Power consumption of (a) a SRAM-based FPGA and (b) a RRAM-based FPGA.

(2) Fig. 1.3 illustrates that *Low Resistance State* (LRS) and *High Resistance State* (HRS) of RRAMs can be exploited to replace pass-gate logic in programmable routing multiplexers and propagate datapath signals [9, 26, 27, 28]. Combining functionality of memory and pass-gate logic in one unique device, RRAMs can narrow the gap between programmable routing multiplexers and long metal wires. Replacing both SRAMs and pass-gate logics, RRAMs greatly reduce area since they are fabricated on the top of transistors. Implanting RRAMs into datapaths leads to less parasitic capacitance than SRAM-based multiplexing structures, contributing to smaller delay [3]. RRAM-based implementations enable area and speed of programmable routing multiplexers to be comparable or even smaller than a long metal wire, fundamentally changing the cost functions considered in FPGA architectures [28].

(3) RRAMs have stable resistances when exposed below programming threshold voltage. As

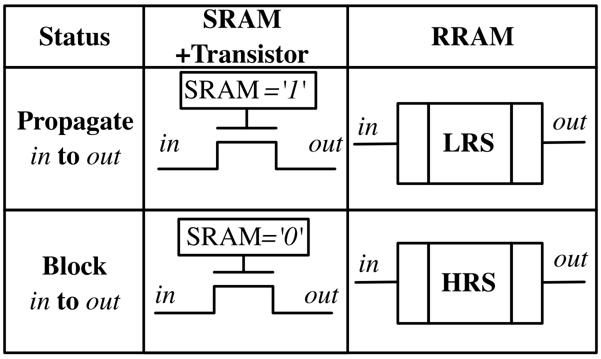


Figure 1.3 – Use SRAM + transistors or RRAMs to propagate and block datapath signals.

long as the working voltage is kept lower than threshold voltage of RRAMs, RRAM-based circuits and systems can exhibit resistive property independent from their work voltage, beyond the limitations on transistors [6]. Hence, using RRAMs in datapaths can have a better trade-off between power and delay than transistors. For instance, RRAM-based circuits operating in the near-V_t regime keep the same performance level as if they were operated at a nominal working voltage, while their power consumption is sharply reduced. Overall, the energy efficiency of FPGAs can be profoundly improved when adapted to RRAM technology [28].

Note that ASICs cannot benefit large improvements from RRAMs as FPGAs, because they seldom use programmable routing multiplexers . Therefore, RRAM-based programmable routing multiplexers open an exclusive opportunity for FPGAs to catch up with ASICs in performance and power. Furthermore, physical features of RRAMs may also expand FPGA's application fields. For example, FPGAs would become popular in aerospace applications since RRAMs are more robust to high-energy radiations than SRAMs.

1.4 Contributions and Organization

This thesis provides a thorough study of the fundamentals of RRAM-based FPGAs, starting from essential circuit designs, i.e., programming structures to architecture-level optimizations and prototyping with novel *Electrical Design Automation* (EDA) tools. In order to reveal important characteristics of RRAM-based FPGAs, our researches are conducted in three aspects: circuit design, architecture exploration tool development and architecture-level optimizations.

The rest of this thesis is organized as follows.

Chapter 2 provides background knowledges covering

(1) RRAM technology: We explain working principles, electrical characteristics and unique technology features of RRAMs, which bring both benefits and challenges to RRAM-based circuit designs.

(2) modern FPGA architectures: We describe basic principles and important enhancements in modern FPGAs, which are the baseline FPGA architecture considered in Chapter 5.

(3) previous works about RRAM-based circuit designs and FPGA architectures: We analysis significance and limitations of circuit topologies, including memory cells, flip-flops and routing multiplexers.

(4) FPGA architecture exploration tools: We introduce EDA techniques of current state-of-art academic tool, i.e., VPR [44] and discuss limitations of power analysis with analytical models.

Chapter 3 aims to propose efficient RRAM-based programming circuits and routing multiplexers. The RRAM-based circuits are studied through both theoretical analysis and electrical simulations with physical design considerations. A low R_{LRS} is commonly considered as the key to guarantee high-performance for RRAM-based circuits. This chapter argues that the high-performance and energy-efficiency of RRAM-based circuits are actually impacted by many other factors, e.g., programming transistors, well organization and physical location of RRAMs. The first study is about how to program RRAMs into LRS with transistors efficiently. Most popular programming structure, i.e., 2T(ransitor)1R(RAM), cannot leverage the full driving strength of transistors, which potentially causes low circuit speed due to a higher R_{LRS} than expected. A more efficient programming structure, namely 4T(ransitor)1R(RAM), is proposed and it demonstrate significant improvements in programming current, guaranteeing a low R_{LRS}. Experimental results prove that using pairs of p-type and n-type transistors are better in driving programming current and also more flexible to diverse RRAM devices, than purely using n-type transistors. By exploiting 4T1R, high-performance and low-power RRAM-based routing multiplexer designs are proposed by considering various physical design parasitics, such as intrinsic capacitance of RRAMs and well organization. Chapter 3 draws three crucial conclusions:

(a) despite from R_{LRS}, parasitics of programming transistors is another important factor to guarantee high-performance for RRAM-based circuits. To obtain the best trade-off between R_{LRS} and parasitics of programming transistors, programming transistor sizing technique is proposed. Experimental results validate that best performance is often achieved with a R_{LRS} larger than its lowest value.

(b) By sharing programming transistors in multiplexing structure, performance of RRAM-based routing multiplexer is underlinear to input size, encouraging the use of large multiplexers. Actually, in large RRAM-based routing multiplexer, circuit design topology becomes the major source of high-performance, rather than a low R_{LRS}.

(c) When RRAMs are embedded in datapath, performance of RRAM-based circuits is not sensitive to working voltage. As a result, operating at near-V_t regime, RRAM-based circuits can keep the same performance level as nominal working voltage, meanwhile their power consumption is sharply reduced. This implies outstanding energy-efficiency and can be generalized to any circuit with RRAMs in datapaths.

With a commercial 40nm technology, we investigate area, delay and power improvements of RRAM-based multiplexing structure by comparing to best SRAM-based implementations. To ensure the accuracy of comparisons, layouts of RRAM-based and SRAM-based routing multiplexers are generated with industrial EDA tools, i.e., Cadence Virtuoso [45] and layout-level parasitic effects are back-annotated in electrical simulations. We believe that the conclusions are generic and instructive when developing novel RRAM-based circuits.

Chapter 4 introduces generic FPGA architecture exploration tool, FPGA-SPICE, for emerging technologies. Current state-of-art FPGA architecture exploration tool, i.e., VPR [44, 46], evaluates area, delay and power with analytical models, which cannot accurately capture the trends of FPGAs based on emerging technologies, such as RRAMs. In addition, VPR provides limited support in prototyping novel FPGA architecture. FPGA-SPICE is developed to enable accurate power analysis and fast prototyping for diverse FPGA architectures, including both SRAM-based and RRAM-based. FPGA-SPICE can auto-generate *Simulation Program with Integrated Circuit Emphasis* (SPICE) netlists, modeling a full FPGA fabric. With SPICE netlists and electrical simulator, i.e., HSPICE [47], accurate power analysis can be conducted. To accurate model physical designs in SPICE netlists, FPGA-SPICE extends the FPGA architectural description language [48] by providing rich transistor-level modeling parameters. Large SPICE netlist, e.g., the one containing a full FPGA fabric, requires a long simulation time. FPGA-SPICE provides different levels of testbenches and techniques in split large SPICE netlists, in order to obtain better trade-off between simulation time and accuracy. In addition, FPGA-SPICE is also capable of auto-generating synthesizable Verilog netlists containing a full FPGA fabric. Verilog netlists can be used to verify the functionality of FPGA designs and also allows engineers to prototype FPGA architectures through a semi-custom design flow. FPGA-SPICE can be useful in many research topics, including but not limited to the following. The power results from FPGA-SPICE can be a baseline when examining the accuracy of analytical power models for FPGA. The accurate power results are an important benchmarking metric when evaluating novel FPGA architecture. SPICE netlists help validating the functionality and performance of circuit designs based on emerging technologies. Synthesizable Verilog netlists simplify the processes in examining the feasibility of novel FPGA architectures.

Chapter 5 focus on architecture-level optimizations in FPGA to leverage the potential of RRAM-based multiplexers proposed in Chapter 3. The architectural parameters, routing architectures and buffering strategy are modified to exploit the high-performance of large RRAM-based multiplexers. We propose that local routing architecture should be unified to connection blocks, in order to achieve high-performance when using RRAM-based multiplexers. Connectivity parameters F_s and best length of routing wire L should be tweaked because

RRAM-based multiplexers are faster in delay than long metal wires. In addition, we propose configuration circuits for the novel RRAM-based FPGA architecture and verify its efficiency with FPGA-SPICE. With cutting-edge EDA tools, VPR and FPGA-SPICE, we believe that the architectural-level results are realistic enough to validate the area, delay and power benefits of RRAM-based FPGAs. We believe that the methodology in architecture evaluation can be generalized to developing FPGA architectures based on emerging technologies.

Chapter 6 summarizes important conclusions in circuit designs, FPGA-SPICE and RRAM-based FPGA architectures. It concludes what is the basis of high-performance and energy-efficiency of RRAM-based FPGAs, and also provides suggestions for future work.

Appendix A includes an example of modern FPGA architectures modelled by FPGA-SPICE architecture description language, which is also the baseline FPGA architecture considered in this thesis.

2 Background and Previous Works

As motivated in Chapter 1, RRAMs are promising to advance FPGA technology. The research on RRAM-based FPGA requires a wide range of background knowledge including RRAM technology, circuit designs, FPGA architecture and EDA techniques. Without any of these, evaluating RRAM-based FPGAs would not be possible with a proper level of accuracy. This chapter aims at providing the sufficient background information required for studying RRAM-based FPGAs and therefore consists of four parts. Section 2.1 introduces *Resistive Random Access Memory* (RRAM) technology, covering device structures, physical mechanism and electrical characteristics. These important features of RRAMs help us understanding their potentials in circuit designs. Section 2.2 presents detailed conventional FPGA architectures, including a few crucial architectural enhancements, circuit design topologies and memory technology. These details provide a solid foundation for developing RRAM-based FPGAs in Chapter 5. Section 2.3 reviews previous works about RRAM-based circuit designs and FPGA architectures, which stands as baseline in Chapter 3 and Chapter 5. Last but not least, we discuss current state-of-art FPGA architecture exploration tools and their limitations especially in terms of power analysis, motivating us to develop FPGA-SPICE in Chapter 4.

2.1 RRAM Technology

Resistive Random Access Memory (RRAM) device technology typically relies on a three-layer material stack, namely a *Metal-Insulator-Metal* (MIM) structure [1]. As depicted in Fig. 2.1(a), a RRAM cell is a two-terminal device, consisting of a *Top Electrode* (TE), a metal oxide insulator and a *Bottom Electrode* (BE). RRAMs can be programmed into two stable resistance states, a *Low Resistance State* (LRS) and a *High Resistance State* (HRS) respectively by modifying the conductivity of the metal oxide layer. Applying a combination of programming voltages and currents between TE and BE can trigger switching events between HRS and LRS. The switching event from HRS to LRS is called the "set" process. Conversely, the switching event from LRS to HRS is called the "reset" process. We denote the resistance of a RRAM in LRS and HRS as R_{LRS} and R_{HRS} respectively.

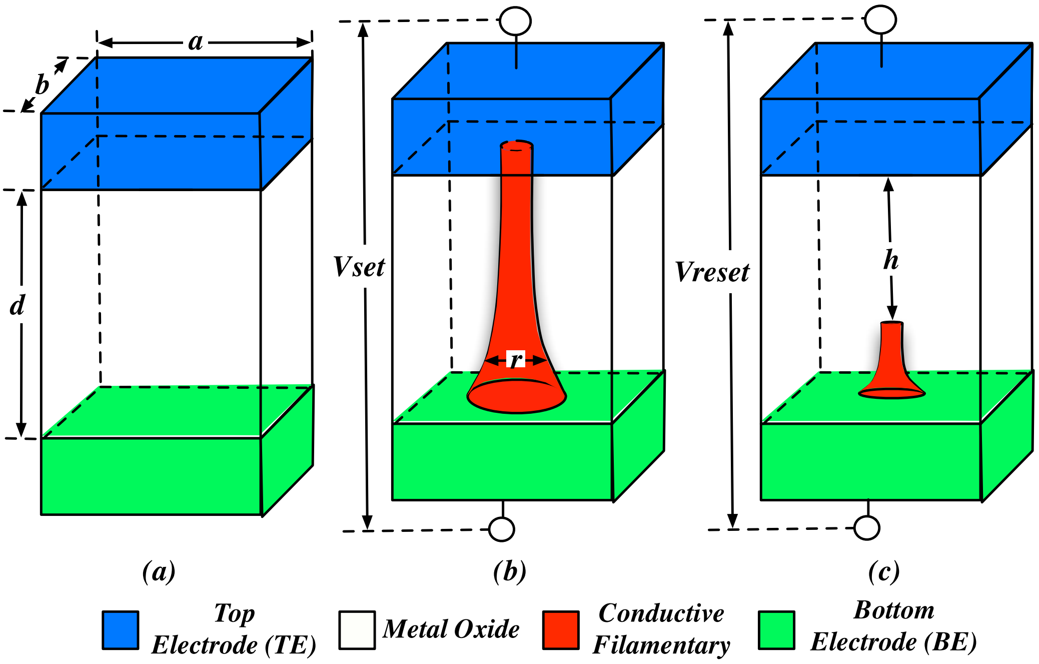


Figure 2.1 – (a) RRAM in pristine state; (b) RRAM in *Low Resistance State* (LRS); (c) RRAM in *High Resistance State* (HRS).

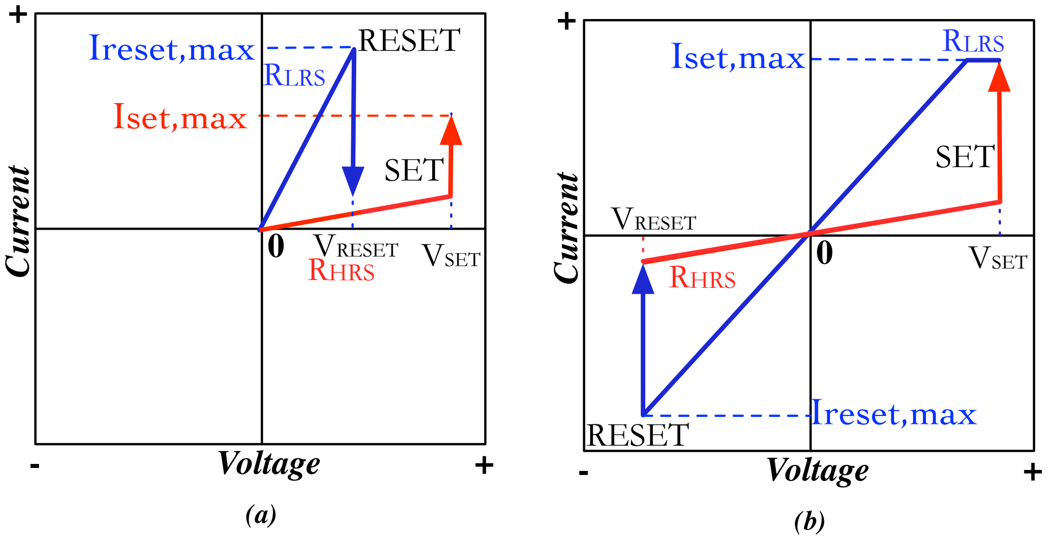


Figure 2.2 – I-V characteristic of (a) a URS RRAM; (b) a BRS RRAM.

In terms of the polarity of programming voltages, RRAMs can be categorized into *Unipolar Resistive Switching* (URS) and *Bipolar Resistive Switching* (BRS) [1]. Fig. 2.2(a)(b) compare the I-V curves of URS and BRS RRAMs. Take the example in Fig. 2.2(a), resistive switching of URS RRAMs depends on the amplitude of V_{set} and V_{reset} but not the polarity, in order to trigger **set** and **reset** processes. In contrast, BRS RRAMs account on the polarity as well as the amplitude of V_{set} and V_{reset} in programming. Take the example in Fig. 2.2(b), a set process can only be triggered by a positive programming voltage, while a subsequent reset process can only be invoked by a negative programming voltage. The minimum programming voltage inducing a positive programming current is defined as V_{set}, while the minimum programming voltage leading to a negative programming current is V_{reset}. In principle, for both types of RRAMs, a programming process can only be triggered by a proper programming voltage while the achieved R_{LRS} and R_{HRS} are determined by the provided programming current. The rest of this thesis will focus on BRS RRAMs because that they are widely adopted in RRAM-based FPGA researches.

In order to **set**/**reset** the RRAM into a stable resistance state, programming voltages should be applied for a given time [1]. The minimum pulse width of programming voltage determines the writing speed of the RRAM [1]. Besides, RRAMs should be able to afford a reasonably large number of writing operations, expressed by the endurance [1], and also should be able to maintain the resistance state for a long period without degradation, expressed by the retention [1].

In the following subsections, we present in-depth knowledge about the RRAM technology from five major aspects: resistive characteristics (subsection 2.1.1), capacitive properties (subsection 2.1.2), fabrication issues (subsection 2.1.4), process variations (subsection 2.1.5) and material engineering (subsection 2.1.6).

2.1.1 Resistive Characteristics

The metal oxide material is the key component of a RRAM that can exhibit resistive switching, whose working principle is mostly based on filamentary conducting mechanism.

In its pristine state (Fig. 2.1(a)), the oxide material is a pure insulator without any *Conductive Filament* (CF). In this case, a RRAM has an extremely high resistance and can be approximately treated as a pure capacitor. A pristine RRAM first go through the "forming" process, after which the device can be freely switched between HRS and LRS. The forming process is to initialize a conductive path in metal oxide, which is achieved by polarizing the memory to a positive bias. The formation of the initial conductive path requires a high electric field in the purpose of knocking the oxygen atoms out of the lattice and creating defect-rich regions in the metal oxide. The localized defects can be generated by **set** processes or recovered during **reset** process, and hence they are regarded as the sources of configuring CFs. To establish such strong electric field, the forming voltage should be high enough, which is typically larger in amplitude than normal **set** voltage. To some extent, the forming process is a special set process because the forming voltage has the same polarity as the set voltage. By carefully controlling the size and materials of the oxide, RRAMs can get rid of forming process, which

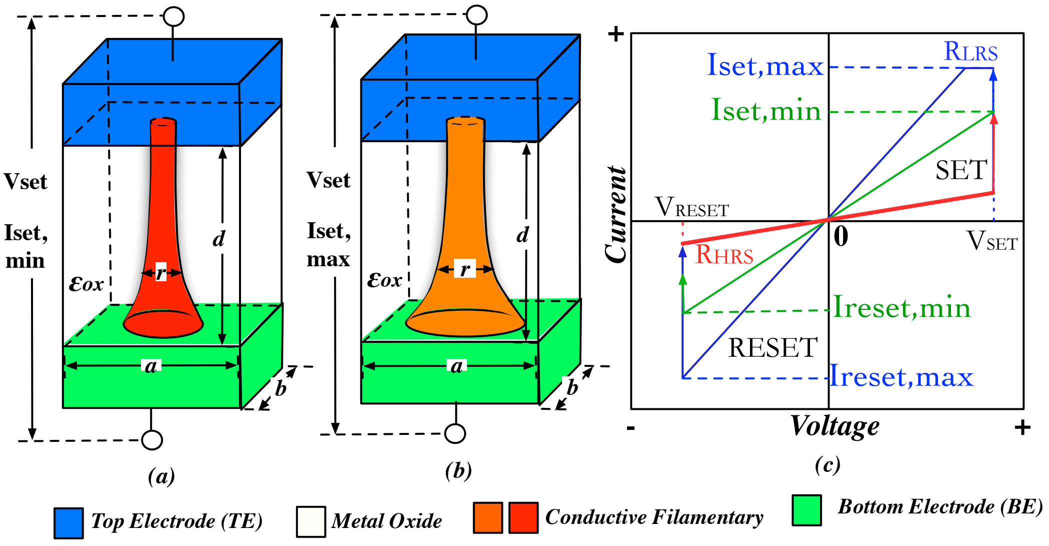


Figure 2.3 – (a) Size of filaments inside a RRAM achieved by $I_{set,min}$; (b) Size of filaments inside a RRAM achieved by $I_{set,max}$; (c) I-V characteristics of a RRAM with Bipolar Resistive Switching

are so called "forming-free" devices [49, 50].

After the forming process, a RRAM device is initialized to LRS, with a CF through the oxide as shown in Fig. 2.1(b). When a reset voltage V_{reset} is applied, the CF created by the set/forming process is partially or fully ruptured to the low-conductivity oxide, leading to an increment in resistance. During the reset process, when the CF is separated from the TE, the RRAM is considered to be in HRS and the minimum I_{reset} required is defined as $I_{reset,min}$. Fig. 2.1(c) exemplifies the resulting CF and oxide during the reset process. The exhibited R_{HRS} depends on the distance between the top of the CF and the TE, denoted as h in Fig. 2.1(c). Because a large reset current leads to a strong rupture of CF and thus increases h, R_{HRS} are positively related to the reset current. Note that I_{reset} should be correlated to the I_{set} in last switching, in order to restore the oxide to its original state before set. A small I_{set} leads to weak CFs, which requires a small I_{reset} to be ruptured. In the example of Fig. 2.3(c), a set process achieved by $I_{set,min}$ requires at least $I_{reset,min}$ in the subsequent reset process [1].

In the subsequent resistive switching cycles, a RRAM in HRS can be configured to LRS with a set voltage, which is smaller than the forming voltage. When a set voltage V_{set} is applied across the two electrodes, part of the oxide is transformed to the CFs, as illustrated in Fig. 2.1(b). When there is a CF through the oxide, the RRAM is considered to be in LRS and the minimum I_{set} required is defined as $I_{set,min}$. In addition, a current compliance $I_{set,max}$ is often enforced to avoid a permanent breakdown of the device. In practice, current compliance is usually provided by the programming transistors. Note that the I_{set} modulates the diameter of CF, and thus impacts on the achieved R_{LRS}. Fig. 2.3(a)(b) illustrates two CFs which are shaped by two programming currents $I_{set,min}$ and $I_{set,max}$, corresponding to the green and blue set curves in Fig. 2.3(c) respectively. The R_{LRS} of a RRAM is typically following a linear or ohmic relationship with the programming current passing through it, when the applied voltage is lower than V_{set} [49]. Therefore, the higher programming current we drive, the lower R_{LRS} we obtain. This reveals one of the most important feature of RRAMs: by adjusting I_{set}, its R_{LRS} can be controlled in the range of $[V_{set}/I_{set,max}, V_{set}/I_{set,min}]$. This means that RRAMs can be sized just as transistors, creating large design space to be explored in circuits and architectures. Tunable R_{LRS} is an unique advantage of RRAM over other NVMs, such as MRAM [39] and PCRAM [40], strongly motivating the studies in the rest of this thesis.

2.1.2 Capacitive Modeling

Resistive property is the major interest of RRAMs to be exploited in applications, meanwhile their capacitive parasitics are often regarded as a negative aspect. For instance, when placed in datapath, capacitances of RRAMs cause additional propagation delay in critical paths, negatively impacting circuit speed. As a result, it is necessary and important to consider the capacitive part when designing circuits with RRAMs. The capacitive effect of a RRAM is induced by the MIM structure, which is naturally a parallel-plate capacitor. Considering a parallel-plate model, capacitance of a pristine RRAM in Fig. 2.1(a) is

$$C_P = \epsilon_{ox}\epsilon_0 \frac{a \cdot b}{d}, \tag{2.1}$$

where ϵ_{ox} is the dielectric constant of the oxide material, ϵ_0 is the electric constant ($\approx 8.854 \times 10^{-12} F \cdot m^{-1}$), $a \cdot b$ represents the contact area between the metal oxide and the electrodes, and d denotes the height of the metal oxide.

The capacitance of a RRAM is influenced by CF, whose dielectric constant ϵ_{CF} is smaller than oxide. Consider a RRAM in Fig. 2.1(b) and (c) and assume that CF can be modeled as a cylinder with an average radius r_{CF}. For a RRAM in LRS, the filaments create a conductive path between TE and BE, resulting in the capacitive effect to be negligible ($C_P \approx 0$). For a RRAM in HRS, the capacitance of a RRAM in HRS is approximately

$$C_P = \epsilon_{ox}\epsilon_0 \left(\frac{a \cdot b - \pi {r_{CF}}^2}{d} + \frac{\pi {r_{CF}}^2}{d-h}\right). \tag{2.2}$$

In practice, (2.1) can be accurate enough because that the size of CF r_{CF} is often much smaller than metal oxide [29, 51], which will be explained in subsection 2.1.5. In this thesis, we estimate the capacitance of RRAMs with (2.1).

2.1.3 Trade-off between R_{LRS} and C_P

As explained in Section 2.1.1, R_{LRS} is determined by the size of *Conductive Filament* (CF):

$$R_{LRS} = \rho_{CF} \frac{d}{\pi r_{CF}^2}, \tag{2.3}$$

where ρ_{CF} denotes the electrical resistivity of CF, d represents the height of CF, and r_{CF} is the radius of CF.

For simplicity in analysis, we assume the shape of CF to be a cylinder, and the area of RRAM device $a \cdot b$ to be fixed under a given technology node, which is limited by the size of contacts (See Section 2.1.4). Combining equation 2.3 and equation 2.2, we see a trade-off between R_{LRS} and C_P. When a smaller R_{LRS} is achieved by decreasing d, a larger C_P is seen in HRS. To be more intuitive, we compute the product of R_{LRS} and C_P:

$$R_{LRS} \cdot C_P = \epsilon_{ox} \epsilon_0 \rho_{CF} \left(\frac{a \cdot b - \pi {r_{CF}}^2}{\pi r_{CF}^2} + \frac{1}{1 - h/d} \right) \tag{2.4}$$

When h/d is fixed, $R_{LRS} \cdot C_P$ can be independent from d. And, increasing the size of CFs can efficiently reduce $R_{LRS} \cdot C_P$. Actually, the product of R_{LRS} and C_P can be regarded as the RC delay of a RRAM device, which significantly impacts the performance of RRAM-based circuits (See Chapter 3). The smaller the $R_{LRS} \cdot C_P$, the better performance of RRAM-based circuits can be achieved.

2.1.4 Co-Integration with CMOS Technology and Scaling Trends

Compatible with *Back-End-Of-Line* (BEOL) technology, RRAMs can be efficiently fabricated using two alternative integrations:

1. Fabricating a memory in the contact of an access transistor [52, 53], as illustrated in Fig. 2.4(a); In this case, the BEs of RRAMs share the same material with source/drain of transistors, enabling RRAMs and transistors to be fabricated with one lithography step. The BE of $RRAM_0$ is built with n-doped S_i, which is also the source/drain of transistors. Indeed, the BEs are natively connected to the source/drain of transistors, bringing conveniences in RRAM-based circuit designs. But in this fabricating choice, RRAMs have to occupy silicon area as transistors, limiting their interests in area-hungry designs.
2. Fabricating a memory on the top of or between metal layers in the process of a via [54], as depicted in Fig. 2.4(b). Compared to native integration with transistors, this methodology allows RRAMs to be 3-D stacked anywhere on the top of transistors, no longer occupying silicon area. This can bring significant reduction on footprints but carry a cost in parasitic effects and fabrication. RRAMs are connected to transistors through contacts, metals and VIAs, causing parasitic resistances and capacitances in

interconnection. To minimize the parasitics, RRAMs should be located close to transistors, i.e., between metal layer $MET1$ and $MET2$. Due to different materials, RRAMs require additional lithography masks than conventional VIAs, increasing fabrication cost. Actually, this fabrication methodology is more commonly adapted than the native integration, because of more flexibility in choosing materials and strong interests in area reduction.

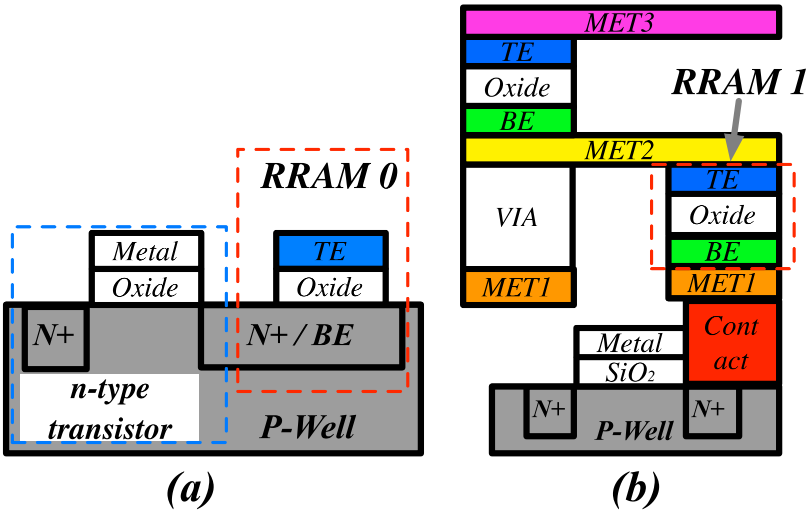


Figure 2.4 – Alternative integrations: (a) Natively combine with source/drain or gate of transistors; (b) Locate between metal layers.

For both integration methods, the size of RRAMs is supposed to be consistent or comparable with contacts and VIAs, in order to simplify Back-End process. Thanks to filamentary conducting mechanism, RRAM can be fabricated with an theoretical cell area as small as $4F^2$, where F is the feature size [55], following the scaling trends of CMOS technology. In principle, device size of RRAMs can potentially reach sub-10nm dimensions as Lee *et al.* reported successful resistive switching events in a CF whose size is $< 10nm$ [29, 51]. In recent years, plenty of research works have demonstrated that device size of RRAMs is scalable between $10nm$ and $180nm$ [52, 50, 56, 57, 49, 55, 58, 59, 60, 61, 62, 63, 64, 29]. Particularly, many efforts have been spent on cooperating with advanced CMOS technology, such as 16nm, 28nm and 40nm, in a good yield rate [52, 58, 60, 61, 64, 62, 63]. These pioneering works are meaningful to RRAM-based FPGA researches as regularity of FPGA architectures is advantageous when adapting to new technology.

Similar to transistors, RRAMs can benefit from the scaling down on their device size, proved by Fig. 2.5. The R_{HRS} is inverse proportional to device area, roughly following the Ohm's law. A small device area can increase R_{HRS} and thus effectively suppress the leakage power of RRAM-based circuits. As shown in (2.1), the parasitic capacitance is linear with the device area. The parasitic capacitance C_P can also be reduced by the scaling down, potentially contributing to delay and dynamic power improvements. Different from R_{HRS} and C_P, R_{LRS} is mainly determined by filamentary conducting current [1]. Since size of filaments is less sensitive to the feature size, R_{LRS} only has a limited dependency on device scaling. The trend on R_{LRS} is superior than transistors, whose equivalent resistance actually increases when scaling down.

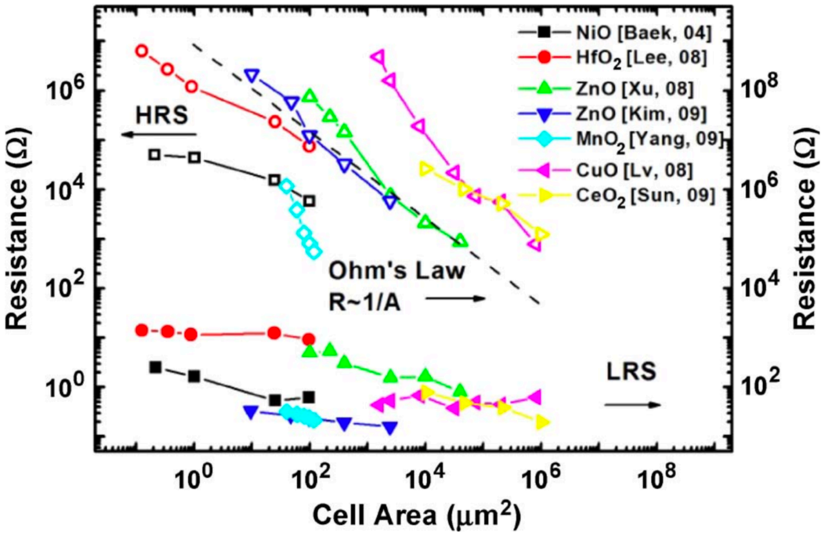


Figure 2.5 – Impact of cell area on R_{HRS} and R_{LRS} [Courtesy by [1]].

2.1.5 Process Variations

Filamentary conducting mechanism brings good scalability but also variation problems. It is believed that the formation and rupture of CFs is stochastic [65]. Variations can impact key parameters negatively. For instance, fluctuations on V_{set} and V_{reset} may cause R_{LRS} and R_{HRS} to be larger than expected, which directly influence performance metrics. There are two sources of the variations:

(1) device-to-device: Similar to transistors, RRAMs on the same die/wafer suffer spatial differences in device geometry.

(2) cycle-to-cycle: A RRAM may exhibit various resistances during each switching. This is an intrinsic property of RRAM devices, coming from the stochastic nature of filamentary conducting. Consequently, the size of CFs is different from cycle to cycle, resulting in R_{LRS} and R_{HRS} variations.

From a device perspective, the variation can be confined mainly by (a) carefully selecting the materials of TE, BE and oxide [66, 67, 68, 69]; and (b) using multi-layers of metal oxides

[70]. Lee *et al.* reported that reducing device size is also an effective way [71]. Through device engineering, both device-to-device and cycle-to-cycle variations are reported to be well controlled between 10-20% [72, 73, 74]. Variation problems can also be addressed by programming methods. To be more robust in cycle-to-cycle variations, programming RRAMs can borrow the program-verify strategy for Flash memory [75, 76, 77].

In this thesis, we will focus on examining the robustness of RRAM-based circuits to process variations.

2.1.6 Material Engineering for Application Requirements

The parameters of a RRAM, such as the R_{LRS}, R_{HRS}, V_{set}, V_{reset} and endurance, are highly dependent on the chosen metal oxide materials, the stack architecture and the fabrication techniques. Therefore, the device properties of RRAMs can be tuned to meet different application needs. For instance, RRAMs for memory applications and FPGAs require different device properties. Table 2.1 lists a few bipolar RRAMs fabricated with different metal oxide materials.

Table 2.1 – Bipolar RRAMs with different metal oxide materials

Metal Oxide Material	C_u/Z_rO_2 [57]	AlO_x [49]	H_fO_x [58]	T_aO_x [56]
R_{LRS} (Ω)	~ 200	$\sim 100k$	$\sim 10k$	~ 100
R_{HRS} (Ω)	$\sim 100M$	$\sim 100M$	$\sim 60k$	$\sim 1k$
Endurance	N/A	10^5	5×10^7	10^9
Retention	10 year @25$^{\circ}C$	10 year @125$^{\circ}C$	30h @250$^{\circ}C$	10 year @85$^{\circ}C$
Peak Current	$\sim 5mA$	$\sim 50nA$	$\sim 50\mu A$	$\sim 170\mu A$
Peak Voltage	$< 2.5V$	$< 2V$	$< 1.5V$	$< 2V$
Speed	$\sim 100ns$	N/A	$\sim 10ns$	$\sim 10ns$
Cell Area (μm^2)	~ 9	~ 1	$1e^{-4}$ (10nm)	~ 0.25

In memory applications, RRAMs typically requires (a) compact cell size (F^2) for high density, (b) fast speed in programming ($1 - 10\mu s$) for high-speed memory access, and (c) excellent endurance ($> 10^9$) for frequent writing operations. There are no specific requirements for R_{LRS} and R_{HRS}/R_{LRS} ratio as long as the states '0' and '1' can be properly differentiated. However, the FPGA architecture that is described in the thesis requires relaxed RRAM parameters, with typically (a) medium endurance ($\sim 10^6$) and long retention period(> 10 years@ 85°), (b) low R_{LRS}($\sim 1 - 4k\Omega$) along with high R_{HRS}/R_{LRS} ratio ($> 10^3$), (c) low programming current ($< 800\mu A$) and (d) medium density ($>\sim 4F^2$). In addition, FPGAs are configured to customized circuit designs but are not programmed frequently. Practically, FPGAs see only limited write cycles ($\sim 10^4$) [78]. Hence, the RRAMs in an FPGA application do not require excellent endurance. Furthermore, the performances of the implemented circuit designs are not determined by the programming cost of the memory. Therefore, fast programming speed is not a necessity for the RRAMs in the presented context. Instead, a long retention period is mandatory because the programmed FPGAs should hold its configurations unless there is a request to re-program. We will discuss in the chapter that the RRAMs will have two different functionalities in the proposed architectures. First, RRAMs will be employed in the data path of the routing multiplexer (as a replacement of the transmission-gates). Their R_{LRS} should be low enough to propagate signals in high speed while R_{HRS}/R_{LRS} ratio should be large to limit the perturbations between the inputs and to avoid parasitic leakage currents. Second, RRAMs will be used *flip-flops* (FFs), and serve as standalone memories only. their R_{HRS} and R_{HRS}/R_{LRS} ratio could be more relaxed as in memory applications. Last but not the least, since FPGA area is typically dominated by the transistors, and programming transistors in particular, the cell size could be relaxed to medium density.

In this thesis, we consider the integration method in Fig. 2.4(b), because that it can significantly narrow the area gap between FPGAs and ASICs. We will consider a RRAM device with the following parameters: $R_{LRS} = 1.6k\Omega, R_{HRS} = 27M\Omega$, as per [50][79]. However, in electrical simulations, we may use degraded parameters to emphasize on certain aspects of the study.

For more details about RRAM technology, we refer the interested reader to [1].

2.2 Conventional FPGA Architectures

In this section, we will first review classical FPGA architectures, whose principles are still used in modern FPGAs. Then, we will introduce critical architectural enhancements and circuit design techniques routinely used in commercial FPGA products. Last but not least, we will analyze the use of memory technologies in modern FPGA architectures.

2.2.1 Classical Architectures

FPGA architectures typically follow a regular organization, which contains highly repeatable modules. A generic island-style FPGA architecture, shown in Fig. 2.6, consists of an array of *Configuration Logic Blocks* (CLBs), which are surrounded by a sea of routing resources [4].

Configurable Logic Block

CLBs are the key module to implement combinational and sequential logic. Fig. 2.7 illustrates a detailed CLB architecture, where a number of *Basic Logic Elements* (BLEs) are tightly connected by a local routing architecture. A BLE is the primitive module implementing logic functions, including a *Look-Up Table* (LUT), a *Flip-Flop* (FF) and a 2-input routing multiplexer. By configuring SRAMs properly, a K-input LUT can realize **any** K-input single-output logic function. The FFs enable BLEs to implement not only combinational but also sequential logic. By configuring the 2-input routing multiplexer, a BLE can operate in either combinational or sequential mode. The local routing architecture, which is actually a group of programmable

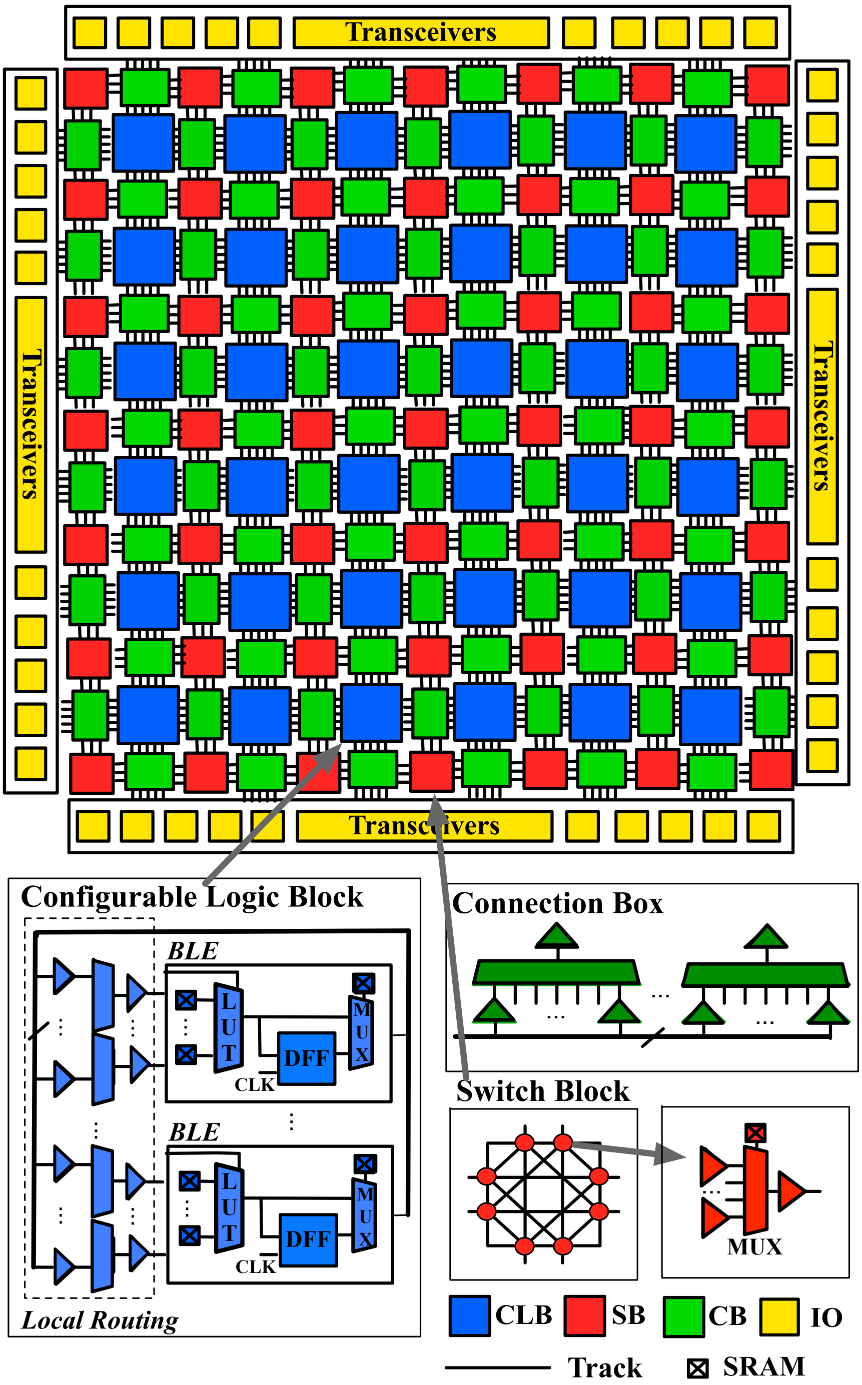


Figure 2.6 – Generic FPGA Architecture.

routing multiplexers, provides interconnections among CLB inputs, BLE inputs and outputs. As depicted in Fig. 2.7, each BLE input is driven by a local routing multiplexer, whose inputs come from all the CLB input pins and BLE outputs. The local routing architecture guarantees that BLEs can be fully connected to each other and also to every CLB input pin. Thanks to such full connectivity, a CLB can implement **any** large logic function by interconnecting LUTs and FFs.

The logic capacity of a CLB is defined as the amount of combinational and sequential logic that can be mapped to a CLB, which is mainly determined by the following parameters: (1) input size of LUTs K; (2) the number of BLEs in a CLB N; (3) the number of inputs of a CLB I. Indeed, large K, N and I improves CLB logic capability but also increases the area, delay and power of CLBs linearly. For instance, area, delay and power of local routing multiplexers are correlated to N and I, because their input size is $N+I$. Large CLBs can reduce the use of global routing architecture, but the saving may be null due to the increase in CLB area. Therefore, there exists a best trade-off between CLB logic capacity and its performance metrics. In modern FPGAs, the best CLB architecture is typically featured by $K=6$, $N=10$ and $I=K(N+1)/2=33$.

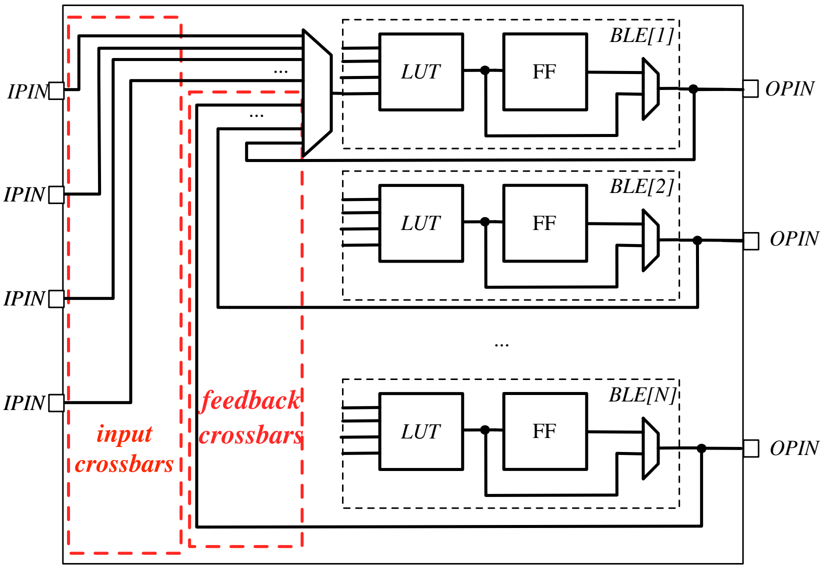


Figure 2.7 – Detailed CLB Architecture.

Global Routing Architecture

The global routing resources outside CLBs consist of two types of blocks, the *Connection Blocks* (CBs) and the *Switch Blocks* (SBs). Both CBs and SBs consist of programmable routing multiplexers but have different interconnecting topologies. CBs connect routing tracks to CLB inputs and outputs, while SBs interconnect routing tracks. Differently from local routing architecture, global routing multiplexers usually have sparse connectivity. In other words, a routing multiplexer can only connect to a subset of the routing tracks. Using sparse connections leads to better trade-off between routing area and routability. Indeed, full connectivity ensures perfect routability but results in large routing multiplexers. In global routing architecture, the number of point-to-point connections is linear to the FPGA array size, which is much larger than local routing architecture. It will cause large routing area and lead to difficulties in wiring if all the routing multiplexers are fully-connected. C. Clos has proved that multi-level sparse crossbars can also achieve perfect routability as fully-connected solutions, while the routing area can be significantly reduced [80]. Therefore, in global routing architectures, point-to-point connections are realized through multiple sparse CBs and SBs.

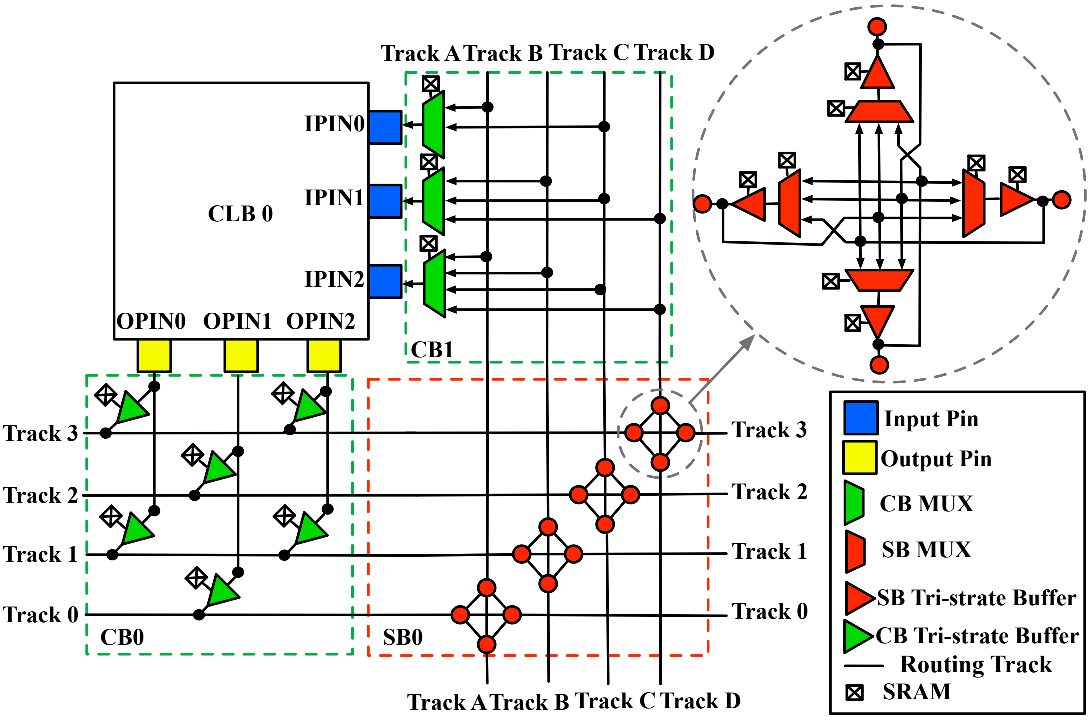


Figure 2.8 – Bi-directional global routing architecture.

The following parameters are widely used to quantify the sparse connectivities in global routing architecture: As routing tracks are grouped in channels, the number of routing tracks per channel is called channel width, denoted by W. In the context of CBs, the fraction of routing tracks that can be connected to a CLB input pin is defined as $F_{c,in}$. The fraction of routing tracks that can be connected by a CLB output pin is defined as $F_{c,out}$. In a SB, the number of routing tracks to which each incoming routing track can connect is defined as F_s. Fig. 2.8 provides an illustrative example of global routing architecture, where CLB $CLB0$ is surrounded by a SB, $SB0$, and two CBs, $CB0$ and $CB1$, with a channel width of 4. Connectivity parameters $F_{c,in}$ of input pins $IPIN0$, $IPIN1$ and $IPIN2$ are $2/4 = 0.5$, $3/4 = 0.75$ and $4/4 = 1$ respectively. All the output pins $OPIN0$, $OPIN1$ and $OPIN2$ share the same connectivity parameters $F_{c,out} = 2/4 = 0.5$. Each routing track can connect to three other tracks, leading to $F_s = 3$ in $SB0$. Note that each routing track is bi-directional. Take the example of $Track3$ in Fig. 2.8, a signal can propagate from left side to right side and vice versa. To realize a bi-directional SB, two routing multiplexers with tri-state buffers are required for each routing track. Different from routing tracks, connections for input and output pins of CLBs have to be uni-directional. As a result, tri-state buffers are used for output pins to guarantee that signals can only flow from output pins to routing tracks, while routing multiplexers are used for input pins to guarantee that signals can only pass from routing tracks to input pins. For a bi-directional routing architecture, routing algorithms have to not only determine directionality of each routing track but also show respect to the uni-directionality of tri-state buffers. These additional constraints complicate the routing algorithms. Normally, a routing path starts from a CLB input, connects to a routing track through a CB, then passes through a number of SBs, to finally reach a CLB output through another CB. However, when the CLBs are far from each other, the routing path may contain many SBs, causing large delay. To overcome this limitation, routing tracks are allowed to span multiple CLBs without passing through any SB. The number of CLBs spanned by a routing tracks is defined as the length of routing track L. Fig. 2.9(a) and (b) describe how to realize a long connection with either cascaded $L = 1$ routing tracks or a single $L = 2$ routing track. The $L = 2$ solution removes one SB on the routing path, potentially leading to a performance improvement. Indeed, while $L = 2$ architecture is less routable than $L = 1$, its circuit speed can be 24% faster [4]. The routability of $L \geq 2$ architecture can be fully compensated by adding more routing tracks and distributing equally their starting points over the length of the track. Take the example of Fig. 2.9(b), $CLB[1]$ cannot be routed to $CLB[2]$ through $Track0$ which starts from $CLB[0]$, but it can always be solved by another routing track $Track1$ which starts from $CLB[1]$. In practice, FPGAs include routing tracks with various L, in order to achieve best performance. For instance, Xilinx XC4000X series FPGAs contain 25% $L = 1$ tracks, 12.5% $L = 1$ tracks, 37.5% $L = 1$ tracks and 25% "one-quarter longs", whose length is one-fourth of the chip [81].

$F_{c,in}$, $F_{c,out}$, F_s and L strongly influence not only routability but also area and performance of FPGAs. V. Betz reported that when only one type of routing track is allowed, $F_{c,in} = 0.25 \cdot W, F_{c,out} = 0.5 \cdot W, F_s = 3, L = 4$ contributes to the best trade-off between area and delay [4].

Most frequently-used FPGA architecture parameters are summarized in Table 2.2. We refer interested readers to [4] for more details about classical FPGA architectures.

Table 2.2 – FPGA Architecture Parameters

Parameter	Range	Description
K	$[1, +\infty]$	Input size of a LUT.
N	$[1, +\infty]$	Number of BLEs in a Configuration Logic Block.
I	$[1, +\infty]$	Number of inputs of a CLB.
W	$[1, +\infty]$	The number of routing tracks contained in a channel.
$F_{c,in}$	$[0, 1]$	The fraction of routing tracks to which each CLB input pin connects.
$F_{c,out}$	$[0, 1]$	The fraction of routing tracks to which each CLB output pin connects.
F_s	$[0, 4W]$	The number of routing tracks to which each incoming routing track can connect in a SB.
L	$[1, +\infty]$	The length of a routing track in term of the number of CLBs spanned by the track.

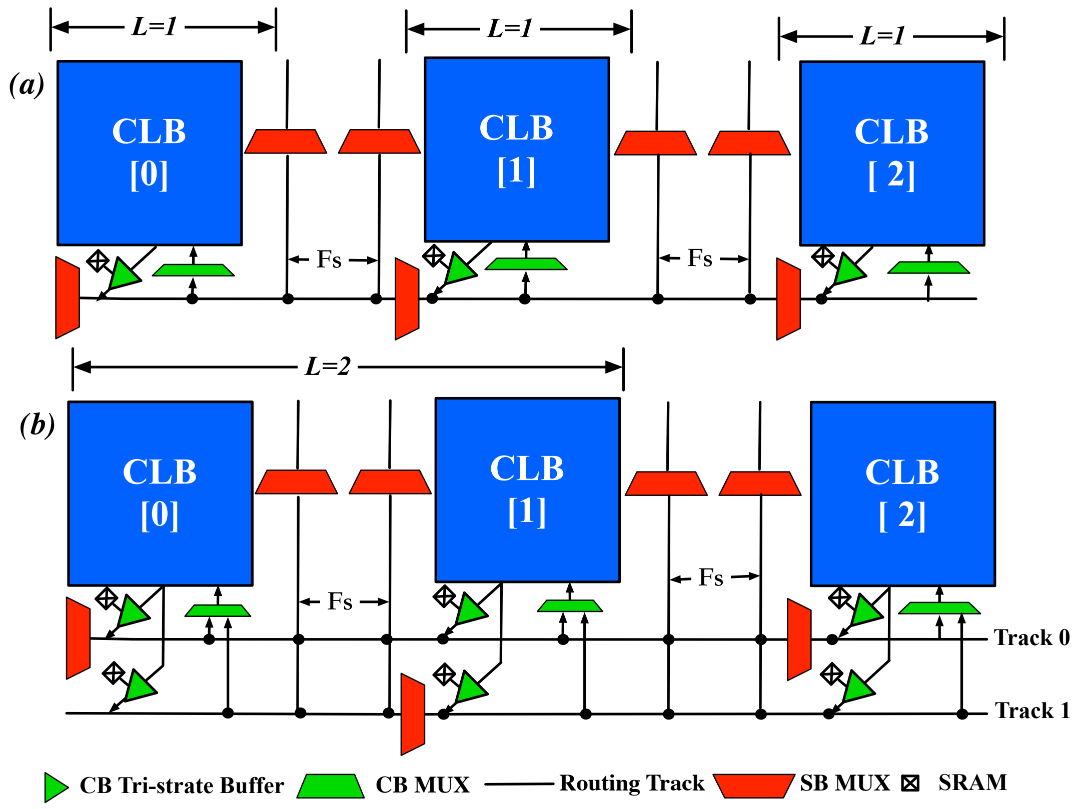


Figure 2.9 – Bi-directional global routing architecture featured by (a) $L = 1$; (b) $L = 2$.

2.2.2 Architectural Enhancements

Since **any** large logic function can be represented by interconnected small partitions, FPGAs can implement **any** circuit by appropriately programming BLEs, global and local routing architectures. However, in reality, a FPGA has resource bounds, e.g., millions of BLEs in Xilinx products [82]. In practice, an extremely large circuit or system may be implemented by a network of FPGAs [83, 84]. The limited capacity of FPGA in implementing large scale computing can be overcome by boosting the capability of a single FPGA, which also narrows the gap between FPGAs and ASICs. Therefore, modern FPGAs have adapted several major architecture enhancements:

(1) **Tile-based heterogeneity**: Modern FPGAs [82, 85] typically employ a tile-based heterogeneous architecture [86], where the entire FPGA is organized in the unit of tile, highlighted blue in Fig. 2.10. A number of tiles or even columns of tiles) are replaced by hard *Intellectual Property* (IP) blocks, such as *Digital Signal Processing* (DSP) blocks and memory banks [82, 85]. The introduction of heterogeneous blocks (highlight brown in Fig. 2.10) aims at a better trade-off between programmability and efficiency. Programmable logics, i.e., LUTs, are considered as soft logic because of their flexibility in mapping logic func-

tions, while compact CMOS logics are considered to be hard logic since their functionality is fixed. Indeed, LUTs are flexible enough to realize **any** multi-input and single-output logic functions but their implementations require more area, delay and power than most compact CMOS logic. For instance, a 2-input NAND gates requires only 4 transistors in CMOS logic, but using a 2-input LUT consumes 28 transistors. This is one of the critical reasons which cause serious overheads of FPGA implementations. Therefore, to alleviate the limitations, modern FPGA architectures embed hard logic to implement most frequently used logic functions. For instance, commercial FPGAs, i.e., Xilinx Virtex Series [82] and Altera Stratix Series [87, 88, 89, 90, 85], feature DSP blocks, various sized memory banks and ARM Cortex CPUs [91], to accelerate arithmetic-intensive applications. Other hard IPs including shifted registers, embedded CPU cores, *Phase Lock Loops* (PLLs) and high-speed transceivers. We refer the interested readers to [87] for more information. By following a tile-based organization, heterogeneous FPGAs can achieve better granularity at layout-level. Commercial FPGAs [86] are manually designed because that their highly repeatable nature are friendly to hand optimization with medium layout efforts[86]. On average, manual FPGA layouts outperform 2× in area and performance than automatically generated layouts [18, 92]. As illustrated in Fig. 2.11, each tile includes a CLB, two CBs and one SB, while routing tracks are interconnected only through SBs. This allows engineers to focus on optimizing the layout of a tile and spend less time on placing and routing tiles.

(2) **Hard carry chain**: In modern FPGAs, heterogeneity is not only applied at the tile-level but also in CLBs. In arithmetic applications, the critical path is highly likely a mixture of the carry part of adders and other regular logic functions. To achieve better delay efficiency, adders should be placed closely to LUTs as much as possible. For this purpose, hard adder chains are embedded in CLBs across all the BLEs, as depicted in Fig. 2.11. The carry parts of the hard adders are connected across BLEs through pins Cin and C_out in Fig. 2.11, while the sum parts are connected to regular BLE outputs. Note that the adder chains are also hard wired in sequence through CLB pins Cin and C_out across all the CLBs in a column. As a result, the hard adder chains are the fastest implementation in FPGAs for adder functions. J. Luu *et al.* reported that embedding adder chains and heterogeneous blocks can improve performance of FPGAs by 15% on average [93]. Further researches [94, 95] focus on exploiting the hard adder chains to improve up to 15% area and 25% delay of general circuit implementations, not limited to arithmetic-intensive ones. We refer interested readers to [93, 94, 95, 96] for more information.

(3) **Fracturable LUT**: Area of a LUT is exponential to its number of inputs. When the mapped function does not exploit all the inputs of a K-input LUT, at least 50% of the LUT is not involved in computing. Consequently, the utilization rate of LUTs in classical FPGAs is often low since they contain one type of LUTs with fixed input size. In modern FPGAs, a K-input LUT can be fractured to two $(K-1)$-input LUTs, boosting its capability in mapping logic functions [97]. Compared to the classical design in Fig. 2.6, the 6-input fracturable LUT in Fig. 2.11 has an additional output, and thus can accommodate two logic functions with up to five common inputs. For instance, the 5-input $LUT[0]$ can accommodate a

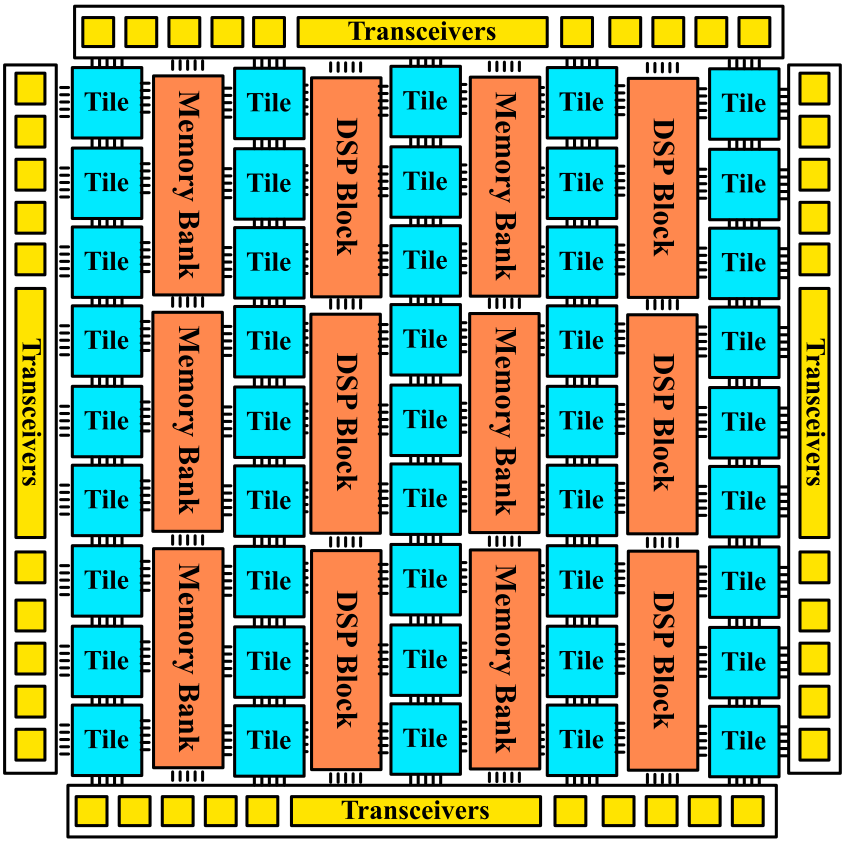


Figure 2.10 – Tile-based FPGA Architecture.

4-input logic function $f_0(x_0, x_1, x_2, x_3, x_4)$ using $in0, in1, in2, in3$ and $in4$. The 5-input $LUT[1]$ can still implement another 4-input logic function $f_1(x_3, x_4, x_5, x_6)$ by sharing $in3$ and $in4$ with 5-input LUT[0]. Alternatively, the 6-input fracturable LUT can implement two small functions without common inputs, whose total number of input is smaller or equal to five. For example, logic functions $f_2(x_0, x_1)$ and $f_3(x_2, x_3, x_4)$ can be mapped to 5-input LUT[0] and 5-input LUT[1] respectively. Such capability is beyond a classical K-input 1-output LUT, significantly improving the capacity of LUTs.

(4) **Uni-directional Global Routing Architecture and Single-Driver Wires**: In the recent decade, we have seen a trend of uni-directional global routing architecture becoming popular in commercial FPGAs [98]. The interests comes from that uni-directional routing architecture can save 25% area and improve delay by 9% as compared to bi-directional classics [98]. Fig. 2.12 depicts an uni-directional global routing architecture featured by the same parameters ($F_s = 3$, $W = 4$, $F_{c,in} = \{0.5, 0.75, 1\}$, $F_{c,out} = 0.5$) as bi-directional example in Fig. 2.8. Just as its name implies, each routing track is directional, as illustrated with arrowed lines in Fig. 2.12. It seems that uni-directional architecture is less flexible than bi-directional architecture because that channel width W have to be doubled

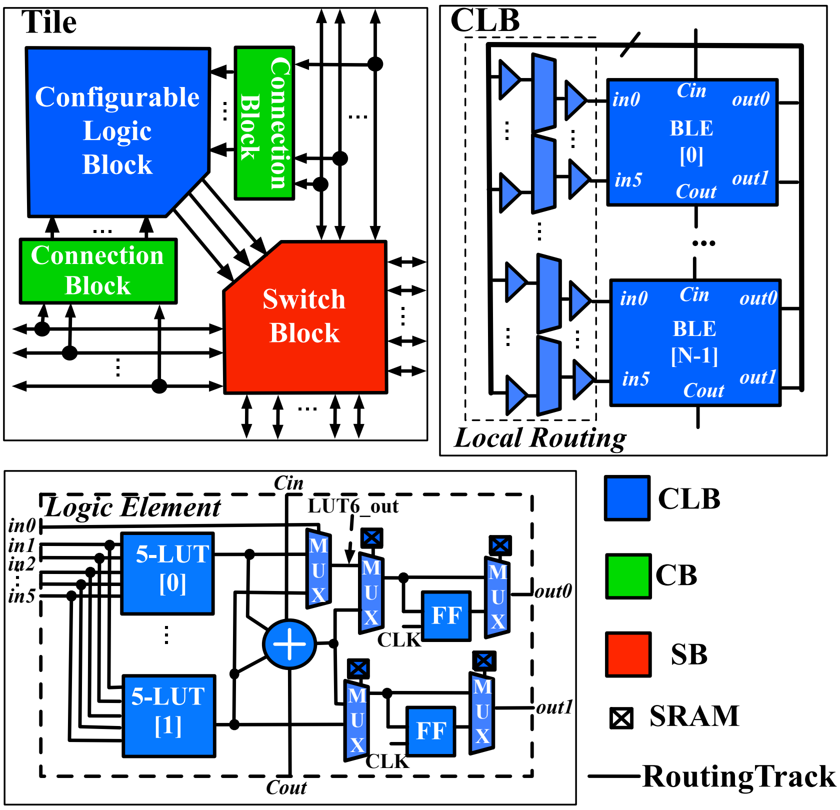


Figure 2.11 – Tile and enhanced CLB architecture.

to reach the same routability. But in fact each routing track will always have a definite directionality in a mapped FPGA. Routing tracks, which are actually metal wires, are on the top of transistors. Doubled channel width has very limited impact of FPGA area. Despite issues of channel width, uni-directional routing architecture has several overwhelming advantages over bi-directional:

(a) Tri-state buffers in SBs can be eliminated, reducing the number of configuration bits and dedicated transistor area. The number of multiplexers is the same for each crosspoint in SBs (See dashed circles in Fig. 2.8 and Fig. 2.12).

(b) CBs for CLB output pins can be merged into SBs. Since each routing track has a specific direction, connections between a routing track and a CLB output pin can be realized by multiplexers, instead of tri-state buffers. As represented with yellow rectangles in Fig. 2.12, CLB output pins are directly wired to an input of SB multiplexers. As such, routing delay from a CLB output pin to a routing track can be reduced, because that only one level of crossbars is needed, rather than the two levels in bi-directional architecture.

(c) The wiring capacitance can be reduced by 37% [98], thanks to single-driver wiring: each uni-directional routing track is driven by only one routing multiplexer. The removal of tri-state buffers contributes to less wire loads of routing tracks. Compared to Fig. 2.9(b), the $L = 2$ uni-directional routing track in Fig. 2.13 only need to drive downstream routing multiplexers.

It is still possible to increase connectivity parameter $F_{c,out}$ in uni-directional architecture. For instance, $OPIN0$ can also drive $Track3$ and $Track2$ by connecting an additional input of SB multiplexers.

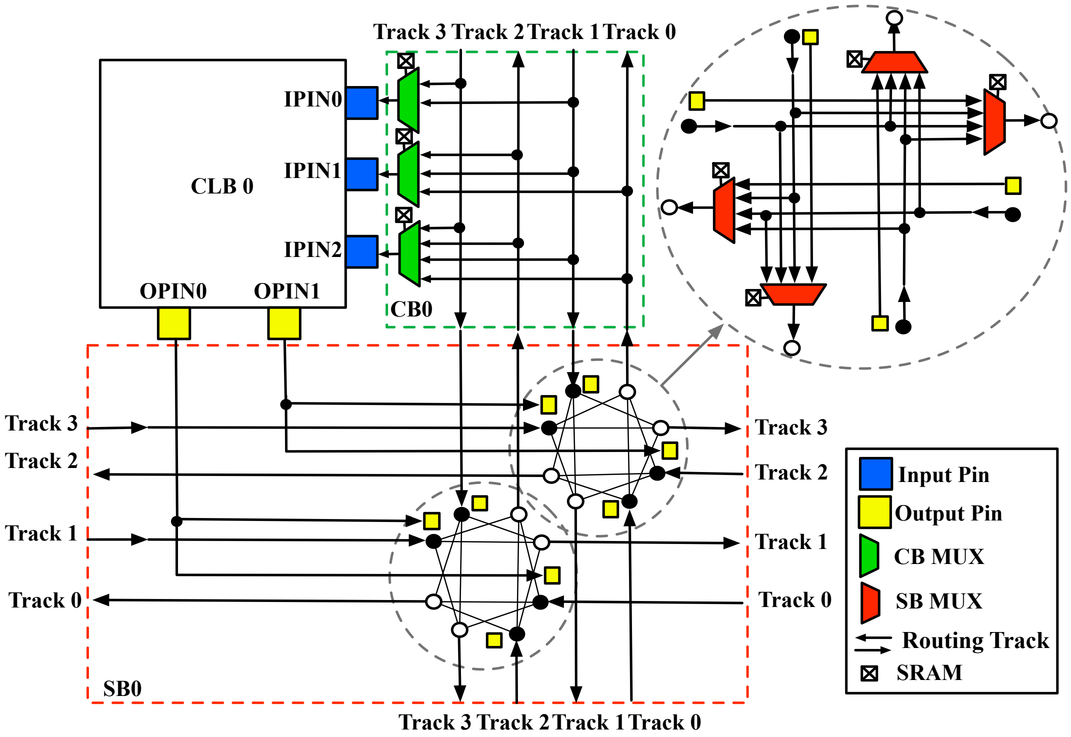


Figure 2.12 – Uni-directional global routing architecture.

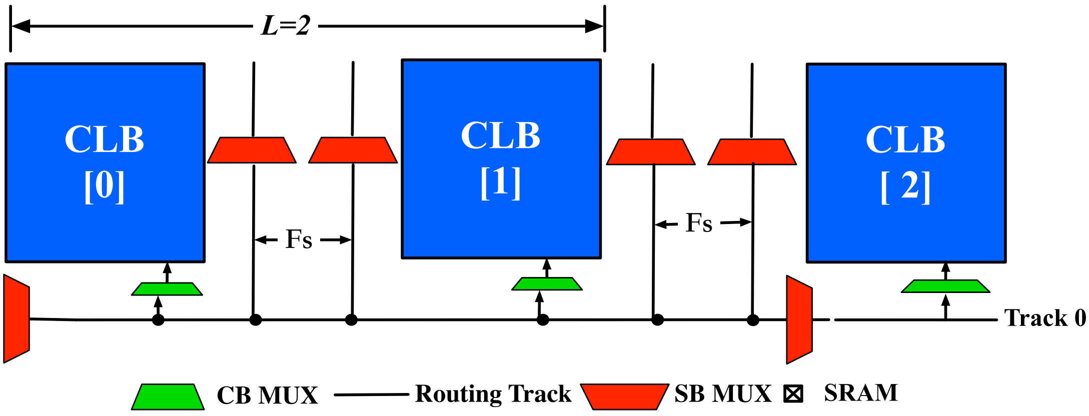


Figure 2.13 – A uni-directional routing track featured by $L = 2$.

Architectural enhancements include but not limited to those introduced here. This section focus on most widely used enhancements in modern FPGAs, which are considered in the architecture-level evaluations throughout this thesis. Other architectural enhancements, such as sparse local routing architecture, time-borrowing FFs and look-ahead/carry-select adder chains, are designed for specific application purposes. We recommend interested readers to see [82, 88, 89, 90, 99, 100, 101, 102, 103] for more details.

2.2.3 Circuit Designs in FPGAs

Actually, the entire FPGA architecture is an assembly of three main circuit primitives: *Look-Up Table* (LUT), *Flip-flop* (FF) and routing multiplexer. Therefore, the design topology for these circuits profoundly impacts the area and performance of FPGAs. This part focuses on introducing current best implementations of LUT, FF and routing multiplexer.

Routing Multiplexer

Routing multiplexers are intensively deployed in both local and global routing architectures, as shown in Fig. 2.6. The functionality of routing multiplexers is to select among several possible input signals. As symbolized in Fig. 2.14(a), a N-input routing multiplexer can propagate any of the N inputs to the output according to the configuration stored in its memory bits. Fig. 2.14(b) shows a straightforward implementation of a N-input routing multiplexer, where each transmission gate can be configured to propagate/block an input independently. The one-level structure requires the least number of transmission-gate, but the number of memory bits required and its critical path delay is linear to the input size N. Consequently, its parasitic capacitance and memory footprint grows linearly to input size N. Therefore, when N is large, one-level multiplexer is area-consuming and low-performance.

Remember that large routing multiplexers are intensively used in local routing architecture. Two-level structure is proposed to achieve better area-delay trade-off in large multiplexers [2]. As illustrated in Fig. 2.15(a), a two-level structure is built by cascading one-level structures. A N-input two-level structure consists of $[\sqrt{N}] + 1$ one-level structures, each of which has $[\sqrt{N}]$ inputs. Note that all the one-level structures can share $[\sqrt{N}]$ memory bits. In a two-level structure, the number of memory bits and critical path delay is quadratic to input size N. Therefore, two-level structure can be area-efficient and high-performance when N becomes large.

It is possible to generalize the topology to multi-level structures, such as three-level, *etc.* A tree-like structure shown in Fig. 2.15(b) is a special case of multi-level structure where each one-level structure has only two inputs. A 2-input one-level multiplexer only requires one memory bit because the two transmission gates are always in opposite states. As a result, a tree-like multiplexer is most compact in terms of the number of memory bits, which is logarithmic to input size. But, due to their large number of stages, tree-like multiplexers

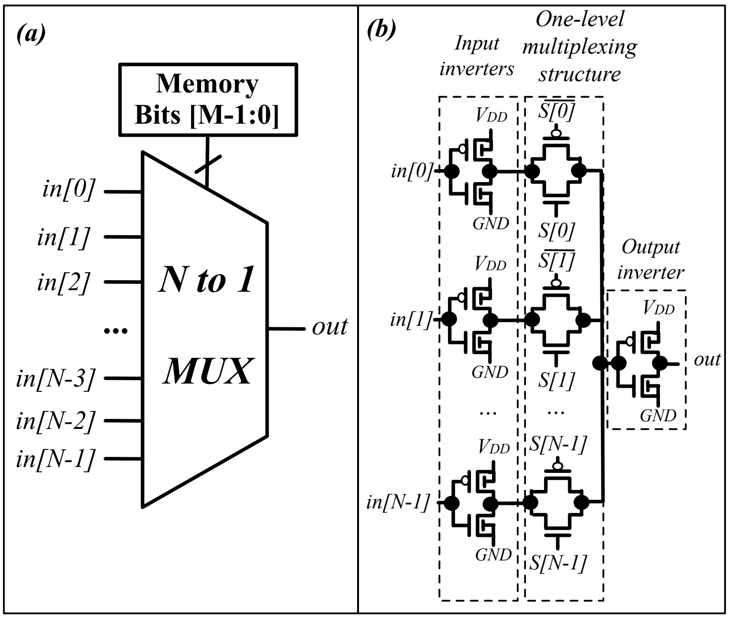


Figure 2.14 – (a) Symbol of a N-input routing multiplexer; (b) One-level implementation [2, 3].

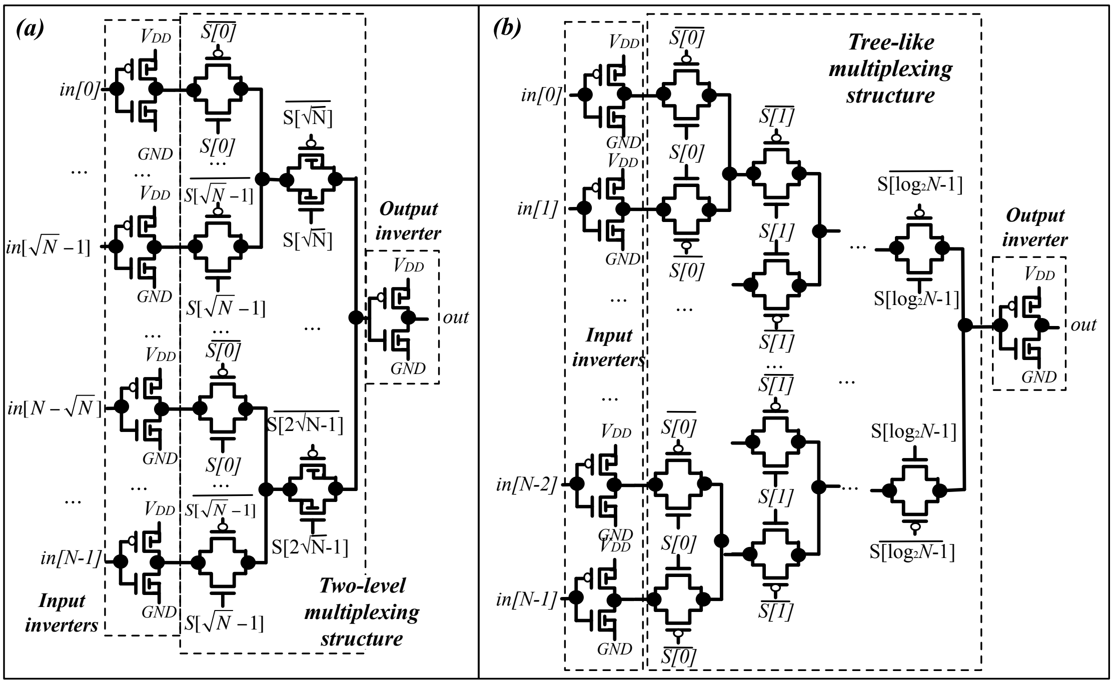


Figure 2.15 – Alternative routing multiplexer design topologies: (a) two-level; (b) tree-like [2, 3].

perform worse in area, delay and power than others.

Table 2.3 – Analytical comparison between CMOS one-level, two-level and tree-like multiplexers

Multiplexer	Transistor Area[1]	Critical Path Delay [2]	Switching Energy [3]
One-level	$N \cdot A_{mem} + N \cdot A_{tgate}$	$R_{tgate} \cdot N \cdot C_{tgate}$	$0.5 \cdot \alpha \cdot N \cdot C_{tgate} V_{DD}^2$
Two-level	$2[\sqrt{N}] \cdot A_{mem} + (N + \sqrt{N}) \cdot A_{tgate}$	$R_{tgate} \cdot (3[\sqrt{N}] + 1) \cdot C_{tgate}$	$0.5 \cdot \alpha \cdot 2[\sqrt{N}] \cdot C_{tgate} V_{DD}^2$
Tree-like	$log_2 N \cdot A_{mem} + (2N-2) \cdot A_{tgate}$	$R_{tgate} \cdot \frac{1}{2}([log_2 N]^2 + [log_2 N]) \cdot C_{tgate}$	$0.5 \cdot \alpha \cdot (3[log_2 N] - 1) \cdot C_{tgate} V_{DD}^2$

[1] Area of input and output inverters are not included here.

[2] Elmore delay model [104] is considered here. [3] Only the switching energy of multiplexer structures is considered here. α is the switching activity.

* A_{mem} is the transistor area of a memory bit. A_{tgate}, R_{tgate} and C_{tgate} are the area, equivalent resistance and source/drain capacitances of a transmission gate.

Table 2.3 summaries an analytical comparison among CMOS one-level, two-level and tree-like multiplexing structure. One-level structure is the best choice for small input size. When N grows, two-level structure becomes the best in terms of area-delay-power product as compared to one-level and other multi-level structures [2]. A tree-like structure is preferred when there is a tight constraint on the number of memory bits. Note that transmission gates in Fig. 2.14 and 2.15 can be replaced by pass-transistors or other pass-gate logics but the results in Table 2.3 and conclusions on best multiplexing structure remain true. In this thesis, we will consider transmission-gate-based routing multiplexer designs because they guarantee best area, delay and power results.

Look-Up Table

Large number of memory bits gives a K-input *Look-Up Table* (LUT) the capability to realize **any** K-input single-output logic function. Fig. 2.16(a) shows the most popular implementation of a K-input LUT, where a 2^K-input tree-like multiplexer is used in a different way than it is for the routing multiplexer in Fig. 2.14(a). Inputs of a LUT are wired to the control lines of the multiplexer while memory bits become the inputs of the multiplexer. By properly configuring the 2^K memory bits, a complete truth table can be built for any K-input single-output logic function. Depending on inputs, the multiplexer of a LUT can output any bit of a truth table. As such, LUTs can realize the functionality of **any** single-output logics.

Fig. 2.16(b) illustrates the transistor-level circuit design of a 2-input LUT based on transmission gates. Note that each input employs three inverters to drive the multiplexer, which can balance the delay from an input to every gates of transmission gates. Note that the area of LUTs is exponential to their input sizes:

$$A_{LUT} = 2^K \cdot A_{mem} + (2^{K+1} - 2) \cdot A_{tgate}, \tag{2.5}$$

where K denotes the number of inputs while A_{mem} and A_{tgate} is the transistor area of a memory bit and a transmission gate respectively. In other words, the logic capacity and area of a LUT is doubled when number of input is increased by one. For instance, a 6-input LUT is built with two 5-input LUTs and a 2-input multiplexer, as shown in Fig. 2.11. In addition, the delay of a LUT comes from the tree-like multiplexer and hence is approximately linear to the input size. Compared to standard CMOS logic, LUTs are expensive in terms of area and delay due to heavily using memory bits and tree-like multiplexers.

In this thesis, we will consider transmission-gate-based multiplexer designs for LUTs in the same perspective as routing multiplexers.

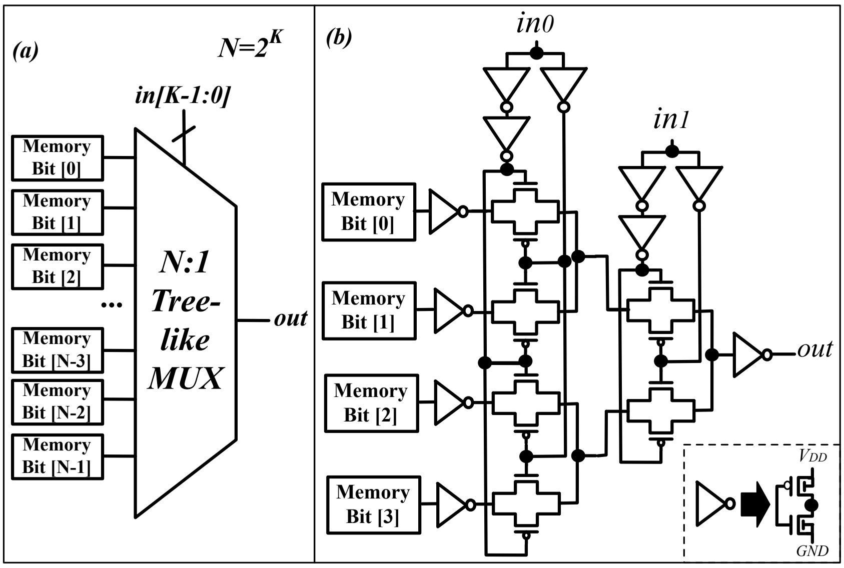


Figure 2.16 – Look-Up Table (LUT): (a) principle internal structure; (b) transistor-level design of a 2-input LUT [4].

Flip-Flop

Flip-Flops (FFs) are an essential hard logic in FPGAs to implement sequential logics. FPGAs typically employ D-type FFs in order to simplify timing constraints in sequential logics. The date stored in a D-type FF can be changed only at the rising/falling edge of the clock signal. Fig. 2.17 shows the transistor-level design of a master-slave D-type FF with asynchronous set and reset. Both the master and slave parts are CMOS latches based on cross-coupled inverter pair. Unless a strong write voltage is applied, the two inverters can hold a stable voltage, either '0' or '1'. When clock signal CLK is disabled (logic low '0'), the first stage (master) is transparent to the D input, but the second stage (slave) cannot change its storage. When the clock signal is enabled (logic high '1'), the first stage is read-only and its storage is transferred to the second stage (slave). As a result, output Q can only change state when the clock signal CLK makes a transition from logic low to logic high. The set and reset signal can force a overwrite to both master and slave parts regardless of input D and clock CLK. In this thesis, we consider the FF design in Fig. 2.17 in conventional FPGA architectures.

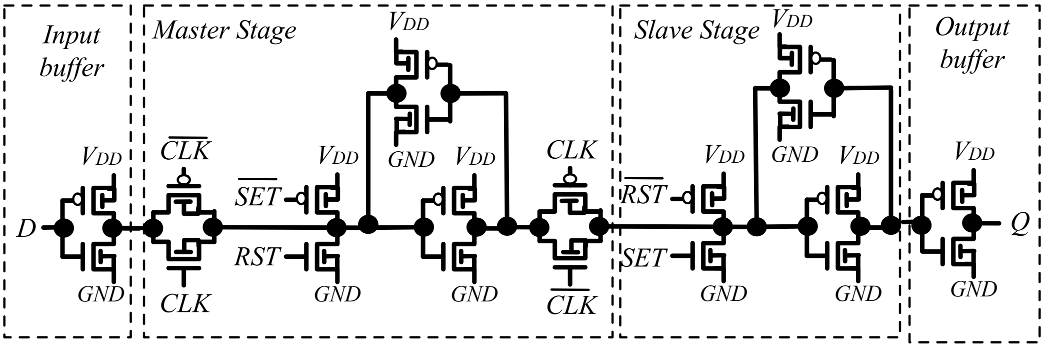


Figure 2.17 – Transistor-level design of a master-slave D-type *Flip-Flop* with asynchronous set and reset [4].

2.2.4 Memory Technologies for FPGAs

It is memory bits that enable FPGAs to be configurable to **any** circuits. As a crucial component in LUTs and routing multiplexers, memory cells can occupy 35% of FPGA area and consume 38% of total static power [105]. Their characteristics are key factors determining merits of FPGAs. Most popular memory technologies used in FPGA can be classified to two categories: (1) Volatile memories, i.e., *Static Random Access Memories* (SRAMs), and (2) Non-Volatile memories, i.e., Flash.

SRAM Technology

Most commercial FPGAs are based on SRAM technology because of its good reliability. Fig. 2.18(a) shows a six-transistor SRAM design, where a CMOS latch based on cross-coupled inverter pair is accessed by two n-type transistors. When control lines *Word Line* (WL) is enabled, a SRAM can be programmed by *Bit Line* (BL) voltages. When control lines WL is disabled, a SRAM can hold its storage whatever BL is. Note that six-transistor SRAM is preferred in FPGA because it is more resistant than five/four-transistor designs to state flipping due to crosstalk or charge sharing. The SRAMs in FPGAs are typically placed in an array and accessed by decoders, like a memory bank. As depicted in Fig. 2.18(b), SRAM cells belonging to the same row share a BL signal while each column is controlled by a WL signal. All the BL and WL signals are controlled by two decoders. Each SRAM cell can be individually programmed by manipulating the two decoders. Note that with efficient sharing BLs and WLs, n SRAMs only require $\sqrt{n}$ BLs and $\sqrt{n}$ WLs. Therefore, area of configuration circuits in FPGAs can be quadric to the number of SRAMs.

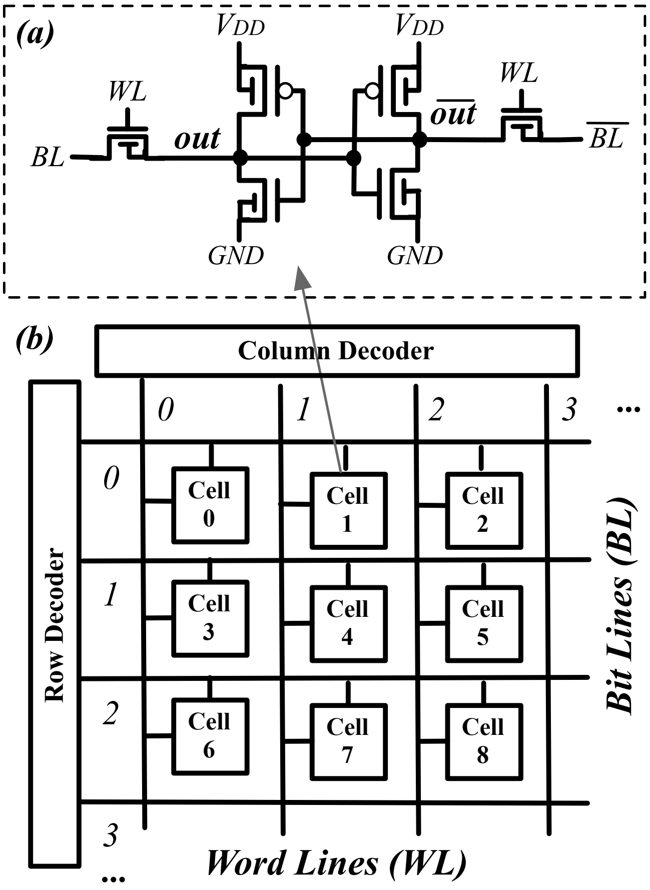


Figure 2.18 – (a) 6-Transistor SRAM design [4]; (b) Configuration circuits for SRAM arrays.

As SRAMs share the same storage mechanism as FFs, SRAM cell can also be embedded in FFs and accessed by a scan-chain. Fig. 2.19 shows the transistor-level design of a *Scan-Chain FF* (SCFF) and associated configuration circuit to program SRAMs. The configuration circuit is actually a cascade of SCFFs, which behaves as shift registers. When programming clock $prog\_clock$ is enabled, all the SRAMs are writable by the output of previous SCFF. As a result, during each programming clock cycle, the data is shifted from one SCFF to another which its output is connected to. It takes n clock cycles to programming the n SRAMs in the scan-chain. Memory bits are fed to a scan-chain in reversed sequence. In the first cycle, memory bit for the last SRAM is given to the head of chain. In the following cycles, the first memory bit is shifted from one SCFF to its next. After n cycles, the first input is propagated to the last SCFF and all the SCFFs receive their desired memory bits.

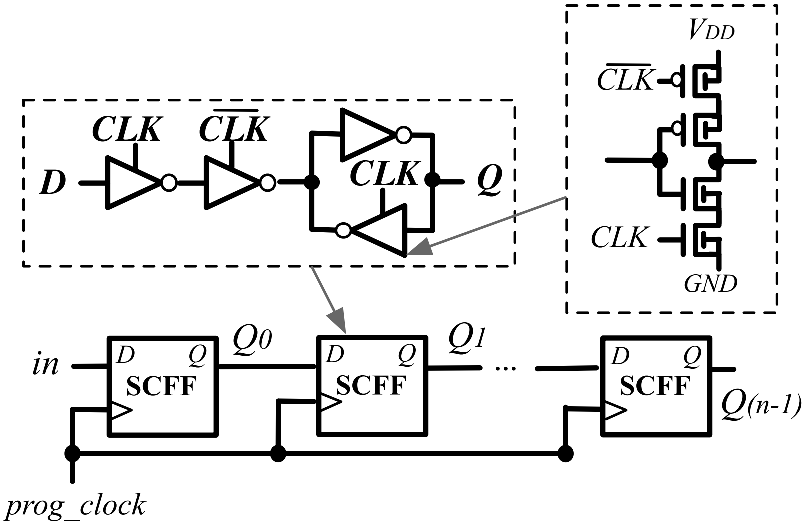


Figure 2.19 – Scan-Chain Flip-Flop (SCFF) design and associated configuration circuits [5, 6]

Flash Technology

As a well-developed non-volatile technology, Flash transistors have been exploited in FPGA architectures to achieve low power consumption. A Flash transistor can retain its configuration with zero leakage, which motivates commercial Flash-based FPGAs replace SRAMs and also pass-gate logics [7, 106].

Fig. 2.20(a) presents the cross-section of a embedded Flash transistor, where CMOS transistors are located in regular wells while the flash transistor is placed in a deep N-well. By applying a negative voltage difference across the floating gate (Fig. 2.20(b)), electrons are removed from the floating gate by Fowler-Nordheim tunneling mechanism [107], which turns the device on. A positive programming voltage inject electrons to the floating gate and turns off the device, as illustrated in Fig. 2.20(c). Because of the voltages required for programming and erasing, flash processes include special high-voltage transistors with thicker oxides, resulting in more complicate process than logic transistors.

Because Flash transistors can retain their on/off state without constant power supplies, they can be regarded as a combination of memory and transistor. By exploiting the features, two Flash transistors sharing a same control gate and a common floating gate (Fig. 2.21(b)) can realize the same functionality as a SRAM-controlled transmission gate in Fig. 2.21(a). The sense device (minimum-sized flash transistor) programs the floating gate voltage while the switch device (a larger flash transistor) turns on/off the data path. When the sense device undergoes a programming sequence illustrated in Fig. 2.20(b)(c), the floating gate of the switch device is programmed simultaneously. In other words, switching on/off the sense device also turns on/off the switch device, leading to propagating/blocking datapath signals.

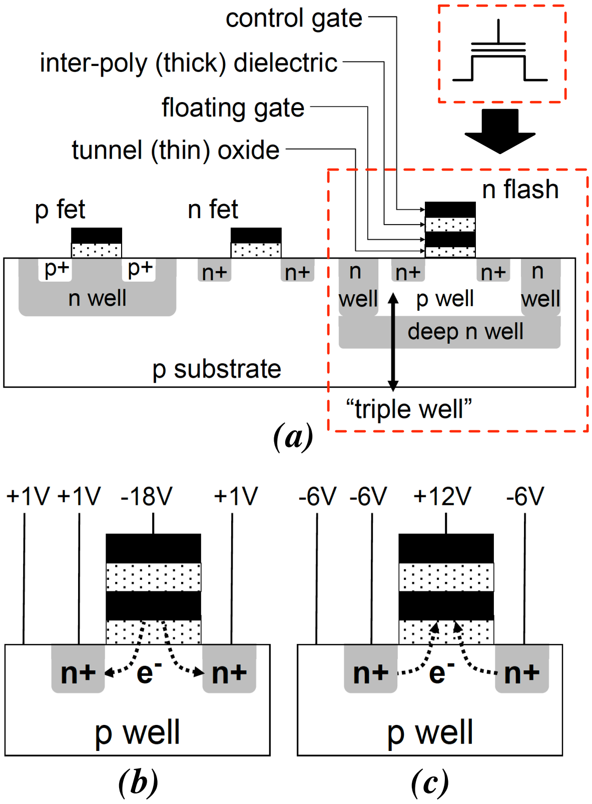


Figure 2.20 – (a) Embedded Flash Process (Courtesy by [7]); (b) Erasing operation of a Flash transistor (Courtesy by [7]); (c) Programming operation of a Flash transistor (Courtesy by [7]).

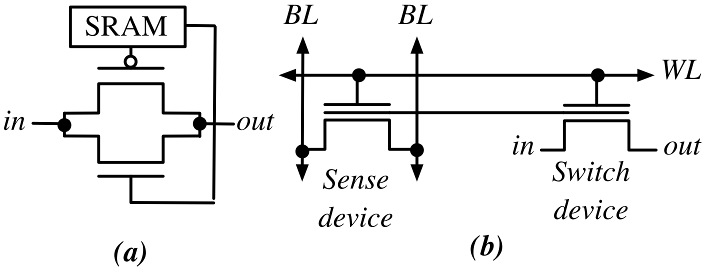


Figure 2.21 – (a) A transmission gate controlled by a SRAM; (b) Equivalent Flash-based programmable switch. (Courtesy by [7])

However, Flash transistors typically require a long configuration time ($\sim msec.$), a high programming current ($\sim mA$) and a large programming voltage ($> 10V$). To keep a short configuration time for the whole FPGA and also a low current budget, Flash transistors are programmed individually and in series. As configuration can be activated by applying a voltage difference between BL and WL, the Flash-based programmable switch in Fig. 2.21(b) is compatible with the configuration circuit in Fig. 2.18.

Indeed, Flash-based FPGAs are better in power consumption than SRAM-based counterparts, thanks to non-volatility. But the drawbacks are also obvious, including low-speed, complicated fabrication process and area overheads, due to the limitation of Flash technology. Therefore, mainstream FPGA products are still based on SRAMs while Flash-based FPGAs are preferred only when power budget is an more important factor than others.

In this thesis, our baseline FPGA architecture resembles a well-optimized commercial SRAM-based FPGA [88], including the following essential architectural enhancements: (1) tile-based architecture, (2) heterogeneous blocks, (3) fracturable LUT, (4) embedded adder chains and (5) single-driver uni-directional global routing architecture.

2.3 Previous works about RRAM-based Circuit Designs and FPGA Architectures

As summarized in Section 2.1, RRAM technology is appealing to FPGA researches owing to their low and tunable R_{LRS}, BEoL integration and non-volatility. This section aims at reviewing previous works related to RRAM-based FPGAs, including both novel circuit designs and architectures. These previous works provide important insights, e.g., inserting RRAMs in datapaths, which strongly motivates our works throughout this thesis. The first part of this section will focus on RRAM-based circuit designs related to FPGA architectures. We will first review programming structure, which is the basis for all essential circuits in FPGA architectures. Then, we report previous works about RRAM-based memory cell, *Flip-Flop* (FF) and routing multiplexer designs. The second part of this section introduce previous works about RRAM-based FPGA architectures, exploiting the circuit designs.

2.3.1 Programming Structures

Programming structures are the elements that configure the resistance states of RRAMs, which are actually the basis for all RRAM-based circuit designs and systems. The quality of programming structures directly determines the configuration time, achieved R_{LRS} and R_{HRS}, profoundly impacting the performance of circuits and systems. Therefore, programming structures are the most important and essential circuit designs and are worth intensive elaborations.

Typically, programming structures employ transistors to provide programming voltage and drive programming current for RRAMs. A programming structure is named according to the number of transistors dedicated to programming a RRAM, e.g., 1T(ransistor)1R(RAM). The transistors in programming structures are called programming transistors. Fig. 2.22 shows three most commonly used programming structures in RRAM-based FPGAs:

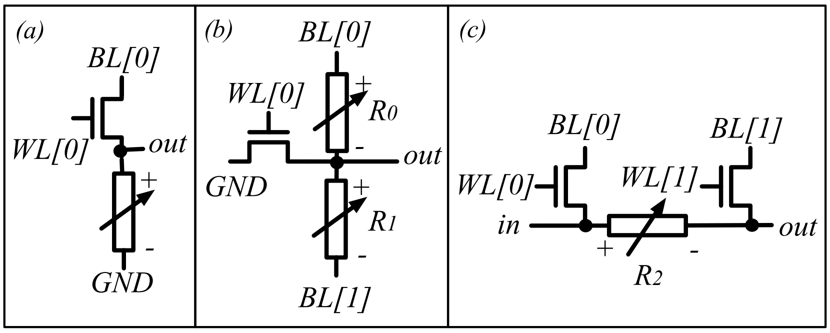


Figure 2.22 – Three most commonly used programming structures: (a) 1T(ransistor)1R(RAM), (b) 1T(ransistor)2R(RAM) and (c) 2T(ransistor)1R(RAM).

(1) **1T(ransistor)1R(RAM)**: The 1T1R programming structure is the most compact implementation, where a RRAM is programmed by a n-type transistor [1, 36, 108]. When $WL[0]$ is enabled in Fig. 2.22(a), the RRAM can be programmed by the voltage of $BL[0]$. When $BL[0] \geq V_{set}$, the RRAM is set to LRS. When $BL[0] \geq V_{reset}$, the RRAM is reset to HRS. During operation, $WL[0]$ is disabled and $BL[0] = V_{DD}$, the data of the RRAM can be read out through the output voltage $V_{out} = V_{DD}\frac{1}{1+R_{RRAM}/R_{trans}}$, where R_{RRAM} is the resistance of RRAM while R_{trans} represents the off-resistance of the programming transistor. Note that V_{DD} should be kept smaller than V_{set} and V_{reset}, to avoid parasitically programming RRAMs. Because each RRAM is accessed by an individual transistor, a 1T1R RRAM cell can eliminate serious problems in RRAM-based crossbar, e.g., the sneaking current and the disturbances during write and read [108].

(2) **1T(ransistor)2R(RAM)**: To improve the reliability, the 1T2R in Fig. 2.22(b) is proposed [8, 109]. The two RRAMs R_0 and R_1 are programmed simultaneously when programming transistor is turned on. Note that the polarity of the two RRAMs are always opposite. By applying $BL[0] = V_{set}$ and $BL[1] = V_{reset}$, RRAM R_0 is set to LRS while RRAM R_1 is reset to HRS. In contrast, $BL[0] = V_{reset}$ and $BL[1] = V_{set}$ configure RRAMs R_0 and R_1 to HRS and LRS respectively. During operation, programming transistor is switched off and $BL[0]$ is connected to V_{DD} while $BL[1]$ is connected to GND. The output voltage V_{out} is determined by $V_{DD}\frac{1}{1+R_1/R_0}$. The 1T1R is most robust to process variations than 1T1R because V_{out} is only related to on/off ratio of RRAMs R_{HRS}/R_{LRS}, whose variability is smaller than R_{HRS} and R_{LRS} [110, 8]. The 1T2R programming structures are proposed to replace SRAMs but they require a very high R_{HRS} ($\sim 10G\Omega$) for RRAMs to suppress the leakage power [111]. For instance, the leakage power of a 1T2R element is $P_{leakage} = V_{DD}^2/(R_{LRS} + R_{HRS})$. Since typically $R_{HRS} >> R_{LRS}$, the leakage power is dominated by R_{HRS}. Assume in 45nm technology node, $V_{DD} = 1.2V$ and an optimistic $R_{HRS} = 100M\Omega$, the leakage power of a RRAM structure is $14.4nW$, far more than the leakage power of a SRAM ($\sim 0.073nW$ [112]).

(3) **2T(ransistor)1R(RAM)**: To overcome the leakage issue, many works focus on embed-

ding RRAMs in the datapath along with two n-type programming transistors [26, 113, 9, 27, 8, 110, 6, 114, 111]. The 2T1R programming structure in Fig. 2.22(c) is proposed to provide equivalent functionality as a SRAM-controlled transmission gate. When $WL[0]$ and $WL[1]$ are enabled, RRAM R_2 can be programmed to HRS/LRS by setting $BL[0]-BL[1]=V_{reset}/V_{set}$. During operation, $WL[0]$ and $WL[1]$ are disabled and RRAM R_2 can propagate/block datapath signal from in to out. When inserted in the datapaths, RRAMs can introduce a low $RLRS$ ($\sim 1k\Omega$), which is ~ 75% less than transmission gates ($\sim 4k\Omega$ at 45-nm technology node) [26, 113, 9, 27, 8]. In addition, compared to a SRAM-controlled transmission gate occupying eight transistor area, the 2T1R programming structure requires only two transistors. By exploiting $RLRS$, the 2T1R programming structure opens an opportunity in area-efficient and high-performance routing architecture [26, 113, 9, 27, 8, 110, 6, 114, 111].

Controlled by BLs and WLs, the 1T1R, 1T2R and 2T1R programming structures can be accessed by the configuration circuits in Fig. 2.18, compatible to existed FPGA architectures.

In previous works [26, 113, 9, 27, 8, 110], evaluations of the 1T1R, 1T2R and 2T1R programming structures focus on functionality verification only, where the achieved R_{LRS} is always assumed to be lowest possible value. However, such simple analysis ignores crucial factors in circuit designs, i.e., electrical characteristics of RRAMs and transistors:

(1) Parasitic capacitances of RRAMs C_P are ignored, which has a strong impact on the circuit performance. Especially when RRAMs appear in datapath, C_P causes delay degradation of routing architecture, mitigating the performance gain from R_{LRS}.

(2) Side effects of programming transistors are also ignored. In order to achieve a low R_{LRS} or a high R_{HRS}, the sizes of programming transistors have to be large enough to drive sufficient programming current. For instance, to achieve the programming current required by [9] ($\sim 2mA$) with 45-nm transistor technology node (I_{set} = ~$200\mu A$ at minimal width), the size of the programming transistor should be ~ 10, far more than the size of a transmission gate (typically ~ 3). In this case, the parasitic capacitances of the programming transistors become non-negligible and may seriously threaten the performance of RRAM-based routing architecture. Therefore, RRAM-based circuits have to trade off between low R_{LRS} and large programming transistors.

(3) Programming structures are designed and verified based on ideal operating conditions. Previous works assume that during programming, the voltage across the RRAMs is stable and n-type transistors can always operate in saturation region, providing maximum programming current. However, these assumptions violate realistic electrical characteristics of transistors and RRAMs in two major aspects: (a) resistance switching of RRAMs leads to that the voltage across RRAMs is changing throughout the programming processes. A RRAM in HRS takes more voltage share than a RRAM in LRS. (b) transistors requires a large

source-to-drain voltage V_{DS} when operating at saturation region. But such V_{DS} may not be always achievable during resistance switching.

In short, instead of pure functionally verification, programming structures should be studied electrically by analyzing operating conditions of RRAMs and transistors. This motivate us to give a detailed study on programming structures in Section 3.

2.3.2 Non-Volatile Flip-Flop and SRAM

Rather than memory arrays, RRAMs can also enhance conventional FFs and SRAMs with non-volatile data storage.

Fig. 2.23 illustrates a *Non-Volatile Flip-Flop* (NVFF) design based on the master-slave FF in Fig. 2.17 [5, 115, 116]. The master stage of NVFF is same as the conventional FF, while the slave stage is modified to store data in RRAMs. During normal operation, the NVFF works the same as a conventional FF, where data storage purely relies on CMOS transistors. Prior to an active-to-sleep transition, the data stored in the slave latch needs to be written to the non-volatile RRAM devices. To this end, the clock is silenced and kept low for the entire duration of the RRAM write operation, thereby forcing the slave latch to be non-transparent and isolated from the master. During write, the RRAM devices are completely disconnected from the slave latch and from the read circuits, so that the voltage drop across their terminals can be set by the write drivers. Note that the two RRAM devices are always used in a complementary fashion, i.e., one device is programmed to the HRS, while the other one is programmed to the LRS. During system wake-up (power-on), the slave latch would ideally be directly restored, based on the data stored in the RRAM devices. Both internal storage nodes Q and $\overline{\mathrm{Q}}$ are first pre-charged and equalized using three dedicated PMOS transistors controlled by $\overline{\mathrm{EQ}}$. Following this pre-charge phase, the internal nodes Q and $\overline{\mathrm{Q}}$ are connected to ground through the RRAM devices. Note that the NVFF can also be used in *Scan-chain* configuration circuit (Fig. 2.19).

The slave latch of a NVFF can be simplified to be a NV SRAM, as shown in Fig. 2.24. The NV SRAM can be configured like the memory array in Fig. 2.18. Similar to NVFFs, the storage is transferred to RRAMs before system power down and also can be loaded from RRAMs after system wake-up. The NVFF and NV SRAM have the same performance as conventional circuits because they share the same working principle during normal operation. Thanks to non-volatility, the energy consumption of NVFF and NV SRAM is 67% smaller than volatile versions.

2.3.3 Multiplexer and Crossbar Designs

Earlier works [22, 117, 110, 109, 118] used 1T1R and 1T2R memory structures to replace the configuration memories in the routing structures. These modifications grant non-volatility to

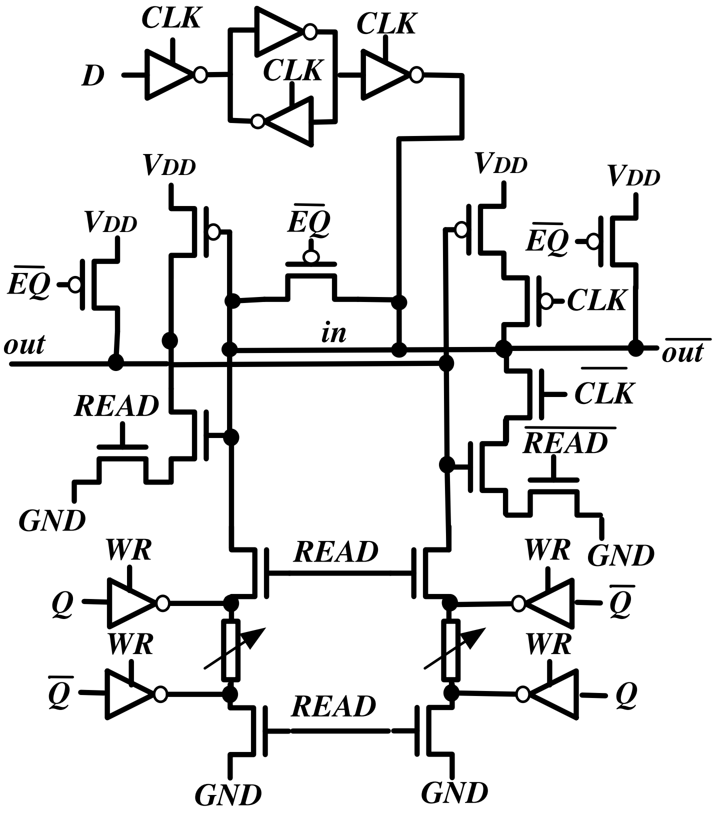


Figure 2.23 – A non-volatile master-slave Flip-Flop design [5, 6].

the FPGA and enable *instant-on normally-off* operations. However, the multiplexer structures in [22, 117, 110, 109, 118] were still based on CMOS multiplexers, leading to no improvements on performance.

To leverage the potential of the 2T1R programming structure, non-volatile routing multiplexer design have been intensively studied in [9, 26, 27, 8, 113]. Fig. 2.25(a) shows a one-level N-input 2T1R-based multiplexer [9, 26, 8, 113], where all the programming structures share a common n-type transistor at the output node. The 2T1R-based multiplexers in Fig. 2.25 depend on n-type transistors to provide high programming current, in order to achieve a low R_{LRS}. For instance, when $WL[0] = WL[N] =' 1'$, $BL[0] =' 1'$ and $BL[N] =' 0'$, RRAM R_0 is programmed to LRS. Fig. 2.25(b) presents an illustrative example of a two-level/tree-like 2T1R-based multiplexer [27], whose input size is 4. Note that every two RRAMs are opposite in polarity, which enables complementary programming. RRAMs belonging to the same stage are programmed simultaneously. Take the example in Fig. 2.25(b), when $BL[0] =' 1'$, $BL[1] =' 1'$, $BL[2] =' 0'$, $WL[0] =' 1'$, $WL[1] =' 0'$ and $WL[2] =' 0'$, RRAMs of the first stage sharing the same polarity with R_0 are programmed to LRS, while those sharing the same polarity with R_1 are programmed to HRS. Note that, every two RRAMs are always different in the resistance states

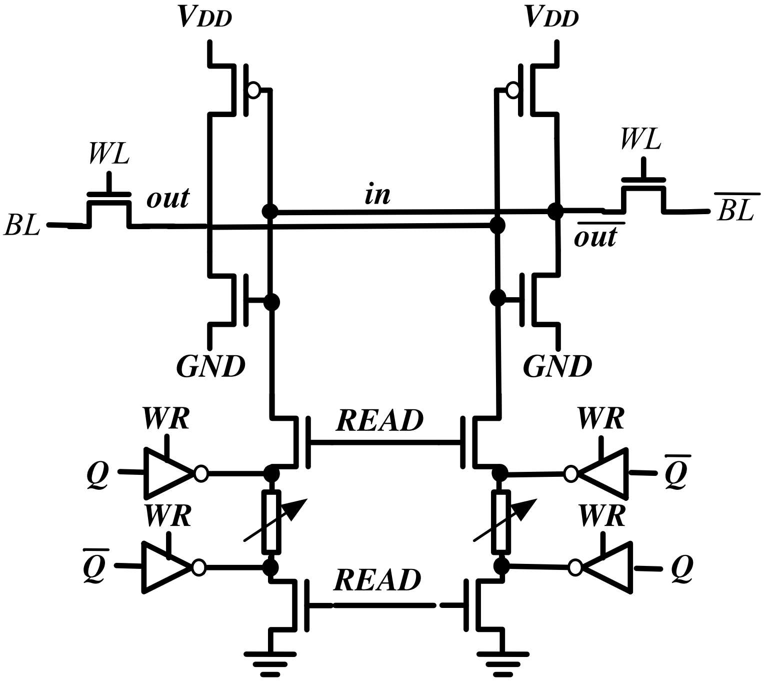


Figure 2.24 – A non-volatile SRAM design [5, 6].

and RRAM programming is conducted stage by stage, which is similar to the tree-like CMOS multiplexers in Fig. 2.15(b).

By efficiently sharing programming transistors in multiplexing structure, the ratio between the number of programming transistors and RRAMs approaches 1 : 1 when input size increases. In addition to better granularity, sharing programming transistors also contribute to better performance. For instance, whatever input size is, the 2T1R-based multiplexer in Fig. 2.15(a) only need a n-type transistor at the output node. As a result, the parasitic capacitance on critical paths and the delay of multiplexers are independent from input size, which cannot be achieved by **any** CMOS multiplexers in Fig. 2.14 and Fig. 2.15. Such relationship between input size and performance becomes a strong motivation for chapter 5 which explores RRAM-based FPGA architectures.

2.3.4 RRAM-based FPGA Architectures

FPGA architecture can benefit from the *non-volatility* as well as the area and performance gains coming from the BEoL integration and the low R_{LRS} achieved by RRAMs. Previous works [109, 110, 8, 26, 9] proposed novel FPGA architecture based on two principles: (a) replace the SRAMs in LUTs with RRAMs, and (b) replace the SRAMs as well as the transmission-gates in routing structures with RRAMs.

Fig. 2.26 illustrates early RRAM-based FPGA architectures where bi-directional routing ar-

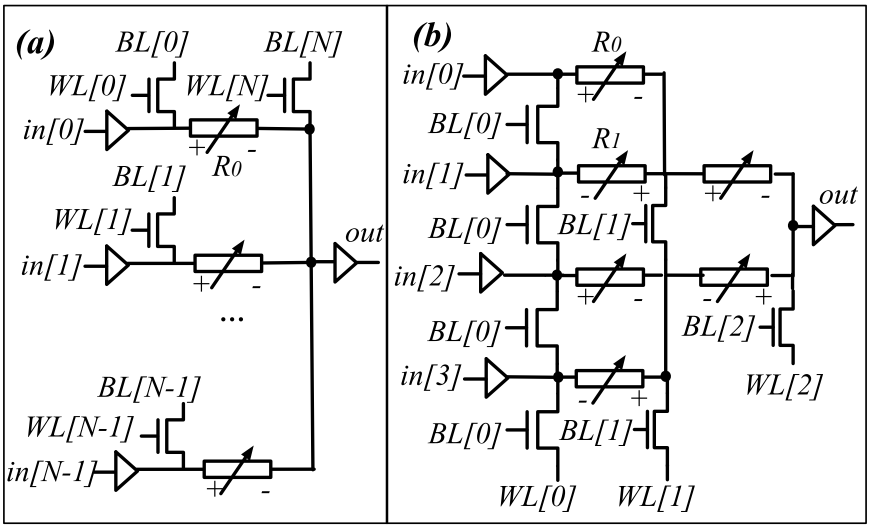


Figure 2.25 – Early designs of 2T1R-based multiplexers: (a) A N-input onelevel structure [9]; (b) An illustrative example of two-level and tree-like 4:1 structure [10].

chitecture is employed. As a direct approach, SRAMs can be replaced by 2T1R programming structures (Fig. 2.26(b)), as proposed by P.-E. Gaillardon *et al.* [119]. Y. Chen *et al.* study a RRAM-based FPGA using such scheme [109], while Y. Yang-Liauw *et al.* recently demonstrated a functional prototype [117]. 2T1R programming structures can also be employed to realize RRAM-based LUT structures (Fig. 2.26(a)) as proposed by P.-E. Gaillardon *et al.* [110]. Efficient CB and SB design as proposed by S. Tanachutiwat *et al.* [26] and J. Cong *et al.* [9] further improve the granularity of bi-directional routing architecture through sharing programming transistors and eliminating tri-state buffers. As illustrated in Fig. 2.26, all the programmable switches that connected to either a routing track or a CLB pin to share a programming transistor. Without tri-state buffers, the transistor area of global routing architecture only is dominated by programming transistors, since RRAMs are fabricated above transistors. As global routing architecture typically occupies more than 50% area of a FPGA, the predicted area gain of RRAM-based FPGAs is $2-3\times$ [26, 9]. However, the absence of tri-state buffers causes the sneak path problems [120, 121, 122] in routing architecture, which is hard to be addressed. During programming, RRAMs in LRS can distribute the programming currents for other RRAMs on the same routing track. Consequently, some RRAMs have a higher R_{LRS} than expected, decreasing the speed of routing paths.

Previous RRAM-based FPGA studies [26, 113, 9, 27] also follow the trends of uni-directional routing architecture and single driver wiring technique, where one/multi-level RRAM-based multiplexers is the key to achieve area, delay and power reduction. More than global routing architecture, the local routing architecture can also benefit from the 2T1R-based multiplexers in Fig. 2.25. Compared to bi-directional routing, uni-directional solution can avoid sneak path problems because RRAMs are separated by buffers. Therefore, this thesis will consider only uni-directional routing architecture for the exploration of RRAM-based FPGA architectures

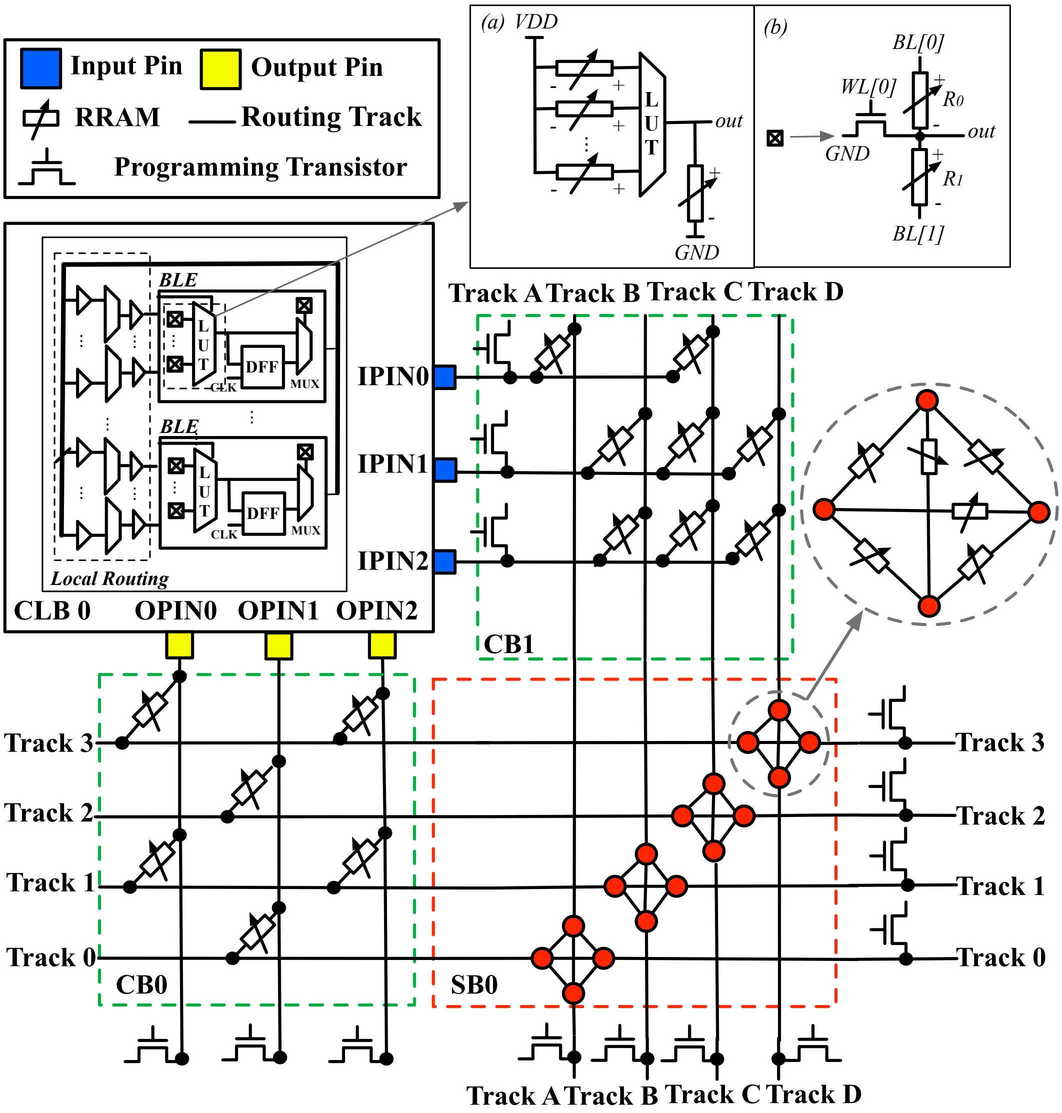


Figure 2.26 – Early RRAM-based FPGA architectures (a)LUTs embedded with 2T1R programming structures; (b)SRAMs are replaced by 2T1R programming structures.

(See Chapter 5).

However, most RRAM-based researches overlook the challenges coming from programming structures (see Section 2.3.1), which may lead to a strong bias in the estimation of any performance metric improvements. Previous works [26, 113, 9, 27, 109, 118, 110, 8] predict that RRAM-based FPGAs can reduce the area by 7%-15%, increase the performance by 45%-58%, and save the power consumption by 20%-58%, compared to SRAM-based FPGAs. However, these architectural improvements are obtained by simply replacing SRAM-based transmission gates in classical FPGA architectures with RRAM-based programming structures. Very limited work studies the impact on novel RRAM-based FPGA architectures that exploit the circuit-level features of RRAM-based multiplexers. Therefore, it is worthy to investigate specific architectural optimizations for RRAM-based FPGAs that would derive from realistic RRAM-based multiplexer designs (See chapter 5).

2.4 FPGA Architecture Exploration Tool and Power Modeling Technique

The most accurate approach to evaluate a FPGA architecture is to manufacture a FPGA chip and then measure its performance by implementing a set of benchmark circuits. However, the architecture of FPGA is dependent on a large number of parameters, as listed in Table 2.2, resulting in a large design space to be explored. As manufacturing and testing all the FPGA architectures in the design space is not practical, modeling FPGA architectures with EDA tools and estimate their performance with analytical models is necessary. Sophisticated EDA tools can reduce the large design space to a few candidates of best FPGA architectures. To guarantee reliable results, the analytical models should be accurate enough to capture the characteristics of diverse FPGAs architectures. Otherwise, the EDA tools would lead to misleading conclusions on the best FPGA architectures. This section is devoted to the EDA techniques used in current best academic FPGA architecture exploration tools. This section consists of two parts. The first part introduces current state-of-art FPGA architecture exploration tools, while the second part discusses the limitation of mainstream power estimation techniques in the context of emerging technologies.

2.4.1 FPGA EDA flow

The purpose of FPGA architecture exploration is to search the best FPGA architecture for a specific technology. Typically, merits of a FPGA architecture are judged by evaluating their area, delay and power consumption average over a set of benchmark circuits. The evaluation is performed with a complete EDA tool suite, where a benchmark circuit is virtually implemented by a hypothesized FPGA.

Fig. 2.27 illustrates the *Verilog-To-Routing* (VTR) flow, which is current state-of-art academic EDA flow for the purpose of FPGA architecture exploration [4, 44]. First of all, the logic synthesis tool, ABC [123], optimizes the benchmark circuits and performs a technology mapping. Then, the activity estimator ACE2 [124] computes the signal activities of all the internal nodes in the benchmark circuits. Finally, the tool VPR [44] packs, places and routes the circuits onto a virtual FPGA architecture defined by the architecture description language. In the packing stage, LUTs, FFs and hard adders are clustered into CLBs. Placement determines the physical positions of CLBs in the FPGA fabric. Routing maps the nets of CLBs into routing architectures. The routing stage contains two steps. In the first step, VPR performs a binary search to determine the minimum channel width W_{min} required for a given benchmark circuit

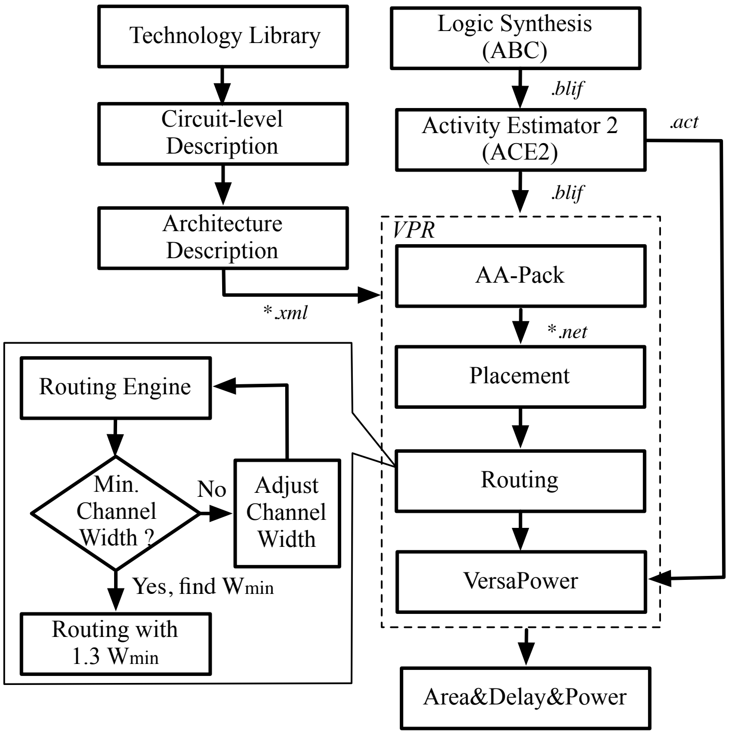


Figure 2.27 – Classical EDA flow for FPGA architecture exploration purpose.

and the FPGA architecture under evaluation. In the second step, a 30% slack is added to the minimum routable channel width W_{min}, in order to simulate a low-stress routing [4]. This comes from the fact that commercial FPGAs are normally built with sufficient routing tracks that "average" circuits have some spare routing available. After routing, VPR reports area and delay by using *Minimum Transistor Width Area* (MTWA) model [4, 125] and Elmore delay model [104] respectively, while power consumption is estimated by VersaPower [46]. The best FPGA architectures are in general determined by overall performance, such as *Area-Delay Product* (ADP).

2.4.2 Probability-based Power Estimation Techniques

Very Large Scale Integration (VLSI) power estimation techniques can be classified into two categories: simulation-based and probability-based [126, 127]. On the one hand, simulation-based methods are the most direct ways to do accurate power analysis. They typically rely on SPICE-based simulations to analyze the power consumption of a given circuit netlist. However, in the 1990s, SPICE simulations were regarded to be only applicable for small-scale circuits due to the low simulation speed and high memory usage [126, 127]. On the other hand, probability-based methods are based on signal activity estimation and analytical power models. Average power consumption is calculated by combining signal switch density and switching power. Compared to a simulation-based method, a probability-based method is faster but trades off accuracy due to the approximate errors in analytical power models and signal activity estimations.

In the specific context of FPGAs, the power estimation engines embedded in academic architecture exploration tools are typically based on probabilistic activity estimation [124] and analytical power models [128, 41, 46].

Signal Activity Estimation

The probability activity estimation models the transitions of a signal with two parameters: the static probability and the transition density. The static probability $P(x)$ at node x is defined as the average fraction of clock cycles in which the steady state value of x is a logic high. The transition density $D(x)$ is the average number of transitions per clock cycle at node x. Fig. 2.28 exemplifies two signals A and B and also the clock signal as reference. Table 2.4 lists the corresponding static probability and transition density of signals.

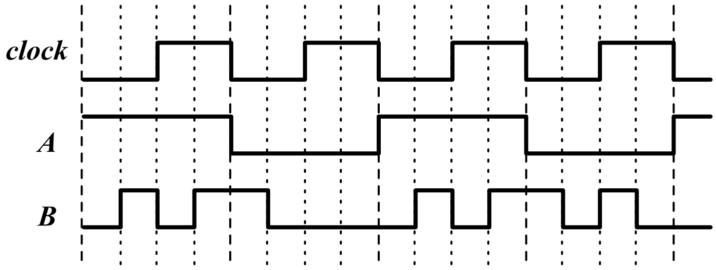


Figure 2.28 – Examples of signals for switching activity modeling.

Table 2.4 – Static probability and transition density of the signals in Fig. 2.28.

Signal	Static Probability	Transition Density
Clock	0.5	2
A	0.5	1
B	0.43	2.5

The transition density can be propagated through a logic gate. Assume a logic gate with n inputs $x_i, 1 \leq i \leq n$, an output y, and a function $y = f(x)$. The $P(x)$ and $D(x)$ at the output node y is determined by the Boolean Difference.

$$\begin{aligned} D(y) &= \sum_{i=1}^{n} P(\frac{\partial f(x)}{\partial x_i})D(x_i), \\ \frac{\partial f(x)}{\partial x_i} &= f(x)\,|_{x_i=1} \oplus f(x)\,|_{x_i=0} \end{aligned} \tag{2.6}$$

When transition density is known for every primary inputs of a circuit, it is possible to compute the transition density of all the internal nodes and primary outputs by applying (2.6) to each logic gate. More details about switching activity modeling and associated algorithms can be found in [128, 124].

Analytical Power Models

The total power of a circuit is the sum of two parts: leakage power and dynamic power[126, 127].

Leakage power is the power dissipation of a circuit with zero transition density. It is well known that the leakage power strongly depends on various factors, including process technology, circuit topology and the state of inputs. Developing a purely analytical leakage power model has to involve many technology parameters, whose numbers keep increasing for modern CMOS technologies [128, 41, 46]. Therefore, previous works [126, 127, 128, 41, 46] commonly estimate leakage power with simulation-based approaches. For each circuit primitive, a leakage power library is built from simulation results with a specific CMOS technology, different circuit designs featured by various transistor sizes. The total leakage power is obtained by identifying the leakage power of circuit primitives in their associated library and then summing up. Even though it is time-consuming to build a leakage power library due to a large number of electrical simulations, such method guarantees good accuracy as compared to purely analytical leakage power models [128, 46]. In VersaPower [46], the average error between estimated leakage power and SPICE results is within 5%.

However, the majority of total power comes from dynamic power consumption, which has two sources: (1) the switching power resulting from charging and discharging parasitic capacitances, and (2) the short-circuit power dissipated by temporary *Direct Current* (DC) paths during signal transitions.

Fig. 2.29 provides an illustrative example to understand the sources of the switching and short-circuit power. The CMOS inverter in Fig. 2.29(a) can be modelled by the RC tree in Fig. 2.29(b), where C_g is the total gate capacitance of transistors P_1 and N_1, R_A and R_B are the equivalent channel resistance of transistors P_1 and N_1 respectively, and C_o is the total parasitic capacitance at node out. Note that C_o includes both parasitic capacitance of transistors and the load capacitance C_L in Fig. 2.29(a). During the transition of input in, there is two types of currents flowing from V_{DD}: capacitance charging current I_{sw} and short-circuit current I_{sc}.

The switching power results from I_{sw}, which charges C_o until $V_{out} = V_{DD}$. Considering transition density in Fig. 2.29(b), the average switching power of node out is

$$P_{sw}(out) = \int_t i_{sw}(t) V_{DD} dt = \frac{1}{2} D(out) \cdot C_o \cdot V_{DD}^2 \cdot f_{clk}, \tag{2.7}$$

where $D(out)$ represents the transition density of node out, V_{DD} denotes the supply voltage and f_{clk} is the clock frequency. The accuracy of the switching power model in (2.7) mainly

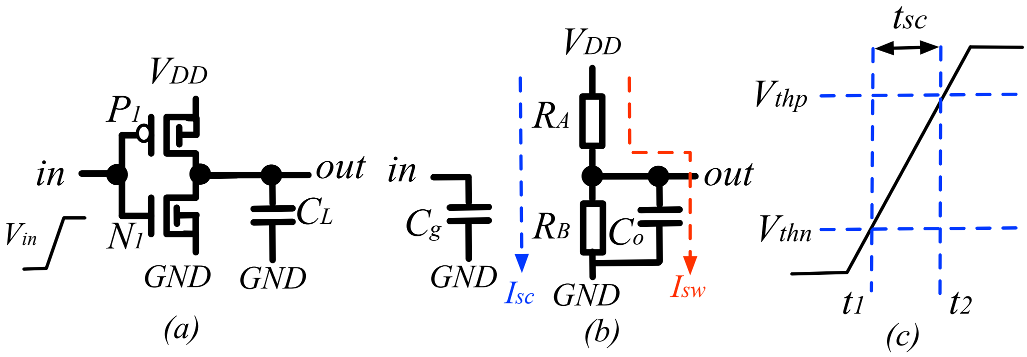


Figure 2.29 – Dynamic power modelling: (a) an CMOS inverter with a load capacitance C_L; (b) Equivalent RC model; (c) Input transition from low to high voltage level.

depends on the value of C_o. Since the parasitic capacitance of a transistor is in general a function of the source-to-drain voltage V_{DS}, which is actually changing during a transition. In practice, power estimation tools build a library for the average parasitic capacitance of a transistor, by extracting from a large number of simulation results [128, 41, 46].

Note that during the input transition, transistors P_1 and N_1 are not fully turned on or off. As depicted in Fig. 2.29(c), when input voltage V_{in} swings from the threshold voltage of transistor N_1, V_{thn}, to the threshold voltage of transistor P_1, V_{thp}, transistors P_1 and N_1 operate at sub-threshold regime and both of them are considered to be in *on* state. Consequently, there is a short-circuit current I_{sc} flowing from V_{DD} to GND during the time period t_{sc}. The short circuit power during a transition can be calculated by

$$P_{sc}(out) = \int_{t_1}^{t_2} i_{sc}(t) V_{DD} dt. \tag{2.8}$$

However, the short-circuit power is difficult to be accurately estimated, due to that $i(t)$ are changing during the transition and it is strongly dependent on the shape of input voltage V_{in}. For instance, slews of V_{in} lead to large difference in the short-circuit power [129]. The estimated short-circuit power typically has an error as large as 10-20% when compared to simulation results [128, 129].

The total dynamic power of a circuit is the sum of the switching and short-circuit power of each node:

$$P_{dynamic,total} = \sum_{i \in nodes} (P_{sc}(i) + P_{sw}(i)). \tag{2.9}$$

Despite the difficulties in accurate modelling capacitances and shape of voltages, the dynamic power models encounter more serious challenges in accuracy from the reconfigurability of

FPGAs:

(1) The accuracy of these analytical power models is guaranteed for only a few input signal patterns of the different circuit elements. Unfortunately, the input signal patterns of FPGAs may significantly differ from a design to another. For instance, the power differences of a 4-input LUT can reach 69% under diverse input signal patterns [41]. Therefore, current power estimation tools guarantee accuracy on very restrictive conditions.

(2) Transistor-level circuit designs are diverse in FPGA architectures, leading to different dynamic power characteristics. For instance, a routing multiplexer has three different transistor-level implementation as shown in Fig. 2.14 and 2.15, each of which has different power characteristic as list in Table 2.3. Academic FPGA architecture exploration tools [44] employ architecture description language [48] to model highly flexible FPGA architectures. The hierarchy and complex interconnects inside modern FPGA logic block architectures can be precisely described with the architecture description language. The timing parameters of logic and routing elements are richly provided for accurate timing analysis. However, there are very limited transistor-level modeling parameters in architecture description language, that can be exploited for power estimations.

(3) Configuration circuits of FPGA architectures are diverse, strongly depending on the memory technologies. For instance, Section 2.2 introduces two types of configuration circuits, which are based on scan-chain FFs and memory arrays respectively. The choice of configuration circuits leads to different power characteristics of FPGA architectures. However, current FPGA exploration tools neglect the contribution of configuration circuits, leading to inaccurate power analysis for entire FPGA architecture.

These three challenges cause over 20% error between estimated power and SPICE results on average when evaluating individual modules, such as LUTs and routing multiplexers [46]. Note that only a limited input patterns and configurations are considered when evaluating the LUTs and routing multiplexers because it is extremely time-consuming to enumerate all the possible conditions. In terms of full FPGA architectures, the error may be even worse when considering a specific benchmark circuit is mapped to a FPGA, because the configurations of LUTs and routing multiplexers may hit the worst cases of analytical models. Furthermore, the accuracy of estimated power has not been carefully examined for full FPGA fabrics due to the lack of SPICE modeling in VPR tools. Additionally, current FPGA power models are developed exclusively for CMOS logic, while there is very limited work with respect to emerging technologies. When developing novel FPGA power models, providing reliable baseline SPICE results is always a necessity.

Overall, the analytical power estimation method is a difficult problem. Without advanced dynamic power models and versatile EDA supports, current power estimation tools relying on analytical power models cannot capture well the power characteristics of a wide range of novel FPGA architectures. To guarantee accurate power analysis for novel circuit design topologies and general FPGA architectures, the simulation-based approaches are worth a revisit and the FPGA architecture description language needs to be extended for power modeling parameters. In chapter 4, we will introduce FPGA-SPICE, a simulation-based accurate power analysis framework, enabling SPICE modeling for versatile FPGA architectures.

2.5 Summary

This chapter has covered memory technologies, circuit designs, architectures and EDA techniques of both conventional and emerging RRAM-based FPGAs. We first reviewed the basics of RRAM technology, which are exploited intensively from circuit design and architecture perspectives in Chapter 3 and Chapter 5. In the second part, we then detailed circuit designs and architectures of SRAM-based FPGAs, which are the baselines of performance evaluations in Chapter 3, Chapter 4 and Chapter 5. The third part presented important prior researches about RRAM-based FPGAs, whose merits will be discussed detailedly in Chapter 3. Finally, we introduced the EDA techniques for conventional FPGAs and in particular focused on the power estimation techniques, the limitations of which will be overcome in Chapter 4.

3 RRAM-based Circuit Designs

Circuit design is a corner stone of FPGA architectures. Actually, it is one of the most critical factors impacting the overall performance of FPGAs. Without efficient RRAM-based circuit designs, it is hard for RRAM-based FPGAs to demonstrate advantages over SRAM-based counterparts. This chapter proposes novel RRAM-based circuit designs and examines their superiority over SRAM-based circuits through both theoretical analysis and electrical simulations. This chapter is divided into two parts:
1. RRAM-based programming structures: the access circuits for RRAMs, which are the most basic elements in all RRAM-based circuit designs, such as NV SRAMs, NV FFs and multiplexers.
2. RRAM-based multiplexer designs: routing circuits employing RRAMs to propagate datapath signals, which are the most frequent element in FPGA architectures.

Part 1: RRAM-based Programming Structures

Programming structures are the circuit elements devoted to configuring RRAMs. As mentioned in Section 2.3, RRAM-based FPGAs account on the low R_{LRS} of RRAMs to guarantee their high performance. Therefore, the quality of programming structures (their ability to achieve low R_{LRS} while minimizing the area footprint) is a crucial factor of the performance of RRAM-based FPGAs. This part provides a thorough study of RRAM-based programming structures for FPGA architectures. We will focus on three most representative programming structures, which are 2T(ransistor)1R(RAM), 2T(ransmission)G(ate)1R(RAM) and 4T(ransistor)1R(RAM). When analyzing each programming structure, we perform both theoretical analysis and electrical simulations in order to demonstrate their advantages and limitations.

This part consists of four sections: Section 3.1 introduces general experimental methodology in evaluating programming structures. Section 3.2 analyzes the specificities and limitations of 2T(ransistor)1R(RAM) programming structure, and discuss the associated shortcomings, such as low current density and area inefficiency. Section 3.3 studies 2T(ransmission)G(ate)1R(RAM) programming structure and discusses its advantages and limitations compared to 2T1R. Section 3.4 proposes a more advanced 4T(ransistor)1R(RAM) programming structure, overcoming limitations of 2T1R and 2TG1R programming structures.

3.1 Experimental Methodology

When studying programming structures, we consider the RRAM model in [130, 131], whose V_{set}/V_{reset} is 1.3V/-1.3V respectively, R_{LRS} is 500Ω, and R_{HRS} is $20k\Omega$ ($R_{HRS}/R_{LRS} = 40$). The current compliance I_{set} and I_{reset} is set to $1mA$, considered as a way to avoid large thermal damage. The minimum required pulse width for programming the RRAM element is 100ns. The programming structures discussed in the paper are implemented with I/O transistors (W/L=320nm/270nm) from a commercial $45nm$ process technology. The associated transistor model is based on BSIM4. The standard V_{GS} and V_{DS} of transistors are 2.5V. The transistors can be over-driven up to 3.0V. The ratio between *p*/*n*-type transistors β is set to 3. In this part, we also consider the area overhead of the P-Well of *p*-type transistors for which a penalty factor $\gamma = 1.2$ is set.

Electrical simulations are run with HSPICE simulator [47]. The time step of electrical simulations is set to $0.1ps$. In each simulation, the RRAM is initialized to the HRS and then transistors are turned on to program the RRAM into LRS. At the end of programming period, we measure the voltage difference between the RRAM electrodes and the current passing through to calculate the LRS resistance R_{LRS}.

We sweep two parameters: the width of transistors W_{prog} and the programming voltage V_{prog}, to study their impact on the performance of programming structures. W_{prog} is defined as the width of the *n*-type transistors used in the structures expressed by the minimal size transistors. W_{prog} is swept in the range from 1 to 5 with a 0.1 step. V_{prog} is swept in the range from 2.5V to 3.0V with a 0.1V step.

Note that, to achieve significant FPGA improvements, a R_{HRS} of at least $20M\Omega$ must be employed [114]. However, as the presented methodology and structures are general for any device parameters and for the sake of reproducibility, we present results using the base parameters of the RRAM model in [130, 131]. We will consider RRAM parameters meeting the demand of FPGA architectures when studying RRAM-based multiplexer design (Second part of this chapter) and FPGA architecture-level optimizations (Chapter 5)

3.2 Limitations of 2T1R Programming Structure

This section begins with circuit design of 2T1R programming structures including the effects from system-level implementations. Then, theoretical analysis is performed from three aspects: I-V characteristics (Section 3.2.2), physical design (Section 3.2.3) and area consumption (Section 3.2.4). Last but not least, electrical simulation results are presented to validate the conclusions of theoretical analysis.

3.2.1 2T1R Circuit Structure

Practical analysis programming structures should consider the context of system-level implementations. Previous works [26, 9, 110, 27, 8, 6] mainly exploit two different strategies to access the individual 2T1R memory elements. A scan-chain organization, as shown in Fig. 3.1(a), has been proposed in [8] while a memory bank arrangement, as shown in Fig. 3.1(b), has been employed in [9]. With the scan-chain organization that is similar to modern FPGAs, RRAMs are programmed through *Flip-Flop* (FF) outputs when signal *prog* is set to 1. For example, when $Q_0 = 1, Q_1 = 0$, a set process for $RRAM_0$ is started. In a memory bank arrangement, the RRAMs are programmed through *Bit Lines* (BLs) and *Word Lines* (WLs). For instance, when $WL[1] = 1, WL[2] = 1, BL[0] = 1, BL[2] = 0$, a set process for $RRAM$ is initiated. Note that, with this strategy, only one RRAM is programmed at a given time - allowing to limit the programming current to be delivered to the chips.

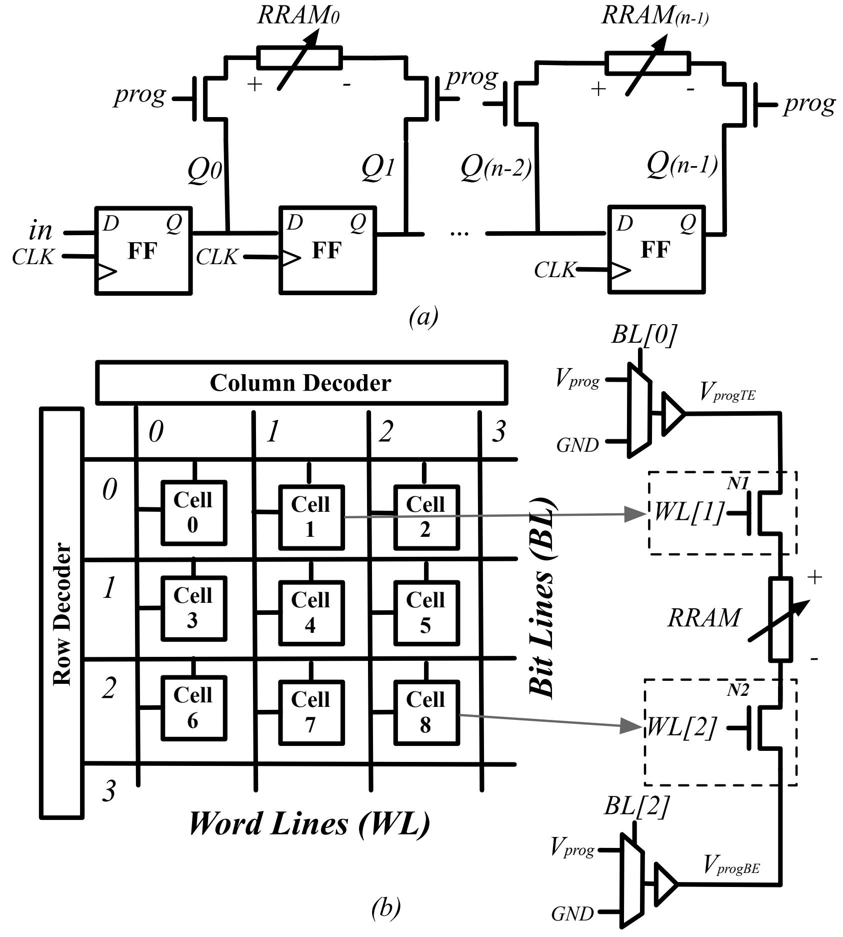


Figure 3.1 – System-level implementations exploiting the 2T1R programming structure: (a) scan chain [8]; (b) memory bank [9].

In Fig. 3.2, we extract a 2T1R structure along with its driving inverters from the system-level implementation shown in Fig. 3.1. A 2T1R structure requires driving inverters to provide the voltage levels of V_{progTE} and V_{progBE} during a programming phase. In a **set** process, the terminals of 2T1R structure V_{progTE} and V_{progBE} are driven by a p-type transistor **P1** and a n-type transistor **N3**, respectively. As illustrated in Fig. 3.2, the driving inverters introduce two potential voltage drops caused by the drain-to-source voltage V_{DS3} and V_{DS4} of transistors **P1** and **N3**, while the 2T1R structure has two built-in voltage drops caused by V_{DS1} and V_{DS2} of transistors **N1** and **N2**. In a **reset** process, the terminals of 2T1R structure V_{progTE} and V_{progBE} are driven by a n-type transistor **N4** and a p-type transistor **P2**, respectively. Similarly, another two drain-to-source voltage drops of transistors **P2** and **N4** are introduced.

Note that the principles in the circuit designs of programming structures are different from logic gates, because the programming structures are driving a resistive load instead of a capacitive one. To drive a resistive load like a RRAM, the source-to-drain voltages V_{DS} of transistors should be large enough in order to ensure a high current. Moreover, when the V_{DS} voltage drops of the transistors take most of the supply range V_{prog} and the voltage difference between the RRAM electrodes goes below the programming threshold voltage, a correct programming cannot be guaranteed. Since driving inverters are shared among programming transistors, their effects on adjusting the programming current is limited. To tune R_{LRS} for each individual RRAM, we should focus on studying how to adjust the driving current through sizing programming transistors **N1** and **N2**. Considering that $V_{prog} = V_{DS1} + V_{DS2} + V_{DS3} + V_{DS4} + V_{RRAM}$, maximize the driving current I_{ds} implies that V_{DS1} and V_{DS2} should be maximized while the effect of V_{DS3} and V_{DS4} should be avoided as much as possible. As a result, the sizes of transistors **P1** and **N3** have to be far larger than **N1** and **N2**, so that V_{DS3} and V_{DS4} can be neglected compared to V_{DS1} and V_{DS2}. We take this assumption and focus on the set process in the rest of the analysis. Without loss of generality, our approach can be applied to the reset process as well.

3.2.2 I-V Characteristics of 2T1R Structure

In this part, we consider the voltage drops V_{DS1} and V_{DS2} in Fig. 3.2 and discuss the I-V characteristics of a 2T1R structure. By considering Kirchhoff circuit laws:

$$\begin{cases} I_{ds} = f(V_{GS1}, V_{DS1}) = f(V_{GS2}, V_{DS2}) \\ V_{RRAM} = I_{ds} R_{RRAM} \\ V_{prog} = V_{DS1} + V_{DS2} + V_{RRAM}. \end{cases} \tag{3.1}$$

where I_{ds} is the current passing through the transistors and RRAM. R_{RRAM} denotes resistance of RRAM. $f(V_{GS1}, V_{DS1})$ and $f(V_{GS2}, V_{DS2})$ represent the I-V relationships of transistors **N1** and **N2** in Fig. 3.2. To give an intuition on the operating points of transistors, we consider the

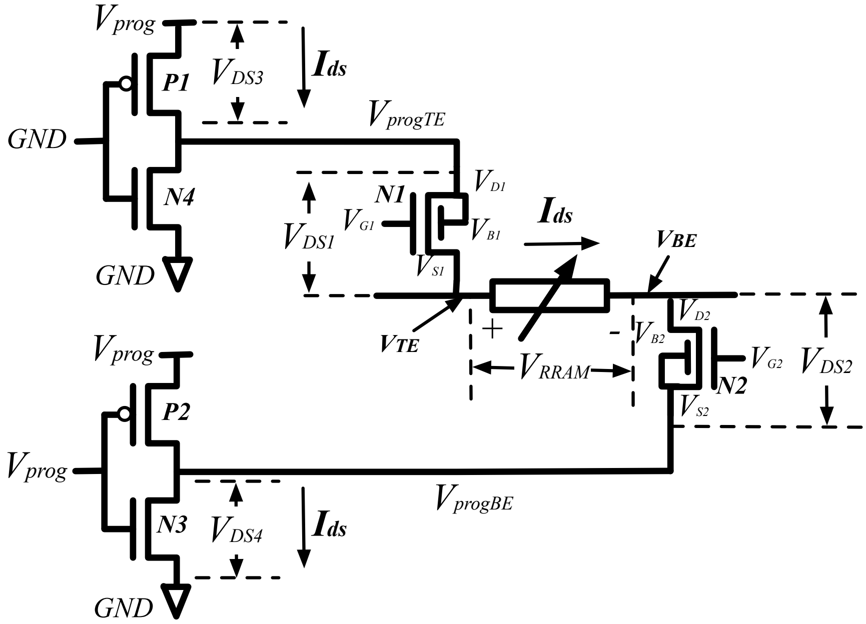


Figure 3.2 – A 2T1R programming structure extracted from system-level implementations in Fig. 3.1

following transistor model:

$$I_{ds} = \begin{cases} k_n \frac{W}{L}[(V_{GS} - V_T)V_{DS} - \frac{1}{2}{V_{DS}}^2], & V_{DS} < V_{GS} - V_T \\ \frac{1}{2}k_n \frac{W}{L}(V_{GS} - V_T)^2, & V_{DS} \geq V_{GS} - V_T \end{cases} \tag{3.2}$$

where k_n denotes the process transconductance parameter of a *n-type* transistor and V_T represents its threshold voltage. W and L are the width and length of channel, respectively. V_{GS} is the voltage difference between the gate and source terminals. V_{DS} is the voltage difference between the drain and source terminals. The intuitive results obtained with the model will be subsequently validated by SPICE simulations. In the theoretical analysis, we focus on studying how the current I_{ds} is changed with V_{GS1}, V_{GS2}, V_{DS1} and V_{DS2} during a set programming phase.

Fig. 3.3 illustrates the I-V curve of the transistors **N1** and **N2** during the programming phase. A programming phase starts when the transistors **N1** and **N2** are turned *on* and the RRAM is in HRS. At the start point **P**, I_{ds} is close to zero because the HRS resistance R_{HRS} of the RRAM typically is very high, leading to V_{DS1} and V_{DS2} approaching zero. V_{RRAM} is above the programming threshold voltage V_{set}, and therefore a resistive transition occurs and the resistance decreases. Note that V_{GS2} equals to V_{G2} because the source voltage of transistors **N2** is *GND*, while $V_{GS1} = V_{G1} - V_{TE}$, is much smaller than V_{GS2}. Then, the resistance of the RRAM is gradually decreasing from R_{HRS} to R_{LRS}, leading to an increase in I_{ds}. The growth in I_{ds} creates a positive feedback: V_{DS1} and V_{DS2} are increasing to provide a higher current which

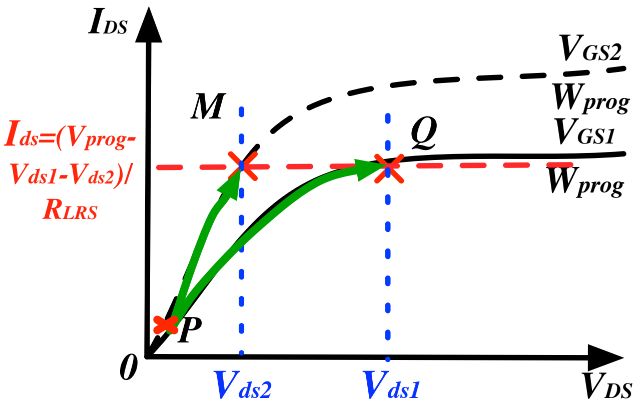


Figure 3.3 – I-V characteristics of the 2T1R structure.

leads the voltage difference across the RRAM to decrease. The positive feedback continues until the V_{RRAM} reaches the V_{set} of the RRAM, i.e., the memory cannot switch anymore. At this point, I_{ds}, V_{DS1} and V_{DS2} reach their peak values. Note that during the programming phase, V_{GS1} is increasing as the source voltage of transistors **N1**, V_{TE}, is decreasing, but it is still smaller than V_{GS2}. The difference in V_{GS} causes a V_{DS} gap because V_{DS1} has to be larger than V_{DS2} in order to drive the same current. Therefore, transistor **N1** may work in deep linear region or even saturation region while transistor **N2** has to work in linear region, causing the programming current to be much lower than what saturated transistors can offer.

Boosting V_{prog} can reduce the difference between V_{DS1} and V_{DS2}, improving the driving strength of transistors. Its effort will be studied by electrical simulations.

3.2.3 Physical Design Difficulties

Typically, in digital circuit designs, the bulks of n-type transistors are connected to GND, as shown in Fig. 3.4(a). However, the regular bulk connections for the 2T1R structure causes serious body effects. In a set process where $V_{progTE}{\approx}V_{prog}$ and $V_{progBE}{\approx}GND$, the $V_{SB} = V_{S1}$ of transistor **N1** in Fig. 3.4(a) is larger than $V_{set} = V_{S1} - V_{D2}$, which leads to a high threshold voltage of transistor **N1** and reduces its driving strength. Note that the V_{SB} of transistor **N2** is negligible due to the V_{DS3} and V_{DS4} and its driving strength is reduced as well. Similar conclusion can be drawn in a reset process where $V_{progTE}{\approx}GND$ and $V_{progBE}{\approx}V_{prog}$.

To alleviate the serious body effect, a symmetric bulk connection can be envisaged as shown in Fig. 3.4(b). When $V_{progTE}{\approx}V_{prog}$ and $V_{progBE}{\approx}GND$, the V_{SB} of transistor **N1** equals to V_{DS} which is smaller than in the previous case and improves the driving strength. The V_{SB} of transistor **N2** is strictly zero, totally eliminating the body effect. Similar conclusion can be drawn when $V_{progTE}{\approx}GND$ and $V_{progBE}{\approx}V_{prog}$.

However, when a symmetric bulk is implemented with a single-well technology as shown in Fig.

3.4(c), the substrate is connected to two voltage sources $V_{progTE} \approx V_{prog}$ and $V_{progBE} \approx GND$, resulting in a high leakage current I_{sub}. Besides, the junction diode at the source of transistor **N1** is positively biased, introducing another high leakage current I_{diode}. I_{sub} can be reduced to zero with a triple-well technology as shown in Fig. 3.4(d), but I_{diode} remains a concern. In short, there exist serious problems in connecting the bulks of 2T1R structure, limiting its feasibility from a physical design perspective.

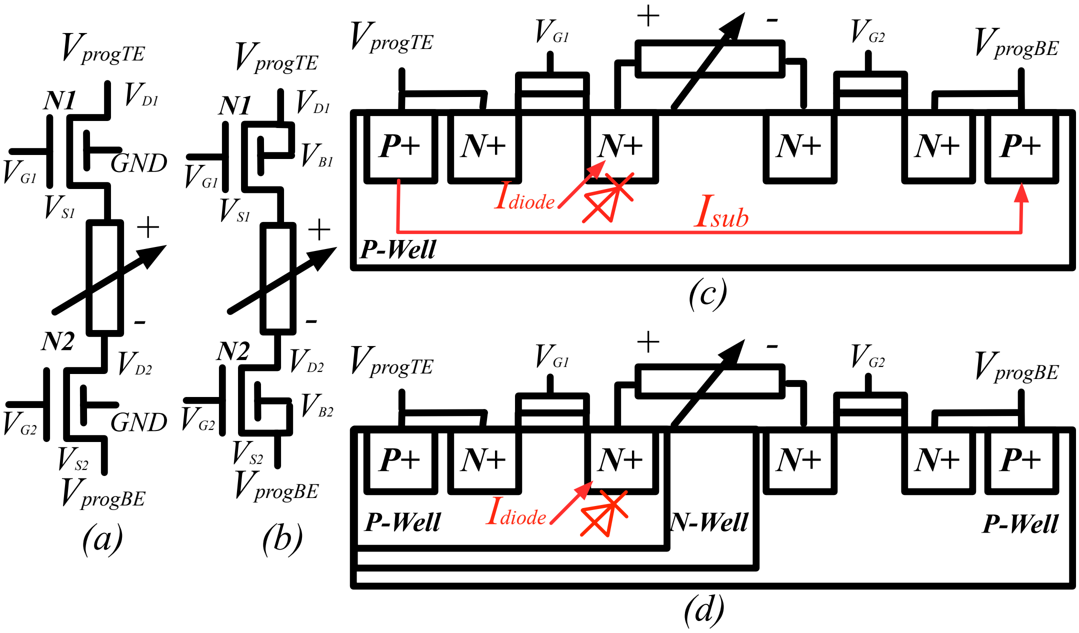


Figure 3.4 – (a) Asymmetric bulk management of the 2T1R structure; (b) Symmetric bulk management of the 2T1R structure; (c) Single well application of layout; (d) Triple well application of layout.

3.2.4 Area Estimation

We estimate the area of the programming structures in terms of minimal size transistors. While we only considered the set process, it is worth noticing that in the 2T1R structure, the same transistors **N1** and **N2** are used in reset process as well. Typically, the reset current is not the same as the set current [1]. To be applicable in both set and reset, the size of transistors **N1** and **N2** should be determined by the largest of set/reset currents. Assume $W_{prog,set}$ and $W_{prog,reset}$ are the transistor sizes required for the set and reset operations, respectively. In the context of a memory bank, we assume that a driving inverter for a BL is shared by N 2T1R structures:

$$\begin{cases} 2W_{prog,set} + 2\cdot(1+\beta\gamma)W_{inv}/N, & I_{set} \geq I_{reset} \\ 2W_{prog,reset} + 2\cdot(1+\beta\gamma)W_{inv}/N, & I_{set} < I_{reset} \end{cases} \tag{3.3}$$

where β is the ratio of p-type and n-type transistors and γ is the penalty factor for the area overhead of the P-Well of p-type transistors. W_{inv} is the size of driving inverters. When the set current is larger than the reset current, the area is determined by $W_{prog,set}$. When the reset current is larger than the set current, the area is determined by $W_{prog,reset}$. In this case, during the **set** process, transistor **N1** and **N2** should be under-drived by reducing V_{G1}, V_{G2} and V_{prog} to respect the current compliance. Unlike the $W_{prog,set}$, a large $W_{prog,reset}$ does not contribute to a high R_{HRS}. In others words, a large $W_{prog,reset}$ does not improve the performance as the $W_{prog,set}$ does. Therefore, when $I_{set} < I_{reset}$, the area consumed by a large $W_{prog,reset}$ is not directly contributing to a performance improvement.

3.2.5 Electrical Simulations

First, we validate our theoretical intuitions by presenting the SPICE transient analysis of the 2T1R structure. Then, we show the SPICE results of the V_{DS} and programming current I_{ds} of the 2T1R structure.

Transient Analysis

Fig. 3.5 illustrates current and voltage waveforms of the 2T1R structure during a set process. After the transistors are turned *on*, a voltage difference V_{MAX} between the RRAM electrodes is applied, initiating the set transition on the memory. The reduction on the resistance of the RRAM leads to an increase in I_{ds}. To support the growing I_{ds}, the V_{DS} of transistors have to increase, leading to V_{TE} is decreasing and V_{BE} is increasing. The RRAM stays in programming phase until $V_{TE} - V_{BE}$ reaches the threshold voltage V_{set}.

V_{DS} of Transistors N1 and N2

Fig. 3.6 shows the trend of V_{DS} in a 2T1R structure by sweeping W_{prog} and V_{prog}, where W_{inv} is 20 in order to keep V_{DS3} and V_{DS4} negligible. The V_{DS} difference reaches 0.65V when $V_{prog} = 2.5V$ on average. Boosting V_{prog} can reduce the V_{DS} difference down to 0.5V. A larger V_{prog} can increase the V_{DS2} by 2.8×. Fig. 3.7 depicts the trend of V_{DS} in 2T1R structure by sweeping W_{prog} and W_{inv}, where V_{prog} is 3.0V. Increasing W_{inv} can effectively reduce the V_{DS} gap by 15%.

Programming Current I_{ds}

The achievable programming currents I_{ds} are determined by V_{DS}. A high V_{prog} can increase the V_{DS}, as explained in Section 3.2.2. Fig. 3.8(a) illustrates that for the same W_{inv}, we can improve 3.4× I_{ds} by boosting V_{prog} from 2.5V to 3.0V on average. W_{inv} is another important factor that influences the I_{ds}. A large W_{inv} can reduce V_{DS3} and V_{DS4} while increase V_{DS1} and V_{DS2}. As shown in Fig. 3.8(b), a large W_{inv}, such as 20, leads to a 3.8× higher I_{ds} than the

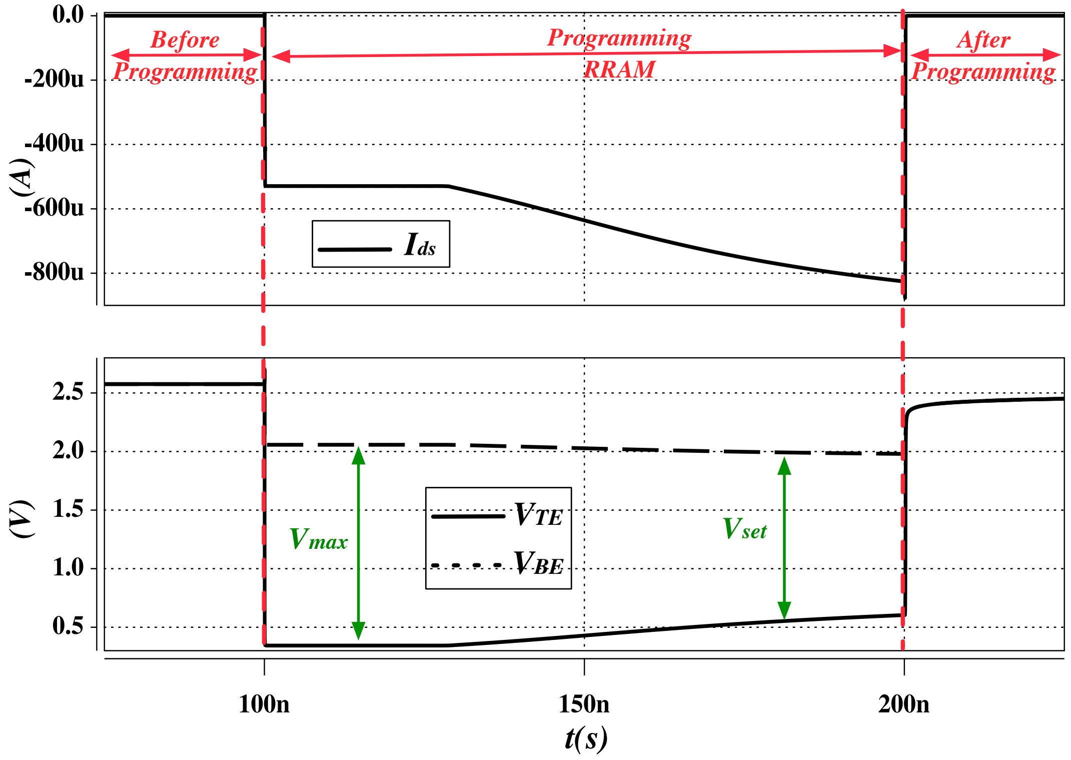


Figure 3.5 – Transient analysis on voltages and current in the 2T1R structure during a set process ($W_{prog} = 5$, $V_{prog} = 3.0V$, $W_{inv} = 20$, 1 $W_{prog} = 320nm$).

smallest $W_{inv} = 1$ on average. In short, boosting V_{prog} is an efficient method in improving I_{ds}, which avoids the use of large transistors. A large W_{inv} (i.e., =20) must be applied to avoid a serious degradation on I_{ds}.

3.2.6 Discussion About Limitations

From theoretical analysis and electrical simulations, we see five major limitations of 2T1R structure:
(1) its current density is low due to the intrinsic low V_{DS2};
(2) its bulk connections lead to a high leakage current;
(3) its current density is weakened by a small W_{inv};
(4) its area is bounded by the maximum of $W_{prog,set}$ and $W_{prog,reset}$, which is not efficient when I_{reset} is large.
(5) it is not manufacturable due to the layout issues shown in Section 3.2.3. Hence, in the rest of the paper, we only refer to it when comparing the current density.

To address the listed limitations (1), (2) and (5), we propose 2TG1R programming structures in Section 3.3.

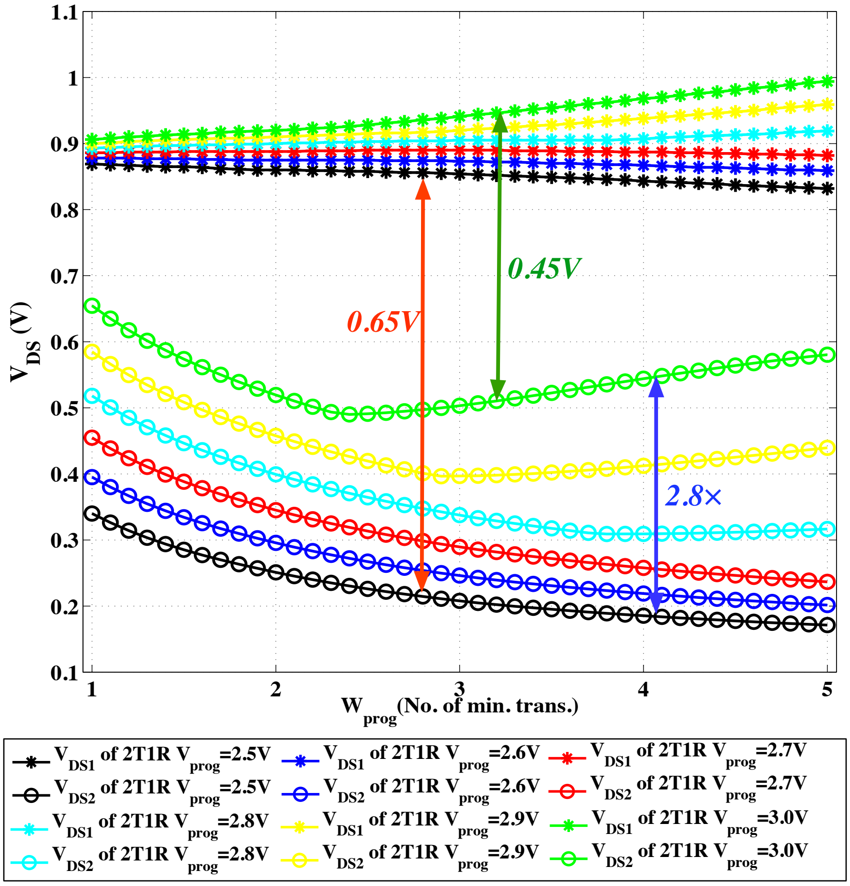


Figure 3.6 – V_{DS1} and V_{DS2} in 2T1R structure under diverse V_{prog} ($W_{inv} = 20$)

3.3 2TG1R Programming Structure

In this section, we improve the previous 2T1R circuit by replacing the n-type transistors and propose a 2TG1R programming structure. The 2TG1R circuit, comprising of four transistors, increases the current density significantly and overcomes the bulk management problem. The solution is validated using the electrical simulations.

3.3.1 2TG1R Circuit Structure

Replacing the n-type transistors in 2T1R structure with transmission gates is a solution to the bulk management and driving strength. As shown in Fig. 3.9, the bulks of the n-type and p-type transistors (in total 4 transistors) are connected respectively to the highest and lowest potentials, similarly to common digital design practice, removing the bulk leakage and body effects. The driving inverters are still required to provide the voltage levels of V_{progTE} and V_{progBE} during the programming phases. Whatever in a set or reset process, there always exist a p-type transistor and a n-type transistor whose $V_{SB} = 0$. Therefore, these two transistors

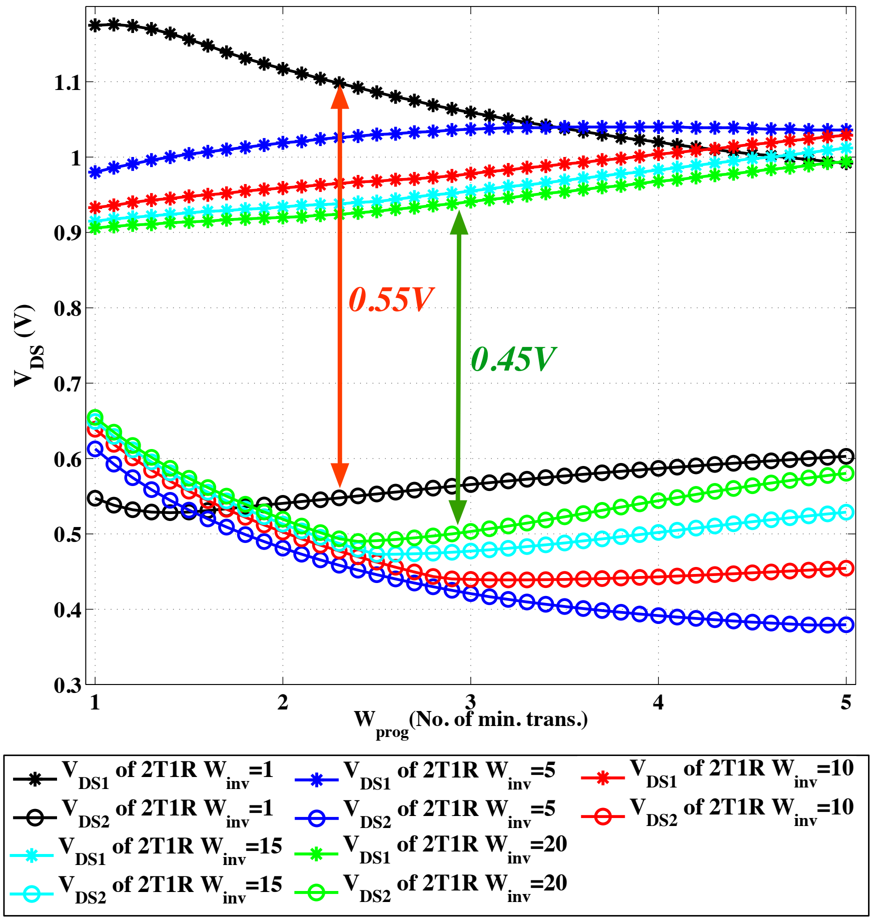


Figure 3.7 – V_{DS1} and V_{DS2} in 2T1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$)

whose $V_{SB} = 0$ can provide higher current than 2T1R structure. Although the other two transistors (weak p-type and weak n-type) suffer serious body effects, they still contribute to the currents. Hence, the total current offered by 2TG1R structure is higher than 2T1R structure.

3.3.2 Area Estimation

We consider the area of a 2TG1R structure in the context of a memory bank as well. By considering the area of two p-type transistors, the area of a 2TG1R structure is:

$$\begin{cases} 2\cdot(1+\beta\gamma)W_{prog,set} + 2\cdot(1+\beta\gamma)W_{inv}/N, & I_{set} \geq I_{reset} \\ 2\cdot(1+\beta\gamma)W_{prog,reset} + 2\cdot(1+\beta\gamma)W_{inv}/N, & I_{set} < I_{reset}. \end{cases} \tag{3.4}$$

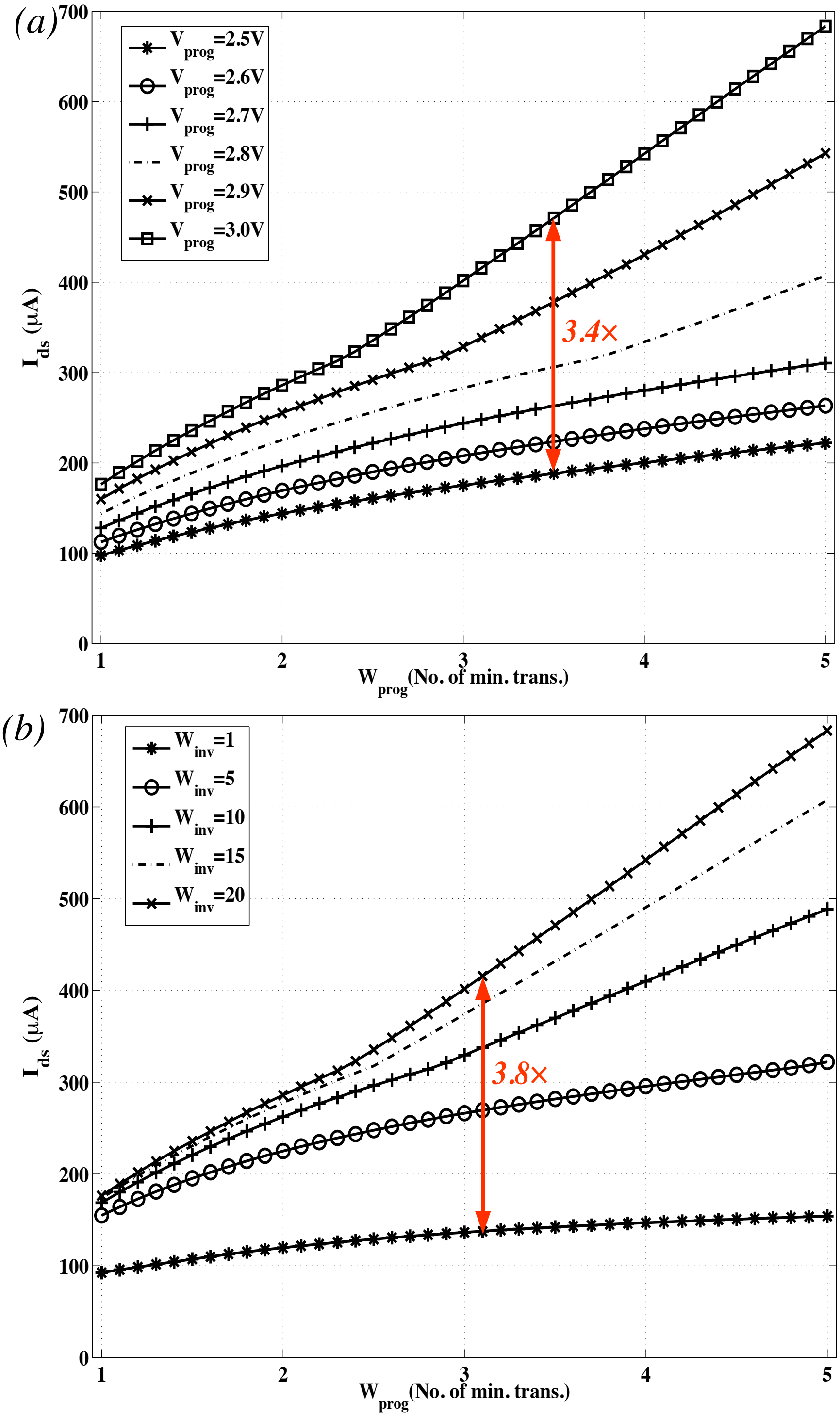


Figure 3.8 – (a) I_{ds} in 2T1R structure under diverse V_{prog} ($W_{inv} = 20$); (b) I_{ds} in 2T1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$)

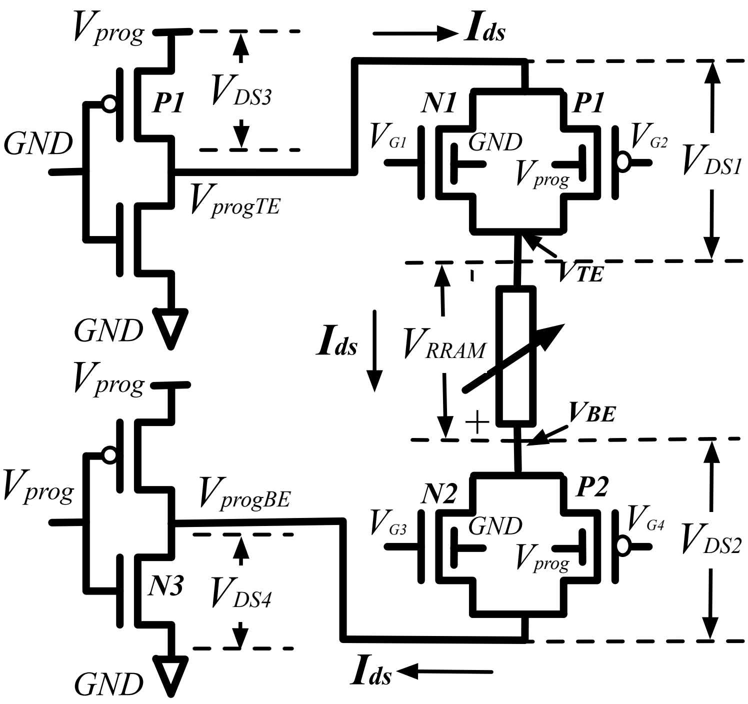


Figure 3.9 – A 2TG1R programming structure extracted from system-level implementations in Fig. 3.1

In summary, the area of 2TG1R circuit is still bounded to the largest of $W_{prog,set}$ and $W_{prog,reset}$. When $I_{set} < I_{reset}$, area investment on $W_{prog,reset}$ does not bring any improvement on performance. This is extremely inefficient when $W_{prog,reset}$ is large. A 2TG1R circuit leads to a even larger area overhead than 2T1R structure due to the use of p-type transistors.

3.3.3 Electrical Simulations

In this section, we show the electrical simulation results of 2TG1R structure. We focus on the improvements on V_{DS} and I_{ds} of 2TG1R structure, compared to the baseline 2T1R element.

Transient Analysis

Basically, the waveforms of the transient analysis on a 2TG1R are the same as 2T1R structure. The only difference lies in the slope rate of V_{TE} and V_{BE} during the programming phase. In 2TG1R, V_{TE} decreases at the same rate as V_{BE} increases. In the other word, V_{DS1} and V_{DS2} in 2TG1R grow at the same rate.

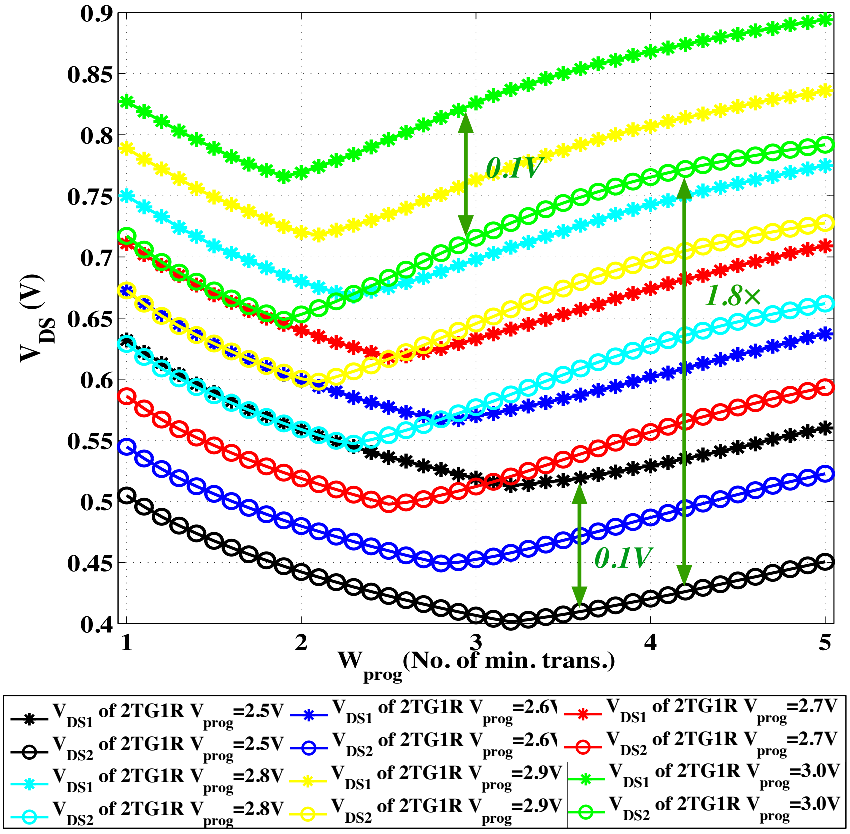


Figure 3.10 – V_{DS1} and V_{DS2} in 2TG1R structure under diverse V_{prog} ($W_{inv} = 20$);

V_{DS} Gap Improvement

As shown in Fig. 3.10 and Fig. 3.11, a 2TG1R structure reduces the V_{DS} gap by 5×, compared to a 2T1R structure. Like the 2T1R structure, boosting V_{prog} can improve V_{DS2} of 2TG1R by 1.8×. However, a 2TG1R still requires a large $W_{inv} = 20$ to avoid the degradation on V_{DS} gap, coming from a non-negligible V_{DS3} and V_{DS4}. When $W_{inv} = 1$, the V_{DS} gap degrades by 2×.

Programming Current I_{ds}

Boosting V_{prog} and W_{inv} achieves a similar effect on the I_{ds} than on the 2T1R structure. Boosting V_{prog} can improve I_{ds} of 2TG1R by 1.8×. Increasing W_{inv} from 1 to 20 can improve I_{ds} of 2TG1R by 4.3×. The I_{ds} of 2TG1R is 1.2× higher than 2T1R structure.

3.3.4 Summary: Advantages and Limitations

From theoretical analysis and electrical simulations, 2TG1R structures have the following advantages over 2T1R structure:

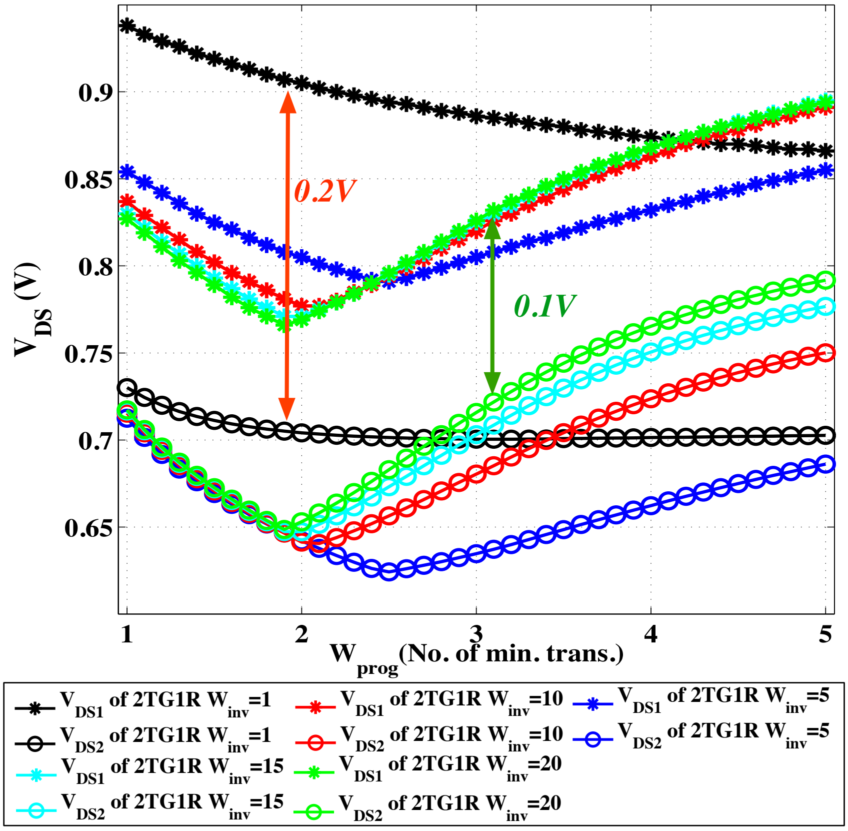


Figure 3.11 – V_{DS1} and V_{DS2} in 2TG1R structure under diverse W_{inv} ($V_{prog} = 3.0V$). (1 $W_{prog} = 320nm$)

(1) the V_{DS} gap is reduced by 5×, contributing to a 1.2× improvement in I_{ds};
(2) its bulk connections are regular, removing the bulk leakage and body effects.
However, the 2TG1R still shares two limitations with the 2T1R structure:
(1) large driving inverters are still needed to avoid current density degradation;
(2) the area is still constrained by the worse case of $W_{prog,set}$ and $W_{prog,reset}$, which is inefficient when $I_{set} < I_{reset}$ and $W_{prog,reset}$ is large.

Note that the 2TG1R programming structure overcomes the limitations (1), (2) and (5) of the 2T1R programming structure (See Section 3.2.6). To fully address the limitations of the 2T1R and the 2TG1R programming structures, we propose 4T1R programming structures in Section 3.4.

3.4 4T1R Programming Structure

In this section, we propose a 4T1R programming structure able to alleviate the addressed limitations of 2T1R programming structures. We first introduce the circuit design and conduct theoretical analysis. Then, we compare the 4T1R structure with 2T1R and 2TG1R structures using electrical simulations.

3.4.1 4T1R Circuit Structure

Fig. 3.12(a) illustrates the schematic of the 4T1R structure which consists of two p-type transistors **P1** and **P2** and two n-type transistors **N1** and **N2**. The sources of the transistors in the 4T1R structure are directly connected to the voltage supplies, eliminating the driving inverters used with the 2T1R and 2TG1R solutions. The programming phase is launched by appropriately biasing the gates of the transistors. In a **set** process, the transistors **P1** and **N2** are turned *on* while the transistor **P2** and **N1** are turned *off*, applying a positive programming voltage between V_{TE} and V_{BE}, as shown in Fig. 3.12(b). Conversely, when the transistors **P2** and **N1** are turned *on* and the transistors **P1** and **N2** are turned *off*, applying a negative voltage between V_{TE} and V_{BE}, a **reset** process is operated. When the programming segment is finished, all the transistors are turned *off*. The 4T1R structure is compatible to the system-level implementations in Fig. 3.1. In a scan-chain organization, V_{G1}, V_{G2}, V_{G3}, V_{G4} can be connected to $\overline{Q_0}$, Q_0, $\overline{Q_1}$, Q_1, respectively. In a memory bank organization, V_{G1}, V_{G2}, V_{G3}, V_{G4} can be connected to $\overline{BL}[0]$, $WL[2]$, $\overline{BL}[2]$, $WL[1]$, respectively.

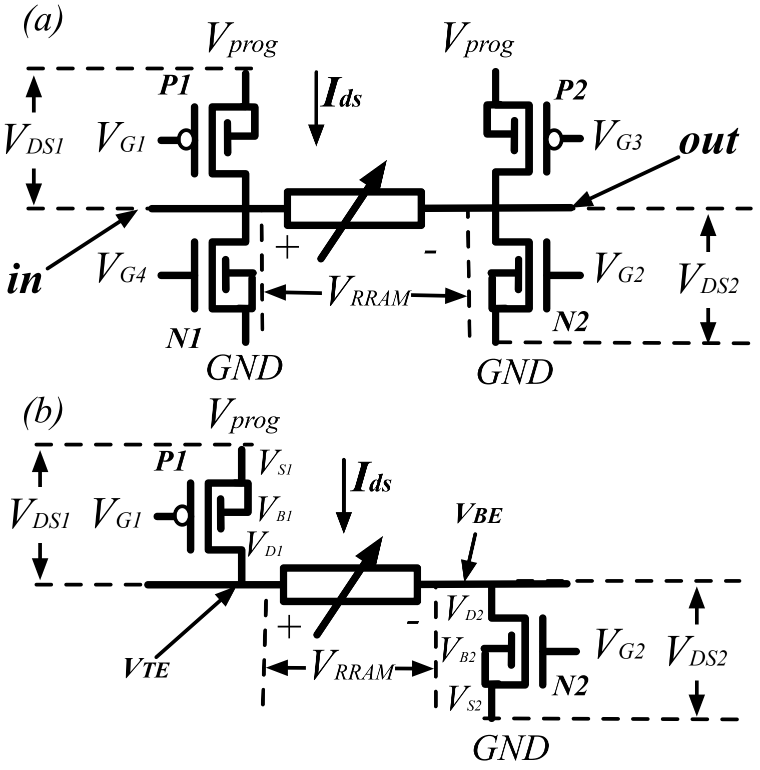


Figure 3.12 – (a) The proposed 4T1R structure (b) Extracted 4T1R structure in a **set** process

3.4.2 Theoretical Analysis on I-V Characteristics

We first focus on the set process (Fig. 3.12(b)). By applying Kirchhoff Circuit Laws, we can express the following relationships:

$$\begin{cases} I_{ds} = f(V_{GS1}, V_{DS1}) = f(V_{GS2}, V_{DS2}) \\ V_{RRAM} = I_{ds} R_{RRAM} \\ V_{prog} = V_{DS1} + V_{DS2} + V_{RRAM}. \end{cases} \tag{3.5}$$

V_{DS1} and V_{DS2} represent the drain-to-source voltages of transistors **P1** and **N2**, respectively. V_{GS1} and V_{GS2} represent the gate-to-source voltages of transistors **P1** and **N2**, respectively. Note that in the 4T1R structure, the sources of the transistors are connected to constant voltage supplies, giving stable V_{GS} during the programming phase. We can set $V_{GS1} = V_{GS2}$. According to the basic transistor model shown in (3.2), when $V_{GS1} = V_{GS2}$, we can find:

$$V_{DS} = V_{DS1} = V_{DS2}. \tag{3.6}$$

Combining (3.5) and (3.6), we can reach

$$I_{ds} = \frac{V_{prog}}{R_{RRAM}} - \frac{2}{R_{RRAM}} V_{DS}. \tag{3.7}$$

We plot the I-V curves of (3.2) and (3.7) in Fig. 3.13(a). The crossing points **P** $(\sim 0, V_{prog}/R_{HRS})$ and **Q** $((V_{prog} - V_{set})/2, I_{set})$ in Fig. 3.13(a) represent the starting and end points of a **set** procedure. From **P** to **Q**, V_{DS} gradually increases to provide a large I_{ds}. On the other side, R_{RRAM} decreases as I_{ds} grows. The increment of I_{ds} further induces a increase in V_{DS} and a decrease in R_{RRAM}. When V_{RRAM} reaches the threshold programming voltage V_{set} of the RRAM, the **set** process stops (point **Q** in Fig. 3.13(a)). We can determine $V_{DS,Q} = (V_{prog} - V_{set})/2$ and $I_{ds,Q} = V_{set}/R_{RRAM,Q}$ at the ending point **Q**. Note that $R_{RRAM,Q}$ is the programmed R_{LRS} of the RRAM while $R_{RRAM,P}$ is R_{HRS} of the RRAM.

In the reset process, let V_{reset} be the threshold programming voltage of the RRAM. The I-V curve of reset process could be different from set process because of the technological constraints (V_{reset} and I_{reset}). Fig. 3.13 illustrates the three cases that could happen during a reset process. Similar to the analysis in set process, we define the operating point **P** $(\sim 0, V_{prog}/R_{HRS})$ as the ending point of a reset process and the operating point **N** $((V_{prog} - V_{reset})/2, I_{reset})$ as the starting point of a reset process. Fig. 3.13(a) is applicable to all the conditions where $V_{set} \geq V_{reset}, I_{set} \geq I_{reset}$, where point **N** overlaps point **Q**. In this case, the reset process is an exact reverse trace of the set process. Fig. 3.13(b) covers the most difficult condition: $V_{set} < V_{reset}$ and $I_{set} < I_{reset}$. Compared to the set, the starting point **N** of the reset process is most stringent. As a result, a $W_{prog,reset}/V_{GS}$,reset larger than $W_{prog,set}/V_{GS}$,set will have to be used to reach point **N**. Note that Fig. 3.13(b) is applicable for other conditions where either $V_{set} < V_{reset}$ or $I_{set} < I_{reset}$ happens. Finally, Fig. 3.13(c) covers another case where

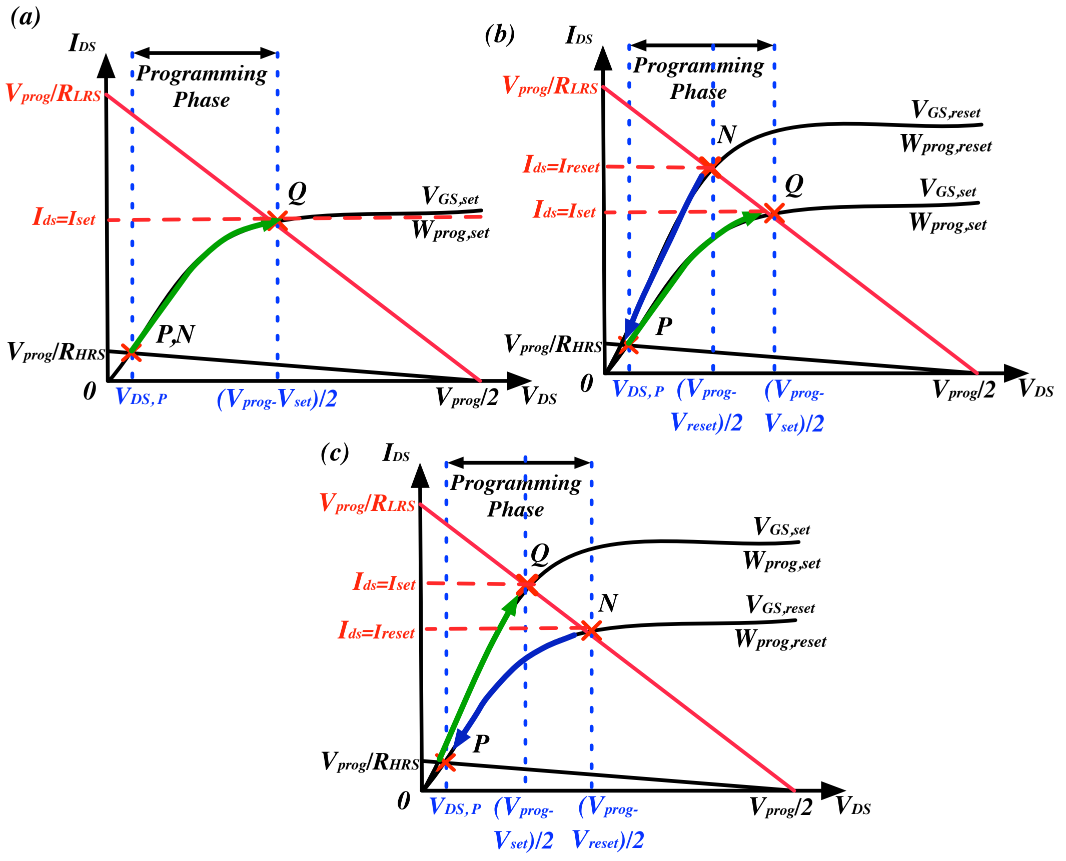


Figure 3.13 – I-V characteristics of the 4T1R structure: (a) V_{set}=V_{reset}; (b) $V_{set} < V_{reset}$ or $I_{set} < I_{reset}$; (c) $V_{set} > V_{reset}$ or $I_{set} > I_{reset}$.

$V_{set} > V_{reset}$ and $I_{set} > I_{reset}$, while the case shown in Fig. 3.13(a) still applies in the case, it would result in an oversizing for the reset process. In the case of Fig. 3.13(c), the starting point of reset process **N** leads to a smaller $W_{prog,reset}/V_{GS}$,reset than $W_{prog,set}/V_{GS,set}$.

Note that Fig. 3.13 reveals another shortcoming of 2T1R and 2TG1R structures, which use the same programming transistors for both the set and the reset processes. Due to this fact, they must be sized according to the worse case $max\{W_{prog,set}, W_{prog,reset}\}$. Hence, for the conditions illustrated in Fig. 3.13(b)(c), the 2T1R and 2TG1R structures have to use two different V_{GS} for the set and the reset processes ($V_{GS,set} \neq V_{GS,reset}$). When two different V_{GS} are needed, the system-level implementations in Fig. 3.1 will require additional circuitry for generating controlling signals, i.e., $WL[1]$ and $WL[2]$ should have three voltage levels: $V_{GS,set}$, $V_{GS,reset}$ and GND.

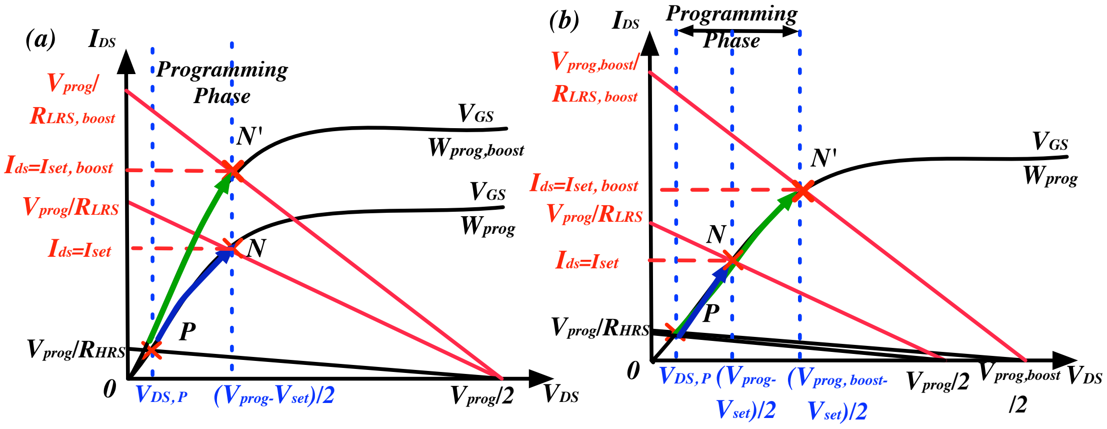


Figure 3.14 – I-V characteristics of the 4T1R structure during set process when: (a) Boosting W_{prog}; (b) Boosting V_{prog}.

3.4.3 Current Density Boosting Methodologies

V_{prog} and W_{prog} are the two controllable parameters for circuit designers to boost $I_{ds,Q}$. In this part, depending on the working regions of the crossing point **Q**, we investigate the boosting methodologies for $I_{ds,Q}$ by tuning V_{prog} and W_{prog}.

Linear Region

When the transistors work in the linear region at the crossing point **Q**, we can obtain the following equations:

$$\begin{cases} I_{ds,Q} = k_n \frac{W_{prog}}{L}[(V_{GS}-V_T)V_{DS,Q} - \frac{1}{2}{V_{DS,Q}}^2] \\ V_{DS,Q} < V_{GS} - V_T \\ I_{ds,Q} = (V_{prog} - 2V_{DS,Q})/R_{RRAM,Q} \\ V_{DS,Q} = (V_{prog} - V_{set})/2 \end{cases} \tag{3.8}$$

From (3.8), we can determine $I_{ds,Q}$:

$$\begin{cases} I_{ds,Q} = \frac{k_n W_{prog}[(V_{GS}-V_T)(V_{prog}-V_{set})-\frac{1}{4}(V_{prog}-V_{set})^2]}{L} \\ R_{RRAM,Q} = \frac{2L \cdot V_{set}/W_{prog}}{k_n[(V_{GS}-V_T)(V_{prog}-V_{set})-\frac{1}{4}(V_{prog}-V_{set})^2]} \\ V_{prog} < 2(V_{GS} - V_T) + V_{set} \end{cases} \tag{3.9}$$

In this case, both W_{prog} and V_{prog} can influence $I_{ds,Q}$. By increasing W_{prog} and V_{prog}, $I_{ds,Q}$ can be magnified, leading to a higher current density.

Fig. 3.14(a) and (b) shows the I-V characteristics of 4T1R programming structure working in linear region when W_{prog} and V_{prog} are boosted respectively. As depicted in Fig. 3.14(a), boosting W_{prog} to $W_{prog,boost}$ leads to that the operating point during set process following another I-V curve, highlighted green in Fig. 3.14(a). Hence, the ending point of set process shifts from N to N' and a higher programming current $I_{set,boost}$ can be achieved, contributing to a reduction in R_{LRS}. Fig. 3.14(b) shows the shift of operating point during set process when V_{prog} is boosted. Increasing V_{prog} to $V_{prog,boost}$ leads to V_{DS} of transistors grows from $V_{prog} - V_{set})/2$ to $V_{prog,boost} - V_{set})/2$. Therefore, we see in Fig. 3.14(b) that the ending point of set process shifts from N to N', contributing to a higher programming current $I_{set,boost}$ than I_{set}. As a result, the achieved R_{LRS} is reduced to $R_{LRS,boost}$.

Saturation Region

When the crossing point **Q** lies in the saturation region, we obtain the following equations.

$$\begin{cases} I_{ds,Q} = k_n \frac{W_{prog}}{L}(V_{GS} - V_T)^2 \\ V_{DS,Q} \geq V_{GS} - V_T \\ I_{ds,Q} = (V_{prog} - 2V_{DS,Q})/R_{RRAM,Q} \\ V_{DS,Q} = (V_{prog} - V_{set})/2 \end{cases} \tag{3.10}$$

From (3.10), we express $I_{ds,Q}$ as follows:

$$\begin{cases} I_{ds,Q} = \frac{k_n W_{prog}(V_{GS}-V_T)^2}{2L} \\ R_{RRAM,Q} = \frac{2L\cdot V_{set}/W_{prog}}{k_n(V_{GS}-V_T)^2} \\ V_{prog} > 2(V_{GS} - V_T) + V_{set} \end{cases} \tag{3.11}$$

In the saturation region, only W_{prog} can boost $I_{ds,Q}$.

Equations (3.9) and (3.11) show that adjusting the W_{prog} and V_{prog} are the two methods in boosting $I_{ds,Q}$. The W_{prog} is linearly proportional to $I_{ds,Q}$ whatever the working region is. When V_{prog} is bound to the linear region, it has a quadratic impact on $I_{ds,Q}$. After V_{prog} meets the need of the saturation region, it has no impact on $I_{ds,Q}$. Therefore, to enhance the current density in the linear region, boosting W_{prog} is effective but requires a large transistor size, while boosting V_{prog} does not increase the transistor size and should be considered as a first choice. When V_{prog} increases, the transistors move from the linear region to the saturation region. In the saturation region, boosting W_{prog} is the only boosting method. Referring to the examples in Fig. 3.14(a)(b), boosting W_{prog} can still shift the I-V curve and lead to a higher programming current even in saturation region, while boosting V_{prog} leads to no difference in programming current since the ending point of set process always lies in the saturation region of the same I-V curve. Similar conclusions can be found for reset process.

Constraints from Breakdown Limitations

As addressed in Section 3.4.3, boosting V_{prog} can increase $I_{ds,Q}$. However, there exists a breakdown voltage V_{break} for the source-to-drain voltage V_{DS} of a transistor that provides an upper-bound. In this section, we discuss the range of V_{prog} that the 4T1R structure can safely afford.

The V_{DS} of all the transistors (**P1,P2,N1,N2**) in Fig. 3.12(a) should satisfy to:

$$\begin{cases} (a): max\{V_{prog} - V_{TE}\} = max\{V_{DS1}\} \leq V_{break} \\ (b): max\{V_{TE}\} = V_{prog} - min\{V_{DS1}\} \leq V_{break} \\ (c): max\{V_{prog} - V_{DS2}\} = V_{prog} - min\{V_{DS2}\} \leq V_{break} \\ (d): max\{V_{BE}\} = max\{V_{DS2}\} \leq V_{break} \\ (e): max\{V_{DS1}\} = max\{V_{DS2}\} = (V_{prog} - V_{set})/2 \\ (f): min\{V_{DS1}\} = min\{V_{DS2}\} = V_{DS,P}. \end{cases} \tag{3.12}$$

Equations 3.12(a)(b)(c)(d) consider the breakdown limitations of V_{DS} of the transistors **P1**,**N1**,**P2**, **N2**, respectively. Equations 3.12(e)(f) are derived from the range of V_{DS} of the transistors **P1**,**N2** in Fig. 3.13. As illustrated in Fig. 3.13, $max\{V_{DS1}\}$ and $max\{V_{DS2}\}$ happen when the RRAM is in LRS (point **Q**), while $min\{V_{DS1}\}$ and $min\{V_{DS2}\}$ happen when the RRAM is in HRS (point **P**). $V_{DS,P}$ can be calculated by applying the transistor model (3.2) to the crossing point **P** in Fig. 3.13:

$$\begin{cases} I_{ds}' = k_n \frac{W_{prog}}{L} [(V_{GS} - V_T)V_{DS,P} - V_{DS,P}^2/2] \\ I_{ds}' = (V_{prog} - 2V_{DS,P})/R_{RRAM,P}. \end{cases} \tag{3.13}$$

Note that here, we only consider the linear region because typically the $R_{RRAM,P}$ is large enough to let the V_{DS} of transistors **P1,N2** less than V_{GS}.

Solving (3.12) and (3.13), we find that the programming voltage V_{prog} constrained by:

$$\begin{cases} P1\&N2: V_{prog} \leq 2V_{break} - V_{set} \\ P2\&N1: V_{prog} \leq V_{break} + V_{DS,P} \\ V_{DS,min} = \frac{2}{R_{RRAM,P}} k_n W_{prog}/L + (V_{GS} - V_T) - \sqrt{\Delta} \\ \Delta = [2 + k_n \frac{W_{prog}}{L}(V_{GS} - V_T)]^2 R_{RRAM,P} \\ -2V_{prog} k_n \frac{W_{prog}}{L} R_{RRAM,P} \end{cases} \tag{3.14}$$

Assume that $R_{RRAM,P}$ of RRAM is large, $V_{DS,P}$ is approximately zero. In such case, the upper-bound of V_{prog} is tied to $V_{prog} \leq min\{2V_{break} - V_{set}, V_{break}\}$.

3.4.4 Area Estimation

In a 4T1R structure, V_{prog} and GND are directly connected to power supplies. Compared to the 2T1R and 2TG1R structures, no driving inverters are needed. The area of a 4T1R structure is the sum of the sizes of transistors used in **set** and **reset** process:

$$2 \cdot (1 + \beta\gamma) W_{prog,set} + 2 \cdot (1 + \beta\gamma) W_{prog,reset}. \tag{3.15}$$

When $W_{prog,reset}$ is much larger than $W_{prog,set}$, all the transistors in the 2T1R and 2TG1R structures have to be as large as $W_{prog,reset}$ while the 4T1R structure can use smaller transistor sizes for set process. Hence, the 4T1R structure brings more flexibilities in transistor sizes than the 2T1R and 2TG1R structures.

3.4.5 Benefits of 4T1R structures

In this section, we compare the 2T1R, 2TG1R and 4T1R structures in terms of three metrics: V_{DS} symmetry, I_{ds} current, area, delay and power.

V_{DS} Gap Reduction

In Fig. 3.15, we compare the V_{DS} of 2T1R, 2TG1R and 4T1R structures, where $W_{inv} = 20$ is considered for the 2T1R and 2TG1R structures. The V_{DS} difference of 2TG1R and 4T1R structures are 75% smaller than 2T1R structure, because they employ p-type transistors to propagate V_{prog}, as explained in Section 3.4.2. Note that if a small W_{inv}, i.e., $W_{inv} = 1$, rather than $W_{inv} = 20$ is used, the V_{DS} gap of the 2TG1R structure would be larger than 4T1R.

Improvement on Programming Current I_{ds}

As a result, the driving current shown in Fig. 3.16 of 4T1R structures is the best of the three solutions. I_{ds} of the 4T1R is 1.1× higher than 2TG1R structure, while 2TG1R improves I_{ds} by $1.3\times$, compared to 2T1R structure. Note that when $V_{prog} = 2.5V$, the improvement in driving current of 4T1R and 2TG1R structures are more significant than $V_{prog} = 3.0V$. When we investigate the driving current density of 2T1R, 2TG1R and 4T1R structures in Fig. 3.17, 4T1R structure is the best, which is 1.1× higher than 2TG1R structure and 1.4× higher than 2T1R structure. Note that the current density of 2T1R and 2TG1R are deceasing when W_{prog} increases, while the current density of 4T1R is increasing. When a larger W_{prog} is used, W_{inv} has to be increased to alleviate the impact of V_{DS3} and V_{DS4}. If W_{inv} does not grow as W_{prog}, V_{DS3} and V_{DS4} becomes non-negligible, resulting a degrading current density. Hence, without re-sizing W_{inv}, when W_{prog} increases, 2T1R and 2TG1R provides a weaker I_{ds} than a 4T1R scheme. As a conclusion, 4T1R structure is more efficient in driving current than 2T1R and 2TG1R structures.

Area, Delay and Power

In this part, we evaluate the area, delay and power of SRAM-based transmission gate and 2TG1R, 4T1R RRAM-based programming structures. The area of RRAM-based multiplexers is estimated with (3.4) and (3.15), where we assume $N = 32$, a typically size for a modern memory bank [132]. The area model in [125] is used to estimate the transistor area. We consider the propagation delay as the delay of the multiplexers, i.e., the signal delay from in to out in Fig.

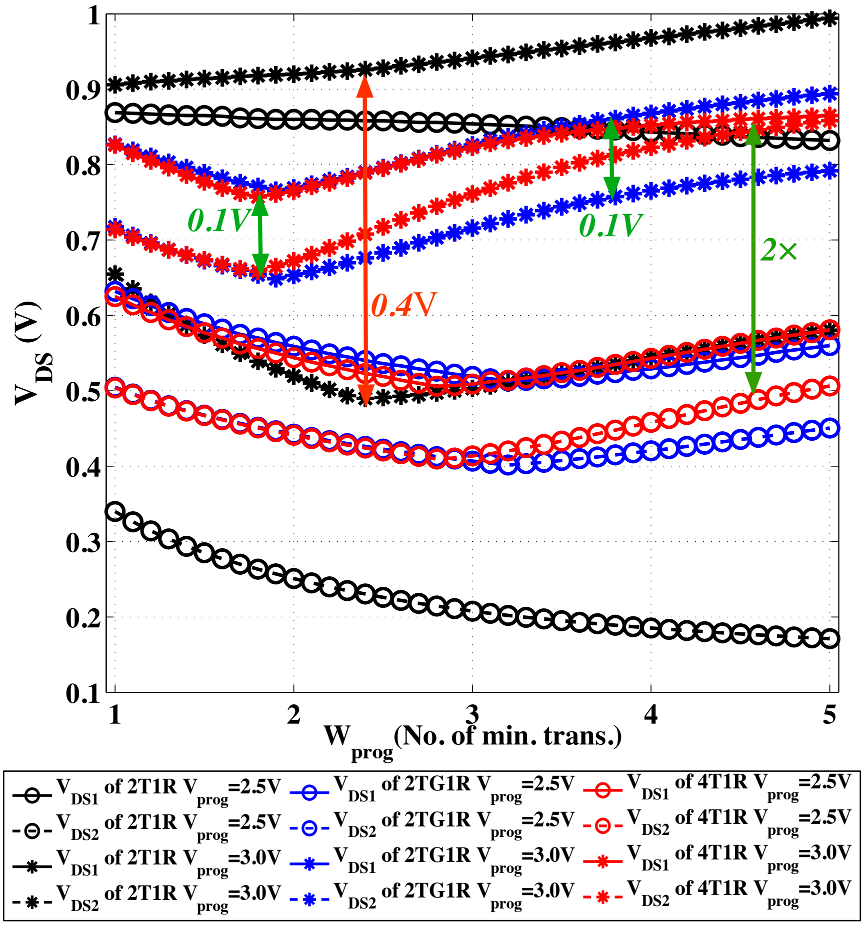


Figure 3.15 – Comparison on V_{DS} of programming transistors under diverse W_{prog} and V_{prog} in 2T1R, TG-based 2T1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$)

3.12(a). To evaluate the switching energy, we assume that 50% of the inputs have switching activities, which is representative in FPGAs [125]. Because I/O transistors are used in 2TG1R and 4T1R structure while SRAM-based circuit use standard transistors, we consider that I/O transistors have twice area than standard transistors.

Fig. 3.18 and Fig. 3.19 illustrate the area-delay product and the power-delay product of 2TG1R and 4T1R structures respectively, when different target R_{LRS} and V_{prog} are considered. A low R_{LRS} requires large programming transistors, which introduces large capacitances to the circuit. When the reduction on R_{LRS} is not as significant as the increment on capacitances, the delay of a RRAM-based circuit increases. In addition, large programming transistors increase the area and large capacitances increase the power consumption. Therefore, a low R_{LRS} does not guarantee the best area-delay and power-delay products [6]. In Fig. 3.18 and Fig. 3.19, we see that the 4T1R programming structure can be more area-delay/power-delay efficient than the SRAM-based multiplexers when $R_{LRS} > 2k\Omega$. Boosting V_{prog} is an efficient method to reduce the area-delay and power-delay products of programming structures. To fully exploit

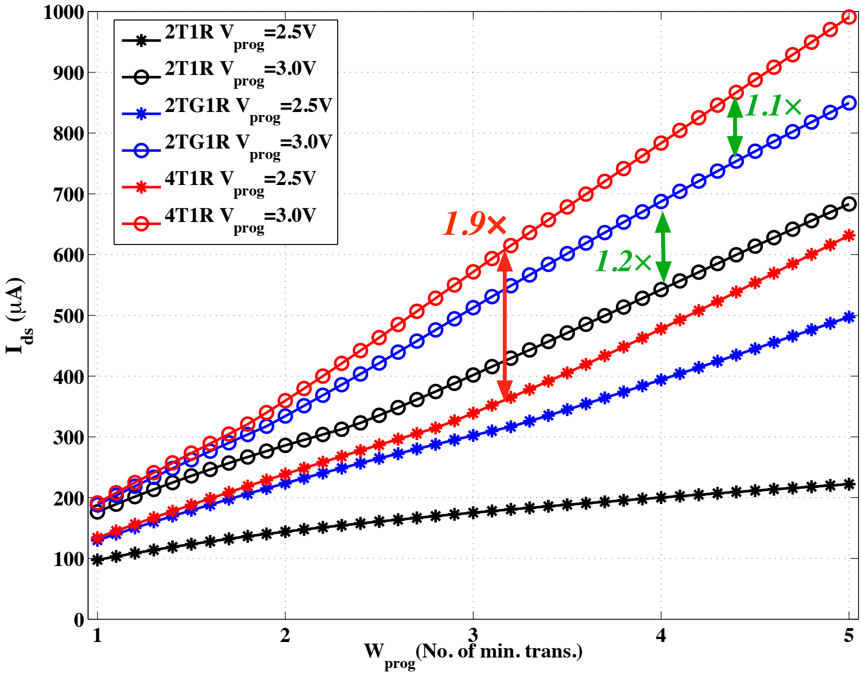


Figure 3.16 – Comparison on I_{ds} in 2T1R, 2TG1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$)

the area and delay of efficiency, it is better to apply the highest possible voltage within the breakdown limit of transistors, i.e., above the standard V_{DD} and close to the breakdown voltage of transistors. It is worth pointing out that the large V_{prog} is only raised during the programming phase, i.e., for a short period of time. As a result, the use of larger programming voltage does not introduce significant reliability hazards.

3.4.6 Summary on the 4T1R programming structures

In summary, the 4T1R programming structures have the following advantages over the 2T1R and 2TG1R structures:
(1) the small V_{DS} gap improves the driving strength of transistors;
(2) Since the **set** and **reset** processes use separated transistors, transistor sizes in 4T1R can be more flexible than 2T1R and 2TG1R, leading to a better area efficiency.
(3) Drain/Source of transistors are directly connected to voltage supplies, eliminating the driving inverters;
(4) the bulk connections of 4T1R structure follow the common digital design practice, and avoid the hazards in 2T1R structure.

Note that the proposed 4T1R programming structure fully overcome the limitations of the 2T1R and 2TG1R programming structures listed in Section 3.2.6 and Section 3.3.4 respectively.

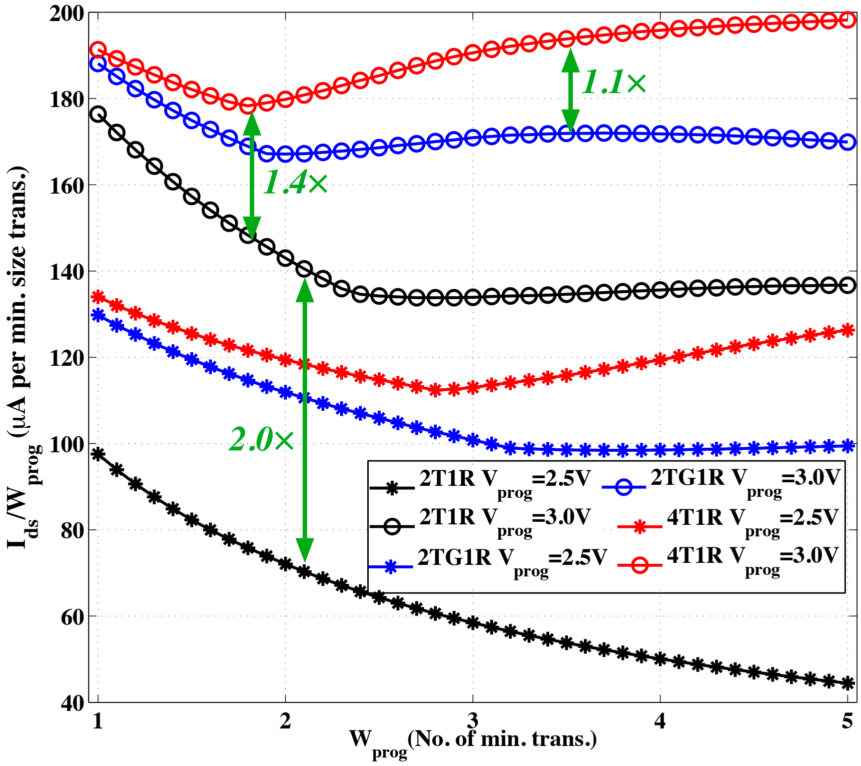


Figure 3.17 – Comparison on driving current per minimum transistor width under diverse W_{prog} and V_{prog} between 2T1R, TG-based 2T1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$)

3.4.7 Discussion

Programming structures are the most basic and common elements of all the RRAM-based circuits, such as NV SRAMs, NV FFs, multiplexers *etc.*. Therefore, performance of programming structures, i.e., the lowest achievable R_{LRS}, transistor area and easiness in physical design, are critical factors impacting the quality of all the RRAM-based circuits. Compared to the 2T1R and the 2TG1R programming structures, the proposed 4T1R programming structure has demonstrated superior capability to achieve lower R_{LRS} with smaller transistor sizes, and also be more friendly to physical designs. The advance in programming structure will case a significant impact on all the RRAM-based circuits and even FPGA architectures.

From a circuit design perspective: Most importantly, a lower achievable R_{LRS} by using smaller transistor sizes leads to smaller resistance and parasitic capacitances on the datapath, meaning that 4T1R-based circuits can achieve better performance than 2T1R-based and 2TG1R-based circuits. Using smaller transistor sizes also leads to that 4T1R-based circuits can be smaller in transistor area than 2T1R-based and 2TG1R-based circuits. Furthermore, the 4T1R programming structure is more adaptive for RRAM devices especially those with asymmetric V_{set} and V_{reset} than its 2T1R and 2TG1R counterparts, leading to better compatibility in

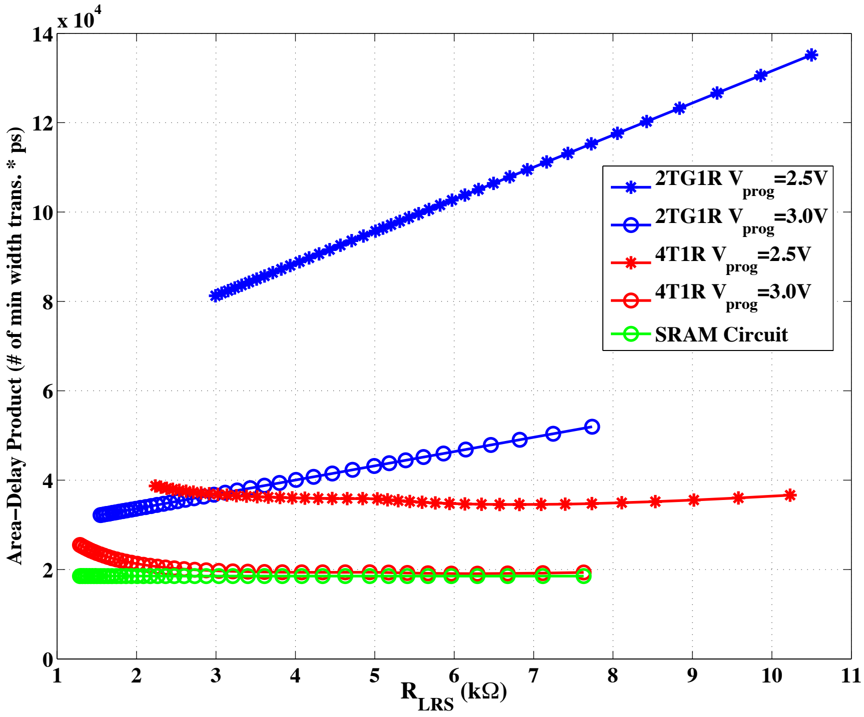


Figure 3.18 – Comparison on area-delay product of 2TG1R and 4T1R structures ($W_{inv} = 20$).

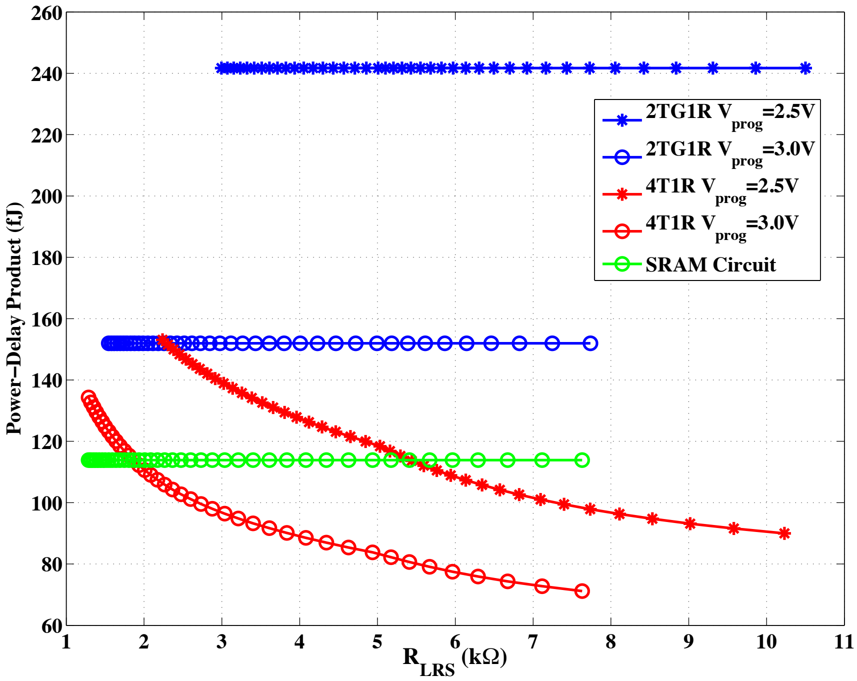


Figure 3.19 – Comparison on power-delay product of 2TG1R and 4T1R structures ($W_{inv} = 20$).

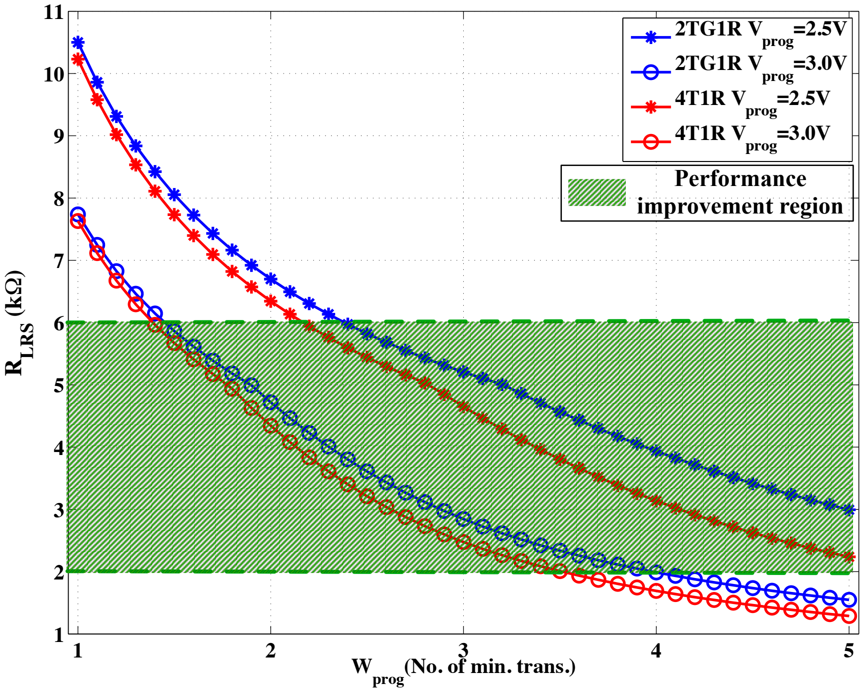


Figure 3.20 – Comparison on R_{LRS} in 2TG1R and 4T1R structures ($W_{inv} = 20$). (1 $W_{prog} = 320nm$)

integrating generic RRAM technology. In addition, the introduced boosting methodologies (increasing V_{prog} and W_{prog}) are effective to all the programming structure. (2T1R, 2TG1R and 4T1R), being generic methods to improve the performance of 4T1R-based circuits. Note that the methodology used in theoretical analysis can generalized to other non-volatile memory technologies, such as *Phase Change Memory* [40], which have similar I-V characteristics as RRAMs.

From an architecture perspective: RRAM-based FPGAs use a low R_{LRS} to improve the performance of routing elements. As it will be presented in Section 3.7 and Chapter 5, a proper R_{LRS} target for FPGA architectures is between $2k\Omega$ and $6k\Omega$ depending on the design context, while R_{HRS} should be at least $20M\Omega$ to mitigate a leakage power increase. The mentioned ranges of R_{LRS} and R_{HRS}, achievable as worst case target in current RRAM technologies, show that, beyond the performance gain, FPGA architectures can tolerate a wide distribution of R_{LRS} and R_{HRS} without delay and power increase [6, 114]. The performance of RRAM-based routing elements are not only determined by the R_{LRS} but also the parasitic capacitances of programming transistors. As a result, programming structures offering a high current density, e.g., the proposed 4T1R programming structure, are preferred. Fig. 3.20 shows the R_{LRS} values that can be driven by 2TG1R and 4T1R structures as a function of W_{prog}. To obtain a proper R_{LRS} in FPGA, the applicable W_{prog} of transistors are between 1.5 and 4. Boosting V_{prog} can significantly reduce the R_{LRS}, which brings opportunities in further area and delay improvement on RRAM-based FPGAs. When considering more advanced technology nodes, such as

28nm, 14nm and beyond, it is expected that lower V_{reset} and V_{set} voltages can be employed as a consequence of the V_{DD} reduction. As a result, the effect of boosting V_{prog} is expected to gain further in efficiency.

Part 2: RRAM-based Multiplexer Designs

As 4T1R programming structure (See Section 3.4) shows outstanding advantages over 2T1R counterparts, it opens opportunities in improving RRAM-based routing multiplexer designs. The second part of this chapter focus on studying how to efficiently integrate the 4T1R programming structure in routing multiplexers. As explained in Section 3.4, both 2T1R and 4T1R programming structures have to employ a high programming voltage, different from nominal working voltage, in order to drive the set and reset currents. Therefore, in physical design, a deep N-well (highlighted red in Fig. 3.21) is required to provide a different voltage domain for the programming structure. However, deep N-wells typically require large spacing between each other and also regular N-wells. This reveals a series of challenges at the physical design level, such as how to co-integration of low-voltage nominal power supply and high voltage programming supply, which have not been evaluated in previous works [6, 114, 26, 110, 9, 27, 133]. This motivates us to take the parasitics into account and study the physical design aspects of integrating 4T1R programming structure into RRAM-based multiplexers.

This part is organized as follows: Section 3.5 introduces and analyzes a naive one-level 4T1R-based multiplexer at the physical design level. Section 3.6 proposes improved one-level, two-level and tree-like 4T1R-based multiplexers, overcoming difficulties in physical design. Section 3.7 deals with a generic optimizing technique for RRAM-based circuits, i.e., programming transistor sizing technique, which enables large design space to be explored. Section 3.8 presents some experimental results and Section 3.9 analyzes the impact of process variations.

3.5 Basic 4T1R-based Multiplexer

In this section, we propose a naive multiplexer structure using 4T1R elements and discuss a few limitations of the structure.

3.5.1 Multiplexer Structure and Programming Strategy

By following the general topology shown in Fig. 2.25, a basic one-level $N:1$ multiplexer can be developed with 4T1R elements. The resulting one-level N-input RRAM-based multiplexer is illustrated in Fig. 3.21 and consists of N pairs of 4T1R programming structures, which are controlled by $N+1$ Bit lines and $N+1$ Word lines. Since RRAMs require a programming voltage which is higher than the nominal one, a Deep N-well isolation (highlighted red in Fig. 3.21) is required for the programming structures, resulting in two power domains. Instead of providing each RRAM with four independent programming transistors, all the RRAMs can share a pair of programming transistors (controlled by $\overline{BL[N]}$ and $WL[N]$ respectively) at node B. As a result, each RRAM can be individually programmed with either positive or negative voltage polarity. For example, we can first set RRAM R_0 by enabling $\overline{BL[0]}$ and $WL[N]$. Note that the rest of bit lines and word lines should be off, to ensure the programming current (highlighted blue in Fig. 3.21) flows only through transistor **P0**, RRAM R_0 and transistor **N0**. Then we can turn off $\overline{BL[0]}$ and $WL[N]$, and turn on $\overline{BL[N]}$ and $WL[N-1]$ to reset RRAM R_{N-1}. Sharing programming transistors in the multiplexer structure is flexible enough from a reconfiguration standpoint. In practice, in a N-input multiplexer, only one RRAM is in LRS while the others are in HRS. Each time a multiplexer is reconfigured, one RRAM is reset from LRS to HRS and another is set from HRS to LRS, implying two steps (one reset process and one set process). Note that set and reset process have to be executed sequentially because set and reset processes require different programming voltages at node B. Whether the multiplexer has shared programming transistors or employs independent programming transistors for each RRAMs, we always need two steps (one reset process and one set process) in each reconfiguration. More importantly, sharing programming transistors can significantly reduce the parasitic capacitances at node B in Fig. 3.21, leading to large delay and power improvements. Independent programming transistors cause that the total parasitic capacitance at node B includes N pairs of programming transistors. In contrast, sharing programming transistors lead to that the total parasitic capacitance at node B includes only a pair of programming transistors.

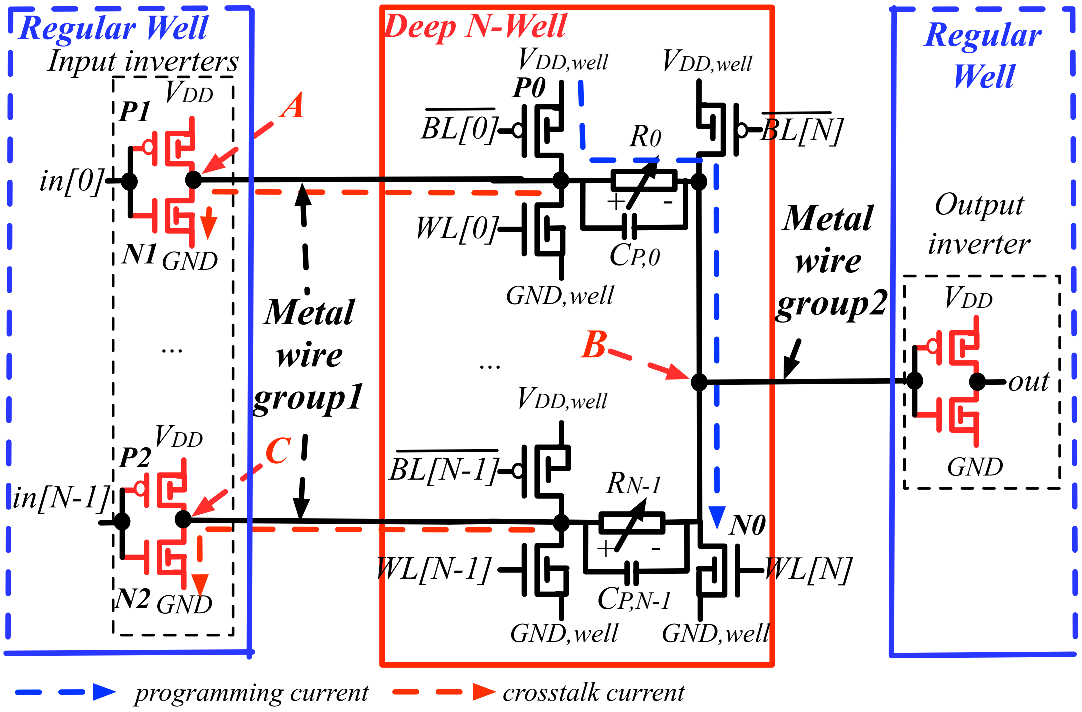


Figure 3.21 – Circuit design and well arrangement of a naive $N:1$ one-level 4T1R-based multiplexer

3.5.2 Limitations from a Physical Design Perspective

Such straightforward design suffers from three possible limitations due to the co-integration of both datapath and programming channels.

Limitation 1: Programming Currents Contribution from Datapath Transistors

Whether a RRAM can be programmed into a reasonable R_{LRS} highly depends on the amount of programming current that can be driven through the RRAM. In order to accurately control the programming current of a RRAM, only a pair of *p*-type and *n*-type transistors is turned on during programming. However, during programming, some datapath transistors in *on* state could inject or distribute the programming currents, leading to the achieved R_{LRS} to be out of specifications. Take the example in Fig. 3.21, assume that RRAM R_0 is being programmed by enabling transistors **P0** and **N0**. Datapath transistors **N1** and **N2** could potentially be in *on* state, sinking part of the programming current, as highlighted by red dashed lines. This would cause the programming current (blue dashed lines) to be smaller than expected, leading to a higher R_{LRS}. Note that not only pull-down transistors, such as **N1** and **N2**, but pull-up transistors of input inverters, such as **P1** and **P2**, can interfere with the programming current. Such interference becomes serious as input sizes increases, which can significantly reduce the programming current passing through RRAMs and even cause failure in configuring RRAMs.

Limitation 2: Breakdown Threats of Datapath Transistors

To achieve a reasonable R_{LRS}, programming voltages $prog\_VDD$ should be large enough to drive a high enough programming current. For instance, [133] considers a programming voltage as high as $prog\_VDD = 3.0V$ while the nominal voltage of the datapath transistors is only $VDD = 0.9V$. Such large gap between $prog\_VDD$ and V_{DD} could cause the datapath transistors to breakdown during RRAMs' programming phases. Take the example in Fig. 3.21, the voltage of node A, V_A, can reach $prog\_VDD$ while programming RRAM R_0, leading to the source-to-drain voltage of transistor **P1** being $prog\_VDD - V_{DD}$. Assume that $prog\_VDD = 3.0V$ and $V_{DD} = 0.9V$, both the gate-to-source voltage V_{GS} and source-to-drain voltage V_{DS} of transistor **P1** are $2.1V$, possibly leading transistor **P1** to breakdown. Note that not only transistor **P1** but also all the transistors belonging to the input and output inverters in Fig. 3.21 can be in a breakdown condition. While exposed to these conditions, even if datapath transistors do not break down, their reliability, i.e., lifetime, would significantly degrade.

Limitation 3: Long Interconnecting Wires between Wells

Since RRAMs require a programming voltage which is higher than the nominal one, a deep N-well isolation (highlighted red in Fig. 3.21) is required for the programming structures, resulting in three N-wells as shown in Fig. 3.21. In physical designs, a large spacing is required between a deep N-well and a regular N-well, which introduces long interconnecting wires. As illustrated in Fig. 3.21, two groups of long interconnecting wires have to be employed: one is between input inverters and programming structures while the other is between programming structures and output inverters. The long metal wires introduce parasitic resistances and capacitances to 4T1R-based multiplexers, potentially causing delay and power degradation.

Therefore, there is a strong need to study how to properly integrate 4T1R programming structures into RRAM-based multiplexers without area and delay overhead while guaranteeing robust operations.

3.6 Improved 4T1R-based Multiplexer

In this section, we address the limitations of the previously introduced naive 4T1R-based multiplexers by employing power-gated inverters and rearranging the power domains. In addition to the one-level 4T1R-based multiplexers, we also investigate two-level and tree-like multiplexer structures, similar to baseline CMOS multiplexers.

3.6.1 One-level Multiplexer Structure

In order to address the identified limitations, we present, in Fig. 3.22(a), an improved one-level N-input 4T1R-based multiplexer, which is different from the one in Fig. 3.21 in two aspects: (a) the datapath input inverters are power-gated in order to eliminate the contribution of the datapath transistors in the programming phase; (b) the two power domains (and the isolation deep N-well) are organized differently to Fig. 3.21. Indeed, the input inverters and part of 4T1R programming structures are driven by a constant voltage domain V_{DD} and GND while the output inverter and the rest of 4T1R programming structures are driven by switchable voltage supplies $V_{DD,well}$ and GND_{well}. During operation, $V_{DD,well}$ and GND_{well} are configured to be equal to V_{DD} and GND respectively, as shown in Fig. 3.22(a). Note that the RRAM programming voltages are typically selected to be larger than V_{DD}, ensuring that RRAMs are not parasitically programmed during operation. When a set operation is triggered, input inverters are disabled and $V_{DD,well}$ and GND_{well} are switched to be $-V_{prog}+2V_{DD}$ and $-V_{prog}+V_{DD}$ respectively, as highlighted red in Fig. 3.22(b). During reset operations, input inverters are disabled and $V_{DD,well}$ and GND_{well} are switched to be V_{prog} and $V_{prog}-V_{DD}$ respectively, as highlighted red in Fig. 3.22(c). As such, the voltage difference across the RRAM during set or reset is $\pm V_{prog}$ and the working principle of the 4T1R programming structure can still be applied. Indeed, to enable the programming current path highlighted blue in Fig. 3.22(b), bit line $\overline{BL[0]}$ is configured to be GND and word line $WL[N]$ is configured to be $-V_{prog}+2V_{DD}$ while other programming transistors should be turned off by configuring $\overline{BL[i]} = VDD, WL[j] = GND, 1 \le i \le N-1, 0 \le j \le N-1$ and $\overline{BL[N]} = -V_{prog}+2V_{DD}$. Table 3.1 summaries the voltages involved in the different operations.

The improved 4T1R-based multiplexer has a major advantage over the initial design in Fig. 3.21: the voltage drop across each datapath transistor can be limited to V_{DD}, allowing the use of

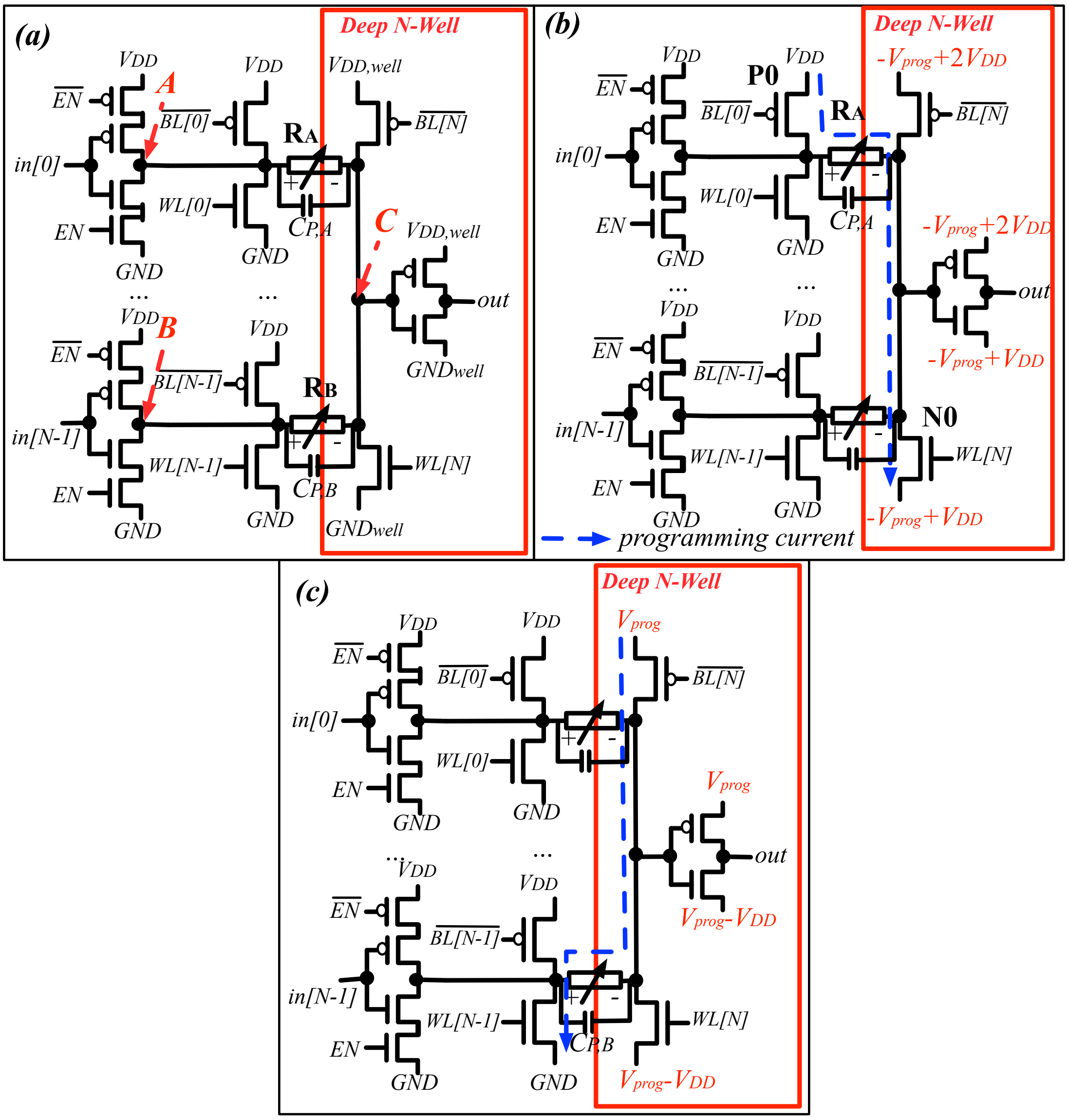


Figure 3.22 – Improved one-level N-input 4T1R-based multiplexer: (a) operating mode ($V_{DD,well} = V_{DD}$, $GND_{well} = GND$); (b) set process ($V_{DD,well} = -V_{prog} + 2V_{DD}$, $GND_{well} = -V_{prog} + V_{DD}$); (c) reset process ($V_{DD,well} = V_{prog}$, $GND_{well} = V_{prog} - V_{DD}$;

logic transistors instead of I/O transistors (thicker oxides and higher breakdown voltage). Logic transistors occupy less area and introduce less capacitances than I/O transistors, potentially improving the footprint and delay of RRAM multiplexers. During the set and reset processes, the voltage drop of each transistor can be boosted from V_{DD} to $V_{DD,max}$, approaching the maximum reliable voltage without breakdown limitation. Boosted $V_{DD,max}$ leads to higher current density driven by transistors, further contributing to a lower R_{LRS} [133]. Note that the set and reset processes typically require short amount of time, i.e., typically 200ns for each

RRAM [133]. Since programming does not occur many times (non-volatility), very low stress is applied on the transistors, further contributing to a robust operation.

Table 3.1 – Voltages arrangements for operation, set and reset examples in Fig. 3.22(a)(b)(c)

Control lines/ Voltages	Operation Fig. 3.22(a)	Set process Fig. 3.22(b)	Reset process Fig. 3.22(c)
$\overline{BL[0]}$	V_{DD}	GND	V_{DD}
$\overline{BL[i]}$, $1 \le i \le N-1$	V_{DD}	V_{DD}	V_{DD}
$\overline{BL[N]}$	V_{DD}	$-V_{prog}+2V_{DD}$	$V_{prog}-V_{DD}$
$WL[i]$, $0 \le i \le N-2$	GND	GND	GND
$WL[N-1]$	GND	GND	V_{DD}
$WL[N]$	GND	$-V_{prog}+2V_{DD}$	$V_{prog}-V_{DD}$
$\overline{EN}$	GND	V_{DD}	V_{DD}
EN	V_{DD}	GND	GND
$V_{DD,well}$	V_{DD}	$-V_{prog}+2V_{DD}$	V_{prog}
GND_{well}	GND	$-V_{prog}+V_{DD}$	$V_{prog}-V_{DD}$

3.6.2 Physical Design Advantages

The improved 4T1R-based multiplexer layout has two major advantages over the initial design in Fig. 3.21:

(1) the voltage drop across each datapath transistor can be limited to V_{DD}, allowing the use of logic transistors instead of I/O transistors (thicker oxides and higher breakdown voltage). Logic transistors occupy less area and introduce less capacitances than I/O transistors, potentially improving the footprint and delay of RRAM multiplexers. During the set and reset processes, the voltage drop of each transistor can be boosted from V_{DD} to $V_{DD,max}$, approaching the maximum reliable voltage without breakdown limitation. Boosted $V_{DD,max}$ leads to higher current density driven by transistors, further contributing to a lower R_{LRS} [133]. Note that the set and reset processes typically require short amount of time, i.e., typically $200ns$ for each RRAM [133]. Since programming does not occur many times (non-volatility), very low stress is applied on the transistors, further contributing to a robust operation.

(2) Only one connection between regular and deep N-Wells is necessary. As a result, only one group of long interconnecting wires is employed, potentially reducing the parasitics from metal wires. To be more illustrative, we depict in Fig. 3.23 and compare the cross-sections of the naive and improved designs at layout level. In each illustrative cross-section, we consider an input inverter $in0$, an output inverter, and a 4T1R programming structure. We assume that,

in the naive design, input and output inverters can be accommodated with a regular N-well, so as to be more area efficient. However, even when the regular N-well is shared, long metal wires are still required because interconnections between datapath logics and programming structures have to include a large space between regular N-well and deep N-well. The length of metal wires $MET1$ and $MET2$ in Fig. 3.23(a) are dominated by the large well spacing L. Fig. 3.23(b) depicts the cross-section of the improved circuit in Fig. 3.22(a). Since RRAMs can be fabricated between metal lines, they can be located in any position between the two wells. Whatever location the RRAM is, there is only one long metal wire ($MET2$ and part of $MET1$) across two wells, while the other metal wires $MET1$ connect transistors inside the same well. Note that the length of interconnecting wires inside the same well is much smaller than those across two wells L. As a result, the length of metal wires in the naive design is dominated by $2 \cdot L$, while the improved design is dominated by L. Therefore, the improved design can reduce 50% the length of interconnecting wire than the naive design, contributing to smaller parasitic resistances and capacitances.

3.6.3 Two-level and Tree-like multiplexer Structure

Based on the circuit topology of CMOS multiplexers shown in Fig. 2.15, we also develop N-input 4T1R-based multiplexers implemented with two-level and tree-like structures. The resulting structures are depicted in Fig. 3.24 and Fig. 3.25 respectively. The two-level and tree-like structures are implemented by cascading elementary one-level multiplexer structures similar to the one shown in Fig. 3.21. Note that even in two-level and tree-like 4T1R multiplexers, only one DNW is needed, as highlighted red in Fig. 3.24 and Fig. 3.25 respectively. To simplify the programming strategies, RRAMs in the even levels have opposite polarities than those in the odd levels. Take the example in Fig. 3.24, the polarities of RRAMs in the second level, highlighted in red, are opposite to the first level. As such, when set processes are required, $V_{DD,well}$ and GND_{well} are switched to $-V_{prog}+2V_{DD}$ and $-V_{prog}+V_{DD}$ respectively; while during reset processes, $V_{DD,well}$ and GND_{well} are switched to V_{prog} and $V_{prog}-V_{DD}$ respectively. Otherwise, if all the RRAMs have had the same polarity, switching $V_{DD,well}$ and GND_{well} depends not only on the type of process (either set or reset) but also on the number of levels (either even or odd), requiring additional circuitry. In addition, DNWs also can be efficiently shared between two cascaded 4T1R-based multiplexers, as illustrated in Fig. 3.26. The input inverters and part of programming structures of $MUX1$ in Fig. 3.26 can share a DNW with the output inverter and part of programming structures of $MUX0$. Note that the polarities of RRAMs of $MUX1$ are opposite to the RRAMs of $MUX0$, allowing a similar programming strategy as highlighted above.

The number of bit lines and word lines can be reduced, as the 4T1R programming structures belonging to the same level can efficiently share control lines, allowing RRAMs to be programmed simultaneously. Take the example of Fig. 3.24, all the multiplexer structures from the first stage can be connected to bit lines $\overline{BL[j]}, 0 \le j \le \sqrt{N}$ and word lines $WL[j], 0 \le j \le \sqrt{N}$. RRAMs that are controlled by $\overline{BL[0]}$ and $WL[\sqrt{N}]$, i.e., R_A and R_B in Fig. 3.24, can be programmed

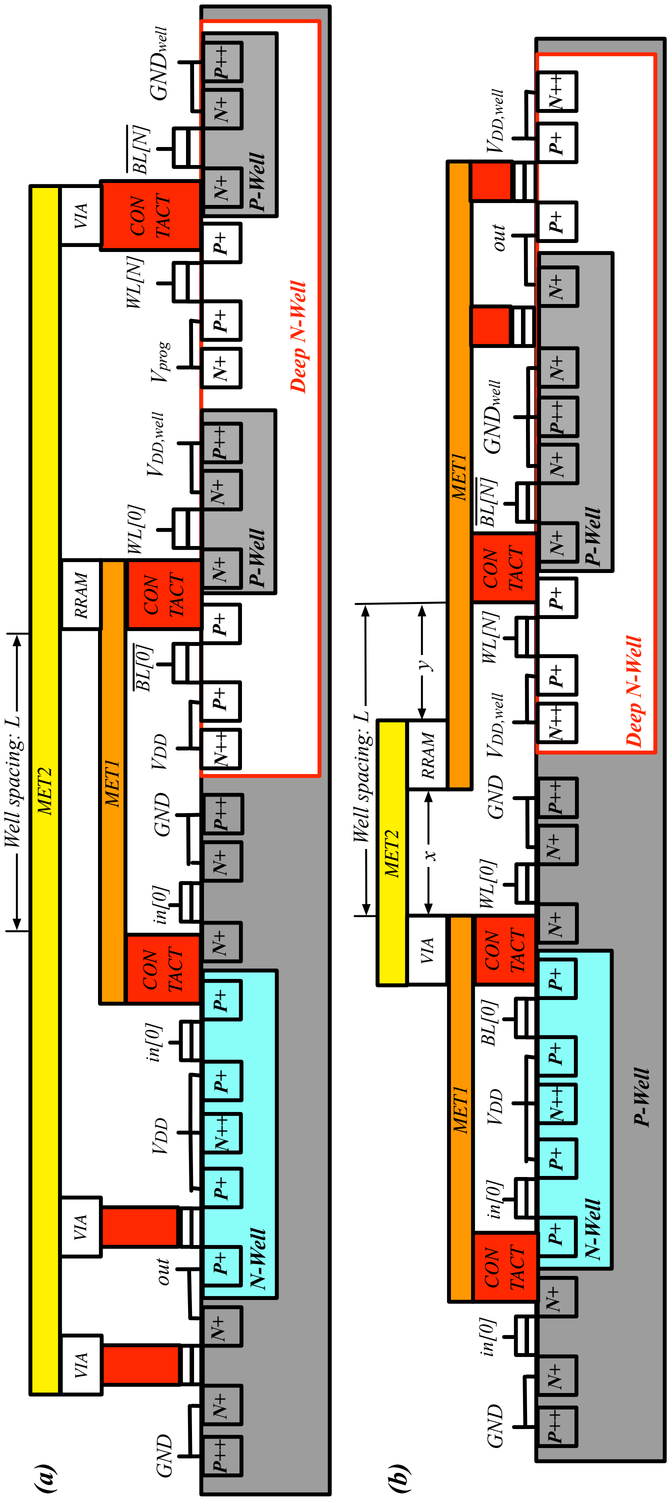


Figure 3.23 – Cross-section of the layout of 4T1R multiplexers: (a) naive design; (b) improved design.

simultaneously, which is resembling to the control sharing in a CMOS multiplexer tree. RRAMs belonging to different stages have to be programmed sequentially. A two-level or tree-like 4T1R-based multiplexer requires $2m$ steps (m reset processes and m set processes) to program all the RRAMs, where m represents the number of stages. In contrast, a one-level 4T1R-based multiplexer, consisting of fewer RRAMs, only need two steps, implying less reconfiguration time and programming energy.

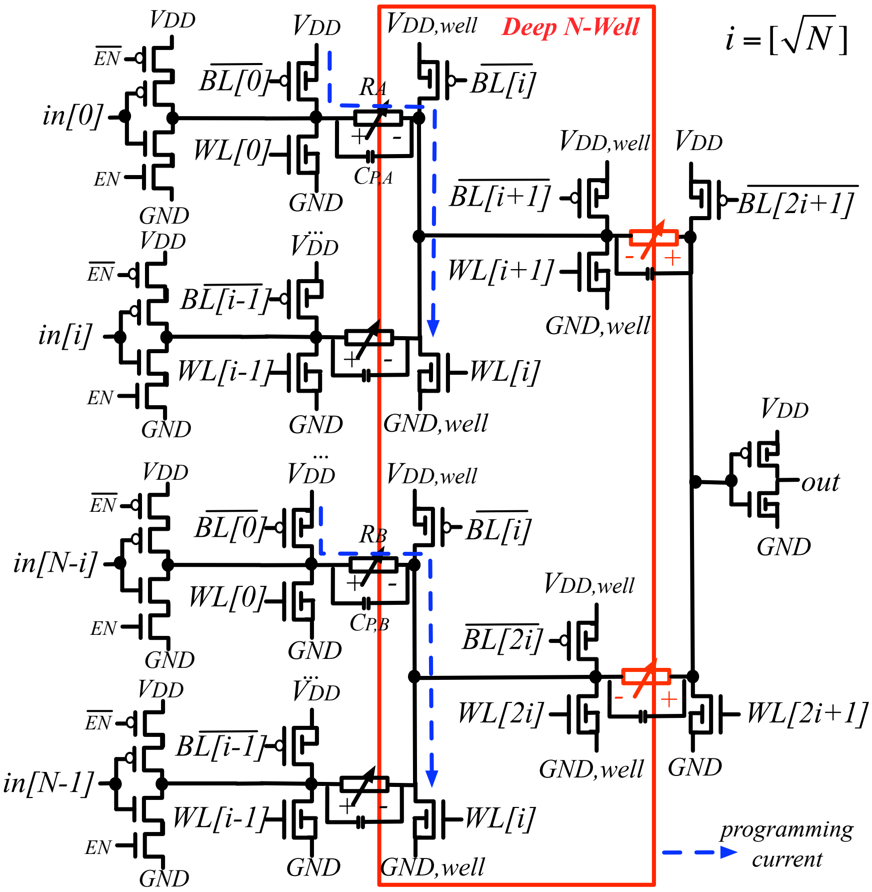


Figure 3.24 – Schematic of a robust two-level N-input 4T1R-based multiplexer.

3.6.4 Sharing deep N-Well between multiplexers

Deep N-wells can be efficiently shared between two cascaded 4T1R-based multiplexers, as illustrated in Fig. 3.26. The input inverters and part of programming structures of $MUX1$ in Fig. 3.26 can share a deep N-well with the output inverter and part of programming structures of $MUX0$. Note that the polarities of RRAMs of $MUX1$ are opposite to the RRAMs of $MUX0$, allowing simple programming strategies. As such, when set processes are required, $V_{DD,well}$ and GND_{well} are switched to $-V_{prog}+2V_{DD}$ and $-V_{prog}+V_{DD}$ respectively; while during reset processes, $V_{DD,well}$ and GND_{well} are switched to V_{prog} and $V_{prog}-V_{DD}$ respectively;

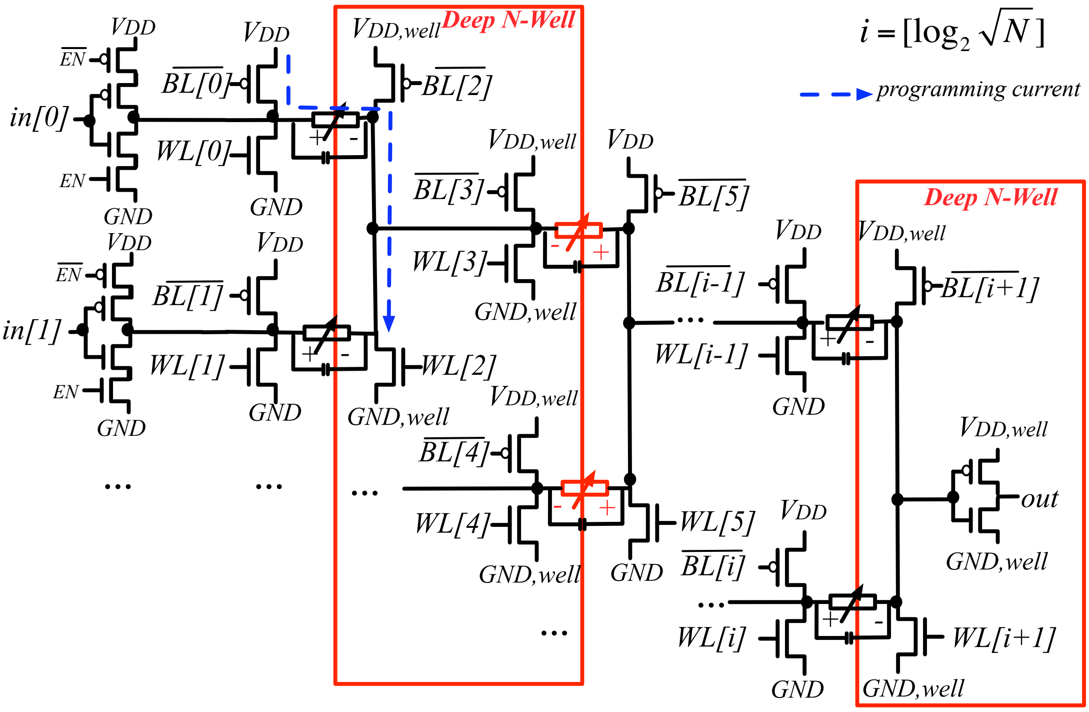


Figure 3.25 – Schematic of a robust tree-like N-input 4T1R-based multiplexer.

Otherwise, if all the RRAMs have had the same polarity, switching $V_{DD,well}$ and GND_{well} depends not only on the programming operation (either set or reset) but also on the location of multiplexers, requiring additional circuitry.

3.6.5 Constraints on the Programming Voltage V_{prog}

During set and reset processes, the necessary programming voltage V_{prog} is determined by the source-to-drain voltage drop across the programming transistors and the programming threshold voltage of the RRAMs. The V_{DS} of the programming transistors should be large enough in order to drive sufficient programming current, but should also be selected under the breakdown conditions. Therefore, there exists a limit for V_{prog} to be respected. For instance, in the set example of Fig. 3.22(b), V_{prog} can be expressed as the sum of the voltages across RRAM A and the programming transistors **P0** and **N0**:

$$\begin{cases} V_{DS,P0} + V_{DS,N0} + V_{set,min} = V_{prog}, \\ V_{DS,P0} = V_{DS,N0} \leq V_{DD,max}, \end{cases} \tag{3.16}$$

where $V_{set,min}$ is minimum programming voltage to trigger a set process for a RRAM. Note that the V_{DS} of the programming transistors should be the same to guarantee the best achievable current density[133]. Similarly, for the reset example in Fig. 3.22(c), one can derive a similar

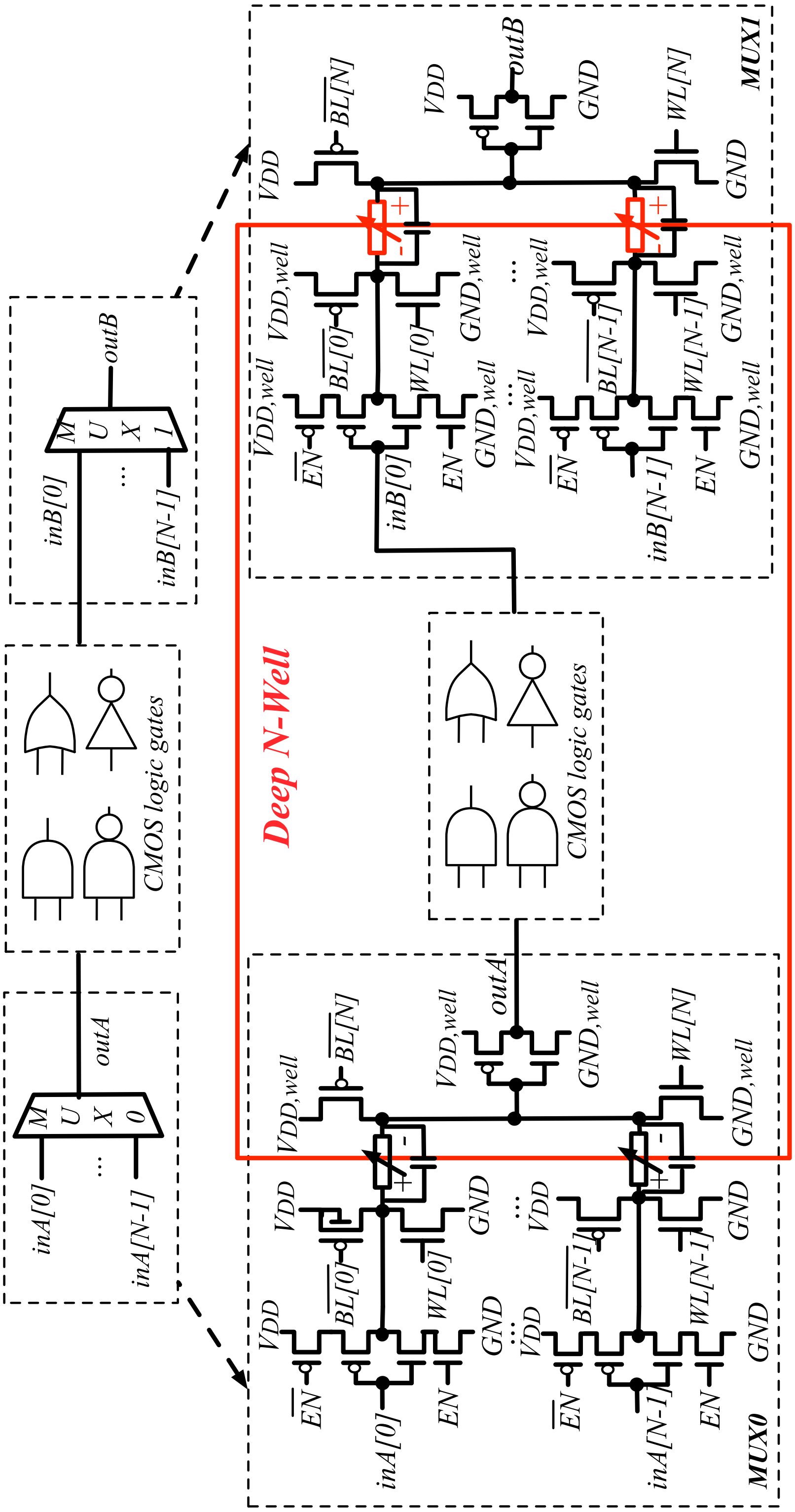


Figure 3.26 – Cascading two N-input one-level 4T1R-based multiplexers: share Deep N-Wells efficiently.

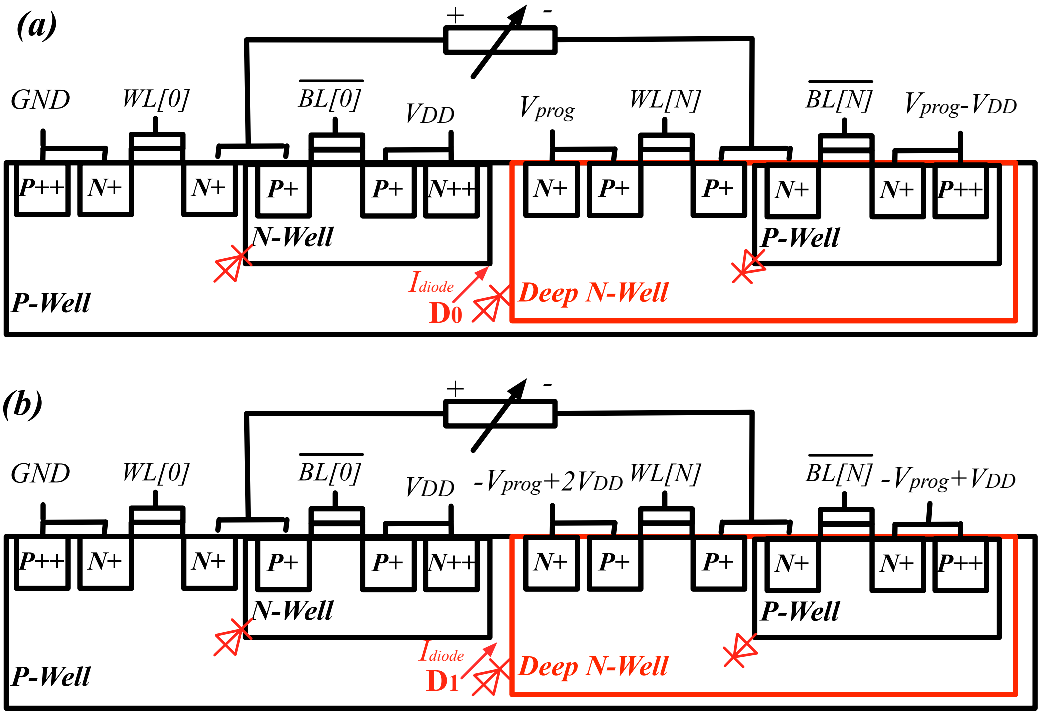


Figure 3.27 – Cross-section of the layout of a 4T1R programming structure: (a) during reset process; (b) during set process.

set of constraints with transistors **P1** and **N1**:

$$\begin{cases} V_{DS,P1} + V_{DS,N1} + V_{reset,min} = V_{prog}, \\ V_{DS,P1} = V_{DS,N1} \leq V_{DD,max}, \end{cases} \tag{3.17}$$

where $V_{set,min}$ is minimum programming voltage to trigger a reset process for a RRAM.

In addition to the limitations mentioned above, the use of different wells also constrains V_{prog} as the diode across *P-Well* and *Deep N-Well* should be reversely biased, as illustrated in Fig. 3.27(a) and (b). During the reset process in Fig. 3.27(a), diode D_0 is always reversely biased because the voltage of *P-Well* is GND and the voltage of *Deep N-Well* is $V_{prog} > GND$. However, during the set process in Fig. 3.27(b), diode D_1 is reversely biased only when:

$$(-V_{prog} + 2V_{DD}) - GND \geq 0. \tag{3.18}$$

If we boost V_{DD} to $V_{DD,max}$ during set and reset process, the constraint becomes:

$$(-V_{prog} + 2V_{DD,max}) - GND \geq 0. \tag{3.19}$$

By combining (3.16), (3.17) and (3.19), we obtain:

$$\begin{cases} V_{prog} \le 2V_{DD,max} + V_{set}, \\ V_{prog} \le 2V_{DD,max} + V_{reset}, \\ V_{prog} \le 2V_{DD,max}. \end{cases} \tag{3.20}$$

As a result, the upper bound for V_{prog} can be expressed as:

$$V_{prog} \le 2V_{DD,max} \tag{3.21}$$

As discussed in [133], a larger V_{prog} leads to a higher programming current and a lower R_{LRS}. In this paper, we consider $V_{prog} = 2V_{DD,max}$ for the electrical simulations.

3.6.6 Analytical Comparison between 4T1R multiplexers

Note that the two-level and tree-like 4T1R-based multiplexers reduce the number of control/programming lines significantly but does not reduce the number of required RRAMs. An analytical comparison of the area, delay and energy between 4T1R-based multiplexers is shown in Table 3.2, and will be verified by electrical simulations in Section 3.8. In CMOS technology, two-level multiplexers produce the best area-delay-power product because their structure reduces not only the number of control lines but also the parasitic capacitances introduced in the critical path. Since the parasitic capacitances of a RRAM is typically smaller than a transistor, the delay and power of one-level 4T1R-based multiplexers scale better with the number of inputs N than CMOS multiplexers. When the input size is small and total capacitance is dominated by programming transistors, the delay and power of one-level 4T1R-based multiplexers are better than two-level and tree-like structures. When the input size is large enough, the total capacitance is dominated by C_P and two-level 4T1R-based multiplexers become better in delay and power.

Table 3.2 – Analytical comparison on area, delay and switching energy of N-input 4T1R-based multiplexers.

Multiplexer	One-level	Two-level	Tree-like
Area[1]	$N \cdot Area_{trans}$	$(N + [\sqrt{N}]) \cdot Area_{trans}$	$(2N-2) \cdot Area_{trans}$
Delay[2]	$R_{LRS} \cdot (C_{trans} + N \cdot C_P)$	$4R_{LRS} \cdot (C_{trans} + [\sqrt{N}] \cdot C_P)$	$0.5 \cdot \alpha \cdot 4 \cdot V_{DD}^2$ $\cdot (C_{trans} + [\sqrt{N}] \cdot C_P)$
Energy[3]	$0.5 \cdot \alpha \cdot V_{DD}^2 \cdot$ $(C_{trans} + N \cdot C_P)$	$\frac{1}{2}([log_2 N]^2 + [log_2 N]) R_{LRS}$ $\cdot (C_{trans} + C_P)$	$0.5 \cdot \alpha \cdot \frac{1}{2}(3[log_2 N] - 1)$ $\cdot (C_{trans} + C_P) V_{DD}^2$

[1] Area of input and output inverters are not included here.

[2] Elmore delay model [104] is considered here.

[3] Only the switching energy of multiplexer structures is considered here.

α is the switching activity.

* R_{LRS} is the equivalent resistance of a RRAM in LRS. C_P is smaller than C_{trans}.

3.7 Optimal Physical Design Parameters

In previous works [26, 9, 110, 27, 8], the sizes of programming transistors are considered uniform to achieve the lowest R_{LRS} of RRAM, which is assumed to produce the best performance of RRAM-based interconnects. However, Fig. 3.18 and Fig. 3.19 demonstrates that the lowest R_{LRS} do not always guarantee the best *Area-Delay Product* (ADP) and *Power-Delay Product* (PDP). Actually, the delay of RRAM-based programmable interconnects is determined by various factors, such as the resistance of RRAMs, the parasitic capacitance of programming transistors and also the parasitics of long interconnecting wires. As the R_{LRS} value is strongly correlated with the size of the programming transistors W_{prog} (See Section 3.4), there is no guarantee that using the lowest possible the R_{LRS} will give the lowest delay. In addition, as RRAMs can be located anywhere on the long interconnecting wire across the two wells as illustrated in Fig. 3.23, the resulting parasitic capacitance is non-negligible and strongly impacts the performance as well. Despite technology factors, such as R_{LRS} and C_P, there are a few design parameters, such as physical location of RRAMs and programming transistor size W_{prog}, which can potentially impact the performance of RRAM-based multiplexers. Therefore, it is worthwhile to study how to improve RRAM-based multiplexers through tuning the design parameters. In this section, we will first introduce our methodology in modeling RRAM-based multiplexers and then focus on studying the optimizing techniques for improving the performance of 4T1R-based multiplexer designs in two aspects: (1) the impact of physical location of RRAMs; (2) the impact of programming transistor size W_{prog}. Note that the methodology developed here is not dependent on the considered RRAM technology or on the transistor technology nodes or even the circuit design topology, but is rather general.

3.7.1 RC modeling of General 4T1R-based multiplexers

Modeling circuits with equivalent RC tree is a widely used method in studying the delay of digital circuit designs [132], which can bring instructive knowledge for circuit optimization. In this part, we introduce the RC modeling for general cases of 4T1R-based multiplexers including layout-level parasitics, based on which we study the optimizing techniques.

The critical path of a RRAM-based multiplexer is the path from an input to the output which contains the largest number of RRAMs in the *Low Resistance State* (LRS) and the largest number of programming transistors. For instance, the highlighted path in Fig. 3.28(a) is the critical path of a N-input RRAM-based multiplexer. Note that the RRAM-based multiplexer in Fig. 3.28(a) is a general case of multi-level multiplexers, which contains n stages of m-input one-level multiplexing structure. Fig. 3.28(b) depicts all the relevant transistors and RRAMs impacting the critical path, considering the general case of a n-stage RRAM-based multiplexer, while its equivalent *RC* model is given in Fig. 3.28(c). Note that the parasitics of long interconnecting wires across N-wells are included in Fig. 3.28(c), which are represented as $R_{x,i}$, $C_{x,i}$, $R_{y,i}$ and $C_{y,i}$, $i = 1, 2, ..., n$. We define the distance between the RRAM and the regular N-well as $x \in [0, L]$ and the distance between the RRAM and the deep N-well as $y \in [0, L]$, as shown in Fig. 3.23(b).

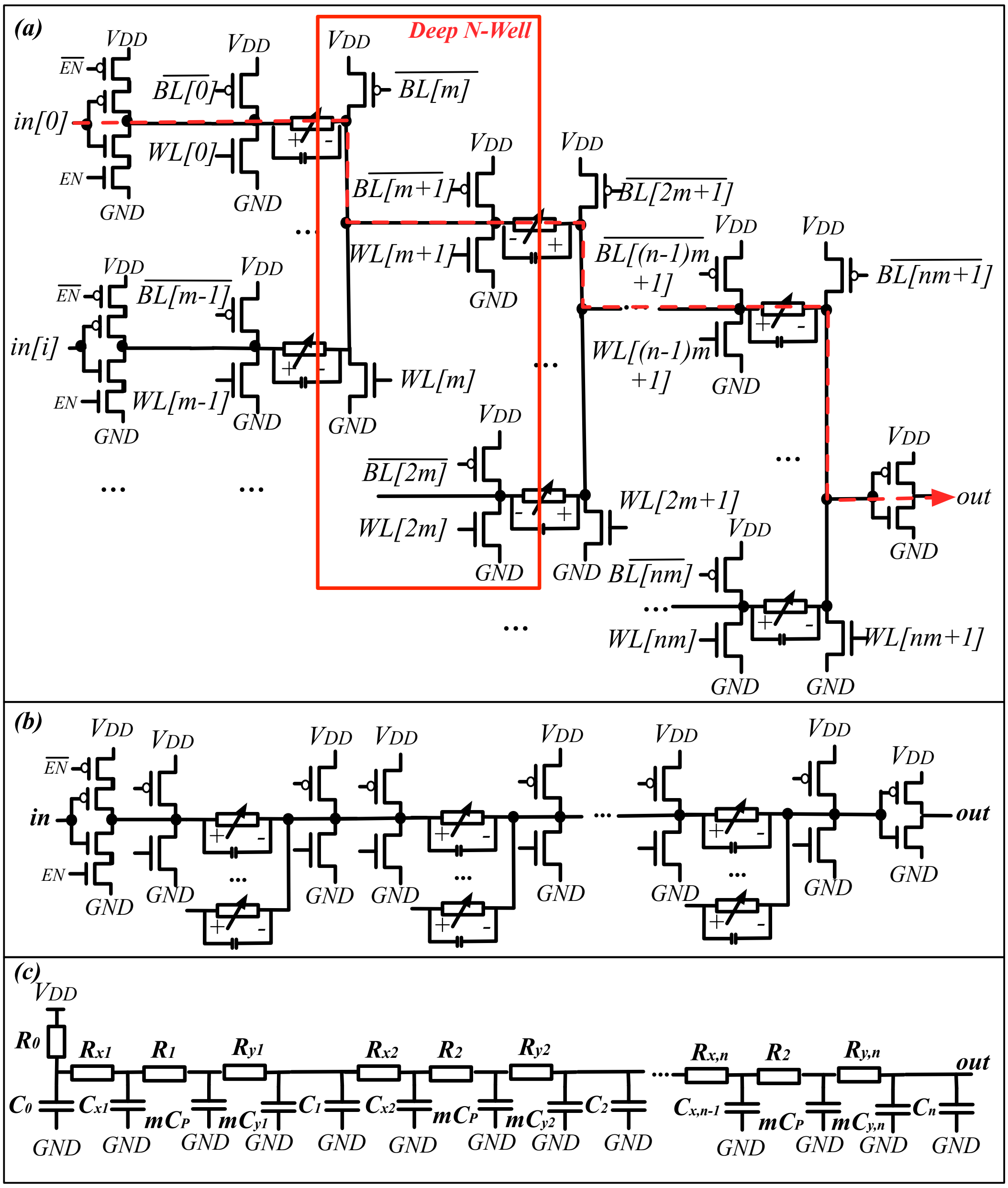


Figure 3.28 – (a) Critical path of a general RRAM-based multiplexer; (b) General critical path of RRAM-based multiplexer; (c) Equivalent RC model.

$(R_{x,i}, C_{x,i})$ and $(R_{y,i}, C_{y,i})$ denote the parasitic resistances and capacitances of the long metal wires at the i^{th} stage of a 4T1R multiplexer, corresponding to (x, y) in Fig. 3.23(b) respectively.

In short, the resistance and capacitance in Fig. 3.28(c) can be extracted from Fig. 3.28(b) and expressed as follows:

$$\begin{aligned}
&R_0 = R_{inv} = \frac{R_{min}}{W_{inv}},\\
&R_i|_{1\leq i\leq n} = R_{LRS},\\
&C_0 = W_{inv}C_{inv} + 2W_{prog}C_{trans},\\
&C_i|_{1\leq i\leq n-1} = 4W_{prog}C_{trans},\\
&C_n = C_L + 2W_{prog}C_{trans},\\
&R_{x,i}|_{1\leq i\leq n} = x_i \cdot R_{\square},\\
&R_{y,i}|_{1\leq i\leq n} = y_i \cdot R_{\square},\\
&C_{x,i}|_{1\leq i\leq n} = x_i \cdot C_{\square},\\
&C_{y,i}|_{1\leq i\leq n} = y_i \cdot C_{\square},
\end{aligned} \tag{3.22}$$

where R_{min} denotes the equivalent resistance of a minimum size inverter, C_{inv} represents the parasitic capacitance at the output of a minimum size inverter, W_{inv} is the size of driving inverter in terms of the minimum width transistor [4]. R_{LRS} denotes the equivalent resistance of a RRAM in LRS, C_P is the parasitic capacitance of a RRAM. W_{prog} represents the width of programming transistor in the unit of the minimum width transistor, and C_{trans} is the parasitic capacitance of a minimum width programming transistor in *off* state. $R_{\square}$ and $C_{\square}$ are the square resistance and capacitance of a unit metal wire respectively. x_i denotes the distance between the RRAM and the left half of 4T1R programming structure at the i^{th} stage of multiplexer, while y_i denotes the distance between the RRAM and the right half of 4T1R programming structure at the i^{th} stage of multiplexer. Note that $x_i + y_i = L$, where L is minimum distance between a regular N-well and a deep N-well.

Considering the Elmore delay [104] of the critical path of a general n-stage RRAM-based multiplexer (Fig. 3.28(b)), we obtain:

$$\begin{aligned}
\tau &= \sum_i C_i \sum_j R_j\\
&= (C_{inv} + 2W_{prog}C_{trans}) \cdot R_{inv}\\
&+ \sum_{i=1}^{n} x_i C_{\square} \cdot [R_{inv} + (i-1)(R_{LRS} + L \cdot R_{\square}) + x_i R_{\square}]\\
&+ \sum_{i=1}^{n} m(L - x_i) C_{\square} \cdot [R_{inv} + i(R_{LRS} + L \cdot R_{\square})]\\
&+ 4W_{prog}C_{trans} \sum_{i=1}^{n-1} [R_{inv} + i(R_{LRS} + L \cdot R_{\square})]\\
&+ (2W_{prog}C_{trans} + C_L) \cdot [R_{inv} + n \cdot (R_{LRS} + L \cdot R_{\square})]\\
&+ m \cdot C_P \sum_{i=1}^{n} (R_{inv} + iR_{LRS} + (i-1)L \cdot R_{\square} + x_i R_{\square})
\end{aligned} \tag{3.23}$$

As we see, despite from technology parameters, i.e., R_{inv}, C_{inv}, $R_{\square}$, $C_{\square}$, C_{trans}, C_P and L, the delay is dependent on many design parameters, x_i, n, m and W_{prog}. To minimize the delay in (3.23), it is worthwhile to study the optimal values of these design parameters. In the rest of this section, we will focus the impact of x_i (See Section 3.7.2) and W_{prog} (See Section 3.7.3).

3.7.2 Physical Position of RRAMs

As illustrated in Fig. 3.23(b), RRAMs are flexible in their location between the two wells. However, the choice of the location of RRAMs lead to different distribution of parasitics inside the 4T1R-based multiplexer, and further resulting in difference in performance. In this part, we study the impact of location of RRAMs on the performance, by using the Elmore Delay in (3.23).

Since our target is to determine the optimal values of variables x_i, we only focus on the terms involving x_i:

$$\begin{aligned}\tau &= f(L, W_{prog}, n, m, R_{inv}, C_{inv}, C_{trans})\\ &+ \sum_{i=1}^{n} R_{\square}C_{\square}{x_i}^2 + [(1-m)R_{inv}C_{\square} + (i-1-mi)(R_{LRS} + LR_{\square})C_{\square} + mR_{\square}C_P]x_i\end{aligned} \tag{3.24}$$

where $f(L, W_{prog}, n, m, R_{inv}, C_{inv}, C_{trans})$ is the sum of terms without x_i.

The delay τ reaches its minimal when x_i is:

$$\begin{aligned}x_{i,opt} &= \frac{(m-1)R_{inv}C_{\square} + (mi+1-i)(R_{LRS} + LR_{\square})C_{\square} - mR_{\square}C_P}{2R_{\square}C_{\square}}\\ &= \frac{m-1}{2}\frac{R_{inv}}{R_{\square}} + \frac{i(m-1)+1}{2}\frac{R_{LRS}}{R_{\square}} + [i(m-1)+1]\frac{L}{2} - \frac{mC_P}{2C_{\square}}\end{aligned} \tag{3.25}$$

Note that $m \geq 2$ and $i \geq 1$, $x_{i,opt}$ is monotonically increasing with respect to i. This implies that $x_{i,opt}$ increases when the number of stages increases. Additionally, in a sophisticated CMOS technology, $C_P \ll C_{\square}$, $R_{inv} \gg R_{\square}$ and $R_{LRS} \gg R_{\square}$. As a result, $x_{i,opt}$ is usually larger than L and Fig. 3.29 depicts the relation between delay τ and x_i in such case.

Our goal is to minimize the delay τ in the range of $x_i \in [0, L]$. As highlighted red in Fig. 3.29, the delay τ is monotonically decreasing when $x_i \in [0, L]$. Hence, the optimal delay is achieved when $x_i = L$. From a circuit design perspective, the optimal location of RRAMs should be close to the right half of 4T1R programming structures, especially in a multi-level multiplexer. In the example of Fig. 3.23(b), the optimal location of RRAMs should be on the top of the deep N-well.

The optimal location of RRAMs will be verified through electrical simulations in Section 3.8.4.

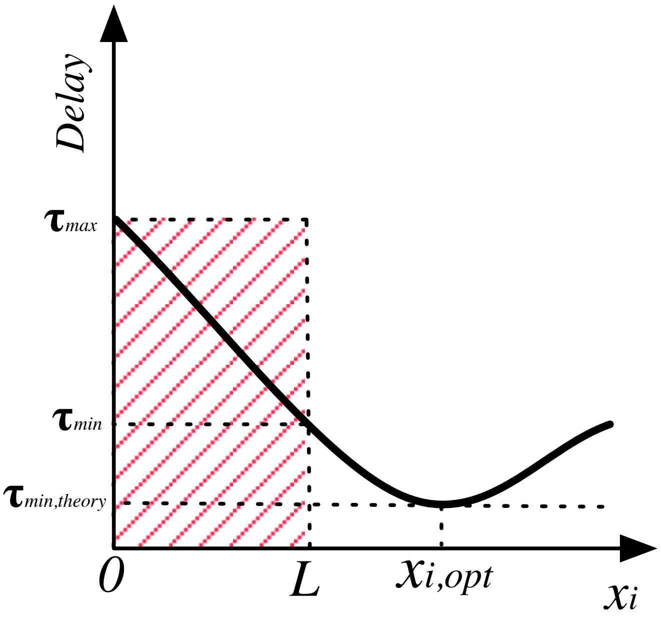


Figure 3.29 – Relation between x_i and delay of a RRAM-based multiplexer.

3.7.3 Programming Transistor Sizing Technique

As we see in (3.23), W_{prog} and R_{LRS} appear in almost every term of the polynomial, implying their tight relationship with delay of RRAM-based multiplexers. This part is devoted to determining the optimal value of W_{prog} and R_{LRS} in the goal of minimizing the delay τ.

As shown in equations (3.9) (3.11), the product of the R_{LRS} of RRAM and the programming transistor size W_{prog} is a function of programming voltage:

$$R_{LRS} = \frac{g(V_{prog})}{W_{prog}} \tag{3.26}$$

Note that the product $R_{LRS}W_{prog}$ is a constant under a specific V_{prog}.

With Equation (3.26), Equation (3.23) is simplified to be related to W_{prog} only. Since our target is to determine the optimal values of variables W_{prog}, we only focus on the terms involving W_{prog}:

$$\begin{aligned}\tau &= h(L, x_i, n, m, R_{inv}, C_{inv}, C_{trans})\\ &+ [4n \cdot R_{inv}C_{trans} + 2n^2 LR_{\square}C_{trans}] \cdot W_{prog}\\ &+ g(V_{prog})[nC_L + m\frac{n(n+1)}{2}(C_P + LC_{\square}) - C_{\square}\sum_{i=1}^{n}(mi - i + 1)x_i] \cdot \frac{1}{W_{prog}},\end{aligned} \tag{3.27}$$

where $h(L, x_i, n, m, R_{inv}, C_{inv}, C_{trans})$ is the sum of terms without W_{prog}.

According to (3.27), the relation between the n-stage multiplexer delay τ and the width of the programming transistor W_{prog} is depicted in Fig. 3.30.

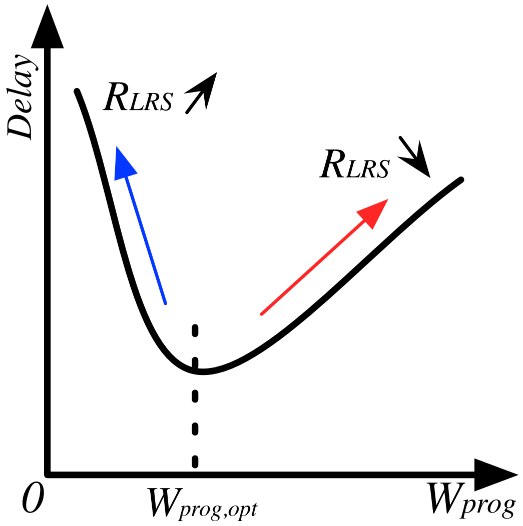


Figure 3.30 – Relation between W_{prog} and delay of a RRAM-based multiplexer.

When W_{prog} is small, the delay increases due to the large R_{LRS} of RRAM. When W_{prog} is large, the delay increases as well. Indeed, while the R_{LRS} is reduced, large parasitic capacitances are introduced by the programming transistors and limit the performances. Therefore, as shown in Fig. 3.30, there exists an optimal $W_{prog,opt}$ giving the best performances by trading off the R_{LRS} with the parasitic capacitances from the programming transistors.

Equation (3.27) reaches minimum value (best delay) when:

$$W_{prog,opt} = \sqrt{\frac{g(V_{prog})[nC_L + m\frac{n(n+1)}{2}(C_P + LC_\square) - C_\square \sum_{i=1}^{n}(mi - i + 1)x_i]}{4n \cdot R_{inv}C_{trans} + 2n^2 LR_\square C_{trans}}} \tag{3.28}$$

In FPGA routing architecture, the number of stages and the number of inputs of multiplexers are diverse. As Equation 3.28 depends on the n and m of the multiplexer, using a uniform size of programming transistors[26, 9, 27, 8] does not ensure the best performance. To achieve the best performances, the multiplexers in FPGA should have different $W_{prog,opt}$.

If we consider the optimal $x_i = L$ as explained in Section 3.7.2, the $W_{prog,opt}$ can be simplified to

$$W_{prog,opt}|_{x_i=L} = \sqrt{\frac{g(V_{prog})[2C_L + (n+1)mC_P + (n-1)LC_\square]}{8 \cdot R_{inv}C_{trans} + 4nLR_\square C_{trans}}} \tag{3.29}$$

Note that $W_{prog,opt}$ is always larger than zero and lies in the valid range of $W_{prog} \in [1, \infty)$.

Since the Elmore delay is an approximation of the delay, the estimated $W_{prog,opt}$ in (3.28) may not always guarantee the best delay. In practice, the best $W_{prog,opt}$ can be found by sweeping W_{prog} in electrical simulations. In Section 3.8.3, we will examine the effect of programming transistor sizing technique.

As input sizes and fan-out loads of multiplexers are diverse in the context of FPGA architectures, the choice of multiplexing structure, transistor sizes and physical locations of RRAMs should be well optimized by considering their architecture context. As a result, the two optimizing techniques are effective methods to achieve optimal performance for multiplexers located in different blocks of a FPGA architecture. Note that the design space of 4T1R-based multiplexer could be even larger than what we have investigated here. For instance, in this thesis, we assume that W_{prog} and R_{LRS} are uniform in a 4T1R-based multiplexer. Actually, W_{prog} and R_{LRS} can be various in different stages, leading to more optimizing opportunity. We leave these as part of our future work.

3.8 Experimental Results

In this section, we will verify the conclusions drawn by our analytical comparison with electrical simulations and further evaluate the performance of the proposed multiplexers. We first explain our experimental methodology. Then, we show and comment the transient behavior of 4T1R-based multiplexers, and finally we compare the area, delay and power between different 4T1R-based and CMOS multiplexer topologies.

3.8.1 Experimental Methodology

We consider a RRAM technology [114] with programming voltages $V_{set} = |V_{reset}| = 1.1V$ and a maximum current compliance of $I_{set} = |I_{reset}| = 500\mu A$. The lowest achievable on-resistance R_{LRS} of a RRAM is $2.2k\Omega$ while the off-resistance R_{HRS} is $23M\Omega$. The parasitic capacitance of a RRAM C_P is estimated to be $13.2aF$ by considering that the RRAMs are embedded in the *MET1* and *MET2* vias of our considered technology. The pulse width of a programming voltage in both set and reset processes is set to be $200ns$. Stanford RRAM compact model [130, 131] is used to model the considered RRAM technology. The TSMC 40nm technology is used in the circuit designs of datapath logics and 4T1R programming structures. Both datapath circuits and the 4T1R programming structures are built with standard logic transistors ($W/L = 140nm/40nm$). The standard logic transistors have a nominal working voltage $V_{DD} = 0.9V$, and can be overdriven to $1.2V$ while staying in their reliability limits.

Transmission gates are implemented with a pair of minimum-width *n*-type and *p*-type logic transistor. Input and output inverters are sized to 3× minimum width in order to resist the parasitics of metal wires. Delay and power results are extracted from HSPICE [47] simulations. The datapath V_{DD} is swept from $0.7V$ to $0.9V$ with a step $0.1V$, in order to study the trade-off between delay and power in sub/near-V_t regime. The programming voltage V_{prog} is selected to be $2.4V$, respecting to the physical design limits, discussed in Section 3.6.5.

The comparison baseline is selected from the CMOS multiplexer topologies in Fig. 2.14 and Fig. 2.15 in terms of best delay. When input size N is lower or equal than 10, we consider one-level CMOS multiplexers as baseline. When input size N is larger than 10, our baseline becomes a two-level CMOS multiplexer. As for 4T1R-based multiplexers, we consider one-level, two-level and tree-like structures for comparison on area, delay and power.

3.8.2 Transient Analysis

In order to validate the analytical comparisons in Table 3.2, we perform transient simulations for 4T1R-based multiplexers, which consist of two phases: (1) the programming phase, where set and reset operations are made to validate the RRAM programming strategy; and (2) the datapath operation phase, where we verify if the multiplexer is functionally correct. Without loss of generality, we focus on a representative example: a 2-input one-level 4T1R-based multiplexer (consider $N = 2$ in Fig. 3.22). Such transient analysis was conducted for every 4T1R-based multiplexer. Before programming, we initialize a 4T1R-based multiplexer in Fig. 3.22 as follows: RRAMs R_A and R_B are formed and configured to HRS and LRS respectively. During the programming phase depicted in Fig. 3.31(a), R_B is first reset to HRS by a reset procedure, then R_A is set to LRS by a set cycle. Fig. 3.31(a) illustrates that both R_A and R_B can be set or reset successfully according to the changes in programming currents I_{vdd0} and I_{vdd1}. Between the programming phase and operating cycles, there are a few idle cycles during which programming transistors are all turned off. After then, input pulses are generated sequentially to the two inputs, as shown in Fig. 3.31(b). We see that the multiplexer is functionally correct, as $in[0]$ is propagated to the output while $in[1]$ is blocked. Transient analysis also verifies that RRAMs can be programmed correctly without interfering each other.

3.8.3 Best W_{prog} for RRAM-based Multiplexers

As explained in Section 3.7, the sizing of programming transistors can significantly impact the delay and power number of RRAM-based multiplexers. In this section, we study the impact of W_{prog} on the delay of the improved 4T1R-based multiplexers through simulation results. Throughout this thesis, W_{prog} is expressed with the number of minimum width transistors. For each 4T1R-based multiplexer structure (one-level, two-level and tree-like), we sweep W_{prog} from 1 to 3 with a step of 0.2, in order to identify the optimal W_{prog} in terms of best delay. Fig. 3.32 shows the delay difference of the improved one-level, two-level and tree-like 4T1R-based multiplexers ($x = L$) when input size is 50. A proper W_{prog} indeed can reduce the

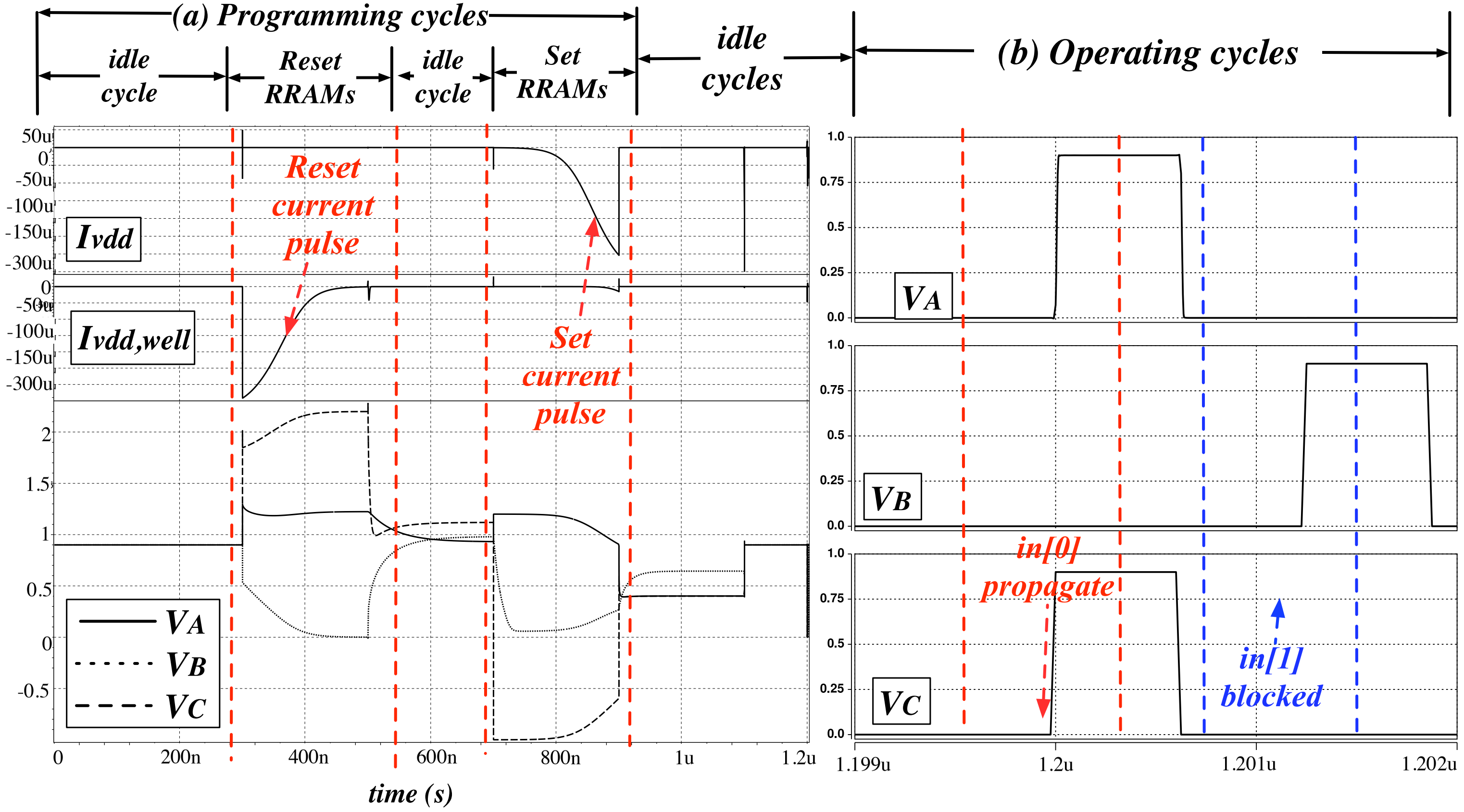


Figure 3.31 – Transient analysis of a 2-input 4T1R-based multiplexer in Fig. 3.22(a): (a) signal waveforms of programming phase; (b) signal waveforms of operation.

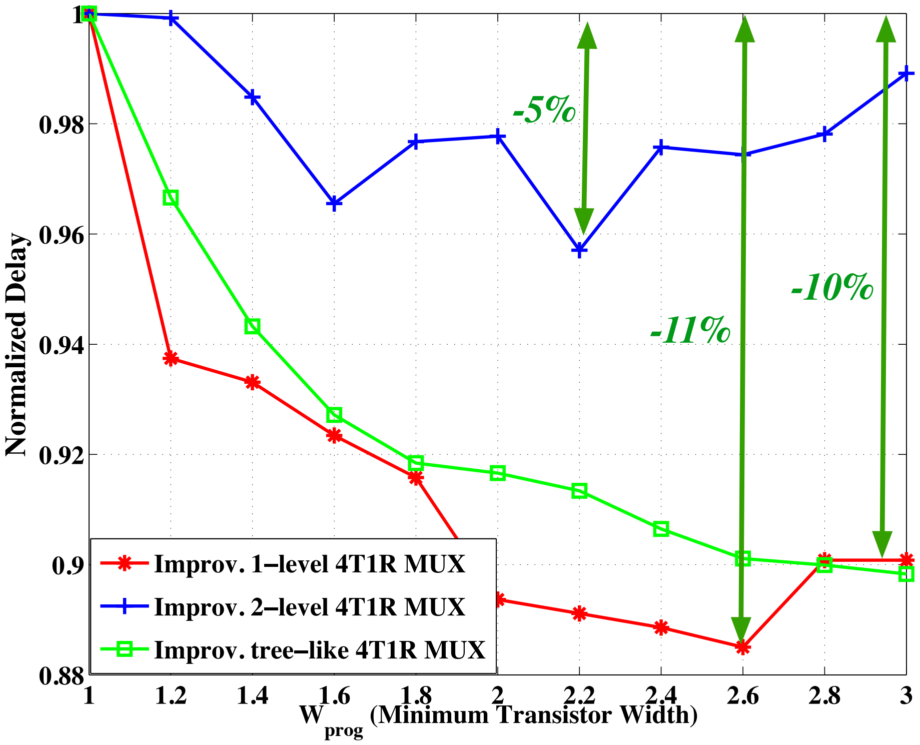


Figure 3.32 – Impact of W_{prog} on the delay of 50-input improved 4T1R-based multiplexers ($x = L$).

delay of 4T1R-based multiplexers by 5%-11%. Fig. 3.32 shows that the best W_{prog} depends on the multiplexing structure because of different n and m, as predicted in Equation (3.29). More than multiplexing structures, Fig. 3.33(a) and (b) present the best W_{prog} is strongly dependent on many other design factors, such as input size and V_{DD}. As depicted in both Fig. 3.33(a) and (b), the best W_{prog} basically increases when input sizes grows. This is consistent to the prediction in Equation (3.29), where optimal W_{prog} is positively related to m. In general, optimal W_{prog} of tree-like multiplexers are larger than two-level and one-level multiplexers, which validates the dependency of $W_{prog,opt}$ on the number of stages n shown in Equation (3.29). Fig. 3.33(b) studies the relation between best W_{prog} and V_{DD}, considering one-level multiplexers. In most cases, operating in near-V_t regime, such as $V_{DD} = 0.7V$ leads to a smaller $W_{prog,opt}$ than nominal working voltages. Indeed, when V_{DD} is decreased, R_{inv} increases due to the degrading current density, leading to a smaller $W_{prog,opt}$ as shown in Equation (3.29).

In short, we see that in Fig. 3.33(a) and (b), the optimal W_{prog} ranges from 1 to 3, strongly influenced by design choices. In addition to delay, the choice of W_{prog} impacts strongly on both area footprint and power consumption. Therefore, to achieve better trade-off in area, delay and power, the optimal W_{prog} can also be determined with respect to various metrics, such as *Area-Delay Product* (ADP) and *Power-Delay Product* (PDP). In the rest of this chapter, W_{prog} of each 4T1R-based multiplexer is properly sized to achieve best delay metric.

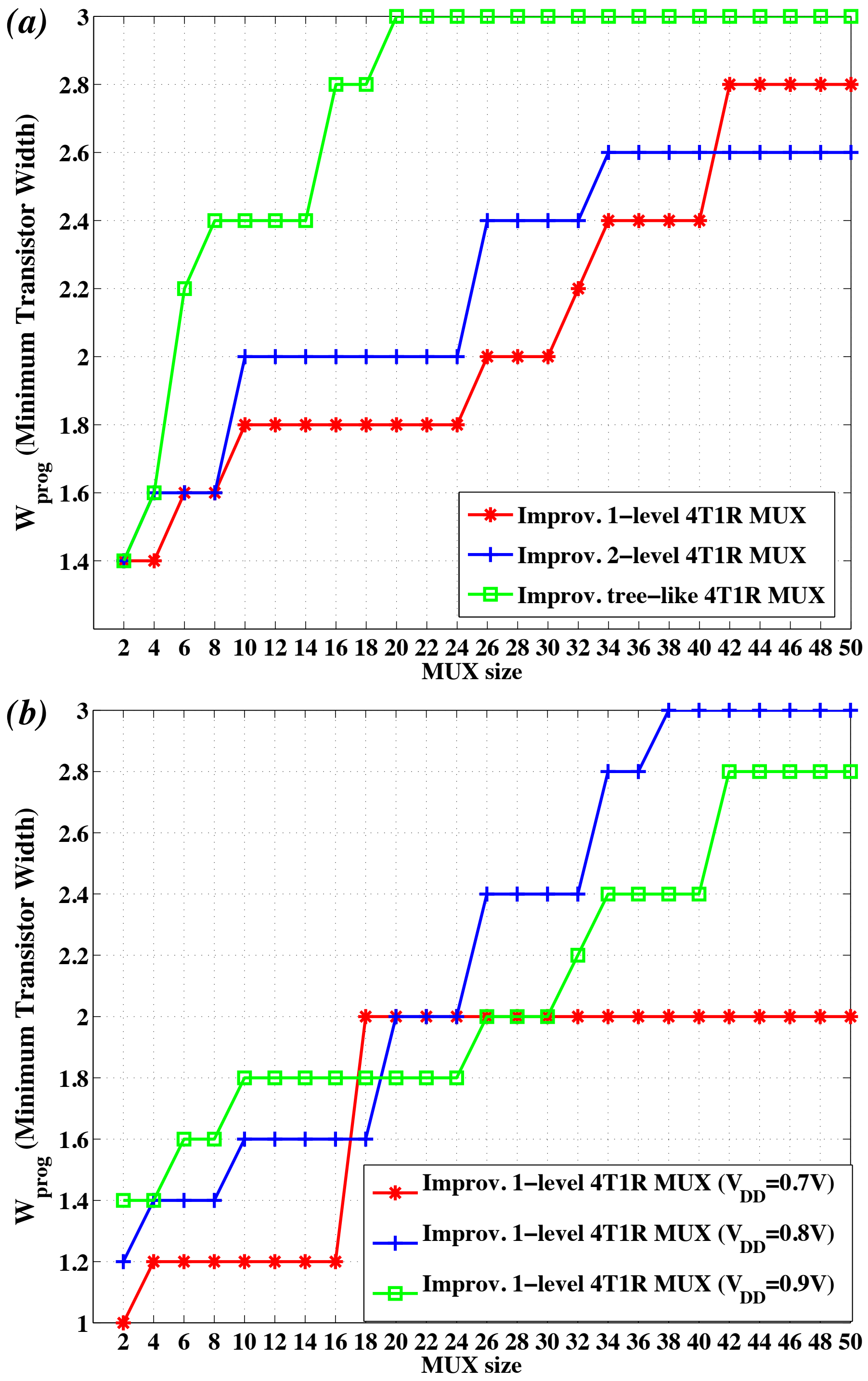


Figure 3.33 – Two case studies on the best W_{prog} of improved 4T1R-based multiplexers ($x = L$): (a) impact of the multiplexing structures when $V_{DD} = 0.9V$ (b) impact of V_{DD}.

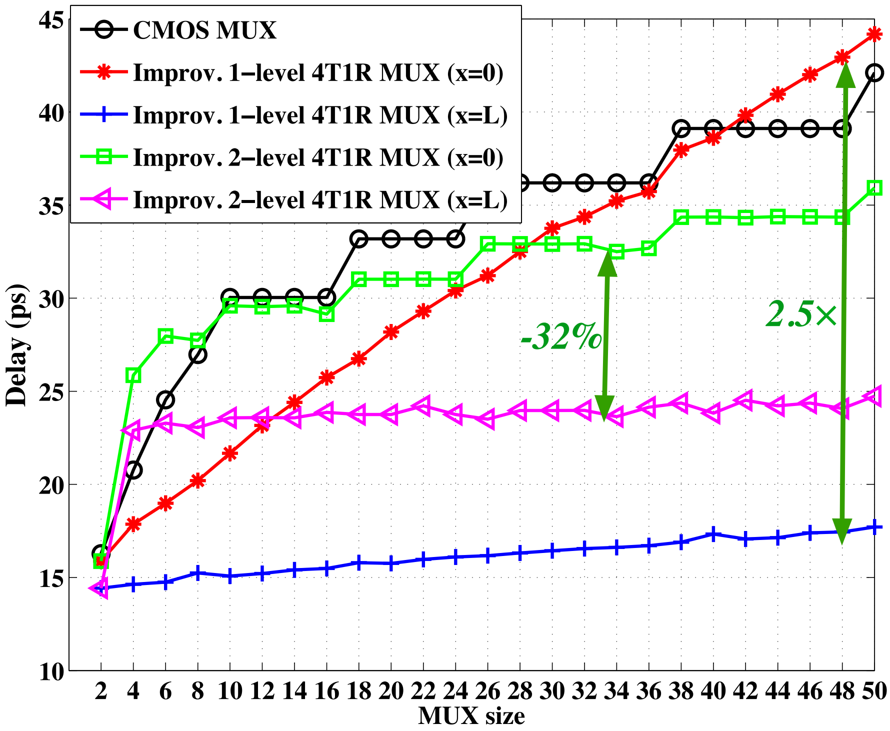


Figure 3.34 – Delay comparison of improved 4T1R-based multiplexers featured by $x = 0$ and $x = L$.

3.8.4 Optimal RRAM Location

As shown in Equation 3.25, the location of RRAMs can influence the delay of 4T1R-based multiplexers. From the consider design kit, we extract process parameters $L = 2.5\mu m$, $R_{inv} = 4.5k\Omega$, $R_{\square} = 2.1\Omega/\mu m$ and $C_{\square} = 72.4aF/\mu m$. According to Equation 3.25, the best location of the RRAMs is $x_{opt} = L$. Therefore, in this part, we study only two locations for RRAMs : $x = 0$ and $x = L$. Fig. 3.34 compares the delay of one-level and two-level improved 4T1R-based multiplexers with different locations of RRAMs $x = 0$ and $x = L$. The improved designs with $x = L$ significantly reduce the delay by 35% – 2.5× as compared to the cases of $x = 0$. In particular, $x = 0$ causes that delay of RRAM-based multiplexers linear to input sizes similar to CMOS counterparts, while $x = L$ can guarantee that delay of RRAM-based multiplexers is almost independent from input size. To be intuitive, such delay characteristic can be explained as follows. In the cases of $x = 0$, long metal wires are all connected to the output nodes of multiplexing structure (See node C in Fig. 3.22(a)). As a result, the parasitic resistances and capacitances at the output node stack at the output node, being linear to the input size. Consequently, the delay of improved 4T1R-based multiplexers $x = 0$ is linear to the input size. Differently, in the case of $x = L$, long metal wires are connected to each input inverter and the parasitics at output node is only impacted by the intrinsic capacitance of RRAMs. Therefore, we see in Fig. 3.34 that the delay of improved 4T1R-based multiplexers is almost independent on the input size.

Note that, thanks to such outstanding feature, improved 4T1R-based multiplexers with large input sizes can be as delay efficient as smallest ones, encouraging the use of large multiplexers in FPGAs. This potentially opens opportunities in optimizing FPGA architectures, which will be explored in Chapter 5. In the rest of this thesis, we consider the improved design with $x = L$ in the comparison with CMOS multiplexers.

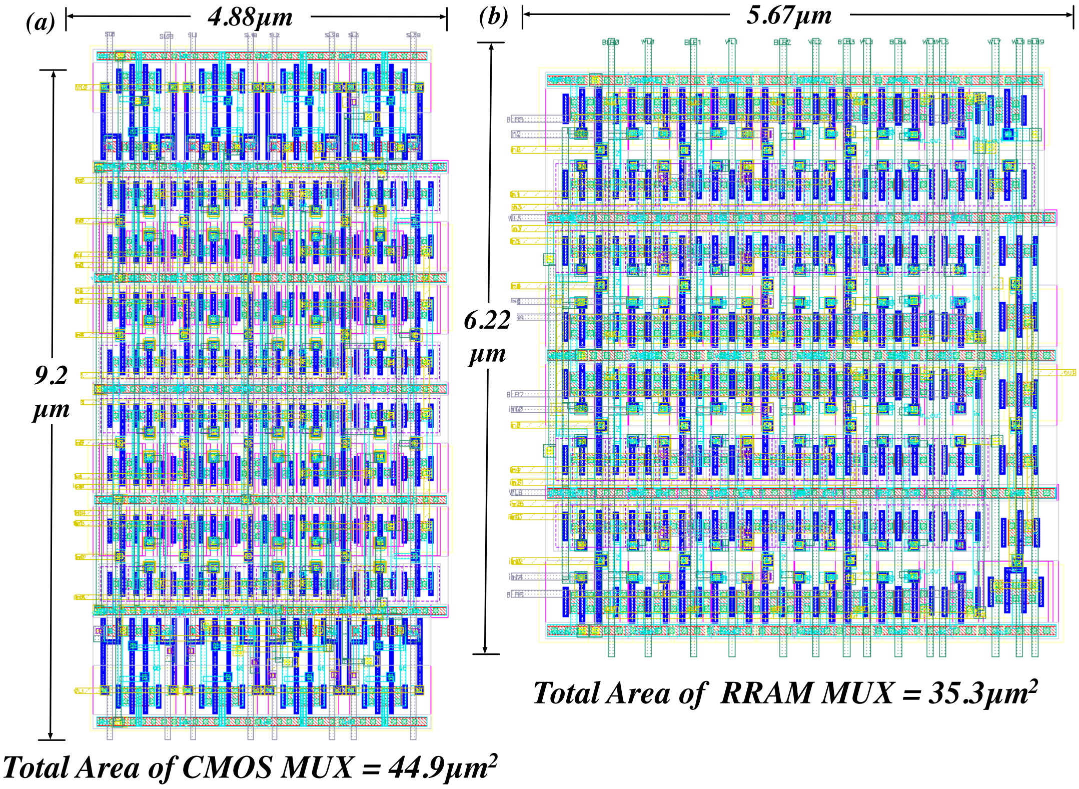

(a)
4.88µm
9.2 µm
Total Area of CMOS MUX = 44.9µm²
(b)
5.67µm
6.22 µm
Total Area of RRAM MUX = 35.3µm²

Figure 3.35 – Layout of 16-input multiplexers: (a) CMOS two-level structure; and (b) 4T1R-based two-level structure.

3.8.5 Area Comparison

In order to properly study the physical area of the proposed structure, i.e., considering routing, well organization etc., and draw fair area comparisons with regular CMOS, we realized the layouts of a 16-input two-level CMOS multiplexer and a 16-input two-level 4T1R-based multiplexer with a semi-custom design flow, as depicts in Fig. 3.35(a) and (b) respectively. Since the different wells can be efficiently shared among multiplexers as shown in Fig. 3.26, the layout of 4T1R-based multiplexer consists of the programming structures and input inverters (*MUX0* in Fig. 3.26) in a regular well. The output and associated programming structure of another multiplexer (*MUX1* in Fig. 3.26) can be shared in this same well. The output inverter and associated programming structure of *MUX0* will be located in a *deep N-well* which also contains programming structure and input inverters of another multiplexer. CMOS multiplexers must employ SRAMs to store their configuration bits, while 4T1R-based multiplexers eliminate the use of SRAMs as their configuration bits are stored in RRAMs. To access either the SRAMs or the RRAMs, we assume a memory bank organization, i.e., using parallel word lines and bit lines. Since CMOS and 4T1R-based multiplexers have similar number of configuration bits, the area of their memory banks are similar and are not included in their layouts. The benefit on removing SRAMs leads to that a 4T1R-based multiplexer ($35.3\mu m^2$) is 21% smaller than its CMOS counterpart ($44.9\mu m^2$). We believe that the area comparison between 16-input multiplexers is representative and also its conclusive trend is also valid for multiplexers with other sizes.

3.8.6 Delay Improvements

Fig. 3.36(a) compares the delay of CMOS multiplexers and the improved 4T1R-based multiplexers with the different structures under analysis. Note that naive 4T1R and 2T1R-based multiplexers are also evaluated with electrical simulations. Due to a low driving current density, RRAM programming of the naive 2T1R-based multiplexers is regarded as a failure because programming structures cannot drive enough current through RRAMs. As a result, the RRAM LRS becomes too high and the multiplexer performance degrades significantly. The performance of the naive 2T1R-based multiplexers are more than 5× worse than the improved 4T1R-based multiplexer and best CMOS multiplexers. To keep a proper scale of axis x and y, we do not plot them in Fig. 3.36(a). In the case of naive 4T1R multiplexers, we consider $W_{prog} = 4$ in order to compensate the loss in programming current due to the input inverters in Fig. 3.21. Such large W_{prog} enables success RRAM programming but at cost of large parasitics of programming transistors. Consequently, the performance of naive 4T1R-based multiplexers is 2.6× worse than the improved ones. In contrast, the improved 4T1R-based multiplexers with one-level, two-level and tree-like structures can guarantee RRAM configuration successful even when W_{prog} is minimized. In the considered input sizes, one-level structure performs better in delay than two-level and tree-like structures due to its smaller parasitic capacitances. One-level structures and two-level 4T1R-based multiplexers achieve up to 2.4× and 42% delay improvements respectively, as compared to their CMOS counterparts. Note that even when the input size is small, i.e., $N = 2$, one-level 4T1R-based multiplexers have similar performance than CMOS implementations.

We also investigate the performance of the multiplexers in the near-V_t regime. As illustrated in Fig. 3.36(b), CMOS multiplexers suffer from 2.25× delay degradation when V_{DD} decreases from $0.9V$ to $0.7V$. However, because, unlike transistors, the resistances of RRAMs are not affected by a reduction of V_{DD}, one-level 4T1R-based multiplexers keep a high-performance-level even in the near-V_t regime. When $V_{DD} = 0.7V$, one-level 4T1R-based multiplexers improve delays by up to 3×, as compared to CMOS multiplexer. Note that, when compared to CMOS multiplexers operating at $V_{DD} = 0.9V$, one-level 4T1R-based multiplexers operating

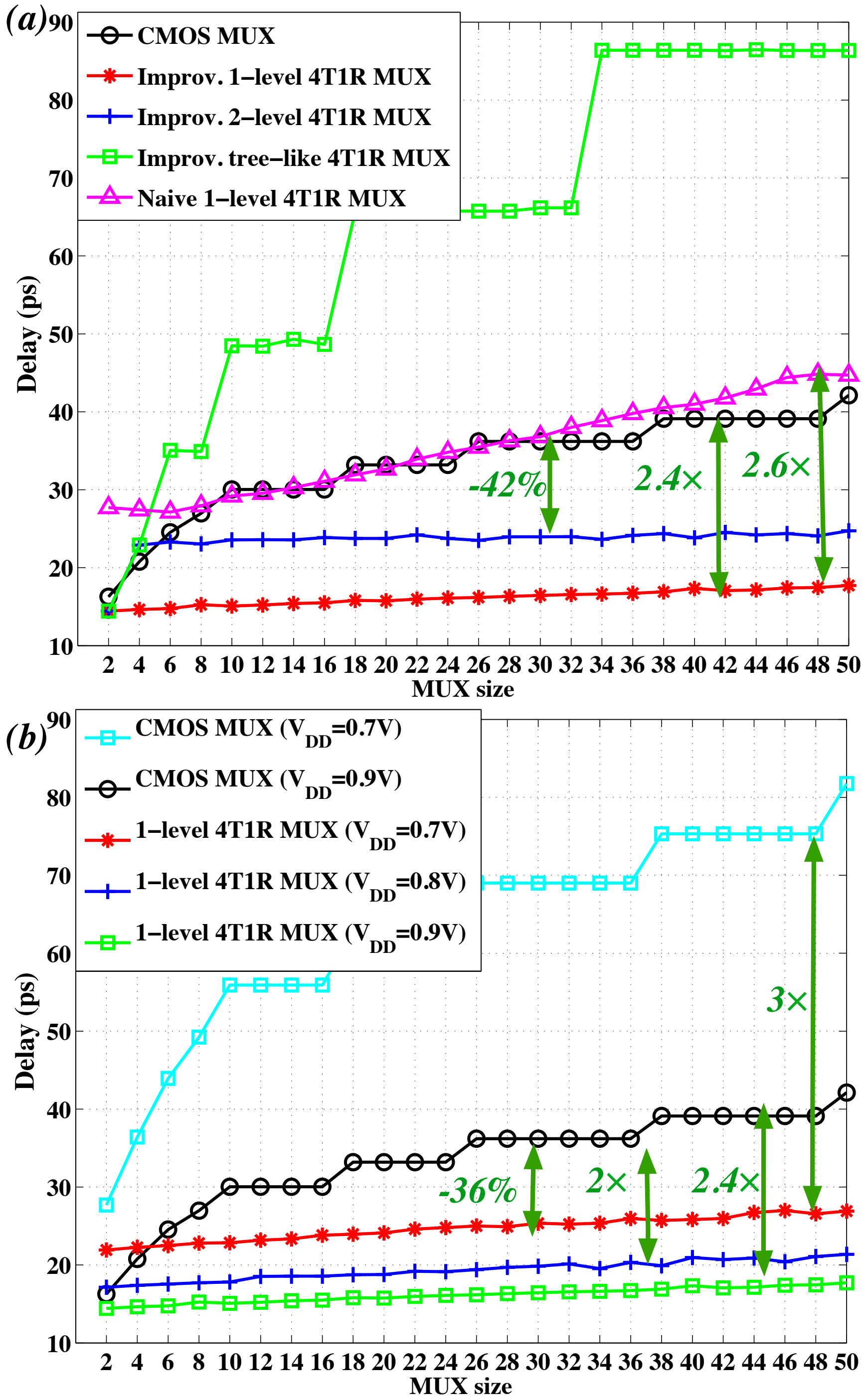


Figure 3.36 – Delay comparison between CMOS and 4T1R-based multiplexers: (a) delay improvements of one-level, two-level and tree-like structures ($V_{DD} = 0.7V$); (b) delay efficiency of one-level structure at near V_t regime.

with $V_{DD} = 0.7V$ outperform up to 36% in delay.

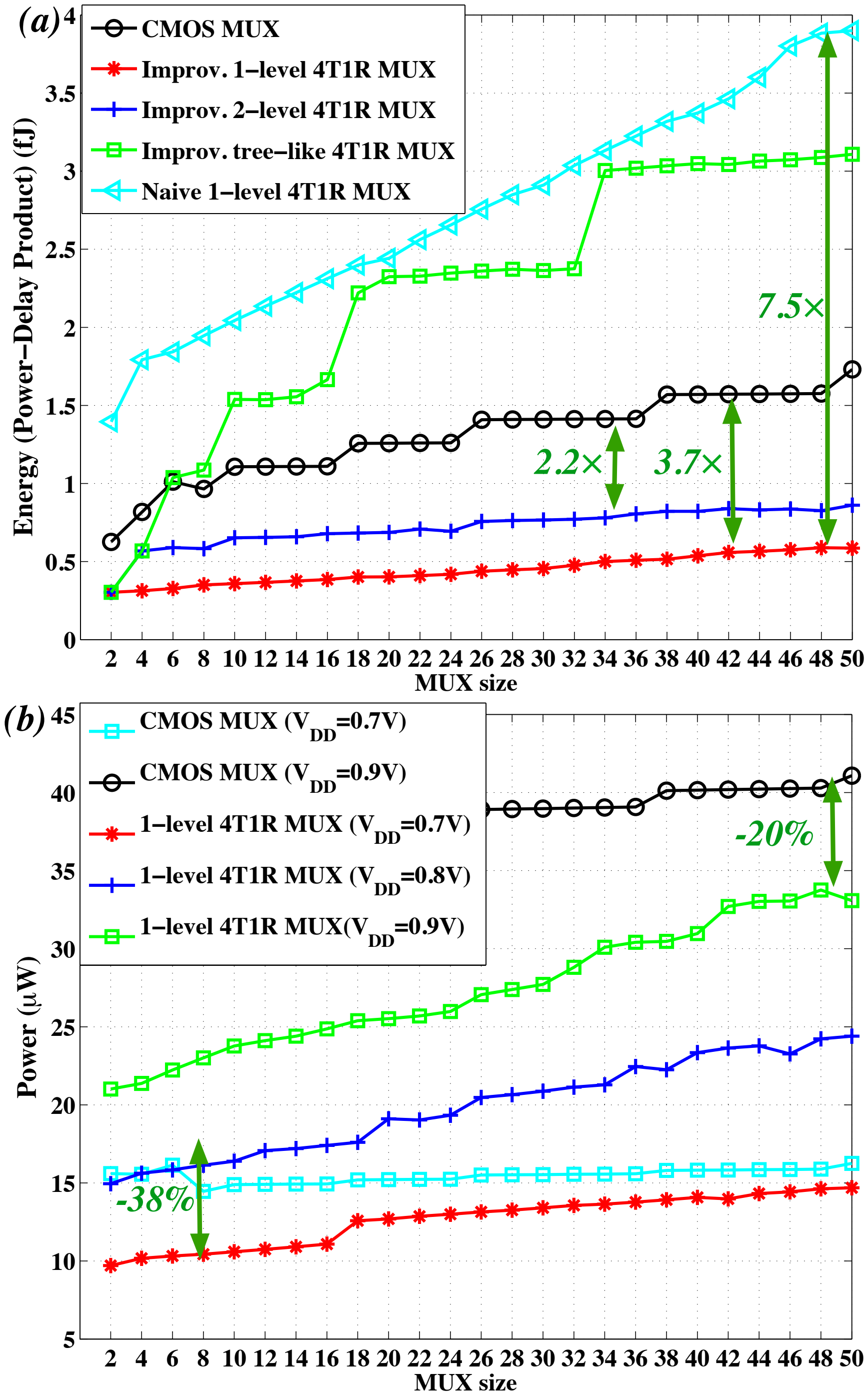


Figure 3.37 – Power comparison between CMOS and 4T1R-based multiplexers: (a) energy improvements of one-level, two-level and tree-like structures ($V_{DD} = 0.7V$); (b) power reduction of one-level structure at near V_t regime.

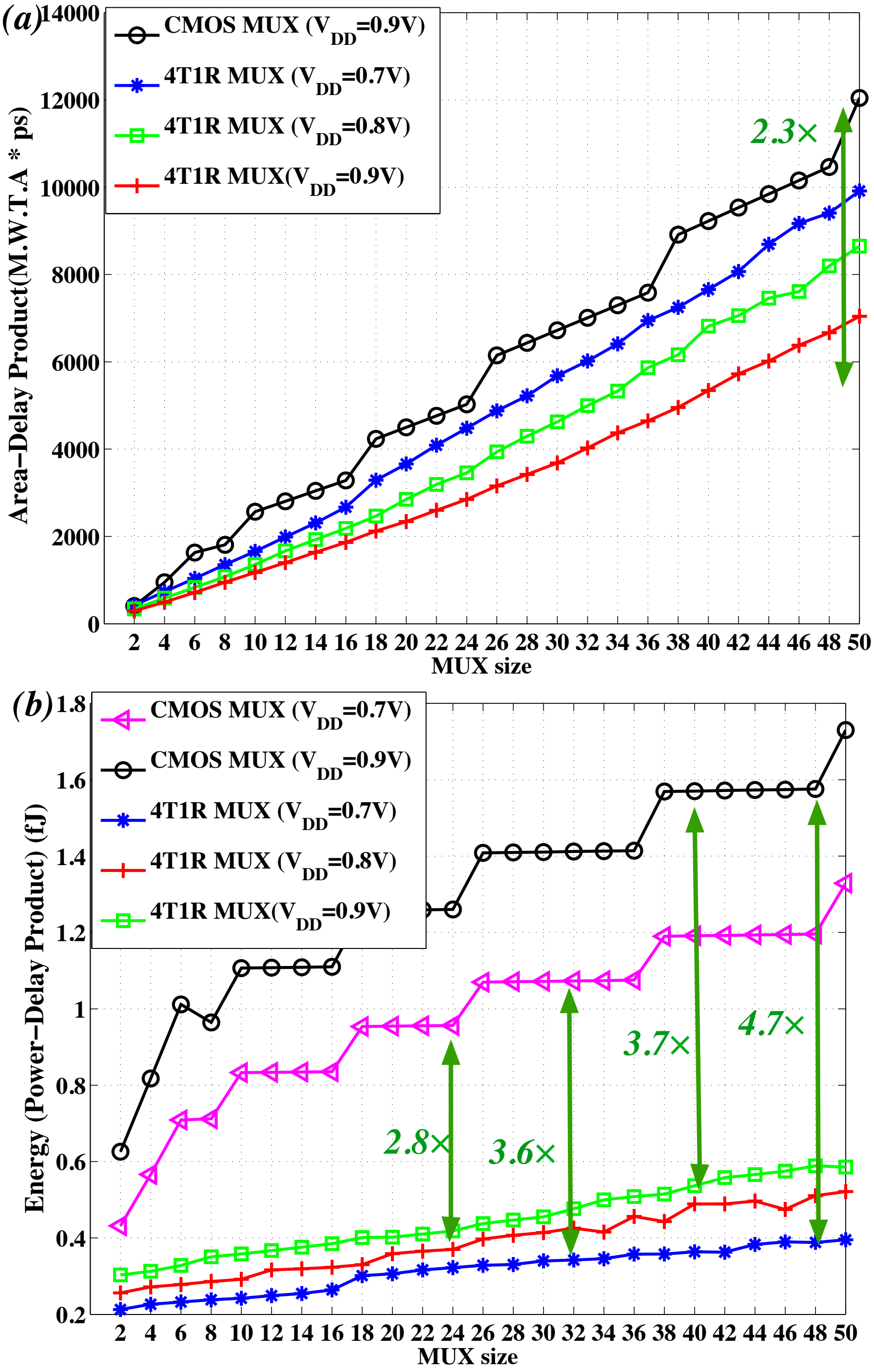


Figure 3.38 – Comparison between CMOS multiplexers and 4T1R-based multiplexers: (a) Area-Delay Product; (b) Power-Delay Product.

3.8.7 Energy and Power Benefits

Fig. 3.37(a) shows the energy efficiency of naive one-level 4T1R-based multiplexers and 4T1R-based multiplexers with different improved structures. Note that naive 4T1R-based multiplexers consumes 7.5× more energy than the improved one-level 4T1R-based multiplexers due to the use of $W_{prog} = 4$. In the considered range of input sizes, a one-level structure multiplexer performs better in terms of energy consumption, bringing up to 3.7× reduction compared to CMOS multiplexers, thanks to the smaller parasitic capacitances. 4T1R-based multiplexers are not only efficient in energy but also in power, as shown in Fig. 3.37(b). At nominal $V_{DD} = 0.9V$, one-level 4T1R-based multiplexers reduce power by 20% as compared CMOS multiplexers. In near-V_t regime, i.e., $V_{DD} = 0.7V$, the power reduction of one-level 4T1R-based multiplexers is 38% as significant as $V_{DD} = 0.9V$. Note that, the 4T1R-based multiplexers operating at $V_{DD} = 0.7V$ can benefit power improvement up to 4× as compared to CMOS multiplexers at nominal $V_{DD} = 0.9V$, and such power reduction is achieved along with significant delay improvements.

3.8.8 Area-Delay and Power-Delay Products Analysis

To explore the inherent trade-offs with area, delay and power, we compare *Area-Delay Product* (ADP) and *Power-Delay Product* (PDP) of CMOS and 4T1R-based multiplexers, as shown in Fig. 3.38. Similar to CMOS multiplexers, we select the best structure for 4T1R-based multiplexers with varying input sizes, in terms of best delay. When input size ranges from 2 to 50, we consider one-level structure. Since 4T1R-based multiplexers reduce both area and delay significantly, *Area-Delay Product* (ADP) of 4T1R-based multiplexers can be up to 2.3× more efficient than CMOS multiplexers than CMOS multiplexers, as illustrated in Fig. 3.38(a). Since 4T1R-based multiplexers are more delay and power efficient than CMOS multiplexers in near-V_t regime, *Power-Delay Product* (PDP) of 4T1R-based multiplexer improves over 4.7× the one of CMOS multiplexers, as shown in Fig. 3.38(b). $V_{DD} = 0.7V$ guarantees the best PDP for 4T1R-based multiplexers. In summary, 4T1R-based multiplexers are delay and power efficient at both nominal V_{DD} and near-V_t regime.

3.9 Impact of Process Variations of RRAMs

RRAMs are more susecptible to device variations than transistors. As their mechanism is physically stochastic, there is a large observed cycle-to-cycle variability[1]. The variations on RRAM parameters, such as V_{set} and V_{reset}, could lead to a degradation of RRAM-based multiplexers performance. Therefore, it is necessary to understand, for a given technology node, what is the range of variations that the RRAM multiplexers can tolerate without significant degradation in delay and power. In this section, we study the effect of three representative RRAM parameters: C_P, V_{set} and V_{reset}, coupled with a commercial 40nm technology.

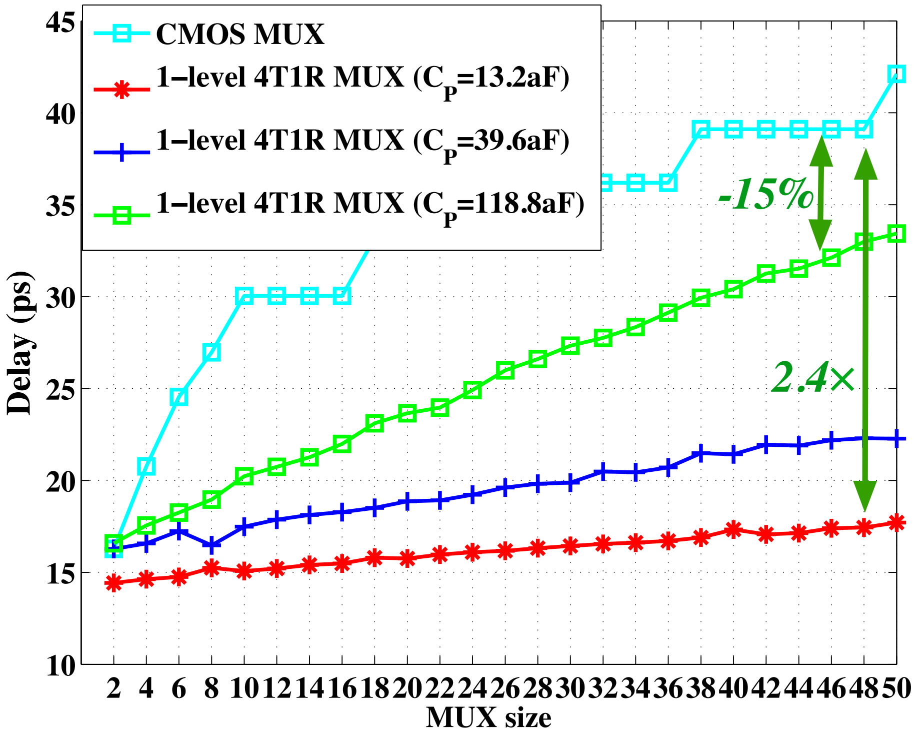


Figure 3.39 – Impact of parasitic capacitance of RRAM C_P on the delay of one-level 4T1R-based multiplexers ($V_{DD} = 0.9V$).

3.9.1 Impact of Variations on C_P

As shown in equation 3.23, the parasitic capacitance of RRAM C_P is one of the crucial factor impacting the delay of 4T1R-based multiplexers. A large C_P introduces more capacitance into datapath and therefore negatively influence the delay of 4T1R-based multiplexers. As presented in Fig. 3.39, the delay of one-level 4T1R-based multiplexers degrades as C_P is increased from $13.2aF$ (the default value used in this thesis) to $118.8aF$. A variation on C_P can indeed reduce the performance gain of 4T1R-based multiplexers from 2.4× to only 15%. More importantly, an increased C_P causes that the delay of 4T1R-based multiplexers becomes strongly linear to the input size, similar to CMOS multiplexers. Therefore, the variation on C_P should be well controlled as it significantly impact not only the performance improvement but also the performance characteristic of 4T1R-based multiplexers.

Note that, in this part, we assume that the increase in C_P does not impact other device parameters of RRAMs, i.e., R_{LRS}. As explained in Section 2.1.1, a increased C_P can lead to a smaller R_{LRS}, which may potentially limit the delay degradation on 4T1R-based multiplexers. Hence, in practice, the impact of C_P on 4T1R-based multiplexers may be less serious than that shown in Fig. 3.39.

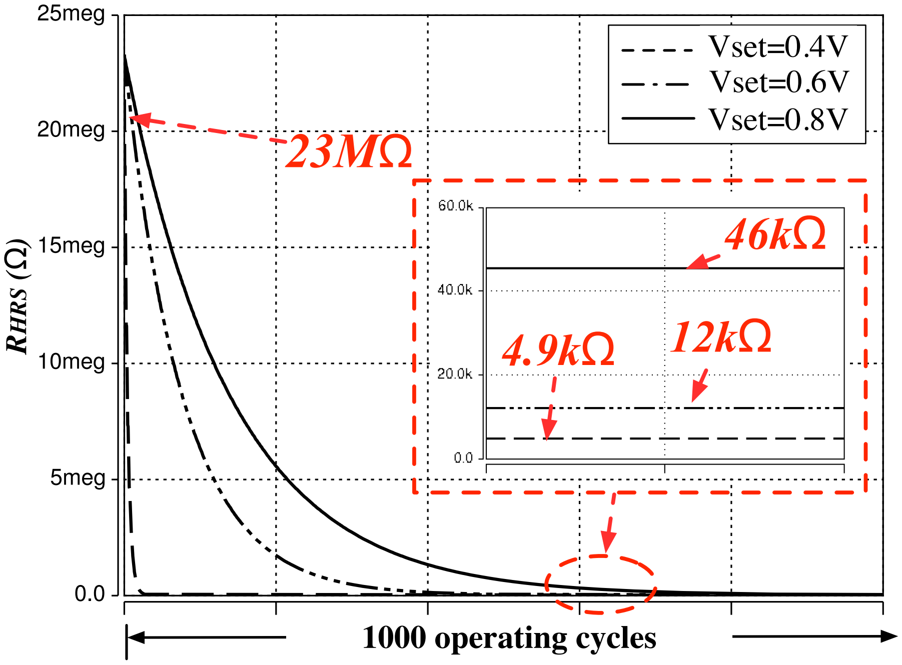


Figure 3.40 – R_{HRS} degradation when $V_{set} = \{0.4, 0.6V, 0.8V\} < V_{DD} = 0.9V$.

3.9.2 Impact of Variations on V_{set}

Process variations on V_{set} may cause $V_{set} < V_{DD}$, where RRAMs could be parasitically set during operation. Take the example in Fig. 3.31(b), during regular operation (highlighted in red), where $V_A = GND, V_B = V_{DD}$ and $V_C = GND$, the voltage drop across RRAM R_B could be large enough to trigger a set process. The RRAM R_B in HRS could be gradually set to LRS after a certain amount of time. In this part, we consider three representative cases of RRAM technologies where V_{set} are $0.4V$, $0.6V$ and $0.8V$ respectively, which are smaller than $V_{DD} = 0.9V$. Using electrical simulations, we run a fatigue test for a 2-input RRAM multiplexer by running one thousands operating cycles, whose input waveforms are similar to the one shown in Fig. 3.31(b). Fig. 3.40 illustrates the degradation trend of R_{HRS} of RRAM R_B, where R_{HRS} decreases gradually from $23M\Omega$ to $4.9-46k\Omega$ and then no further degradation is observed. The lower bound of R_{HRS} degradation remains to be $4.9-46k\Omega$ even when 100 thousands and $1M$ operating cycles are further applied. The existence of a lower bound of R_{HRS} can be explained as following: The voltage at node C in Fig. 3.22(a) is dependent on the resistance of R_B,

$$V_C = V_{DD} \cdot \frac{R_{R_B}}{R_{R_A} + R_{R_B}}, \tag{3.30}$$

where R_{R_A} and R_{R_B} represent the resistances of RRAM R_A and R_B in Fig. 3.22(a) respectively. As R_{R_B} degrades, V_C decreases as well, leading to the voltage drop across RRAM R_B decreases. When the voltage drop across RRAM R_B is reduced to be lower than V_{set}, the parasitic set process is stopped. The lower bound of degradation is independent from the number of operating cycles but is related to V_{set}. In Fig. 3.40, we see that a high $V_{set} = 0.8V$ leads to less degradation on R_{HRS} than $V_{set} = 0.4V$. Note that the degradation on R_{HRS} could cause significant leakage overhead [114]. In this paper, we consider a 20% margin between nominal V_{DD} and V_{set}. Additionally, the excellent performance of 4T1R-based multiplexers in near-V_t regime allows the use of low V_{DD}, i.e., $= 0.7V$, further increasing the margin to 60%. We believe such margin is sufficient to resist V_{set} variations.

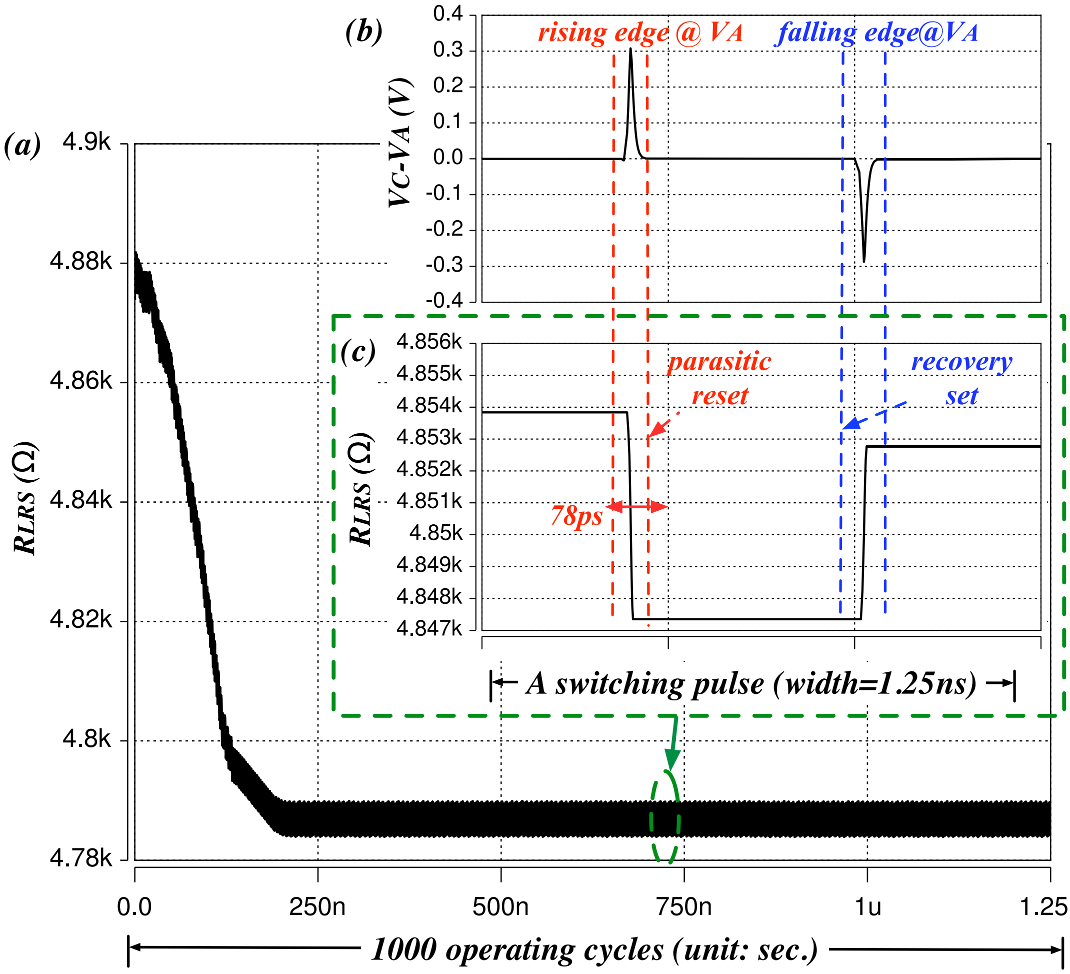


Figure 3.41 – (a) R_{LRS} degradation when $V_{reset} = 0.3V$ over $1k$ operating cycles; (b) Voltage across a RRAM in LRS (V_A and V_C in Fig. 3.22(a)) during operation; and (c) R_{LRS} degradation when $V_{reset} = 0.3V$ in a switching cycle.

3.9.3 Impact of Variations on V_{reset}

A parasitic reset process could also happen to a RRAM in LRS when the voltage drop across RRAM $|V_{RRAM}| < |V_{reset}|$. However, during normal operation, the voltage drop across a RRAM is typically smaller than $0.3V$, as shown in Fig. 3.41(a), and the duration of such voltage drop is as short as $78ps$. Hence, as long as V_{reset} varies to be above $max\{V_C - V_A\}$, i.e., $= 0.3V$, a parasitic reset process can be fully avoided. Using electrical simulation, we consider $V_{reset} = 0.4V$, $0.5V$ and $0.6V$ in the same torture test as described in Section 3.9.2, and the resistances of RRAM in LRS remains unchanged in all the conditions. Even if V_{reset} is smaller than $max\{V_C - V_A\}$, R_{LRS} degradation is much less serious than R_{HRS}. Fig. 3.41(b) illustrates that when V_{reset} is below $0.3V$, the parasitic reset caused by a rising edge of V_A ($V_C > V_A$) can be partly recovered by a falling edge of V_A ($V_C < V_A$), resulting in a $\sim 10\Omega$ R_{LRS} degradation per operation cycle. However, as compared to nominal $V_{reset} = 1.1V$ considered in this paper, process variation can be well controlled to ensure $V_{reset} > 0.3V$ and thus parasitic reset can be fully avoided.

3.10 Summary

In this chapter, we investigated essential RRAM-based circuit designs for FPGA architectures. To the best of our knowledge, this is the first work contributing to systematical studies on the programming structures and efficient integrating RRAMs into routing multiplexers by considering physical design details. The proposed 4T1R programming structure and routing multiplexer design have profound impacts on the RRAM-based circuit designs and also FPGA architectures. We first studied the programming structures for RRAMs through both theoretical analysis and electrical simulations. The proposed 4T1R programming structure outperforms the widely-used 2T1R programming structure by a significant improvement of driving current density. Thanks to the significant advance in area efficiency, lowest achievable R_{LRS} and physical designs, the proposed 4T1R programming structure can be widely used in all the RRAM-based circuits, including but not limited to routing multiplexers. For instance, the 4T1R programming structure is adapted to non-volatile SRAM designs in Chapter 5. The methodologies in analyzing and boosting programming structure is rather general and can be extended to other non-volatile memory technology, e.g., *Phase Change Memory* [40]. This implies that the 4T1R programming structure can be exploited for other non-volatile memory technologies.

We then presented one-level, two-level and tree-like multiplexer designs based on the 4T1R programming structure, addressed the physical design challenges in RRAM-based circuit designs and analyze the impact of process variations. In addition, we proposed generic optimization techniques, i.e., programming transistor sizing and optimal RRAM location, which can significantly improve area, delay and power of RRAM-based multiplexers. Note that the methodologies in analyzing programming transistor sizing and optimal RRAM location are not limited to the proposed multiplexer design, but are rather general to all RRAM-based circuits. Electrical simulations demonstrate the superiority of 4T1R-based multiplexers over best CMOS multiplexers:
(1) their delay can be much less dependent on the input size.
(2) delay improvement is 2× and 3× when considering nominal and near-V_t working voltages

respectively.
(3) energy can be reduced by 2.8× and 3.7× when considering nominal and near-V_t working voltages respectively.
The outstanding performance of 4T1R-based multiplexers can lead to strong architecture impacts, including but not limited to FPGA architectures. For instance, multiplexers are also intensively used in *Network-On-Chips* (NoC) [134]. In particular, the one-level 4T1R-based multiplexers show superior delay and power characteristics over best CMOS multiplexers. As for the RRAM-based FPGA architectures, such paradigm shift in the interconnection topology potentially leads to a revisit of best architecture parameters. Last but not least, the impact of process variations of RRAMs on the proposed 4T1R-based multiplexers are also examined. Experimental results show that variations on V_{reset} should be well constrained due to their remarkable influence on multiplexer performance while variations on V_{set} can be relaxed because of their trivial impact on multiplexer performance.

Chapter 3 hardcores for architecture-level studies about RRAM FPGAs and strongly motivates Chapter 4 and Chapter 5. The improved multiplexer designs will be modelled from a CAD perspective in Chapter 4 and their outstanding charactersitics will be intensively exploiting in FPGA architectures in Chapter 5.

4 Simulation-based Architecture Exploration Tool

As stated in Section 2.4, mainstream *Field Programmable Gate Array* (FPGA) architecture exploration tools, e.g., VTR [44], face serious limitations in capturing the characteristics of FPGAs architectures based on emerging technologies, due to the large design space offered by FPGAs and the limits of analytical models. In addition, the novel RRAM-based circuit designs shown in Chapter 3 bring new physical design constraints and hence require both functional and electrical verification at architecture-level. To enable further studies about RRAM-based FPGA architecture presented in Chapter 5, a novel architecture exploration tool is desired to fill the void in accurately modeling and fast prototyping of FPGAs architectures using unconventional device technologies.

In this chapter, we introduce a simulation-based FPGA architecture exploration tool suite, called FPGA-SPICE, that is tightly integrated with the popular academic architecture exploration tool suite VTR [44]. FPGA-SPICE aims at providing SPICE and Verilog modeling for both SRAM-based and RRAM-based FPGA architectures, in order to perform accurate power analysis, functional verification and prototyping. To support versatile architectures and circuit designs, FPGA-SPICE extends the generic architecture description language of VTR [48] to consider transistor-level parameters related to each module inside the FPGA architecture under evaluation. With SPICE netlists, accurate power analysis can be conducted for large FPGA fabrics through electrical simulators, i.e., HSPICE [47]. Verilog netlists allow full FPGA fabrics to be rapidly prototyped through a semi-custom design flow [45], and also enables functional verification with a HDL simulator [135]. Note that the SPICE and Verilog modeling methodologies of FPGA-SPICE are general, which can be easily extended to studying FPGA architectures based on other emerging technologies, such as *Phase Change Memory* (PCM) [40].

This chapter is organized as follows. Section 4.1 introduces the working principles of FPGA-SPICE. Section 4.2 presents the extended FPGA architecture description language. Section 4.3 discusses the core engine to generate transistor-level designs of circuit modules in FPGA architectures. Section 4.4 covers critical techniques in auto-generating SPICE and Verilog testbenches. Section 4.5 shows the experimental results about accurate area and power analysis of FPGAs.

FPGA-SPICE is available for download at [136].

4.1 Principles

FPGA-SPICE plays a role of interfacing various EDA tools, i.e., SPICE-based electrical simulators and Verilog-based design tools, with the VTR tool suite. In order to accurately model a full FPGA fabric with SPICE or Verilog netlists, FPGA-SPICE requires detailed routing information, such as directionality, connectivity and channel width. Therefore, FPGA-SPICE is invoked after routing stage, similar to VersaPower [46] in the classical EDA flow shown in Fig. 2.27. Depending on the purpose of FPGA-SPICE, either for SPICE or Verilog netlist auto-generation, the organization of EDA flow and even working principles of FPGA-SPICE could be different. In the rest of this section, we will introduce FPGA-SPICE in two separated tracks: SPICE modeling (Section 4.1.1) and Verilog modeling (Section 4.1.2).

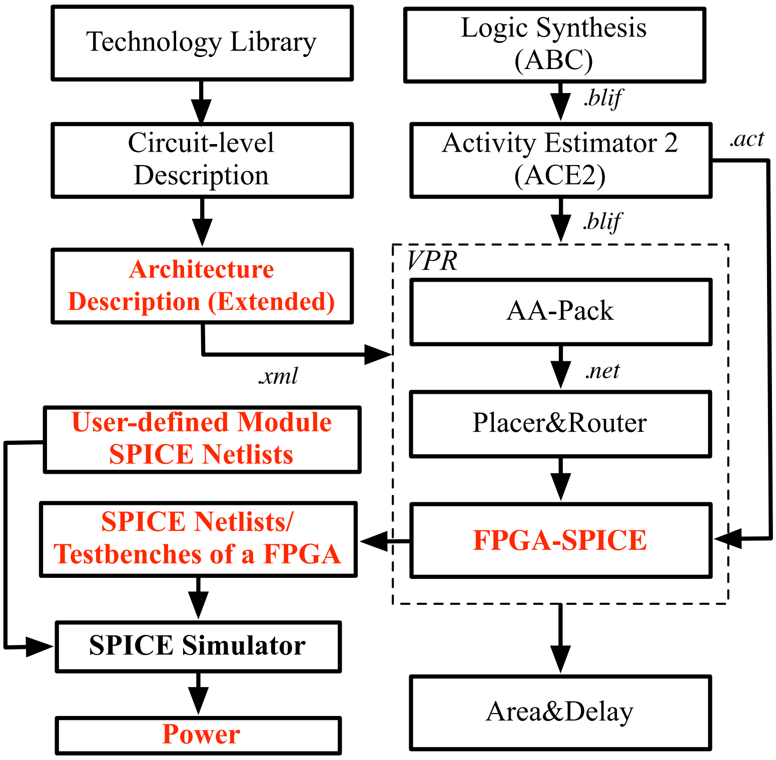

Technology Library
Logic Synthesis (ABC)
.blif
Circuit-level Description
Activity Estimator 2 (ACE2)
.act
.blif
VPR
Architecture Description (Extended)
AA-Pack
.xml
.net
User-defined Module SPICE Netlists
Placer&Router
SPICE Netlists/ Testbenches of a FPGA
FPGA-SPICE
SPICE Simulator
Area&Delay
Power

Figure 4.1 – FPGA-SPICE EDA flow for SPICE modeling purpose.

4.1.1 SPICE Modeling

In a SPICE-oriented design flow, FPGA-SPICE plays a role of automatically generating SPICE netlists and testbenches for a mapped FPGA architecture. As illustrated in Fig. 4.1, FPGA-SPICE exploits the description of the architecture provided by the architect to VTR, the mapped netlists and the estimated signal activities to dump circuit netlists and the associated testbenches for the implemented benchmarks. The tool subsequently invokes a SPICE simulator to conduct power analysis.

FPGA-SPICE reads transistor-level design parameters from an extended architecture description XML file and use them to automatically generate detailed SPICE netlists of the basic circuit elements used in the full FPGA architecture. The proposed extension of the VTR architecture description language will be given in Section 4.2.

Alternatively, FPGA-SPICE can use user-defined SPICE netlists rather than automatically generating them. This is an interesting feature to model fine-grain FPGA components, such as SRAMs, whose performances are highly dependent on the technology and the circuit structure. This brings the capability to study the system-level impact of full-custom optimized circuit elementary blocks, thereby enabling interesting circuit/architecture co-optimization opportunities. Details about transistor-level SPICE netlists generation are introduced in Section 4.3.

FPGA-SPICE can generate its netlists at three levels of complexity: full-chip-level, grid-level and component-level. Fig. 4.2, Fig. 4.3 and Fig. 4.4 illustrate the granularity of each level respectively. In a full-chip-level testbench, all the components, such as CLBs, SBs and CBs, are simulated within a unique top SPICE netlist, leading to an accurate simulation. Nevertheless, a full-chip-level testbench simulation may require long runtime and large memory usage because of the exponential complexity of SPICE solvers. To reduce both runtime and memory usage, FPGA-SPICE can split the evaluation of a full-chip-level testbench into grid-level and component-level testbenches. The grid-level testbenches consider separately each individual CLBs, memory banks, DSP blocks, SB multiplexers and CB multiplexers. In the component-level testbenches, the CLBs are further sliced into finer-grain modules, such as LUTs, FFs and local routing multiplexers, for each of which an associated testbench is created. Section 4.4 focus on the partitioning strategies in grid/component-level testbenches.

4.1.2 Verilog Modeling

Different from SPICE modeling, the Verilog generator of FPGA-SPICE aims at automatically generating synthesizable circuit netlists and testbenches in order to perform functional verification and prototyping. As illustrated in Fig. 4.5, FPGA-SPICE reads the extended architecture description file and dumps synthesizable Verilog netlists, associated testbenches and bitstream for a mapped FPGA fabric. Note that the detailed circuit designs, such as transistor sizing and buffering, are typically handled by a semi-custom design flow. The synthesizable

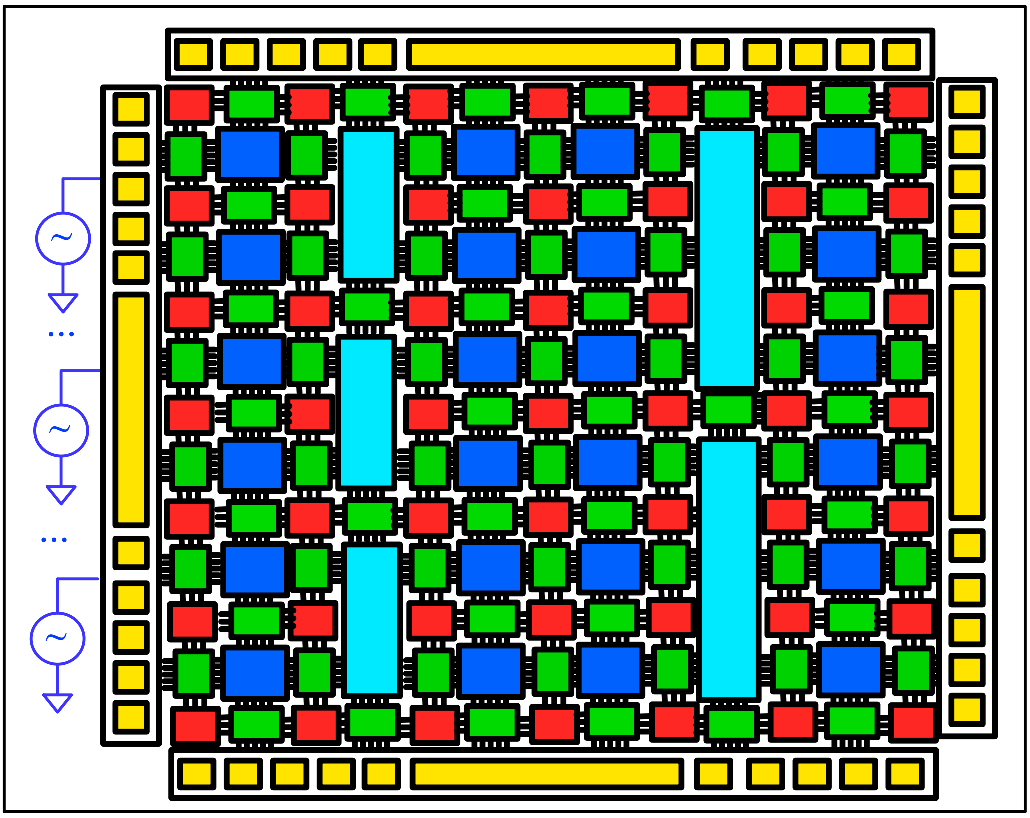


Figure 4.2 – Ilustration of the full-chip-level testbenches.

Verilog netlists are organized at structure-level, and hence FPGA-SPICE requires more circuit-level modeling parameters to capture diverse circuit design topologies than transistor-level modeling parameters. Section 4.3 will introduce the circuit-level modeling enhancements in the VTR architecture description language.

Similar to SPICE modeling, FPGA-SPICE can also use a user-defined Verilog netlists rather than automatically generating them. Thanks to the popularity of Verilog modeling in hard *Intellectual Property* (IP) cores, such feature brings opportunities in modeling coarse-grained FPGA architectures. As Verilog netlist are widely used in EDA tools, the Verilog generator enables various FPGA research opportunities. In this thesis, we focus on exploiting the Verilog generator to perform functional verification and automatic layout generation, as illustrated in Fig. 4.5. The synthesizable Verilog netlists and the associated testbenches can be the input of a *Hardware Description Language* (HDL) simulator, e.g., Modelsim™[135], and therefore be used to verify the functionality of the mapped FPGA implementations. Section 4.5.2 will introduce the techniques used in functional verification. The synthesizable Verilog netlists can be the input of a semi-custom design flow, e.g., Cadence Innovus™[137], where the Verilog netlists are optimized by physical synthesis and then converted to their corresponding layout.

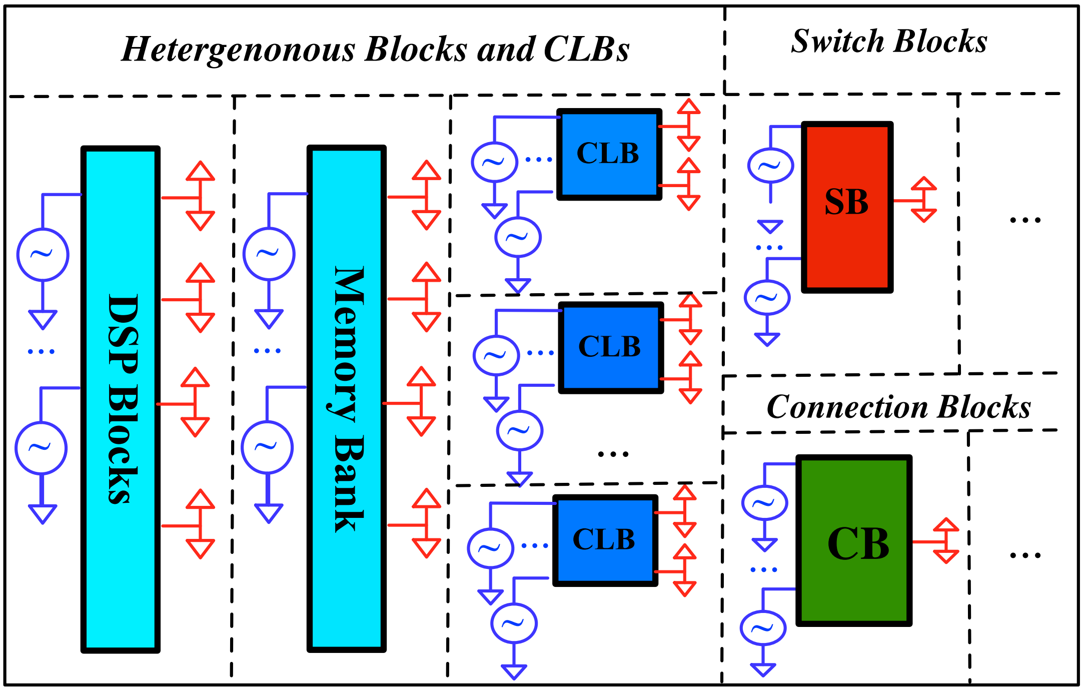


Figure 4.3 – Ilustration of the grid-level testbenches.

The layout-level realization can be directly used for manufacturing and also for realistic area, delay and power analysis for the investigated FPGA architectures. Section 4.5.6 is devoted to present the layout-level results.

4.2 Extended Architecture Description Language

FPGA-SPICE extends the architecture description language of [48]. This architecture description language can model highly-flexible FPGA architectures at an abstract level. In the extension, we add transistor-level circuit design parameters for:

1. elaborating the circuit components of the FPGA modules (See Section 4.2.1);
2. capturing the physical structure of circuit modules (See Section 4.2.2);
3. describing the topology of configuration circuits (See Section 4.2.3).

4.2.1 Transistor-level Module Declaration

First, transistor model and basic geometrical properties are defined in XML nodes `tech_lib` and `transistors`, as follows:

```
<tech_lib lib_path=‘‘45nmHP.pm’’ nominal_vdd=‘‘1.0’’/>
```

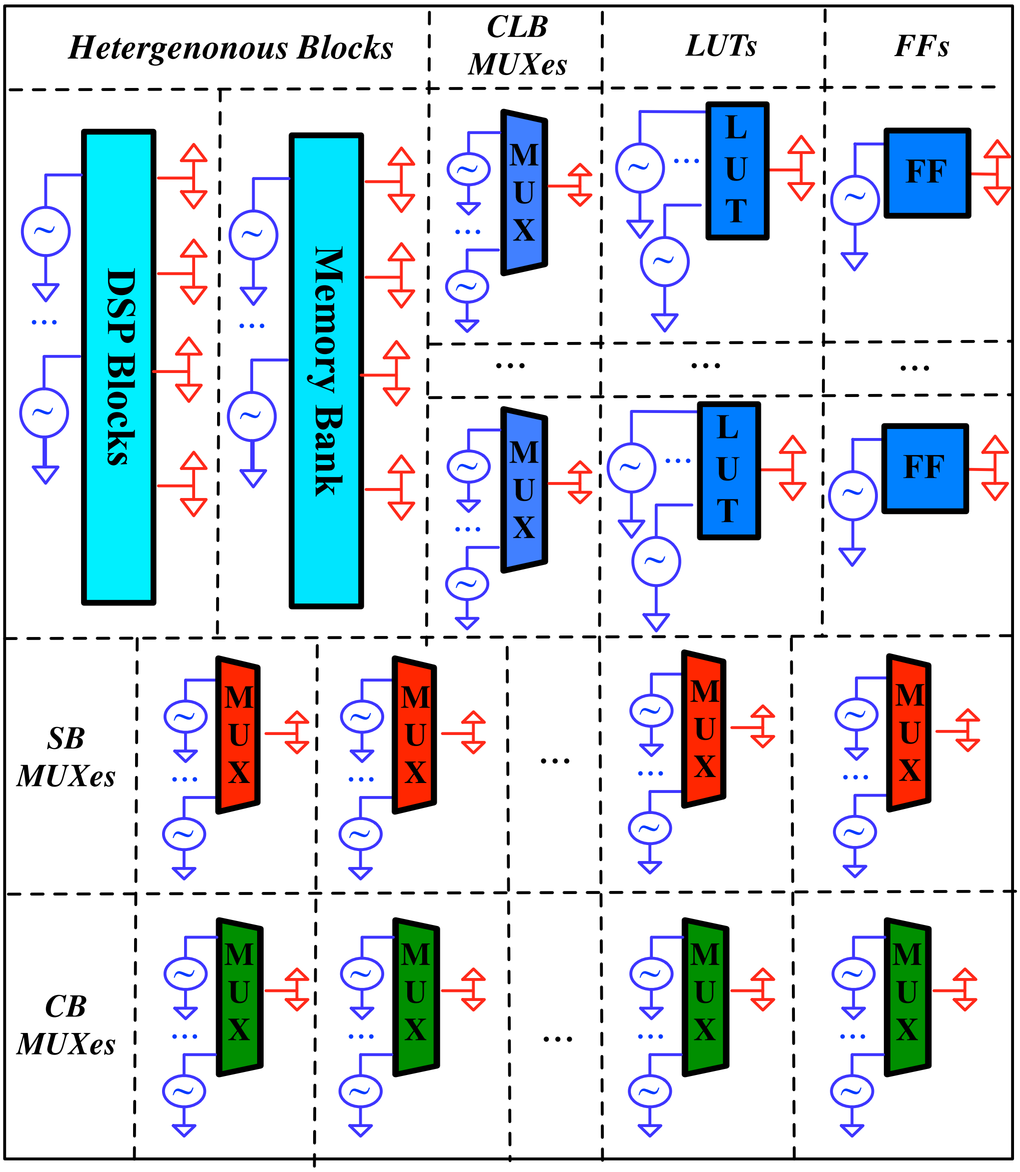


Figure 4.4 – Ilustration of the component-level testbenches.

```
<transistors pn_ratio=“1.5">
    <nmos chan_length=“45e-9” min_width=“140e-9”/>
    <pmos chan_length=“45e-9” min_width=“140e-9”/>
</transistors>
```

The channel length, transistor width and ratio between *p*-type and *n*-type transistors are

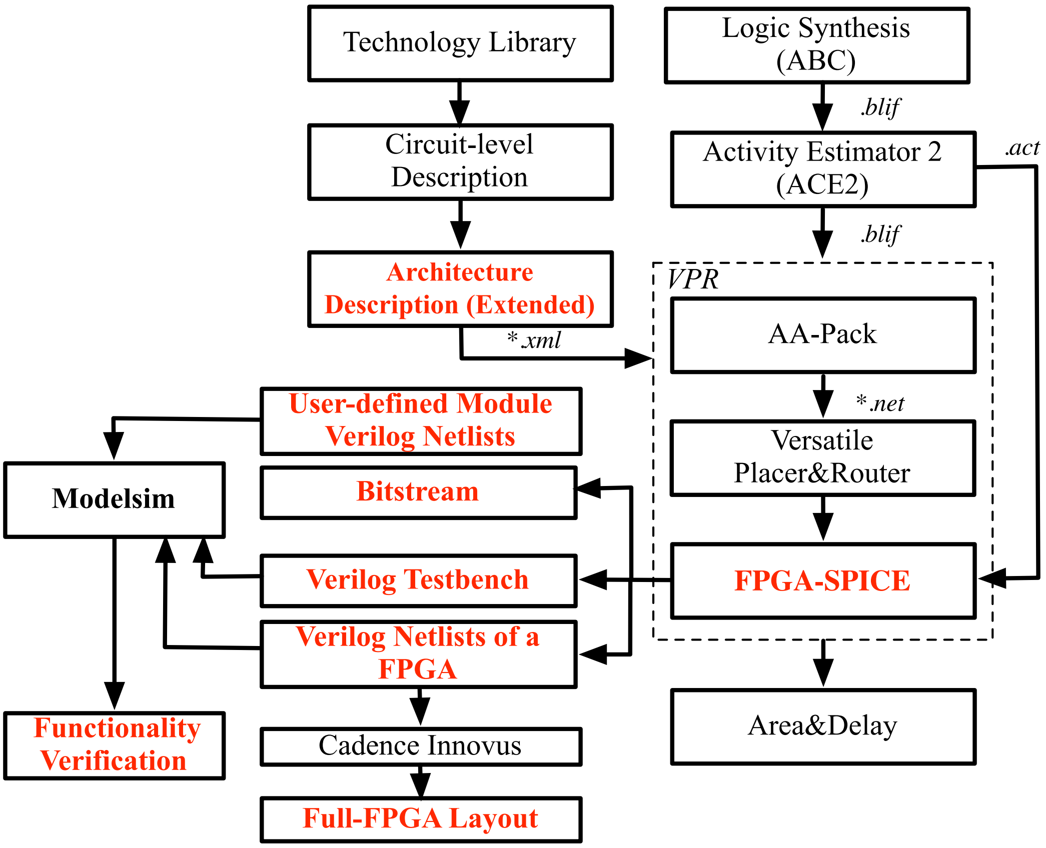


Figure 4.5 – FPGA-SPICE EDA flow for synthesizable Verilog purpose.

defined in the XML properties `nmos` and `pmos`, respectively.

Then, transistor-level circuit design parameters of a FPGA module are defined under a XML property called `spice_model`. The VTR architecture description language models all logic blocks with a hierarchy of XML properties, called `pb_type`. We create a property `spice_model_name` under `pb_type` to link the logic blocks to defined spice models. The following code shows an example, where a 6-input LUT spice model, `lut6`, is defined and linked to a logic block, `n_lut6`:

```
<spice_model type=‘‘lut’’ name=‘‘lut6’’ sp_netlist=‘‘lut6.sp’’

verilog_netlist=‘‘‘lut6.v">

   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘6’’ is_global="false" is_clock="false"/>

   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>

   <port type=‘‘sram’’ prefix=‘‘sram’’ size=‘‘64’’ spice_model_name=‘‘sram6T’’
   default_val=‘‘1’’/>

<spice_model>

<pb_type name=‘‘n_lut6’’ spice_model_name=‘‘lut6’’>

</pb_type>
```

Under the XML property `spice_model`, the ports of a LUT should be defined by providing the size, port type and port name. In addition, whether the port is a global port in FPGA, such as the clock signal, can be defined under the XML node `port`. FPGA-SPICE can automatically identify the functionality of global ports and give proper stimuli in testbenches. Since the circuit designs of some of the FPGA modules are highly dependent on the technology nodes, such as SRAMs, hard logic blocks or FFs, FPGA-SPICE allows user-customized SPICE netlists for each defined spice model. In the above example of `lut6`, user-customized SPICE and Verilog netlists are defined in the XML properties, `sp_netlist` and `verilog_netlist`. Note that, the circuit design of SRAMs used in a `spice_model` can also be customized by assigning the XML property `spice_model_name` in the port. In the example of `lut6`, a `spice_model` named by `sram6T` is declared to be used.

4.2.2 Physical Structure Modeling

To be efficient in mapping logic functions to circuit modules, VPR uses abstract-level modeling to bridge the technology mapping results and FPGA architecture resources. The VPR architecture description language focuses on describing the structure of circuit modules at behavioral-level rather than at structural-level. For instance, an I/O pad is described with two operating modes: input pad and output pad, as illustrated in Fig. 4.6(a). An input of a circuit can be mapped to an input pad while an output of a circuit can be mapped to an output pad. Indeed, the transistor-level design of a I/O pad in Fig. 4.6(b) can operate as either an input pad or an output pad by configuring the SRAM. However, with the abstract-level modeling, the physical structure of I/O pads cannot be accurately described, causing difficulties in transistor-level modeling. Comparing to Fig. 4.6(b), an I/O pad modelled by VPR (in Fig. 4.6(a)) lacks two critical elements: (1) the SRAM controlling the directionality of the I/O module; (2) two ports `direction` and *PAD* of the I/O module. `PAD` is an bi-directional port that interfaces the FPGA to outside world. `direction` determines whether the signal is propagated from `PAD` to `data_in` or from `data_out` to `PAD`. Hence, in the purpose of accurate modeling FPGAs with SPICE or Verilog netlists, the abstract-level modeling should be improved to exactly describe the physical design.

We extend the architecture description language to model the physical design of an I/O pad, as follows:

```
<pb_type name="io" idle_mode_name="inpad" physical_mode_name="io_phy">
```

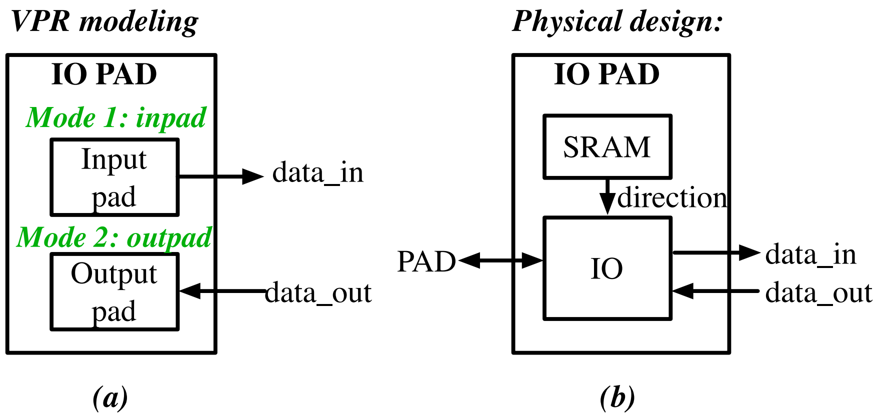


Figure 4.6 – An I/O pad: (a) VPR abstract-level modeling, and (b) actual physical design.

```
    <mode name="io_phy">

      <pb_type name="iopad" num_pb="1" spice_model_name="iopad"/>

    </mode>

    <mode name="inpad">

      <pb_type name="inpad" num_pb="1" mode_bits="1"/>

    </mode>

    <mode name="outpad">

      <pb_type name="outpad" num_pb="1" mode_bits="0"/>

    </mode>

</pb_type>
```

In parallel to the original abstract-level modeling, an extra mode named by `io_phy` is added to the `pb_type`, under which the physical design of an I/O pad is described by the architecture description language. An XML property `physical_mode_name` is added to the `pb_type`, in order to identify which mode describes the physical design of the module. As a module depends on the configuration bits to switch between operating modes, each operating mode, e.g., `inpad` and `outpad`, contains a new XML property `mode_bits`, in order to define its unique configuration bits. For instance, the `mode_bits="1"` under operating mode `inpad` specifies that it is enabled when the SRAM is configured to logic `1`. Note that the new mode `io_phy` is only used by FPGA-SPICE for SPICE and Verilog generator, while the two original modes `inpad` and `outpad` are used in VPR packing, placement and routing. As such, the extended architecture description language does not influence any results of VPR packing, placement and routing.

4.2.3 Configuration Circuitry

As introduced in Section 2.2.4, memory bits of FPGAs can be accessed by different types of configuration circuits, leading to difference in the full-chip area and also other merits. For example, when scan-chain flip-flops are used, area of configuration circuits is linear to the number of memory bits. When using BL and WL decoders, area of configuration circuits is in square root relationship to the number of memory bits. However, since most FPGA researches only focus on the core logics, the exact impact of configuration circuits has not been carefully examined. As FPGA-SPICE aims at accurately model a full FPGA fabric with SPICE or Verilog netlists, the architecture description language is extended to model the configuration circuits. Under the XML node `sram`, details of configuration circuits can be specified separately for SPICE and Verilog generator, as follows:

```
<sram area="6">

   <verilog organization="memory_bank" spice_model_name="sram6T_blwl"/>

   <spice organization="standalone" spice_model_name="sram6T"/>

</sram>
```

Take the example of the XML node `verilog`, the type of configuration circuit can be specified by the XML property `organization`. The supported configuration circuits include memory-bank-style (shown in Fig. 2.18) and scan-chains (shown in Fig. 2.19). The memory model accessed by the configuration circuits can be declared in the XML property `spice_model_name`, which is linked to a defined spice model devoted to the transistor-level designs of a SRAM and a scan-chain flip-flop (See details in Section 4.3.3 and Section 4.3.4).

As a result, FPGA-SPICE can automatically generate the bitstream used to program the configuration circuits, according to the selected implementations.

4.3 Transistor-level Circuit Netlist Generation

In an FPGA, the circuit-level implementations for the different blocks, such as channel wires, multiplexers and LUTs, are highly dependent on the architectural choices. FPGA-SPICE can automatically determine their design parameters and generate the associated SPICE netlists. In this section, we will discuss the details of the circuit netlist generation engine. We will start with the basic circuits, i.e., inverters, buffers and transmission gates, which are commonly used by all the blocks. Then, we will introduce more complicated blocks, such as SRAMs, multiplexers and LUTs.

4.3.1 Inverters/Buffers

Inverters and buffers are essential components of FPGA submodules, such as LUTs and multiplexers, as shown in Fig. 2.14, Fig. 2.15 and Fig. 2.16. FPGA-SPICE allows inverters and buffers to be either fully customized by specifying *sp_netlist* or automatically generated.

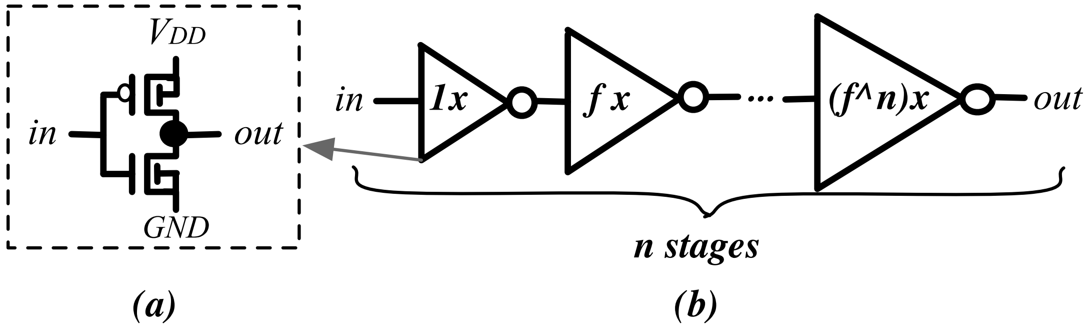


Figure 4.7 – Transistor-level circuit design of (a) an inverter and (b) a tapered buffer.

The transistor-level circuit design of an inverter in Fig. 4.7(a) can modelled by the following code:

```
<spice_model type=‘‘inv_buf’’ name=‘‘inv1’’>

   <design_technology type=‘‘cmos’’ topology=‘‘inverter’’ size=‘‘1’’/>

   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘1’’/>

   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>

</spice_model>
```

The transistor sizes can be specified in the SPICE model definitions.

FPGA-SPICE can also model the transistor-level circuit design of a general multi-stage buffer in Fig. 4.7(b) with the following code:

```
<spice_model type=‘‘inv_buf’’ name=‘‘tap_buf4’’>

   <design_technology type=‘‘cmos’’ topology=‘‘buffer’’ size=‘‘1’’

   tapered=‘‘on’’ tap_buf_level=‘‘3’’ f_per_stage=‘‘4’’/>

   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘1’’/>

   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>

</spice_model>
```

The size and design topology can be customized by properly setting the XML properties `tapered`, `tap_buf_level` and `f_per_stage`.

4.3.2 Pass-gate Logic

Pass-gate logic is the essential component in LUTs and multiplexers, as shown in Fig. 2.14, Fig. 2.15 and Fig. 2.16. The transistor-level circuit design of a transmission gate can be defined with the following code:

```
<spice_model type=‘‘pass_gate’’ name=‘‘tgate’’>
   <design_technology type=‘‘cmos’’ topology=‘‘transmission_gate’’
   nmos_size=‘‘1’’ pmos_size=‘‘2’’/>
   <input_buffer exist=‘‘off’’/>
   <output_buffer exist=‘‘off’’/>
   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘1’’/>
   <port type=‘‘input’’ prefix=‘‘sel’’ size=‘‘1’’/>
   <port type=‘‘input’’ prefix=‘‘selb’’ size=‘‘1’’/>
   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>
</spice_model>
```

The sizes of the transistors used in the pass gate or transmission gate logic can be specified in the XML properties `nmos_size` and `pmos_size`.

4.3.3 SRAM

SRAM is a critical component of SRAM-based FPGA, whose transistor-level design is mostly dependent on the technology node and is usually hand-optimized. Therefore, SPICE and Verilog netlists of SRAMs are required to be user-defined. The following codes exemplify how to define a spice model for the SRAM circuit design shown in Fig. 2.18.

```
<spice_model type=‘‘sram’’ name=‘‘sram6T’’ spice_netlist=‘‘sram6T.sp"
verilog_netlist=‘‘‘sram6T.v">
   <design_technology type=‘‘cmos’’/>
   <input_buffer exist=‘‘off’’/>
   <output_buffer exist=‘‘off’’/>
   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘1’’/>

   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘2’’/>

   <port type=‘‘bl’’ prefix=‘‘bl’’ size=‘‘1’’/>

   <port type=‘‘wl’’ prefix=‘‘wl’’ size=‘‘1’’/>

</spice_model>
```

Note that the modeling method can also support the non-volatile SRAM design in Fig. 2.24.

4.3.4 Scan-chain Flip-Flop

Similar to SRAM, SPICE and Verilog netlists of scan-chain flip-flop are required to be user-defined. The following code exemplifies how to define a spice model for the scan-chain flip-flop design shown in Fig. 2.19.

```
<spice_model type=‘‘sff’’ name=‘‘sc_dff’’ spice_netlist=‘‘scff.sp"

verilog_netlist=‘‘‘scff.v">

   <design_technology type=‘‘cmos’’/>

   <input_buffer exist=‘‘on’’ spice_model_name=‘‘inv4’’/>

   <output_buffer exist=‘‘on’’ spice_model_name=‘‘inv4’’/>

   <port type=‘‘input’’ prefix=‘‘D’’ size=‘‘1’’/>

   <port type=‘‘input’’ prefix=‘‘Set’’ size=‘‘1’’ is_global="true" is_set="true"/>

   <port type=‘‘input’’ prefix=‘‘Reset’’ size=‘‘1’’ is_global="true" is_reset="true"/>

   <port type=‘‘output’’ prefix=‘‘Q’’ size=‘‘1’’/>

   <port type=‘‘output’’ prefix=‘‘Qb’’ size=‘‘1’’/>

   <port type=‘‘clock’’ prefix=‘‘prog_clk’’ size=‘‘1’’ is_global="true"

is_clock="true"/>

</spice_model>
```

The presence or absence of input/output inverters/buffers can be declared by setting the XML properties *exist* and *spice_model_name*. In the example, the input and output buffers are linked to the spice model named by $inv1$, which is defined in Section 4.3.1.

4.3.5 IO Circuits

IO circuits are usually provided as a standard cell in a specific technology library, since their transistor-level designs are strongly dependent on the technology nodes. The following codes define a spice model called *iopad* which is linked to the IO module shown in Section 4.2.2. Note that in the port *sram*, we specify it as a mode selector of IO module (in Fig. 4.6), and declare that it is connected to a SRAM, which is defined in Section 4.3.3.

```
<spice_model type=“iopad” name=“iopad” spice_netlist=“iopad.sp"
verilog_netlist=“‘iopad.v">
   <design_technology type=“cmos”/>
   <input_buffer exist=“on” spice_model_name=“inv4”/>
   <output_buffer exist=“on” spice_model_name=“inv4”/>
   <port type=“inout” prefix=“pad” size=“1”/>
   <port type=“sram” prefix=“en” size=“1” mode_select=“true”
   spice_model_name=“sram6T” default_val=“1”/>
   <port type=“input” prefix=“outpad” size=“1”/>
   <port type=“output” prefix=“inpad” size=“1”/>
</spice_model>
```

4.3.6 Multiplexers

The multiplexers in FPGAs have diverse sizes and fan-outs, depending on their locations, i.e., in local routing or global routing.

In this context, different circuit-level optimization, such as transistor sizing and the use of tapered buffer, may apply. The transistor sizes and buffer allocation can be specified in the SPICE model definitions. The presence or absence of input/output inverters/buffers can be declared by setting the XML properties *exist* and *spice_model_name*. The use of a pass gate logic or a transmission gate logic design style can be specified in the XML property *pass_gate_logic*.

Transistor-level circuit design examples of global routing multiplexers and local routing multiplexers are shown in Fig. 4.8(a) and Fig. 4.8(b), respectively. The tree-like structure of multiplexers is depicted in Fig. 4.8(c). The transistor-level circuit design of a global routing multiplexer in Fig. 4.8(a) can modelled by the following code:

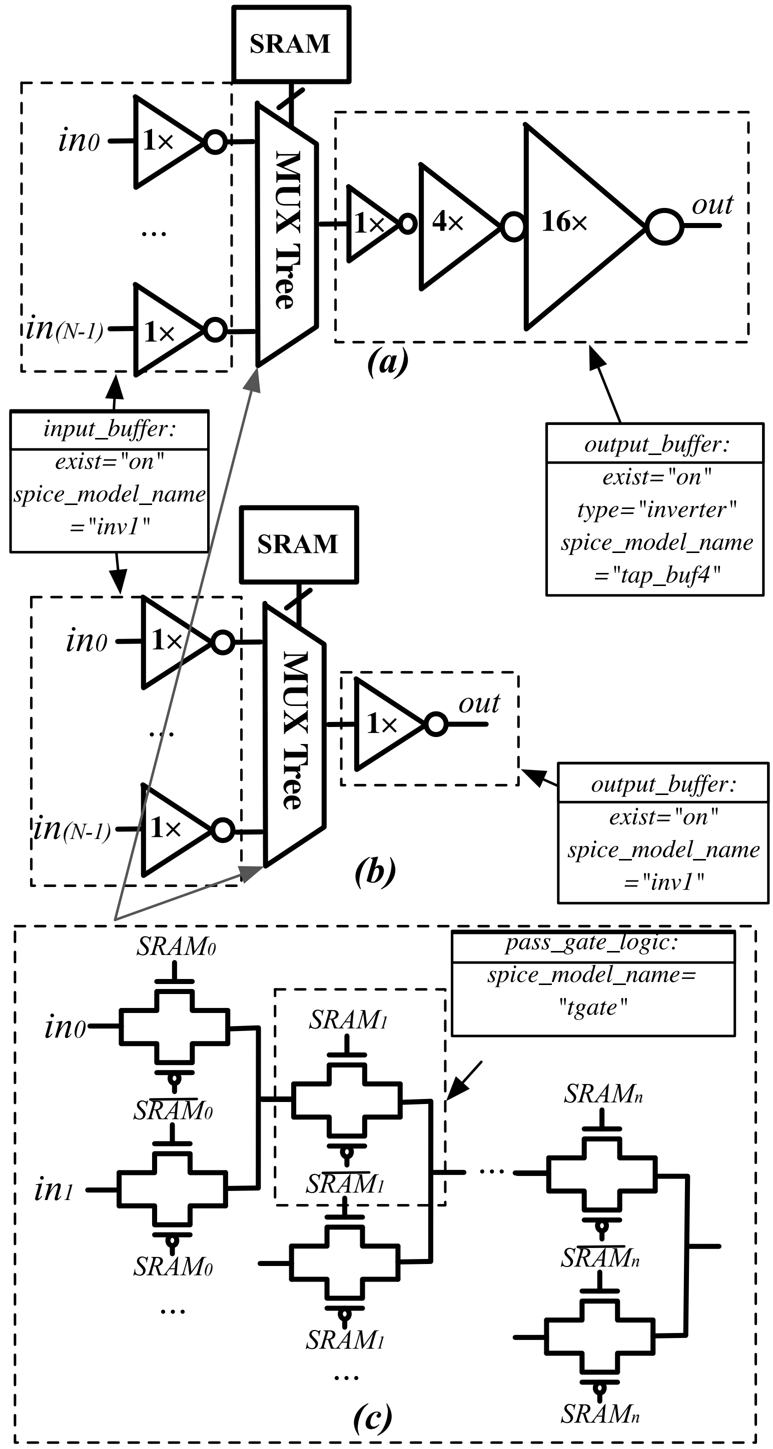


Figure 4.8 – Transistor-level circuit design of (a) a global routing multiplexer, (b) a local routing multiplexer, and (c) the internal tree-like structure.

```
<spice_model type=‘‘mux’’ name=‘‘sb_mux’’/>
   <design_technology type=‘‘cmos’’ structure=‘‘one-level’’/>
   <input_buffer exist=‘‘on’’ spice_model_name=‘‘inv1’’/>
   <output_buffer exist=‘‘on’’ spice_model_name=‘‘tap_buf4’’/>
   <pass_gate_logic spice_model_name=‘‘tgate’’/>
   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘4’’/>
   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>
   <port type=‘‘sram’’ prefix=‘‘sram’’ size=‘‘4’’/>
</spice_model>
```

Global routing multiplexers require an output tapered buffer [132], in order to drive the long routing metal wires as well as downstream loads due to the SB and CB multiplexers [2]. The output tapered buffer in Fig. 4.8(a) consists of three stages and the logical effort between stages is four, whose spice model is defined in Section 4.3.1. Input buffers are added to restore the input signals and drive the tree-like internal structure of the multiplexer. Fig. 4.8(b) depicts the circuit design of a local routing multiplexer which interconnects CLB input pins to BLE input pins. Because the fanout of the multiplexer is typically small (one or two inverters), there is only a minimum-size output inverter.

To enable accurate power analysis for RRAM-based FPGAs, FPGA-SPICE is capable of modeling one-level, two-level and tree-like 4T1R-based multiplexers, presented in Chapter 3. Transistor-level circuit design examples of a one-level 4T1R-based multiplexer are shown in Fig. 4.9. The transistor-level circuit design of a global routing multiplexer in Fig. 4.9 can modelled by the following code:

```
<spice_model type="mux" name="mux_1level">
   <design_technology type="rram" ron="3e3" roff="20e6"
   wprog_set_nmos="1" wprog_reset_nmos="1"
   wprog_set_pmos="2" wprog_reset_pmos="2"
   structure="one-level"/>
   <input_buffer exist="on" spice_model_name="inv1"/>
   <output_buffer exist="on" spice_model_name="inv1"/>
   <port type="input" prefix="in" size="1"/>
```

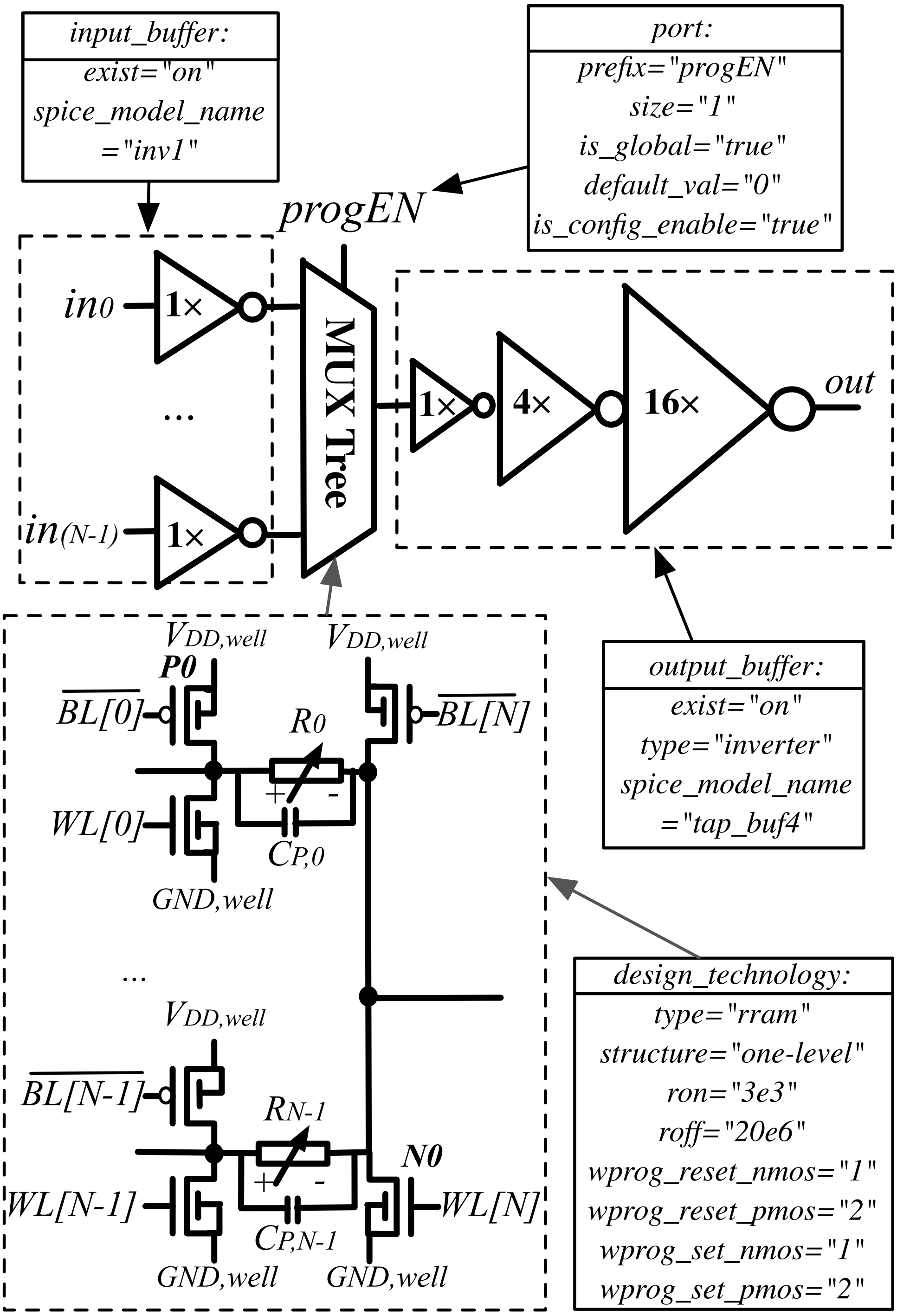


Figure 4.9 – Transistor-level circuit design of a 4T1R-based multiplexer.

```
   <port type="input" prefix="EN" size="1" is_global="true"
   default_val="0" is_config_enable="true"/>
   <port type="output" prefix="out" size="1"/>
</spice_model>
```

Compared to the SRAM-based multiplexers in Fig. 4.8, the 4T1R-based multiplexer has an global port `progEN`, which is shared by all the 4T1R-based multiplexers in a FPGA. As a programming enable signal, `progEN` is enabled periodically during configuration phase, while being disabled during operation (See Chapter 3). In the XML definition, we specify that `progEN` is enabled during configuration phase (`is_config_enable="true"`), while during operation, it is stuck at logic `0` (`default_val="0"`).

FPGA-SPICE translates the architectural needs and design topologies into multiplexer SPICE netlists and initializes the SRAM or RRAM configurations according to VPR routing results.

4.3.7 Look-Up Tables

LUTs are crucial components in FPGAs as they serve as combinational function generators. Fig. 4.10 illustrates the transistor-level circuit design of the LUT structure considered in this chapter, including the configuration SRAMs, the decoding multiplexers, and buffers [125].

The following XML properties are used to describe the circuit characteristics of the implementation shown in Fig. 4.10. The `input_buffer` properties model the buffers between the inputs of internal multiplexer and SRAM outputs. The `lut_input_buffer` properties describe the buffers at LUT inputs, where `f_stage` denotes the logical efforts of the input buffers. By setting the `spice_model_name` property under XML node `pass_gate_logic`, the type of pass-gate logic used in the decoding multiplexers can be specified. In the example, the LUT circuit employs the transmission gate defined in Section 4.3.2. FPGA-SPICE decodes technology mapping results of LUTs to properly initialize the SRAM bits.

```
<spice_model type=‘‘lut’’ name=‘‘lut6’’>
  <lut_input_buffer exist=‘‘on’’ spice_model_name="buf_size2"/>
  <input_buffer exist=‘‘on’’ spice_model_name=‘‘inv1’’>
  <output_buffer exist=‘‘on’’ spice_model_name=‘‘inv1’’>
  <pass_gate_logic spice_model_name=‘‘tgate’’/>
   <port type=‘‘input’’ prefix=‘‘in’’ size=‘‘6’’ is_global="false" is_clock="false"/>
   <port type=‘‘output’’ prefix=‘‘out’’ size=‘‘1’’/>
```

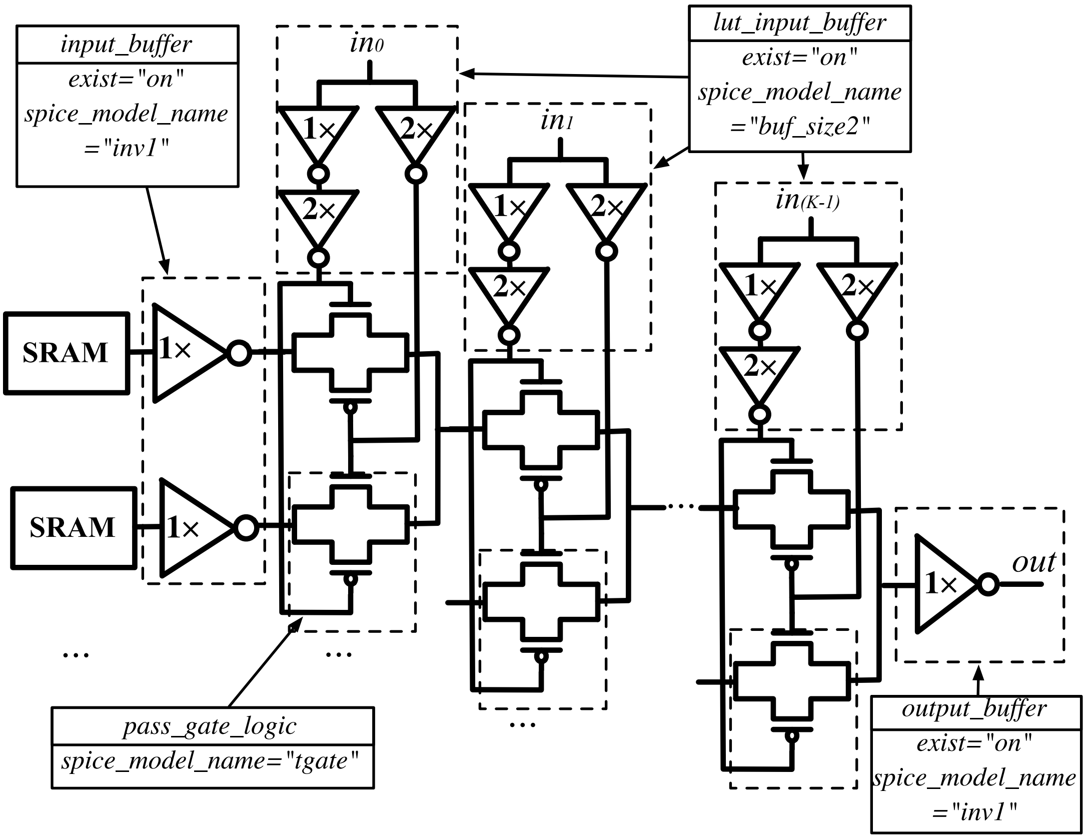


Figure 4.10 – An example of the transistor-level design of a LUT

```
   <port type=‘‘sram’’ prefix=‘‘sram’’ size=‘‘64’’ spice_model_name=‘‘sram6T’’
   default_val=‘‘1’’/>
</spice_model>
```

4.3.8 Channel Wire

In modern FPGAs, the CLB area increases to contain heterogeneous blocks, resulting in long interconnecting wires between *Switch Blocks* (SBs) and also inside CLBs. Take the example in Fig. 2.7, the length of metal wires interconnecting between BLE outputs and local routing multiplexers can be as long as the channel wires interconnecting two adjacent SBs. In addition, difficulties in scaling down interconnecting metal wires cause that their parasitics can be as significant as those of transistors [132]. As a result, channel wires have become non-negligible modules when evaluating FPGA architectures. A length-L channel wire is abstracted as L cascaded segments, each of which spans a unique CLB. Fig. 4.11(a) depicts a length-2 channel wire in unidirectional routing architecture [4]. The channel wire is divided into two segments, namely $Segment0$ and $Segment1$.

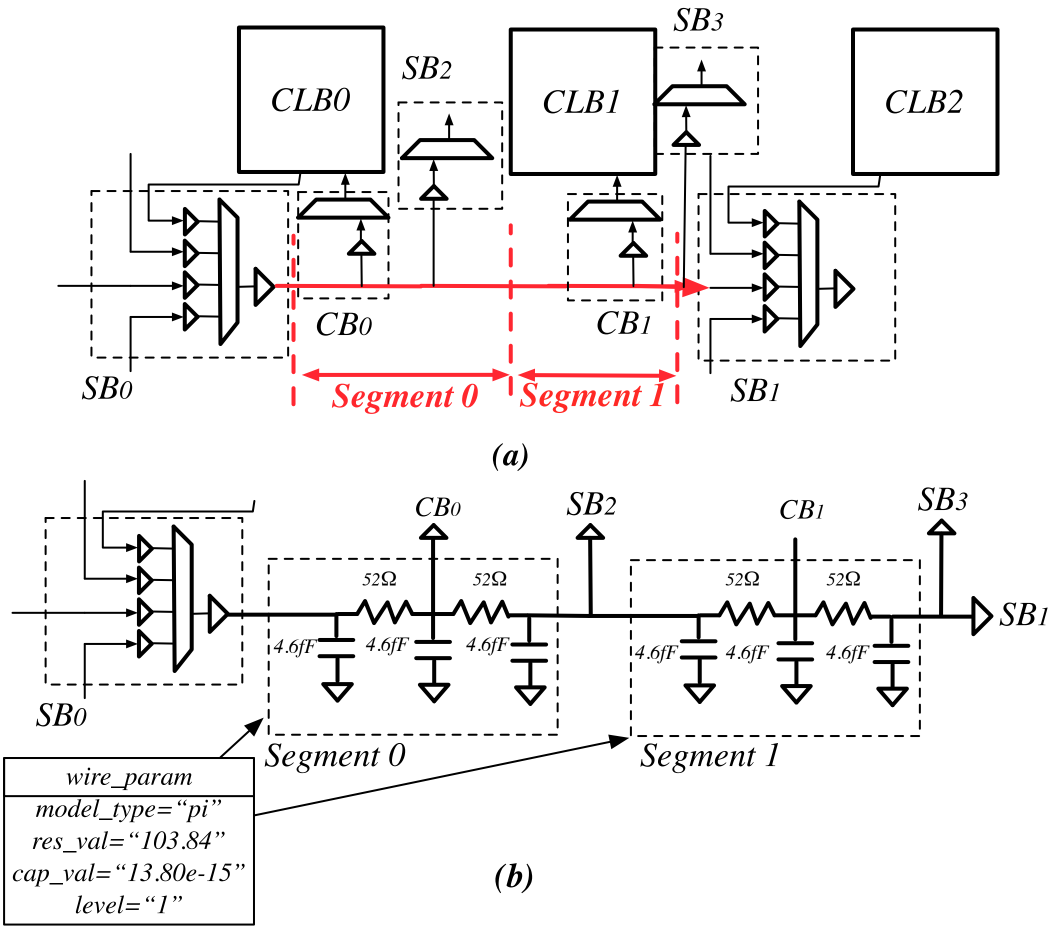


Figure 4.11 – (a) A length-2 unidirectional wire (highlighted in red) within FPGA routing architecture; (b) Corresponding RC modeling of segments

We assume that the inputs of CBs are connected to the middle of segments, breaking segments into two parts. We model each part of segments with distributed RC lines. The type of RC lines, i.e., either π-type or T-type [132], is specified in the XML property `model_type`. The number of levels of a RC line can be customized by setting the XML property `level`. The total resistances and capacitance of a segment can be defined in XML properties `res_val` and `cap_val`, respectively. The following example describes the RC models of segments in Fig 4.11(b), corresponding to the segments in Fig 4.11(a).

```
<spice_model type=‘‘chan_wire’’ name=‘‘chan_segment’’>

   <wire_param model_type=‘‘pi’’ res_val=‘‘103.84’’

   cap_val=‘‘13.80e-15’’ level=‘‘1’’/>

</spice_model>
```

4.4 Netlist Partitioning Strategies

Full-chip-level netlists, that consider the full FPGA fabric in unique SPICE testbenches, would produce accurate analysis but will come at the cost of large simulation time and memory usage. FPGA-SPICE can distribute the individual elements of a full-chip-level testbench (See Fig. 4.2) into separate grid/component-level testbenches (See Fig. 4.3 and Fig. 4.4), significantly reducing the simulation time and memory usage at the cost of a lower accuracy. In this section, we introduce the two techniques, namely voltage stimuli/load extraction and parasitic activity estimation, used in FPGA-SPICE to split a full-chip netlist.

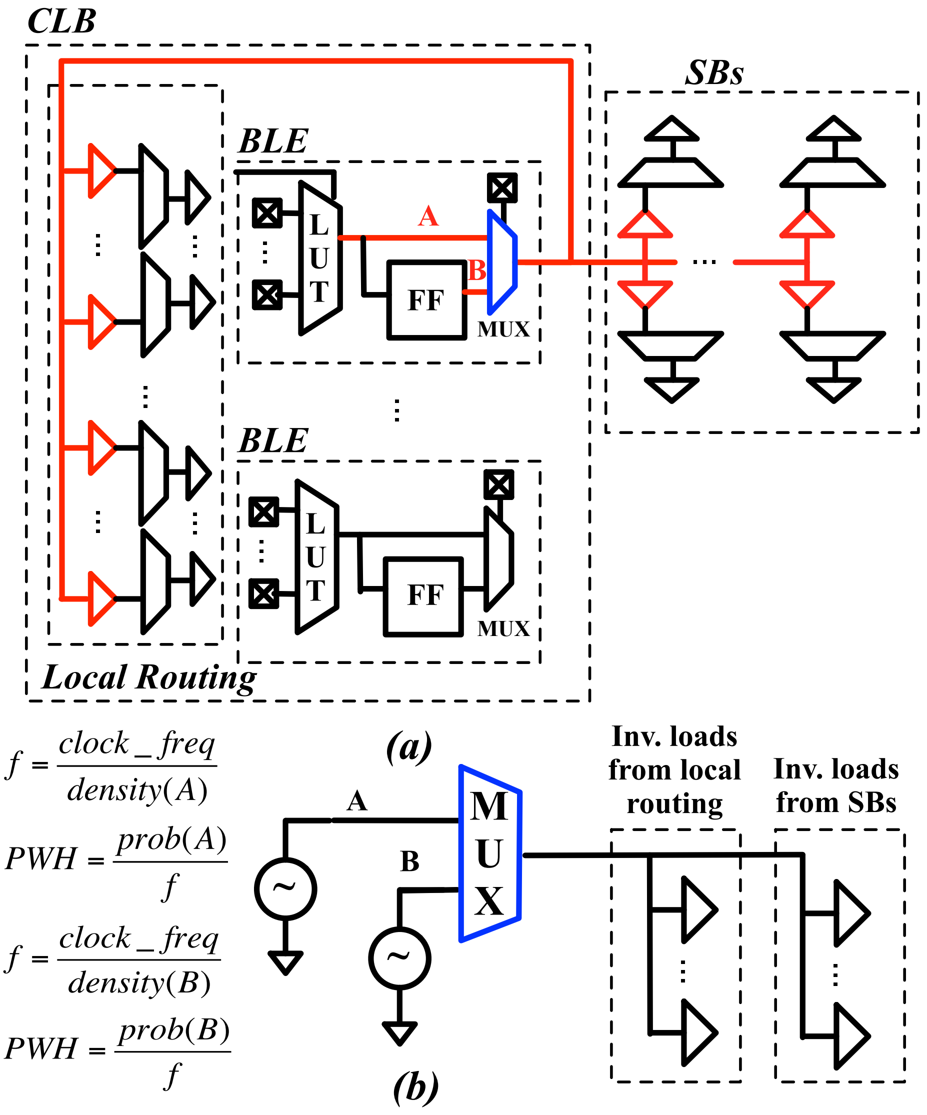

CLB
SBs
BLE
L U T
A
B
FF
MUX
BLE
L U T
FF
MUX
Local Routing
(a)
$f = \frac{clock\_freq}{density(A)}$
$PWH = \frac{prob(A)}{f}$
$f = \frac{clock\_freq}{density(B)}$
$PWH = \frac{prob(B)}{f}$
A
B
M U X
Inv. loads from local routing
Inv. loads from SBs
(b)

Figure 4.12 – Ilustration of the voltage stimuli generation and load extraction techniques. (a) BLE multiplexer with its architectural context; (b) extracted testbench.

4.4.1 Voltage Stimuli and Loads Extraction

FPGA-SPICE generates its individual testbenches by extracting voltage stimuli and downstream loads. To illustrate the technique, Fig. 4.12 shows a BLE multiplexer (in blue) that is driven by signals A and B, and that fanouts to local routing and global routing architectures.

First, voltage stimuli are added to model the signal activities of A and B. Their frequencies and pulse widths are derived from signal density and activities. The signal density defines the number of switching events of a signal in one clock cycle while the probability represents the proportion that the signal is in logic 1 during one system clock cycle. To relate these activity information, we set the frequency of the voltage stimuli to:

$$freq = \frac{clock\_period}{density(Signal)}. \tag{4.1}$$

The pulse width of a voltage stimuli is set to:

$$pulse\_width = freq \cdot probability(Signal). \tag{4.2}$$

Then, FPGA-SPICE adds the loads of the block by extracting the downstream elements in the architecture (highlighted in red in Fig. 4.12(a)). The downstream loads of a grid/component should be included in the testbench for two reasons: (1) these loads are charged/discharged by the element and (2) the power consumption is sensitive to voltage slews, which are highly dependent on the downstream loads [128]. Note that, if the downstream loads include channel wires, the channel wires should be extracted and included to the testbench.

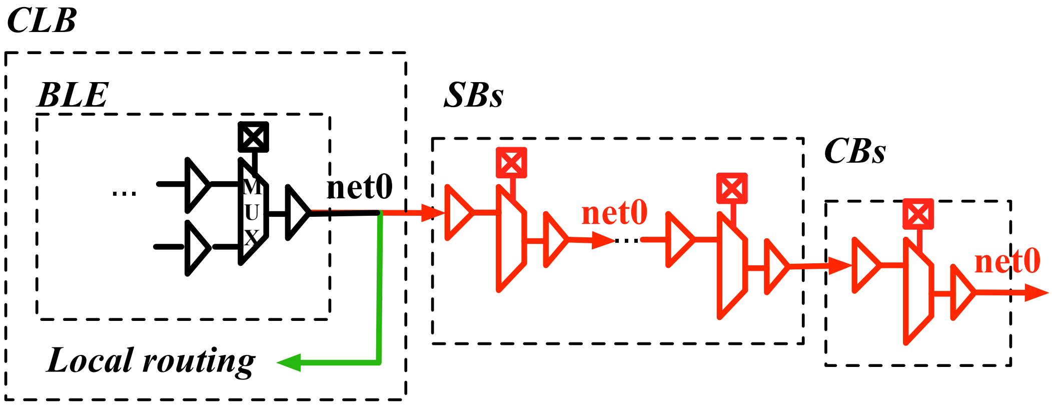


Figure 4.13 – An example for parasitic nets estimation.

4.4.2 Parasitic Activity Estimation

Input signals in grid/component-level netlists should accurately model the internal signal activities of FPGA modules. In an FPGA, the signals of the used nets may be parasitically propagated to unused nets, depending on the topology of the routing architecture. ACE2 estimates the signal activities of the used nets but cannot foresee the parasitically propagated activities because they are only predictable after the routing pass finishes [124]. Fig. 4.13 illustrates the parasitic net signals sourcing from a used net, called $net0$. Assume $net0$ is only used by the CLB through local routing (green path) and not routed to the global routing architecture. VPR assumes that all the downstream components driven by $net0$ are idle and configures them to propagate their first inputs. However, in such condition, $net0$ will be propagated through the routing structure (red path). These parasitic activities will cause extra power consumption and should be taken into account. FPGA-SPICE performs parasitic activity estimation for all the unused nets after routing stage by iteratively using *Depth-First Search* (DFS) algorithms.

4.5 Experimental Results

As shown in Fig. 4.1 and Fig. 4.5, FPGA-SPICE is a versatile tool interfacing VPR with other EDA tools, such as HSPICE [47], ModelSim [135] and Innovus [137], leading to various research interests. In this section, we will first introduce general experimental methodology. Then, we present experimental results by using FPGA-SPICE in four applications, not accessible with standard academic tools:

1. Verify the functionality of FPGA implementations (Section 4.5.2);
2. Study the runtime, memory usage and accuracy of the different levels of testbenches (Section 4.5.3);
3. Study the power breakdown of a modern FPGA architecture under different technology nodes and compare the results to standard analytical models, i.e., VersaPower (Section 4.5.4);
4. Perform a detailed analysis on the full-chip-level area of SRAM-based FPGAs (Section 4.5.6).

4.5.1 Methodology

We use the FPGA-SPICE EDA flows shown in Fig. 4.1 and Fig. 4.5. MCNC big20 benchmarks [138] are selected as the EDA flow inputs. First, ABC synthesizes the benchmarks and ACE2 estimates the signal activities. Then, VPR packs, places and routes. Afterwards, the FPGA-SPICE generates the full-chip/grid/component-level testbenches and also Verilog netlists of the modeled architectures. In the last step, we call different industrial EDA tools for various purposes:

1. we run the HDL simulator ModelSim [135] to verify the functionality of Verilog netlists;

2. we run the electrical simulator HSPICE [47] to analyzing power;
3. we run Cadence Innovus [137] to generate the layouts of a full FPGA chip by running a semi-custom design flow, in order to perform accurate area evaluation.

The experiments are run on a 64-bit RedHat Linux server with 28 Intel Xeon Processors and 256Gb memory.

In this chapter, we resemble the architecture of an Altera Stratix IV FPGA [88], where each CLB contains $I = 33$ inputs pins and $N = 10$ fracturable 6-input LUTs ($K = 6$). Length-4 uni-directional routing architectures are employed to interconnect Wilton's Switch Boxes (SBs), where $F_s = 3$. We set $F_{c,in} = 0.15$ and $F_{c,out} = 0.10$. The channel width, W, is set to 120 by adding 20% margin to the minimum channel width that VPR can route the biggest tested benchmark. All the architecture description files used in this chapter are available in [136]. For the power analysis, we consider three technology nodes, 22nm, 45nm and 180nm using the PTM model cards[139]. For the area analysis, we only consider a commercial 40nm technology node. The transistor-level circuit designs of SRAMs, FFs and multiplexers are derived from [125]. We model routing wire segments with a one-level π-type RC models and the wire parameters are derived from ITRS [140]. We determine the simulation clock period by adding a 20% slack to the VPR critical path delay, in order to consider errors between the timing analysis engine and SPICE simulations [4]. The duration of electrical simulations should be a full operating cycle by considering the least active signal, as follows:

$$sim\_time\_period = \frac{clock\_period}{min\{density(Signal)\}}. \tag{4.3}$$

However, the density of the least active signal is typically very low, which leads to long time period and large simulation time. Instead, we replace the $min\{density(Signal)\}$ with the average density of signals to reduce the the simulation time. The time step of SPICE simulator is set to 0.1ps and fast simulation algorithm is turned *on*.

4.5.2 Functional Verification

Before presenting area and power results, all the SPICE and Verilog netlists generated by FPGA-SPICE have passed functional verification with full-chip-level testbenches, to guarantee that they behave exactly the same as pre-VPR netlists functionally. In this thesis, the functional verification considers random input vectors. Indeed, to be more robust, formal verification can be applied, and we leave this as part of future works.

The functional verification employs the EDA flow shown in Fig. 4.5. In a top-level Verilog testbench, stimulus are automatically added to all the inputs of a full FPGA module, as illustrated in Fig. 4.14. A top-level Verilog testbench includes two phases:

1. Configuration phase, where each memory cell, i.e., SRAM or RRAM, is programmed

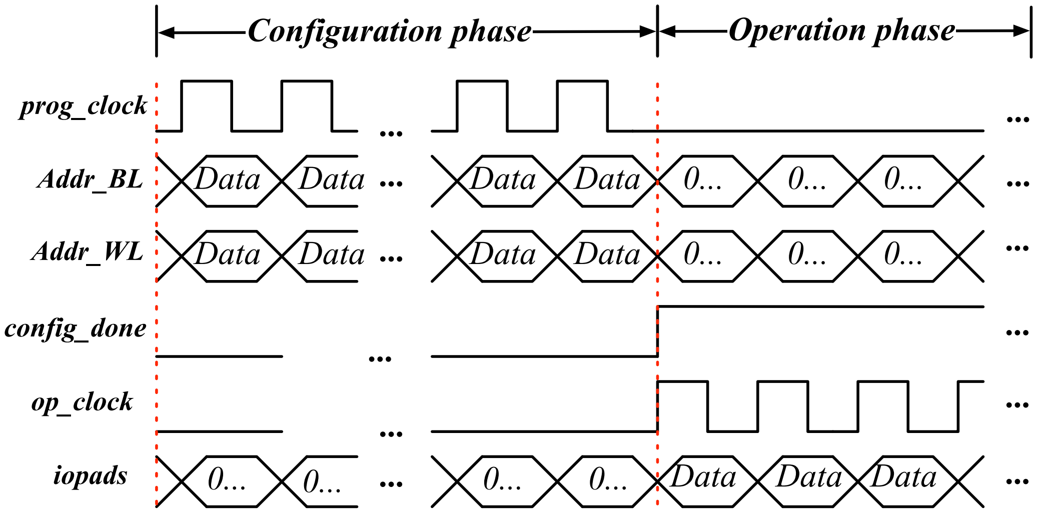


Figure 4.14 – An illustration of the waveforms for functional verification purpose.

serially according to the bitstream. In Fig. 4.14, during each programming cycle, a memory cell is configured by assigning their addresses to BL and WL decoders. During this period, the programming clock is enabled, signal $config\_done$ is disabled and all the I/Os of FPGA stuck at logic `0`.

2. Operating phase, where configuration circuits are powered off and testing input patterns are fed to all the I/Os of FPGA. During this period, the programming clock is disabled and signal $config\_done$ is enabled.

The output waveforms are then compared to the simulation results of post-logic-synthesis netlists, and ensure they are consistent. Fig. 4.15 shows the waveforms of functional verification of a simple benchmark: an inverter. The red rectangle highlights the waveform during configuration phase, while the blue rectangle highlights the waveform during operation phase. Fig. 4.15(b) presents an example of the waveforms during a programming clock cycle. We see that the BL and WL addresses are changed at each rising edge of programming clock $prog\_clock$, resulting in configuring a SRAM. Fig. 4.15(c) presents an example of the waveforms during a operating clock cycle. We see that the output $input\_B$ of FPGA is always an inversion of the input $input\_A$, revealing the correctness of functionality.

4.5.3 Studies on Runtime, Memory Usage and Accuracy

Simulating full-chip-level testbenches is the most accurate approach to power analysis at the cost of runtime and memory usage. Table 4.1 compares the runtime, memory usage and power results of full-chip/grid/component-level testbenches at different technology nodes, obtained for the MCNC big20 benchmark *s298*. Compared to the full-chip-level testbench,

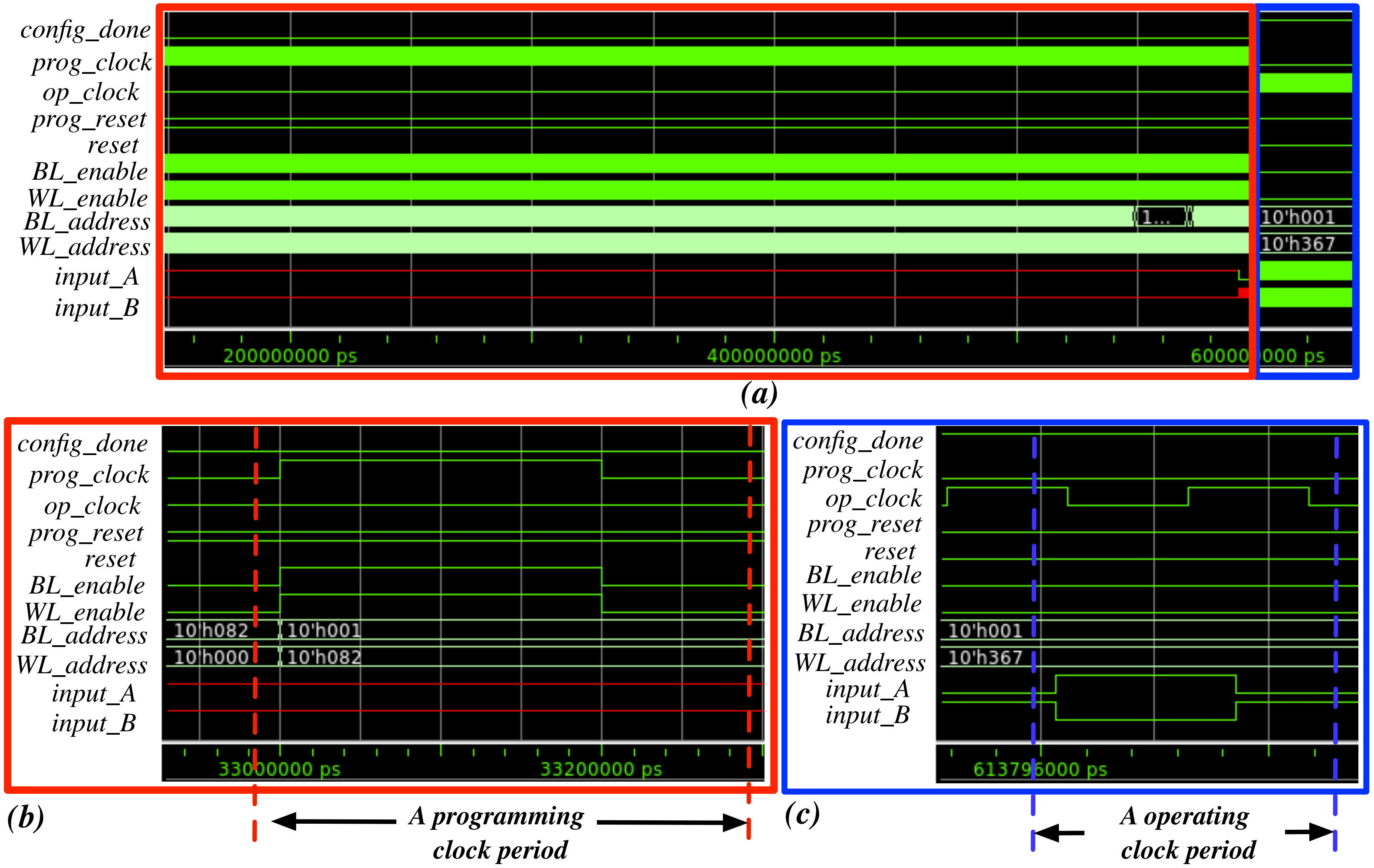


Figure 4.15 – Waveforms of a sample circuit: inverter, achieved by ModelSim simulation: (a) full waveform with configuration phase highlighted in red rectangle and operation phase highlighted in blue rectangle; (b) an example of a programming clock cycle; (c) an example of a operating clock cycle.

the grid-level testbenches achieve 12× speed-up in runtime with a moderate 14.5% error on average over the different technology nodes. Compared to the full-chip-level testbench, the component-level testbenches accelerate 14× in runtime with a 13.6% error on average over the different technology nodes. Component-level testbenches lead to the best trade-off in runtime and accuracy loss thanks to the efficient netlist partitioning strategies discussed in Section 4.4. Therefore, in the following, we use component-level power results to study power breakdowns.

Table 4.1 – Comparison of runtime, memory usage and total power of full-chip/grid/component-level testbenches for 22nm, 45nm and 180nm technology nodes in the case of the MCNC big20 benchmark *s298*.

Benchmark: s298	**Runtime (No. of minutes)**			**Improvement**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	129.48	106.15	102.56	-	-	-
Grid-level	10.27	9.82	8.25	-92%[1]	-91%[1]	-92%[1]
Component-level	7.42	6.97	6.23	-94%[3]	-93%[3]	-94%[3]

Benchmark: s298	**Peak Used Memory (Mb.)**			**Improvement**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	4780	4827	4306	-	-	-
Grid-level	768	768	825	-84%[1]	-84%[1]	-81%[1]
Component-level	589	584	621	-88%[3]	-88%[3]	-86%[3]

Benchmark: s298	**Total Power (mW)**			**Accuracy**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	1.56	4.13	15.63	100%	100%	100%
Grid-level	1.41	3.37	18.03	-9%[2]	-18%[2]	+15%[2]
Component-level	1.45	3.21	17.57	-7%[4]	-21%[4]	+12%[4]

[1] Gain(%) = (Grid-level/Full-chip-level-1) × 100%

[2] Error(%) = (Component-level/Full-chip-level-1) × 100%

[3] Gain(%) = (Grid-level/Full-chip-level-1) × 100%

[4] Error(%) = (Component-level/Full-chip-level-1) × 100%

4.5.4 Power Breakdowns

In this part, we use FPGA-SPICE to study the power breakdowns of the considered FPGA architecture. Fig. 4.16 shows the power repartition by components for the three considered technology nodes. These breakdowns are obtained by averaging the results over the complete MCNC big20 suite. In general, the routing architecture consumes 90% of the total power with the global routing architecture taking 60% of the overall power. When the technology scales down from 180nm to 22nm, the power share of the global routing architecture increases, resulting from the fact that interconnect does not scale down as the same ratio as transistors do. Indeed, the parasitic transistor capacitance decreases by 90% from 180nm to 22nm technology node but the interconnect capacitance per length is reduced by only 70% [46]. Consequently, at 22nm and 45nm technology, the number of stages in the SB tapered buffers in typically

Table 4.2 – Comparison of accuracy by modules in full-chip/grid/component-level testbenches for 22nm, 45nm and 180nm technology nodes in the case of the MCNC benchmark big20 *s298*.

Benchmark: s298	**CLB Power (mW)**			**Accuracy**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	0.42	1.06	7.85	100%	100%	100%
Grid-level	0.44	1.17	10.00	+5%[1]	+10%[1]	+27%[1]
Component-level	0.47()	1.01()	9.54()	+12%[2]	-5%[2]	+22%[2]

Benchmark: s298	**CBs Power (mW)**			**Accuracy**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	0.12	0.23	2.53	100%	100%	100%
Grid-level	0.11	0.22	2.67	-8%[1]	-5%[1]	-5%[1]
Component-level	0.11	0.22	2.67	-8%[2]	-5%[2]	-5%[2]

Benchmark: s298	**SBs Power (mW)**			**Accuracy**		
Testbench/Tech.	**22nm**	**45nm**	**180nm**	**22nm**	**45nm**	**180nm**
Full-chip-level	1.02	2.82	5.26	100%	100%	100%
Grid-level	0.86	1.99	5.37	-15%[1]	-29%[1]	+2%[1]
Component-level	0.86	1.99	5.37	-15%[2]	-29%[2]	+2%[2]

[1] Error(%) = (Grid-level/Full-chip-level-1) × 100%

[2] Error(%) = (Component-level/Full-chip-level-1) × 100%

larger in order to drive the interconnect wires. Therefore, the power share of SBs grows from 180nm to 22nm technology. The obtained results are in accordance with literature [46].

4.5.5 Accuracy Examination vs. VersaPower

In this part, we compare the power breakdown results between FPGA-SPICE and VersaPower, as shown in Fig. 4.16. FPGA-SPICE predicts that the local routing architecture requires as much power as the global routing architecture, which is different from the VersaPower. It can be explained in the following reasons. First, FPGA-SPICE takes the parasitic net activities into account which leads to additional power consumption in routing architectures. VersaPower assumes that unused resources in FPGAs can be regionally powered-off and therefore parasitic net activities can be neglected. Second, FPGA-SPICE uses electrical simulations and real configuration information from VTR, i.e., SRAM configurations in LUTs, used and unused routing multiplexer configurations, to accurately analyze the power of the architectures, while VersaPower only considers worst-case scenario and basic scaling strategies [46]. Therefore, we believe that the power results from FPGA-SPICE are more accurate and realistic.

4.5.6 Area Characteristics of SRAM-based FPGAs

With synthesizable Verilog netslits and a semi-custom design flow, FPGA-SPICE enables accurate area study for FPGAs with realistic layouts at full-chip-level, as well as fast prototyping. In this section, we consider the FPGA architecture described in Section 4.5.1 but with a reduced

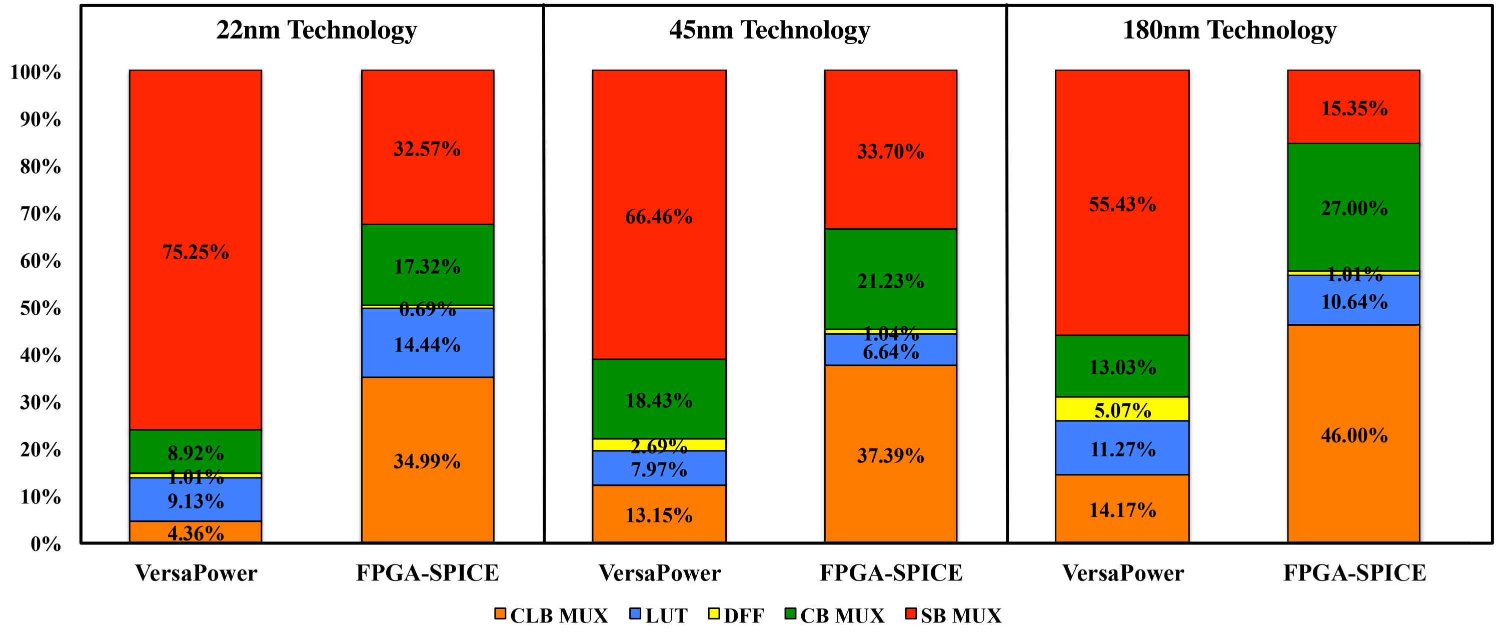
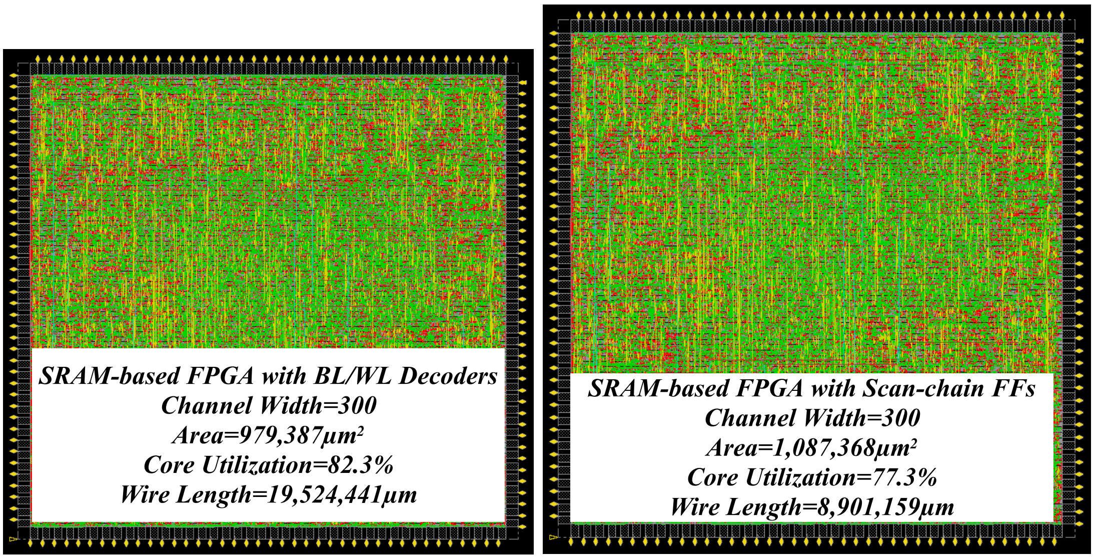


Figure 4.16 – Power breakdown results of the considered FPGA architecture between FPGA-SPICE and VersaPower averaged over the MCNC big20 benchmark suite for 22nm, 45nm and 180nm technology nodes.

CLB array size 5 × 5 and a channel width of 300, in order to fit the capability of our Linux server without losing representativity. We perform semi-custom design flows for two SRAM-based FPGA different in configuration circuits: (1) using BL and WL decoders as illustrated in Fig. 2.18; and (2) relying on scan-chain flip-flops as depicted in Fig. 2.19; The achieved full-chip-level layouts are used in studying area characteristics.



(a) *(b)*

Figure 4.17 – Full-chip layouts of 40nm SRAM-based FPGAs with CLB array size 5 × 5, a channel width of 300.

Fig. 4.17 depicts two full layouts of SRAM-based FPGA chips: (a) configured by BL and WL decoders and (b) scan-chain flip-flops, where we see most area is covered by interconnecting metal wires, illustrating its dominant impact on the total area. It is reported that 8-10% of the total area is exclusively devoted to the metal interconnect.

The total area of a SRAM-based FPGA with BL and WL decoders is reported to be $979,387\mu m^2$, which is 9% smaller than a SRAM-based FPGA with scan-chains ($1,087,368.68\mu m^2$). The area saving comes from that control lines of SRAMs can be efficiently shared among rows and columns, leading to the size of configuration circuit is square root of the number of SRAMs. But scan-chains results in the configuration circuit area to be linear to the number of SRAMs. Note that, even if the considered FPGA (array size 5 × 5 and contains 250 LUTs) is far smaller than commercial FPGAs (which can contain millions of LUTs), the number of SRAMs has already reached 180,470 ($\sim 22MB$). The choice of configuration circuits can indeed significantly impact the total area. Additionally, when a larger array size is applied, the area difference between the two FPGAs should be more significant.

Fig. 4.18 compares the area breakdown between SRAM-based FPGAs configured by BL/WL decoders and scan-chain flip-flops. The scan-chain SRAMs can occupy 46.7% of the total area, which is the major overhead. Note that the obtained area breakdown results are accordance

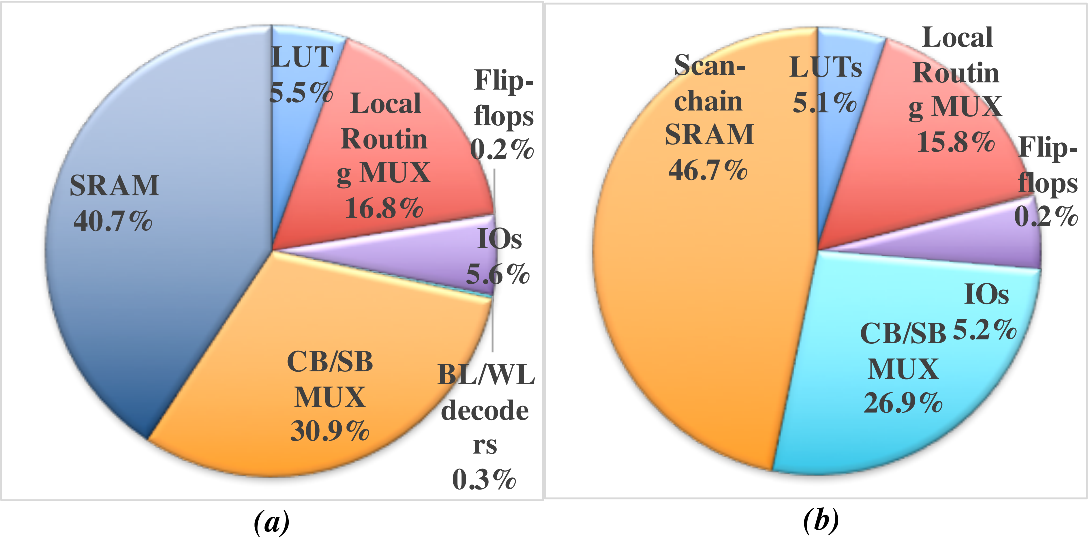


Figure 4.18 – Area breakdown of SRAM-based FPGAs which are configured by (a) BL/WL decoders, and (b) scan-chain flip-flops.

with literatures [141]. LUTs and FFs stand only up to 6% in the total area, while routing multiplexers (25-46%) are the major contributor in both SRAM-based FPGAs. Actually, the share of routing multiplexer may be even larger if we consider the area of SRAMs associated to the routing multiplexers. Note that BL and WL decoders only take 0.3% of the total area, but their share would increase when array size and channel width of FPGA increases due to the heavy use of SRAMs.

4.6 Summary

This chapter introduced FPGA-SPICE, a simulation-based architecture evaluation tool suite, enabling accurate area and power analysis. This tool extends the VTR architecture description language to include transistor-level modeling parameters of FPGA components, to capture the physical structure of I/O circuits and to model different types of configuration circuits. Tightly embedded within academic architecture exploration tool suites, FPGA-SPICE generates SPICE and Verilog netlists at different levels of complexity, considering precise technology mapping, placement and routing information as well as technological data. SPICE and Verilog netlists can be subsequently exploited for different research purposes:

1. use HDL simulator to verify the functionality of implementations;
2. use SPICE simulators to perform accurate power analysis;
3. feed a semi-custom design flow to achieve full FPGA layouts and perform accurate area analysis and enable fast prototyping.

As a general-purpose architecture evaluation framework, FPGA-SPICE can support more transistor-level circuit design topologies, such as one-level/two-level multiplexers, and such support covers peripheral circuits, such as I/O circuits and configuration circuits. FPGA-SPICE is also capable of one-level, two-level and tree-like 4T1R-based multiplexer designs presented in Chapter 3, enabling accurate architecture-level evaluations for RRAM-based FPGAs. In addition to accurate modeling for transistor-level circuit designs, FPGA-SPICE adapts netlist partitioning strategies to better trade off the runtime and memory usage of simulations with accuracy. Thanks to various techniques developed for accurate SPICE and Verilog modeling, the area and power results provided by FPGA-SPICE are more accurate and realistic, when compared to analytical power models, i.e., VersaPower. In the case study, FPGA-SPICE are used to capture the area and power characteristics of SRAM-based FPGAs with different configuration circuits. In Chapter 5, we will exploit FPGA-SPICE in studying area and power characteristics of RRAM-based FPGA architectures and compare to their SRAM-based counterparts.

5 RRAM-based FPGA Architectures

As presented in Chapter 2, SRAM-based FPGA architectures typically employ multiple levels of small crossbars, instead of large multiplexers, due to a strong limitation of SRAM-based multiplexer: whatever multiplexer structure is employed, their area, delay and power increase linearly with the input size [4]. However, in Chapter 3, we have seen an outstanding feature of RRAM-based multiplexers: their delay and power scale better with the input size and therefore the architectural design space can be extended beyond the limitations of SRAM-based multiplexers. Indeed, the properties of RRAM-based multiplexers allow the FPGA architect to size differently its routing multiplexers by: privileging one-level crossbars, made of large multiplexers, as much as possible. This paradigm shift in the interconnection topology also requires to rethink the optimal architectural parameters, which have been well determined for classical SRAM-based architectures. Hence, it is worthwhile to identify properly-sized RRAM-based FPGA architectures which can exploit the full potential of RRAM-based multiplexers, and determine the associated optimal architectural parameters.

In this chapter, we will study and optimize RRAM-based FPGAs from an architecture perspective. By exploiting VPR [44] and FPGA-SPICE (introduced in Chapter 4), we perform architecture-level simulations to:

1. determine the proper R_{HRS} for RRAM-based FPGA architectures;
2. study area and power characteristics of RRAM-based FPGAs over their SRAM-based counterparts;
3. validate the impact of architecture-level optimizations.
4. investigate the delay and power efficiency of near-V_t RRAM-based FPGAs

This chapter will be divided to two parts: Section 5.1 presents the generality of RRAM-based FPGA architectures studied in this chapter and demonstrates the area and power characteristics of general RRAM-based FPGA architecture by using FPGA-SPICE. Section 5.2 proposes three architecture-level optimizations for RRAM-based FPGAs and validate their impacts.

5.1 General Vision

The RRAM-based FPGA introduced in this thesis has no architectural difference with respect to the conventional SRAM-based FPGA shown in Fig. 2.6. It remains an island-style FPGA where the cluster-based CLBs are surrounded by SBs and CBs. The differences lie in the circuit design of those modules heavily relying on SRAMs, i.e., LUTs and multiplexers. Fig. 5.1 and Fig. 5.2 compare the circuit designs of LUT and multiplexer between a conventional SRAM-based FPGA and the RRAM-based FPGA introduced in this thesis.

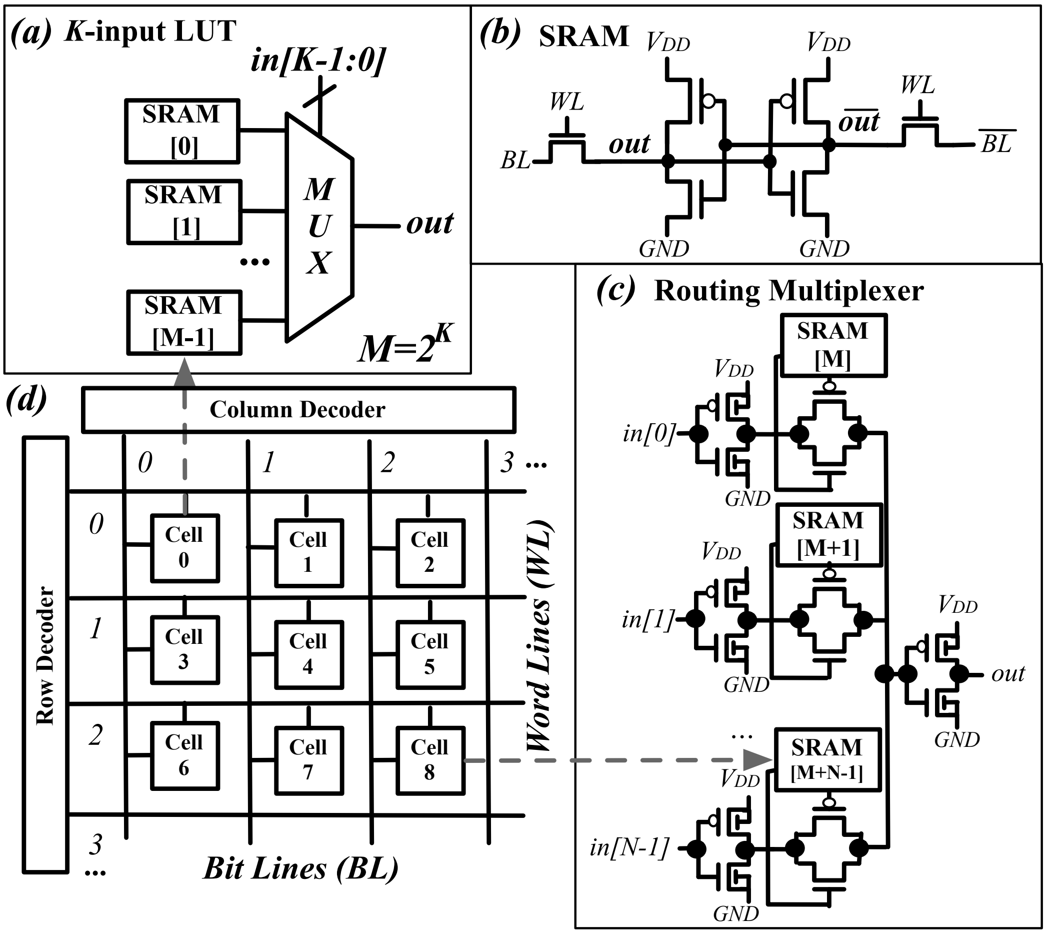


Figure 5.1 – Memory access organization in SRAM-based FPGA: SRAMs are placed in an array and SRAMs in the same column/row share the same BL/WL.

5.1.1 Choice of Non-volatile Modules

In our FPGA, the logic elements exploit *Non-Volatile* (NV) LUTs. Such FPGA does not need to be re-programmed during each power on and can benefit instant-*on* and normally-*off* properties. Typically, a LUT consists of a bank of SRAMs and a multiplexer as shown in Fig

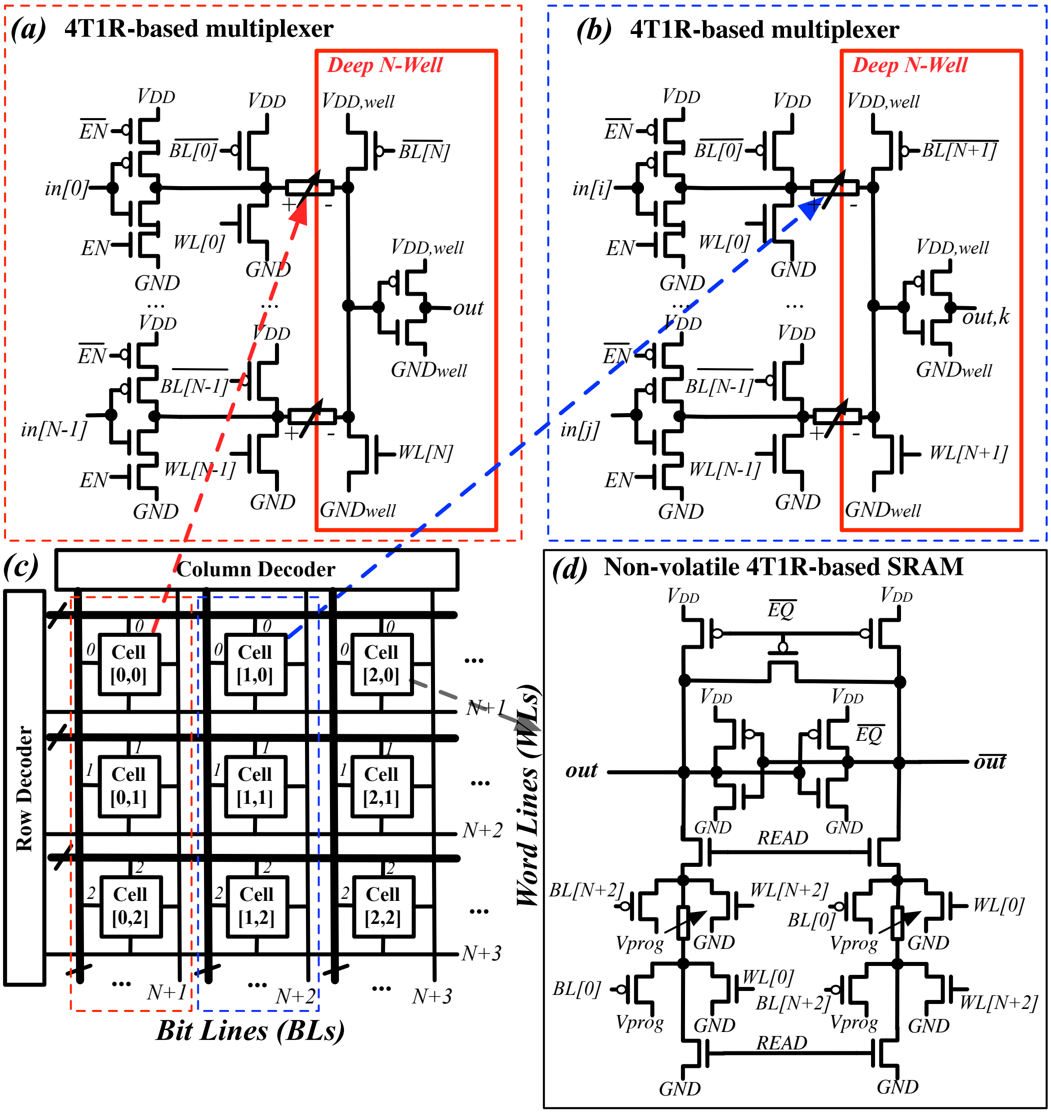


Figure 5.2 – Memory access organization in RRAM-based FPGA: RRAMs belonging to the same multiplexer/NV SRAM are placed in the same column and share BL/WL.

5.1(a). The SRAM bank stores a truth table which is decoded by the multiplexer, enabling LUT to realize any logic function. In this chapter, we replace the SRAMs (Fig. 5.1(b)) in LUTs with *Non-Volatile* (NV) SRAMs borrowed from previous work [5]. Note that the NV SRAM used in this thesis (Fig. 5.2(b)) employ 4T1R programming structures to configure RRAMs, instead of 2T1R programming structures in [5].

The multiplexers in LUTs are still implemented by pass-transistors considering that their decoding results keep changing when the FPGA is operating. If RRAMs are inserted in the data path of LUTs for decoding, their operating speed will drastically limit frequency. Compared to SRAM-based, the NV LUTs have no difference in performance because of the same decoder implementation. Data path DFFs are also *Non-Volatile* with the same circuit elements. These FFs operate as standard volatile CMOS FF during regular operation but they are also capable to store the data *non-volatily* on demand before a sleep period. Data stored in the NV DFFs can then be restored during wake up. In these flip-flops, RRAMs are written only before the sleep period. These events have very low frequency and are compatible with the endurance capabilities of RRAMs. While supported by the presented architecture, instant-*on* and normally-*off* operation will not be evaluated in this thesis. Similar to NV SRAM, the NV FFs in this thesis also employ 4T1R programming structures to configure RRAMs. More details about the NV DFF architecture can be found in [5].

While the decoded paths of the LUT multiplexer change at runtime, the selected paths in the routing multiplexers (i.e., in BLE output selector, local routing, SBs and CBs) remain unchanged during runtime. Note that we do not consider partial reconfiguration during runtime for FPGA architectures in this thesis. Therefore, RRAMs can be inserted in the data path of routing architecture without challenging the endurance. Fig. 5.2(a)(b) illustrate the 4T1R-based multiplexer introduced in Chapter 3, which replaces the SRAM-based multiplexer shown in Fig. 5.2(c). Compared to the SRAM-based multiplexers, the 4T1R-based multiplexers exhibit both high performance and low-power accounted to the low R_{LRS} of the RRAMs and smaller parasitic capacitances introduced in the data path.

5.1.2 Configuration Circuits

SRAMs in FPGAs can be configured through *Bit Lines* (BLs) and *Word Lines* (WLs), similar to the principle of memory bank, as depicted in Fig. 5.1. SRAMs are organized in an array, where SRAMs in one column share the same BL, while SRAMs in one row share the same WL. As such, the number of BLs and WLs are square root to the number of SRAMs, leading to small BL and WL decoders. To configure a SRAM, the associated WL is enabled while the configuration bit is fed to the corresponding BL. Note that during configuration, other BLs and WLs should be disabled in order to avoid mistakenly accessing other SRAMs in the same column/row. In this rest of this chapter, our baseline SRAM-based FPGAs employ the BL/WL decoders in Fig. 5.1 to access each SRAM.

In our RRAM-based FPGA architecture, each RRAM is accessed by BLs and WLs as well but requires a different BL and WL sharing strategy. BLs and WLs of each 4T1R-based multiplexer and each RRAM of LUTs are divided into two groups:

1. Common BLs and WLs that are shared by all the 4T1R-based multiplexer and also the RRAM of LUTs. Take the example in Fig. 5.2(b), (c) and (d), the two N-input 4T1R-based multiplexers share $BL[0...N-1]$ and $WL[0...N-1]$, and the NV SRAM share $BL[0]$ and $WL[0]$ with the multiplexers. Considering the different input size of multiplexers in FPGA architecture, the number of shared BLs and WLs is determined by the largest input size of multiplexers.

2. Independent BLs and WLs, which are unique for each 4T1R-based multiplexer and also RRAM of LUTs. As shown in Fig. 5.2(c) and (d), the programming transistors close to output inverters in the two 4T1R-based multiplexer are controlled by two unique BLs and WLs, $(BL[N], WL[N])$ and $(BL[N+1], WL[N+1])$, respectively. Similarly, the NV SRAM in Fig. 5.2(b) has an unique pair of BL and WL, $BL[N+2], WL[N+2]$.

As such, each RRAM can be configured in the same way of assigning BL and WL signals as SRAM-based FPGAs. Since each RRAM has a unique address, it is accessible only when its associated couple of BL/WL is activated, providing the programming current exclusively for one RRAM. Therefore, the BL and WL sharing strategy in Fig. 5.2 can avoid parasitic programming and guarantee the number of BLs and WLs linear to the number of NVSRAMs and 4T1R-based multiplexers.

Indeed, our RRAM-based FPGA architecture requires more BLs and WLs than SRAM-based, leading to large decoder circuits and potentially area overhead. However, our RRAM-based FPGA eliminates the use of SRAMs in routing multiplexers, bringing significant area reduction. Considering that in general routing multiplexers occupies more than 50% of the total area, the area overhead from decoder circuits can be fully compensated by the 4T1R-based multiplexers. Overall, our RRAM-based FPGA will be area efficient as its SRAM-based counterpart or even better, depending on the scale of routing architecture. In Section 5.1.4, we will focus on study the area characteristics of proposed RRAM-based FPGA architecture with layout-level results.

5.1.3 Experimental Methodology

To be representative, both the SRAM-based and the RRAM-based FPGA architectures consider the same set of architectural parameters: $K = 6$, $N = 10$, $I = 40$, $F_{c,in} = 0.15$, $F_{cout} = 0.1$, $F_s = 3$ and $L = 2$, with unidirectional routing architecture. SRAM-based and RRAM-based FPGAs employ the configuration circuits depicted in Fig. 5.1 and Fig. 5.2 respectively. Note that in this section, we focus on studying the difference in area and power characteristics of SRAM-based and RRAM-based FPGAs. The area and power of hard adder chains and heterogeneous blocks are highly dependent on the choice of *Intellectual Property* (IP) blocks, and hence they are not included in the evaluated FPGA architectures here. In addition to the core logic of FPGAs, i.e., LUTs, FFs and routing multiplexers, the architecture evaluation in this section includes peripheral circuitry, i.e., I/O pads, BL and WL decoders, in order to draw realistic conclusions.

In terms of the circuit designs, both SRAM-based and RRAM-based FPGAs are built with a commercial 40nm technology. All the multiplexers and LUTs use transmission gates and are also buffered according to their realistic fan-out in the architectural context. For SRAM-based FPGAs, LUTs employ the design in Fig. 2.16 (Section 2.2.3). For best area-delay-power product, routing multiplexers in local routing architecture adopt a two-level multiplexing structure, as shown in Fig. 2.15(a) (Section 2.2.3). Routing multiplexers in global routing architecture, i.e., CBs and SBs, consider a one-level multiplexing structure, as shown in Fig. 2.14(b) (Section 2.2.3). For RRAM-based FPGAs, routing multiplexers uniformly adopt a one-level 4T1R-based multiplexing structure for best area-delay-power product, which has been introduced detailedly in Chapter 3. The 4T1R-based multiplexers are properly sized by the optimization techniques introduced in Section 3.7. Similar to Section 3.8.1, we consider the Stanford RRAM model [130] with the following parameters: $R_{LRS} = 5k\Omega$, R_{HRS} ranging from $1M\Omega$ to $200M\Omega$, $I_{set} = 500\mu A$, $V_{set} = V_{reset} = 1.1V$. The parasitic capacitance of a RRAM is considered to be $C_P = 13.2aF$. RRAM-based FPGAs follow the principle explained in Section 5.1.1. Both SRAM-based and RRAM-based FPGA architectures have passed the functionality verification with FPGA-SPICE, validating that they can be configured and also operate correctly.

Area results are based on analyzing full FPGA layouts generated by a semi-custom design flow. FPGA-SPICE are used to provide Verilog netlists containing a full FPGA chip for the semi-custom design flow (See Section 4.1). The experiments are conducted on a workstation with 256G memory and Xeon processors. For sake of the capability of our workstation, we consider a CLB array size of 5 × 5 and swept the channel width from 50 to 300 with a step of 50 for both FPGA architectures, which are surrounded by 160 I/O pads. Note that the achieved area results with a 5 × 5 CLB array can be representative because large FPGAs can be regarded as an assembly of the small CLB arrays. Studying area characteristics of large FPGAs will be part of the future works.

Power results are achieved by SPICE simulations. FPGA-SPICE automatically generates the component-level testbenches and latest HSPICE simulator (Version 2017.03) perform power analysis. The power analysis considers FPGA architectures implemented with the twenty biggest MCNC benchmarks [138]. Note the power analysis will focus on the core logic of FPGAs, that is LUTs, FFs and routing multiplexers, in order to examine the architectural impact of RRAM-based circuit designs. I/O pads and configuration circuits are not included.

Note that the methodology developed here is not dependent on the considered RRAM technology or on the transistor technology nodes or even the circuit design topology, but is rather general.

5.1.4 Area Characteristics

Fig. 5.3 compares the full-chip layouts of SRAM-based and RRAM-based FPGAs, both of which contain a 5 × 5 CLB array and a global routing architecture with a channel width of 300, as well as I/O pads and BL/WL decoders.

Fig. 5.4(a) and (b) compare the area breakdown of RRAM-based and SRAM-based FPGA chips when channel width is set to 300. In both FPGAs, routing multiplexers occupy > 40% of the total area, while LUTs and FFs only have a ~ 6% share. More than 40% of the total area is consumed by SRAMs in the SRAM-based FPGA, while only 15% of the total area is consumed NV SRAMs in the RRAM-based FPGA. Note that BL/WL decoders take 4.5% of the total area in RRAM-based FPGA while they are negligible in SRAM-based FPGA. This is due to the BL and

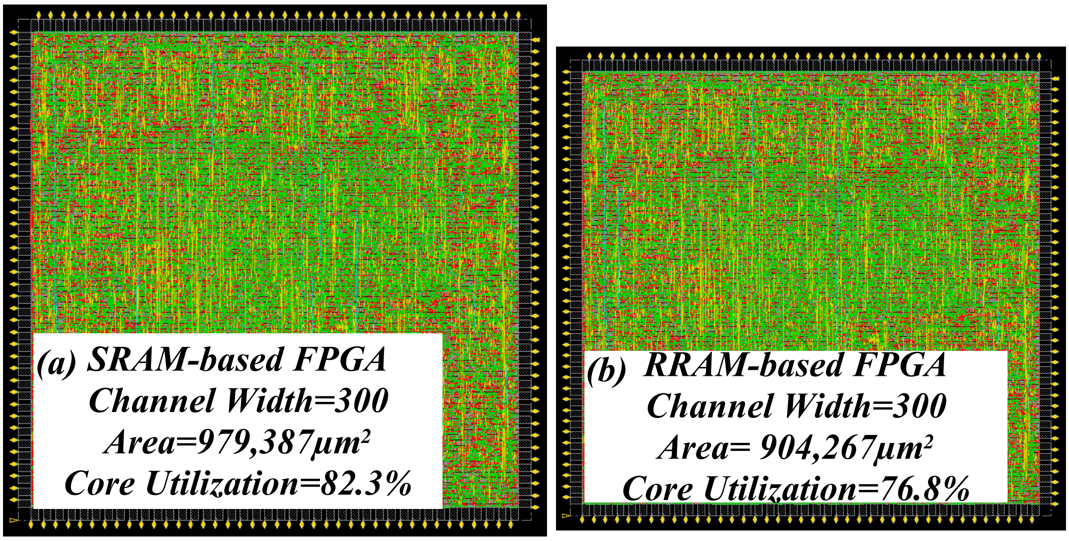
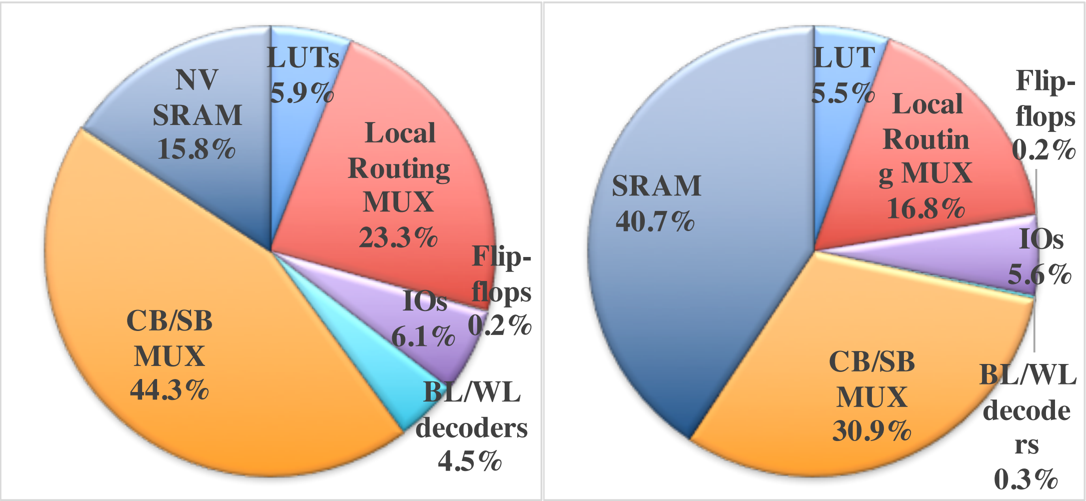


Figure 5.3 – Full-chip layouts of 40nm SRAM-based and RRAM-based FPGAs with CLB array size 5 × 5.



(a) *(b)*

Figure 5.4 – Area breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA.

WL sharing strategy in RRAM-based FPGA is not as efficient as SRAM-based FPGA (See Section 5.1.2). Therefore, improving the BL and WL sharing strategy is well worth investigation and is part of the future work. Thanks to 4T1R-based multiplexers, the area of routing multiplexer in RRAM-based FPGA is smaller than SRAM-based FPGA because the SRAMs are eliminated. Indeed, the SRAM-based FPGA contains 180,470 SRAMs, while the RRAM-based FPGA reduce the number to only 16,160 NV SRAMs. Despite the reduced number of volatile elements, we see the total area of SRAM/NV SRAM is similar due to the large area of NV SRAM. As shown in Fig. 5.1(b) and Fig. 5.2(b), a NV SRAM requires 12 more transistors than a normal SRAM, resulting in an area overhead as large as 6×. Therefore, compact NV SRAM designs can be another challenge in RRAM-based FPGA study.

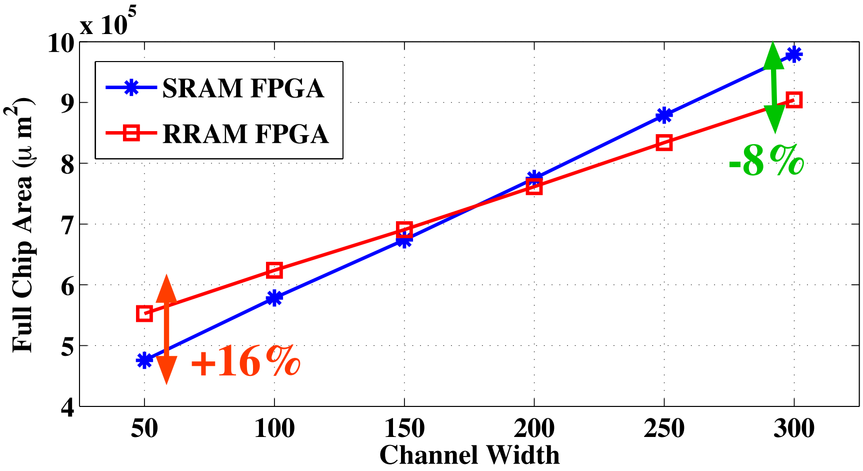


Figure 5.5 – Full-chip area comparison between SRAM-based and RRAM-based FPGAs by sweeping channel widths from 50 to 300.

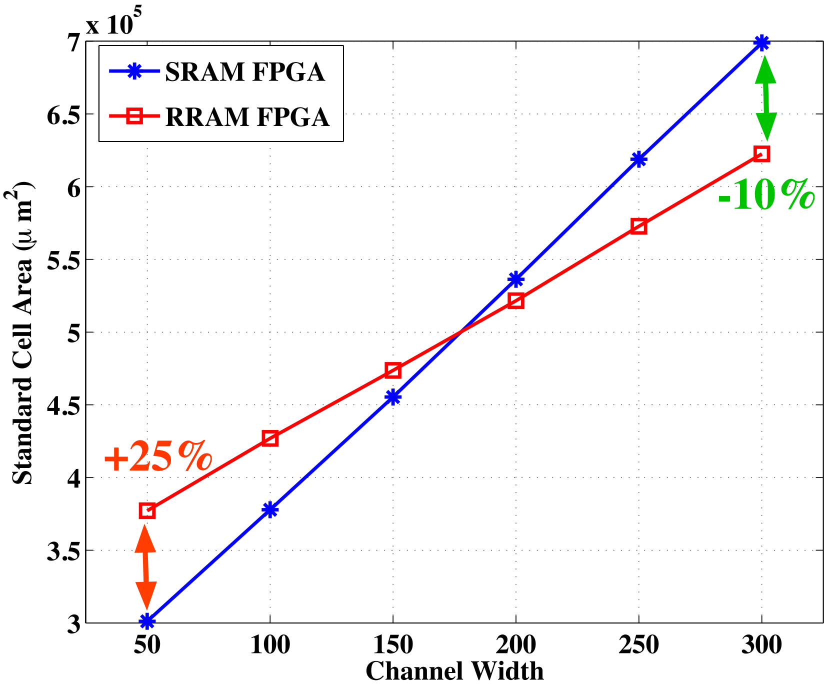


Figure 5.6 – Standard cell area comparison between SRAM-based and RRAM-based FPGAs by sweeping channel widths from 50 to 300.

Fig. 5.5 compares the full-chip area of SRAM-based and RRAM-based FPGAs by considering different channel widths. When channel width is smaller than 250, the proposed RRAM-based

FPGAs require more area than their SRAM-based counterparts. The area overheads results from two factors:

1. The number of routing multiplexers is positively related to the channel width. Indeed, RRAM-based multiplexers are more area efficient than their SRAM-based counterparts. However, when channel width is small, the area saved by RRAM-based multiplexers cannot fully mitigate the area overhead of NV SRAMs.
2. RRAM-based FPGAs potentially requires more area exclusively devoted to routing metal wires. Fig. 5.6 compares the total area of standard cells in SRAM-based and RRAM-based FPGAs by considering different channel widths. Considering the case where channel width is 200, the total area of standard cells in RRAM-based FPGAs is smaller than SRAM-based implementations, while the full-chip area of RRAM-based FPGAs is larger. This implies that RRAM-based FPGAs contain more routing area than SRAM-based FPGAs. According to detailed area reports, there is 30% of the full-chip area is exclusively dedicated to routing wires in RRAM-based FPGAs while only 20% of the full-chip area is exclusively dedicated to routing wires in SRAM-based FPGAs.

Therefore, when channel width is larger than 250, the proposed RRAM-based FPGAs become more area efficient than SRAM-based FPGAs, owing to the increased number of routing multiplexers in global routing architecture. We see the proposed RRAM-based FPGA consumes 4% and 10% smaller in terms of full-chip area and total area of standard cells respectively, as compared to the SRAM-based FPGA, when channel width is set to 300. And we believe that the area reduction can be more significant when larger channel widths are applied.

5.1.5 Power Characteristics

As presented in Chapter 3, SRAM-based and RRAM-based multiplexers have different structures, which lead to differences in power characteristics. Chapter 3 focused on comparing the power characteristics of SRAM-based and RRAM-based multiplexers at circuit-level. However, non-volatility allows RRAM-based FPGAs to be normally powered off and instantly powered on, leading to different power characteristics at architecture-level. As illustrated in Fig. 1.2, RRAM-based FPGA can be simply powered-off during long idle period, consuming zero static power. Therefore, studying the power characteristics of RRAM-based FPGAs should focus on the static and dynamic power consumed during standard operation time.

In addition, similar to SRAM-based multiplexers, whose static power is mainly determined by the *off*-resistance of transistors, static power of 4T1R-based multiplexers is highly dependent on the R_{HRS} of RRAMs. At the first glance, R_{HRS} should be as large as the *off*-resistance of a transistor in order to keep the a low static power consumption [114]. However, thanks to the non-volatility, the lower bound of R_{HRS} can be relaxed owing to the following considerations:

1. Static power of RRAM-based FPGA only occurs during standard operation time, which

is typically along with high dynamic power consumption.

2. RRAM-based FPGAs still include pure CMOS circuits, such as LUTs, FFs and tapered buffers, which can alleviate the impact of R_{HRS} on total static power.

3. Dynamic power of 4T1R-based multiplexers is smaller than CMOS multiplexers (See Section 3.8), leaving more budget in static power during standard operation time.

4. As explained in Section 5.1, RRAM-based FPGA requires less volatile elements, potentially reducing the power consumption.

Therefore, the choice of R_{HRS} should be studied in the context of FPGA architecture, rather than in the context of standalone 4T1R multiplexers.

In this section, we will analyze the power consumption of RRAM-based FPGAs from an architecture perspective. We first study the static power characteristics of 4T1R-based multiplexers by considering the architectural context. We then study the impact of R_{HRS} on the power consumption of RRAM-based FPGAs during standard operation time.

Static Power of 4T1R-based Multiplexers

The static power of a multiplexer is dominated by the number of the leakage paths from VDD to GND and also the resistance of sneak paths. We study the N-input multiplexers in Fig. 5.7 as an example and focus on analyzing what dominates the static power of 4T1R-based multiplexers. We will focus on the leakage paths through input inverters, transmission gates and programming structures since they are highly sensitive to the input size and input patterns. Without losing generality, we assume that the inputs $in[0]$ of both SRAM-based and 4T1R-based multiplexers in Fig. 5.7(a) and (b) are propagated to the output node.

In Fig. 5.7, we see that RRAM-based multiplexers contain more leakage paths than SRAM-based implementations. The pull-up transistors of programming structures introduce additional sources of leakage paths and the pull-down transistors of programming structures lead to additional sources of leakage paths. Note that even though the programming transistors are all turned off during operating period, they indeed increase the leakage current from V_{DD} to GND.

Take the example of Fig. 5.7(a), assume that $in[0]$ is set to GND and $in[N-1]$ is set to V_{DD}, transmission gate **tg0** is turned on while transmission gate **tg1** is turned off. A leakage paths can start from a p-type transistor **p0**, pass through transmission gates **tg0** and **tg1**, and end at a n-type transistor **n1**. We define the resistance of a transistor in *on* state as R_{on} while the resistance of a transistor in *off* state is denoted by R_{off}. Since typically $R_{off} >> R_{on}$, the resistance of the leakage path **p0** to **tg0** to **tg1** to **n1** is dominated by R_{off}:

$$R_{leak1} = R_{on} + R_{on}||R_{on} + R_{off}||R_{off} + R_{on} \approx R_{off}/2 \tag{5.1}$$

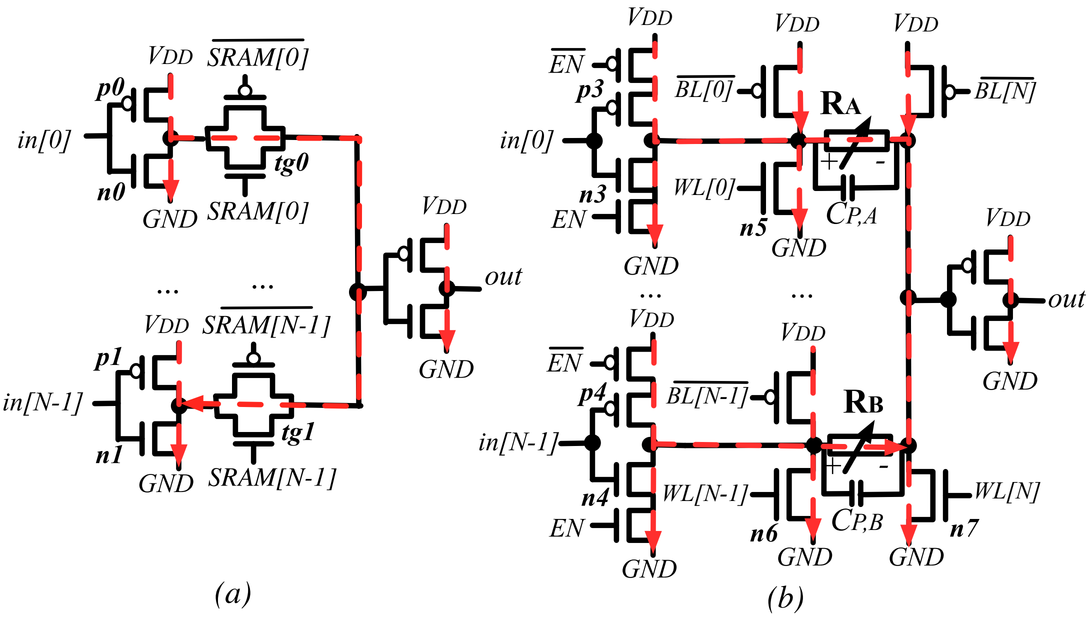


Figure 5.7 – Leakage paths of N-input multiplexers: (a) SRAM-based (b)RRAM-based

The leakage power contributed by **p0** → $tg0$ → $tg1$ → **n1** is:

$$P_{leak1} \approx 2V_{DD}^2/R_{off} \tag{5.2}$$

Similarly, in the 4T1R-based multiplexer (Fig. 5.7(b)), assume that $in[0]$ is set to GND and $in[N-1]$ stuck at V_{DD}, RRAM R_A is in LRS while RRAM R_B is in HRS. Note that all the programming transistors are in *off* state during operating mode. Compared to the SRAM-based multiplexer in Fig. 5.7(a), the leakage path starting from a p-type transistor **p3** in Fig. 5.7(b) has more ending points, due to the programming transistors connected to GND. A leakage path can start from a p-type transistor **p3**, pass through RRAM R_A and RRAM R_B, and end at a n-type transistor, such as **n4**, **n5**, **n6** and **n7**. Table 5.1 lists the leakage paths from **p3** to **n4**, **n5**, **n6** and **n7** and their resistance.

Table 5.1 – Resistance of leakage paths of the 4T1R-based multiplexer in 5.7(b) whose starting point is **p3** and ending points are **n4**, **n5**, **n6** and **n7**

Leakage paths	Resistances on leakage paths
Path 1: **p3** → R_A → R_B → **n4**	$R_{on} + R_{LRS} + R_{HRS} + R_{on}$
Path 2: **p3** → **n5**	$R_{on} + R_{off}$
Path 3: **p3** → R_A → R_B → **n6**	$R_{on} + R_{LRS} + R_{HRS} + R_{off}$
Path 4: **p3** → R_A → **n7**	$R_{on} + R_{LRS} + R_{off}$

Note that $R_{HRS} >> R_{on}$, the resistance of the leakage path listed in Table 5.1 is dominated by

R_{HRS} and R_{off}. As a result, the leakage power contributed by the leakage paths in Table 5.1 is

$$P_{leak1} \approx 2V_{DD}^2/R_{HRS} + 2V_{DD}^2/R_{off} \quad (5.3)$$

which is obviously larger than the leakage power contribution in Equation 5.2. We see that in Equation 5.3, R_{HRS} is one of the important factors influencing the leakage power.

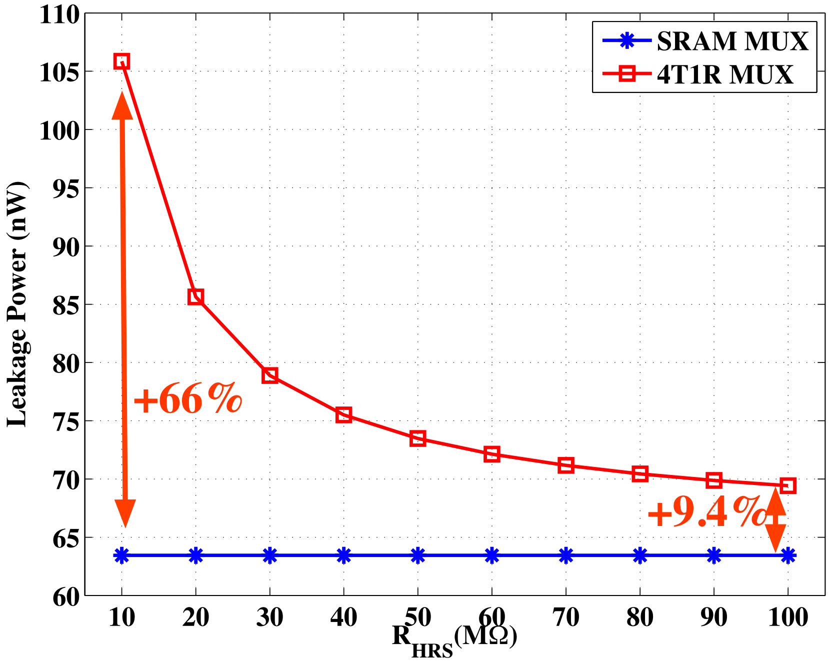


Figure 5.8 – Impact of R_{HRS} on the average static power of a 2-input 4T1R-based multiplexer

In the rest of this section, we will rely on simulation results in studying the impact of R_{HRS} on the leakage power of 4T1R-based multiplexers, rather than a full analysis on the leakage paths. Fig. 5.8 compares the average leakage power of a 2-input 4T1R-based multiplexer to its SRAM-based counterpart by sweeping R_{HRS} from $10M\Omega$ to $100M\Omega$. The leakage power overhead can be limited to 9.5% when $R_{HRS} = 100M\Omega$. Note that the simulation results is achieved by enumerating all the possible input patterns for both SRAM-based and RRAM-based 2-input multiplexers. Additionally, considering the architectural context, multiplexers, such as those of *Switch Blocks* (SBs), usually contain tapered buffers at their outputs, which can also reduce the leakage power overhead. Fig. 5.9 depicts the average leakage power of a 2-input 4T1R-based and SRAM-based multiplexers with tapered buffers at outputs. Note that when $R_{HRS} = 10M\Omega$, the leakage overhead is reduced to 33% as compared to Fig. 5.8.

However, due to that the number of input patterns is exponential to the input size, it is unrealistic to enumerate all the input patterns for a N-input multiplexer, in order to conduct a full simulation-based analysis on the leakage power. Furthermore, due to the diverse configurations, any combination of propagating path and input pattern can happen to all the

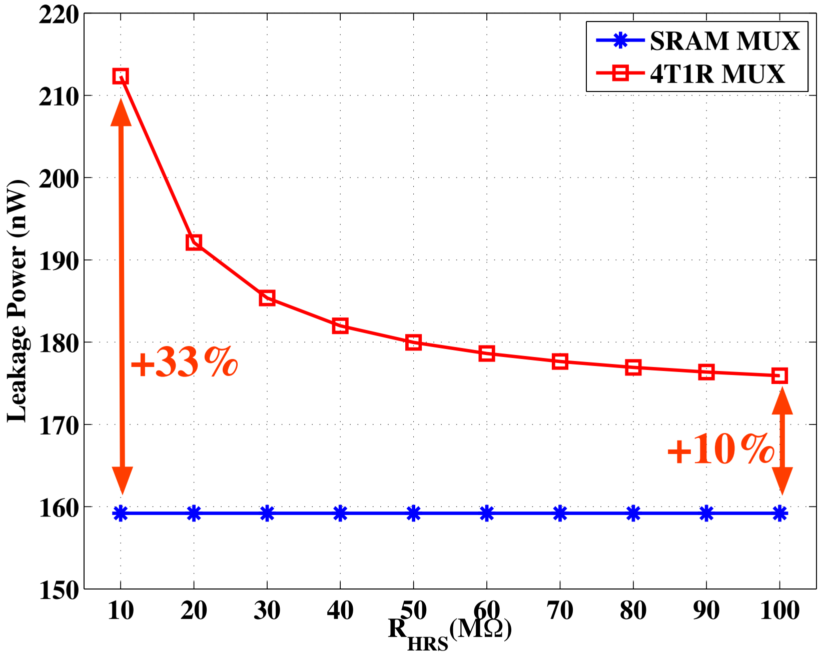


Figure 5.9 – Impact of R_{HRS} on the average static power of a 2-input 4T1R-based multiplexer with tapered buffer at output

multiplexers of FPGA. Therefore, accurate leakage power analysis for RRAM-based FPGAs should consider electrical simulations based on realistic circuit implementations. In addition, thanks to the non-volatility, static power of RRAM-based FPGA only occurs during standard operation time, which is typically along with high dynamic power consumption. Hence, in the rest of this thesis, the power analysis on RRAM-based FPGAs consider both the static and dynamic power consumed during standard operation time.

Impact of R_{HRS} on Power Consumption

As explained in Section 5.1.5, the R_{HRS} can influence the power consumption of RRAM-based routing elements. We evaluate in Fig. 5.10 the impact of R_{HRS} on the average power of the considered FPGA architectures implementing in MCNC big20 benchmarks by using FPGA-SPICE. Basically, the power consumption of RRAM-based FPGA increases as R_{HRS} decreases. Note that the power differences between RRAM-based and SRAM-based FPGAs is within 3% when R_{HRS} is $20M\Omega$. And when R_{HRS} is larger than $20M\Omega$, RRAM-based FPGAs becomes more power efficient than SRAM-based FPGAs. In particular, when $R_{HRS} = 100M\Omega$, RRAM-based FPGAs consumes 23% less power than SRAM-based FPGAs. Indeed, RRAM-based multiplexers consume larger leakage power consumption than their SRAM-based counterparts. Also, in Chapter 3, we have presented that RRAM-based multiplexers are more power efficient in terms of dynamic power than SRAM-based implementations. Therefore, when R_{HRS} is smaller than

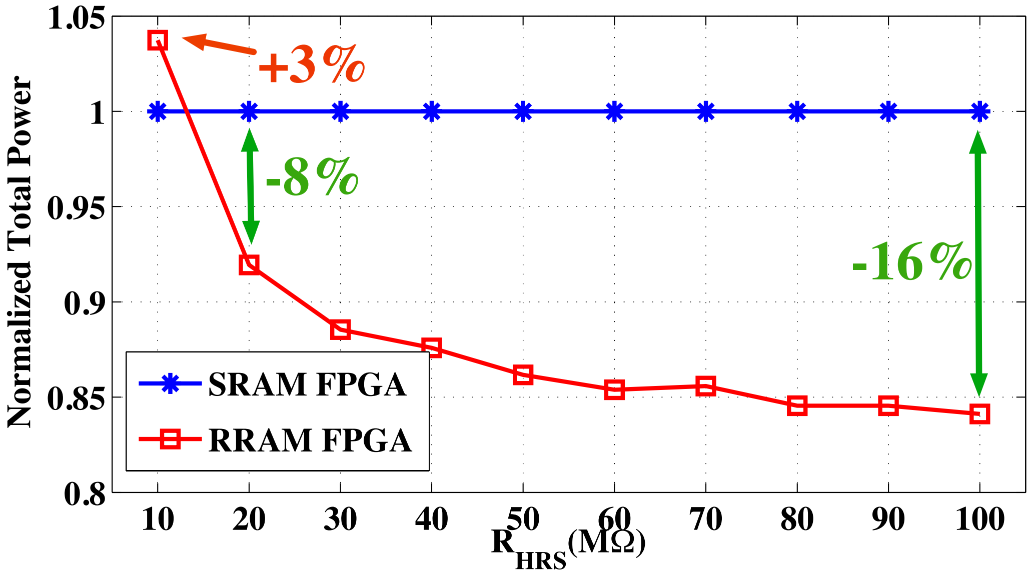


Figure 5.10 – Normalized power consumption of SRAM-based and RRAM-based architectures with different R_{HRS}

$20M\Omega$, the leakage power overheads of RRAM-based FPGAs is too large and shadows the gain in dynamic power, resulting in power overhead in total. When R_{HRS} is large than $20M\Omega$, the dynamic power advantages can fully mitigate the leakage power overhead, contributing to power reduction in total. Note that all the power results in Fig. 5.10 are achieved when both SRAM-based and RRAM-based FPGAs operate at nominal working voltage. Since that RRAM-based circuits exhibit high-performance especially in near-V_t regime (See Chapter 3), RRAM-based FPGAs can be more delay efficient than SRAM-based FPGAs when the working voltage is reduced to near-V_t. Note that such high performance is achieved along with the power reduction. Therefore, in terms of *Power-Delay Product*, the minimum requirements of RRAM devices in FPGAs can be further relaxed.

In this thesis, we consider $R_{HRS} = 20M\Omega$ as the minimum requirement for RRAM devices, in order to ensure the power efficiency of RRAM-based FPGAs.

Power Breakdown

In this section, we study the power breakdown of RRAM-based FPGA and compare to its SRAM counterpart. To be fair, we consider $R_{HRS} = 20M\Omega$ for RRAM-based FPGA, which guarantees zero power difference between RRAM-based and SRAM-based FPGAs average over MCNC big20 benchmarks (See Section 5.1.5). Fig. 5.11 compares the static power breakdown between RRAM-based and SRAM-based FPGAs. In general, routing multiplexers consumes over 40% of the total static power, while LUTs and FFs only consumes up to 20% of the total. Due to the heavy use of SRAMs, 36% of the static power is consumed by SRAMs in SRAM-based FPGA.

Differently, in RRAM-based FPGA, only 14% is required by SRAMs. This reduced share of SRAM power comes from that 4T1R-based multiplexers eliminate the use of SRAMs, giving more power budget to other components.

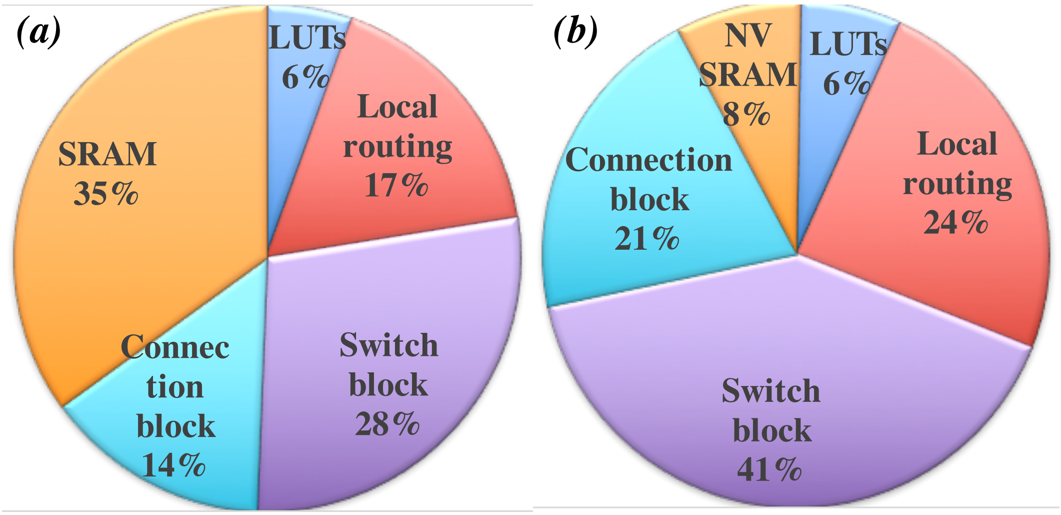


Figure 5.11 – Static power breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA.

Fig. 5.12 compares the dynamic power breakdown between RRAM-based and SRAM-based FPGAs. We see that over 70% of the total power is consumed by routing multiplexers, while only 12% is consumed by LUTs and FFs. By removing the SRAMs in routing multiplexers, the power share of SRAMs is reduced from 14% (SRAM-based FPGA) to 5% in RRAM-based FPGA.

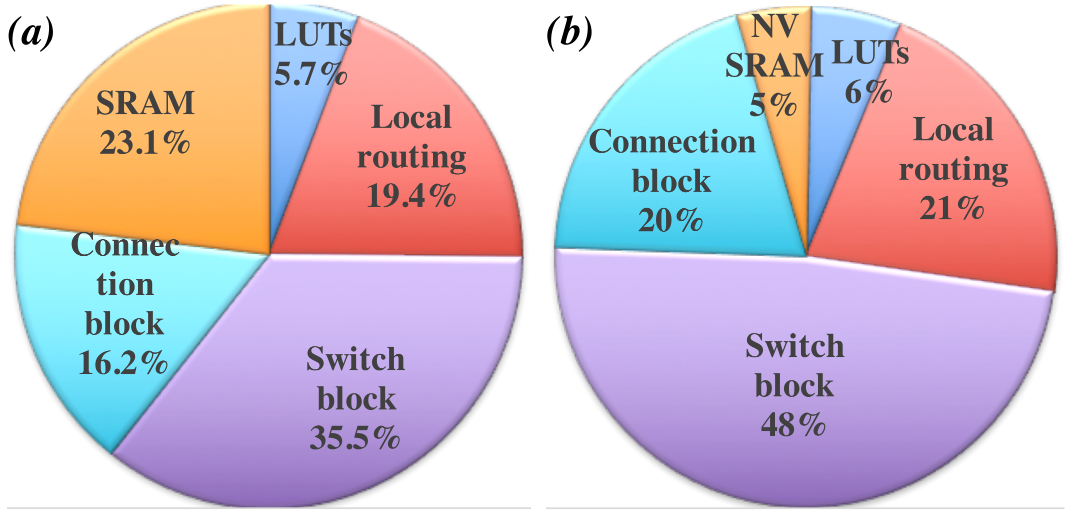


Figure 5.12 – Dynamic power breakdown of (a) RRAM-based FPGA and (b) SRAM-based FPGA.

5.1.6 Overall Performance

Fig. 5.13 compares the overall performance of SRAM-based and RRAM-based FPGAs operating in both nominal and near-V_t regimes. When operating at nominal voltage ($V_{DD} = 0.9V$), RRAM-based FPGA can improve delay by 22% over its SRAM-based counterpart. Even when V_{DD} is reduced to near-V_t regime, i.e., $0.8V$, RRAM-based FPGA remains at the same performance-level as the SRAM-based FPGA at nominal voltage. Significantly, the near-V_t RRAM-based FPGA benefits from energy reduction, leading to a $2-2.3\times$ improvement on energy. Note that the energy of RRAM-based FPGA operating at $V_{DD} = 0.9V$ is similar to the best SRAM-based FPGA ($V_{DD} = 0.7V$). In terms of *Energy-Delay Product* (EDP), SRAM-based FPGA at nominal voltage is the best, while RRAM-based FPGA at $V_{DD} = 0.8V$ is the best with a close to $2\times$ improvement compared to the best SRAM-based FPGA. Note that only runtime power consumption is evaluated in Fig. 5.13. We believe that the energy improvement of RRAM-based FPGA can go beyond $2\times$, when non-volatility is taken into account.

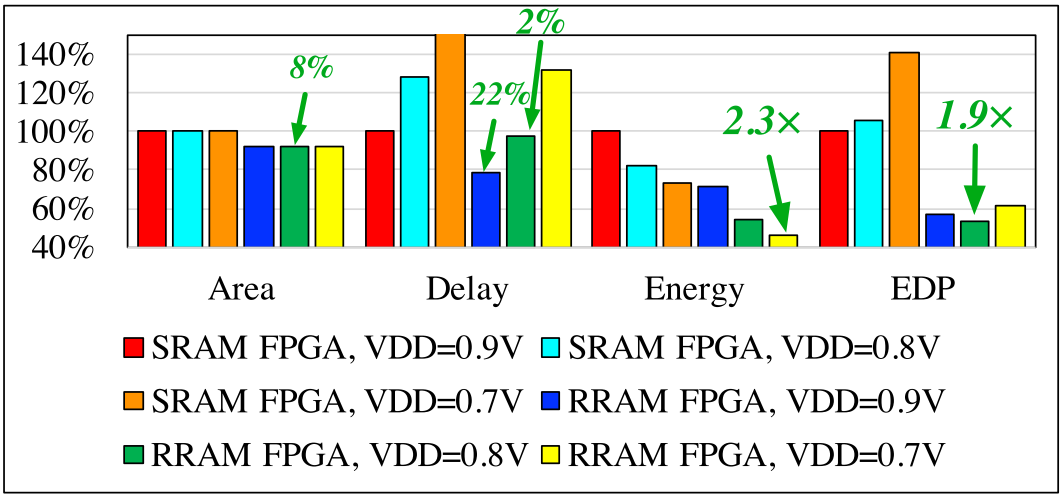


Figure 5.13 – Area, delay and energy comparison between SRAM-based and RRAM-based FPGAs operating at nominal and near-V_t regime.

5.2 Architecture-level Optimizations

Most SRAM-based FPGA architectures typically employ multiple levels of small crossbars, instead of large multiplexers, due to a strong limitation of SRAM-based multiplexer: Whatever multiplexer structure is employed, their area, delay and power increase linearly with the input size [4]. However, we saw in Chapter 3 that the delay of RRAM-based multiplexers is independent from the input size. Table 5.2 compares the delay of SRAM-based and 4T1R-based multiplexer in their architectural context, i.e., by considering realistic sizing and loads. In high fan-in and low fan-out condition, such as local routing, the 4T1R-based multiplexer can achieve 48% reduction in delay. In contrast, when fan-in is low and fan-out is high, e.g.,

Table 5.2 – Delay comparison between SRAM-based and RRAM-based routing multiplexers.

Multiplexer Location	Input Size	fan-out	SRAM-based MUX (ps)	4T1R-based MUX (ps)	Improvements
Local Routing	80	1	57.7	30.4	-48%
BLE output selector	2	70	38.8	42.2	+11%
Connection block	48	60	76.0	48.2	-36%
Switch block	4	124[1]	57.8	49.6	-14%

* Output buffers are considered and sized according to the fan-outs of routing multiplexers in architecture.

[1] The fanout includes the parasitics of long metal wires driven by SBs.

the BLE output selector, 4T1R-based multiplexer guarantees a similar performance level as an SRAM-based implementation. Therefore, considering such feature of one-level 4T1R-based multiplexers, the FPGA architectural design space can be extended beyond the limitations of SRAM-based multiplexer. Indeed, the properties of RRAM-based multiplexers allow the FPGA architect to size differently its routing multiplexers by: privileging one-level crossbars, made of large multiplexers, as much as possible. This paradigm shift in the interconnection topology also requires to rethink the optimal architectural parameters, which have been well determined for classical SRAM-based architectures. Hence, it is worthwhile to identify properly-sized RRAM-based FPGA architectures which can exploit the full potential of RRAM-based multiplexers, and determine the associated optimal architectural parameters.

To exploit the high-performance of 4T1R-based multiplexers, in this section, we propose three architectural optimizations:

1. The realization of a Unified Connection Block;
2. The increase of Switch Blocks capacity;
3. The decrease of the best length of routing wire;

For each architectural optimization, we study its impact on both SRAM-based and RRAM-based FPGAs.

This section will be organized as follows. Section 5.2.1 introduces the general experimental methodology in this part. Section 5.2.2, Section 5.2.3 and Section 5.2.4 present the three architectural optimizations and validate their impacts on both SRAM-based and RRAM-based FPGAs. Section 5.2.5 compares the optimized RRAM-based FPGA to its SRAM-based counterpart, considering both nominal working voltage and near-V_t regime.

5.2.1 Experimental Methodology

In this part, we will base our analysis using a commercial 40nm technology, whose nominal working voltage is $V_{DD} = 0.9V$. Area is estimated and expressed by the number of minimum width transistors, based on the area model in [125]. Delay results are extracted from electrical simulations by running HSPICE simulator[47]. Both datapath logic gates and programming structures are built with standard logic transistors ($W_{logic}/L_{logic} = 140nm/40nm$, $W_{PMOS,logic}/W_{NMOS,logic} = 2$). SRAM-based multiplexers are built with two-level structures and transmission gates for best area-delay product [2]. RRAM-based multiplexers are built with one-level structure and I/O transistors [133]. Electrical simulations use the Stanford RRAM model [130] with following parameters: $R_{LRS} = 2k\Omega$, $R_{HRS} = 27M\Omega$, $I_{set} = 500\mu A$, $V_{set} = V_{reset} = 1.1V$. The parasitic capacitance of a RRAM is considered to be $C_P = 13.2aF$. The considered RRAM parameters are sufficient to guarantee that the RRAM-based circuits are as power efficient as SRAM-based circuits [114]. To determine the size of CB and SB multiplexers, we set the channel width to $W = 320$, which is close to the practical number in commercial products [85, 82].

Since each architectural optimization involves different routing architecture parameters, such as $F_{c,in}$, F_s and L, for a fair comparison, we vary a single parameter in each comparison and find a reasonable value for each parameter. Once we find the best value of one parameter, we set it to this value and vary another. All the investigated tile-based FPGA architectures share the Stratix IV-like CLB architecture [88], which contains 10 BLEs, consisting of 6-input fracturable LUTs and FFs ($K = 6, N = 10$). We consider a uni-directional routing architecture and the CLB output connection flexibility, $F_{c,out}$, is fixed to 0.1. All the baseline architectures have 40 inputs for each CLB ($I = 40$). Because the local routing is removed in the proposed architecture, we provide 60 inputs for each CLB ($I = K \cdot N = 60$). We will focus on studying the effect of the different architectural modifications on both SRAM-based and RRAM-based FPGAs. Both SRAM-based and RRAM-based implementations of the proposed architecture are then investigated and their benefits are examined by comparing to the baseline SRAM-based and RRAM-based architectures, respectively. We believe that such methodology helps to identify where RRAM FPGAs can be improved beyond SRAM FPGAs. Then, we will discuss the benefits of a properly-optimized RRAM-based FPGA compared to the SRAM counterpart.

We use the VTR flow [44] to evaluate the area, delay, power and channel width of the investigated FPGA architectures. The twenty biggest MCNC [138] and VTR benchmarks [44] suites are logic optimized by ABC [123] and then packed, placed and routed by VPR7. We add a 30% slack to the minimum routable channel width W_{min}, in order to simulate a low-stress routing [4]. For a fair comparison, the maximum routing iterations are set to 50 for the classical architecture, while 100 routing iterations are used for the proposed architectures. Indeed, our proposed architecture requires more routing efforts because local routing is removed and more nets have to be routed by the global router.

5.2.2 Unified Connection Block

In SRAM-based FPGA architectures, a routing track has to pass through a CB multiplexer and a local routing multiplexer before reaching a LUT input, as shown in Fig. 5.14. Such routing architecture efficiently reduces the number of CB multiplexer to be used. Indeed, the number of the inputs of a CLB, typically $I = K(N+1)/2$, is smaller than the total number of LUTs inputs, $K \cdot N$, where K is the input size of a LUT and N is the number of BLEs in a CLB. However, it requires tapered buffers at the outputs of CB multiplexers, in order to drive the high fan-outs. Take the example in Fig. 5.14, each CB multiplexer has to drive $K \cdot N$ local routing multiplexers. The use of large tapered buffers potentially increase the delay from a routing track to a LUT input. This situation is extremely inefficient for RRAM-based FPGAs since the delay of a tapered buffer may be far larger than the delay of the RRAM-based multiplexer itself.

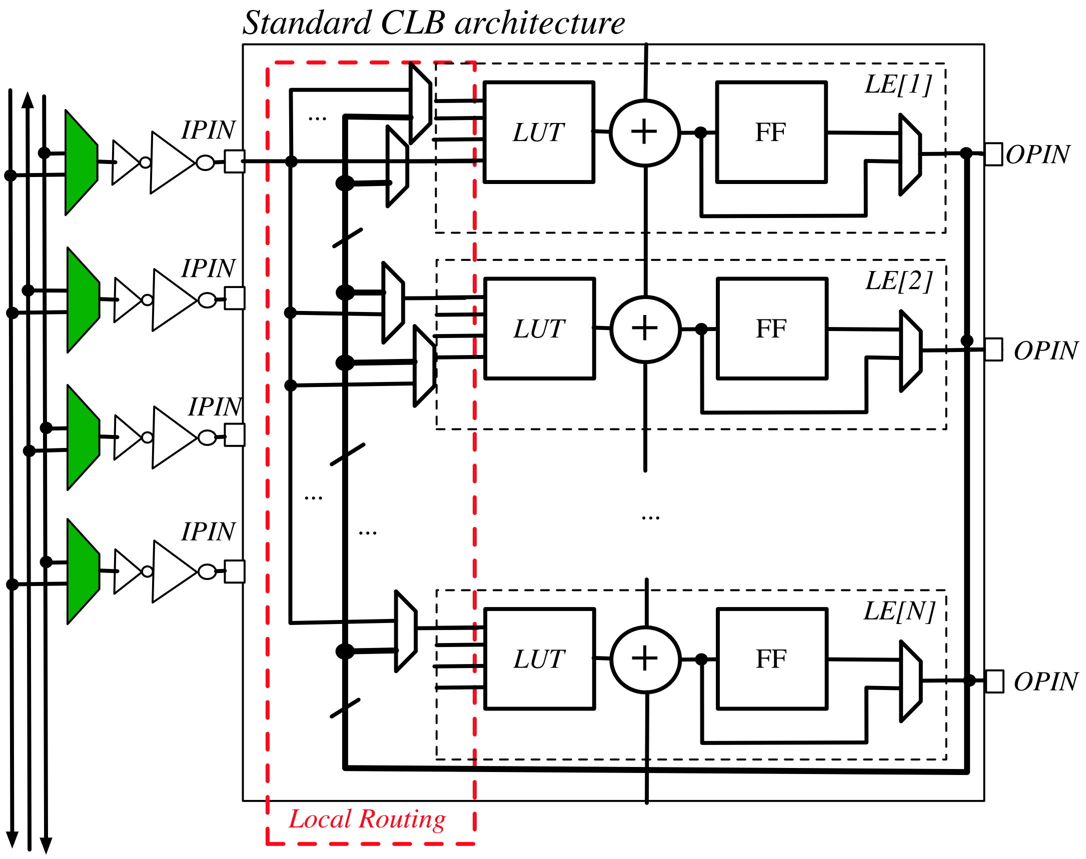


Figure 5.14 – Classical interconnection from routing tracks to LUT inputs.

Therefore, we propose that RRAM-based FPGA should use a one-level RRAM-based crossbar to provide interconnections between routing tracks and LUT inputs, as illustrated in Fig. 5.15. Note that feedback connections are also resolved by the unified Connection Block. The proposed routing architecture is well suited to RRAM-based multiplexers for three reasons: (a) Each CB multiplexers now has a unique fan-out, and tapered buffers can be avoided; (b) Only one large multiplexer interconnects between a routing track to a LUT input; Both routing delay and feedback delay can be significantly reduced when a RRAM-based multiplexer is used; (c) The number of inputs of a CLB is increased to $I = K \cdot N$, which can potentially lead

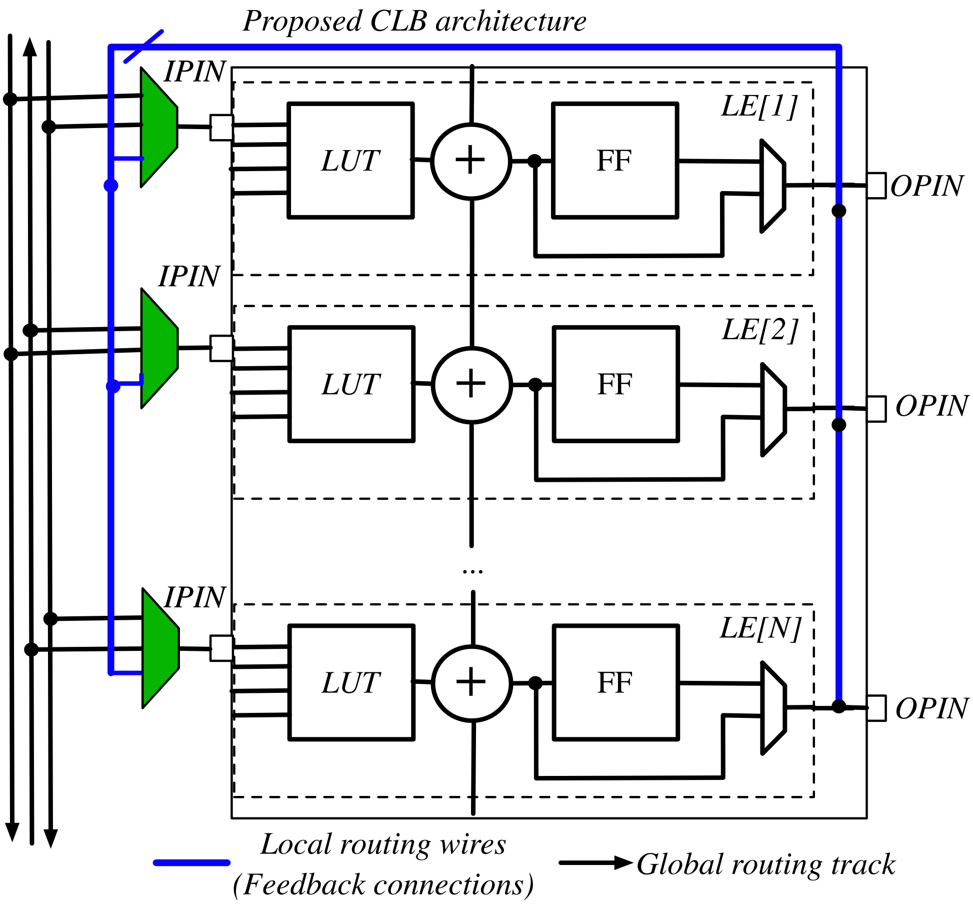


Figure 5.15 – Proposed interconnection from routing tracks to LUT inputs.

to a total area reduction even for SRAM-based FPGAs [142]; Since RRAM-based multiplexers require a smaller footprint, the area reduction could be more significant.

The proposed routing architecture requires to redefine the best fraction of routing tracks that can be reached by each CB multiplexer, $F_{c,in}$. Note that in the classical architecture ($F_{c,in} = 0.15$), all the nets mapped to the inputs of a CLB are different because the local routing can connect a net from a CLB input to multiple LUTs. The proposed architecture may have a net mapped to multiple CLB inputs due to the absence of local routing. Therefore, we need to increase $F_{c,in}$ to allow more CLB inputs to be reached by a single routing track, to compensate the potential loss in routability. In an FPGA tile, all the LUT inputs are connected to the right and bottom sides of a CLB. Each LUT has $K/2$ input connected to the right/bottom side of a CLB. To ensure that different LUT inputs can be connected from a common routing track, $F_{c,in}$ should be at least $2/K$. Fig. 5.16 depicts such an example when $K = 6$. Input $in0$ of $LUT0$ and input $in0$ of $LUT1$ can be reached by the same track $Track0$. Note that there is no need to allow two inputs of the same LUT to share a routing track. The case where two inputs of a LUT share the same net can never happen because the inputs of a LUT are naturally logic

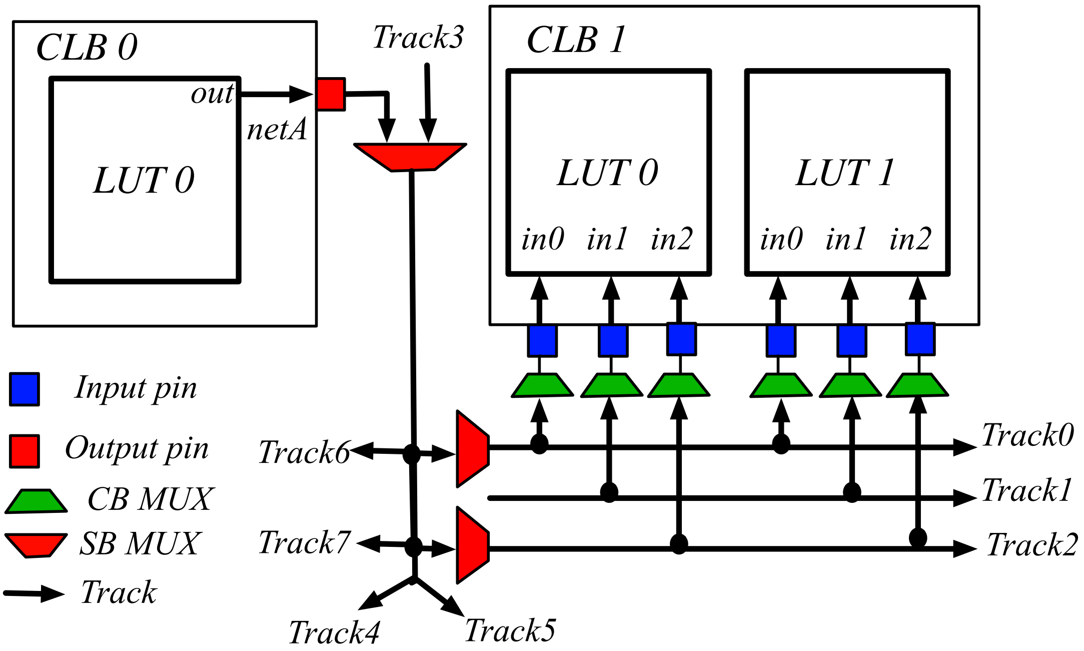
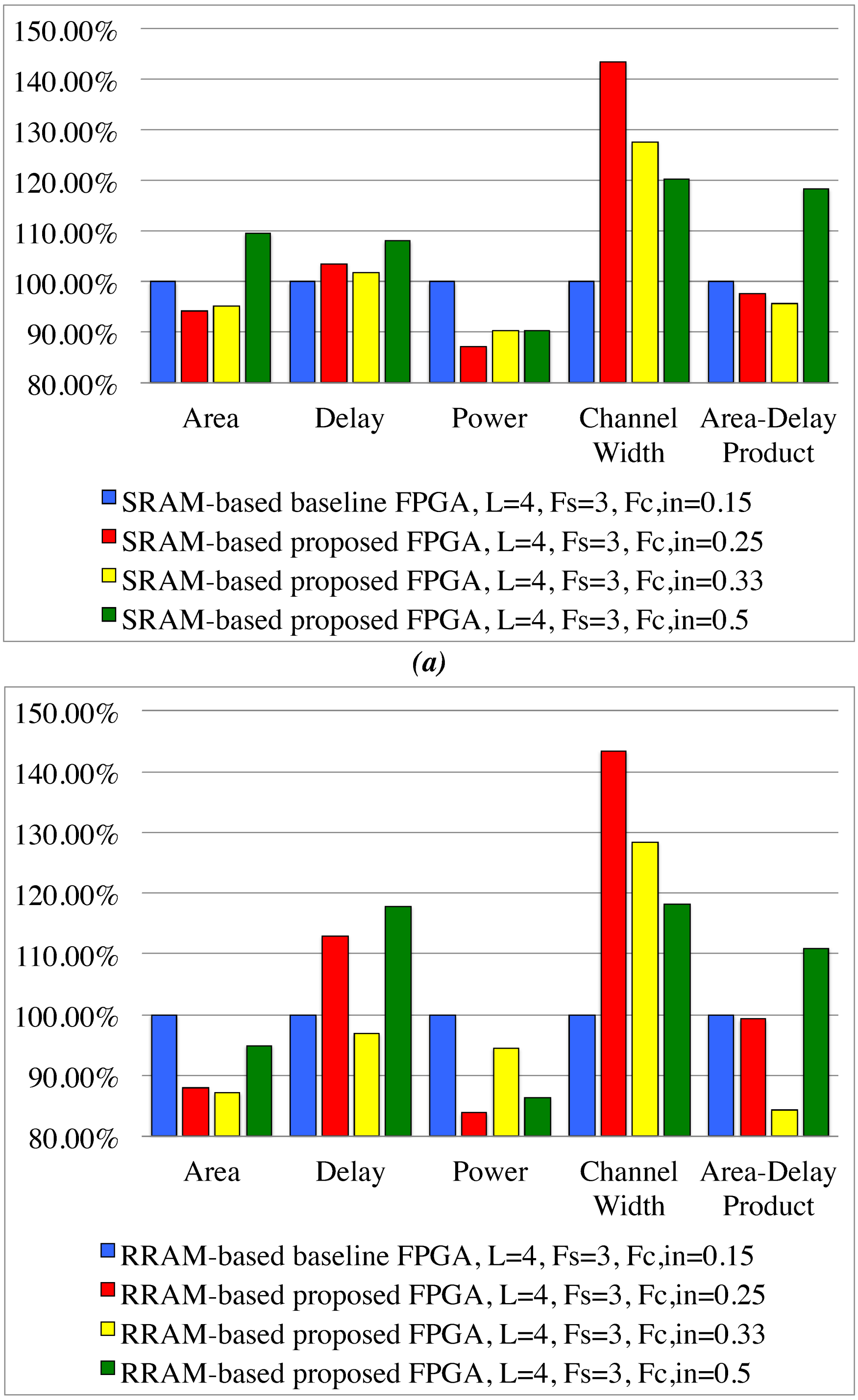


Figure 5.16 – An illustrative example of the proposed routing architecture($K = 6$) with $F_{c,in} = 0.33$ and $F_s = 6$.

equivalent. By considering architecture parameters $K = 6$, the proposed architecture requires $F_{c,in}$ to be at least 0.33, in order to ensure routability. In this part, we sweep $F_{c,in}$ to examine the best $F_{c,in}$ for the proposed architecture.

Fig. 5.17(a) and (b) show normalized area, delay, power and channel width of SRAM-based and RRAM-based proposed architectures with $F_{c,in} = \{0.15, 0.25, 0.33, 0.5\}$, when compared to baseline architectures respectively. The SRAM-based proposed architecture with $F_{c,in} = 0.33$ produces a slightly better area-delay product (-4%) than the classical architecture, but performs worse (+2%) in delay. In contrast, the RRAM-based proposed architecture with $F_{c,in} = 0.33$ reduces delay by 3% and area-delay product by 15%, when compared to the classical architecture. In either SRAM-based or RRAM-based FPGAs, the proposed architecture with $F_{c,in} = 0.33$ produces the best area-delay product. Note that we see a 5% area reduction in both SRAM-based and RRAM-based proposed architectures when $F_{c,in} = 0.33$, which is close to the conclusion of literature [142]. The proposed architecture with varying $F_{c,in}$ reduces power by 10%-13% for SRAM-based and RRAM-based FPGAs. In the classical architecture, there are two-stages of multiplexers (local routing and classical connection blocks) that lead to four levels of transmission gates between the routing tracks and the LUTs. However, in the proposed unified connection block, there is only one-stage of multiplexers (two-levels of transmission gates) between the routing tracks and the LUTs, contributing to power efficiency. Besides, the unified connection blocks eliminates the need for intermediate buffers between the local routing and the connection block, which further reduce the power. Channel width overheads



(a)

(b)

Figure 5.17 – Normalized average area, delay, power and channel width of baseline and proposed architecture by sweeping $F_{c,in}$: (a) SRAM-based architectures; (b) RRAM-based architectures.

are observed in both SRAM-based and RRAM-based proposed architectures, because their routability is lower than their baselines due to the absence of local routing. However, these overheads can be potentially eliminated because the routability can be significantly improved when we increase F_s and decrease L. In terms of the best overall performance, we consider $F_{c,in} = 0.33$ for the proposed FPGA architectures in the rest of this chapter.

Fig. 5.18 compares the tile area of a classical FPGA architecture ($I = 40, F_{c,in} = 0.15$) and the proposed RRAM FPGA architecture ($I = W \cdot F_{c,in}, F_{c,in} = 0.33$) for a sweeping channel width W from 100 to 350. Note that the input size of local routing multiplexers in traditional SRAM FPGAs is fixed for every W, while that of proposed RRAM FPGAs is directly related to W. When a small W, e.g. = 100, is used, the size of the local routing multiplexers in the proposed RRAM FPGAs is smaller than for a classical FPGA architecture. Therefore, when $W < 300$, the proposed RRAM FPGA architecture benefits up to 36% area reduction as compared to classical FPGA architecture. When $W > 300$, the input size of multiplexers in the proposed RRAM FPGAs becomes larger, leading to a 9% area overhead when $W = 350$. The considered $W = 320$ in this part promises that the proposed RRAM FPGAs is as area efficient as classical SRAM FPGAs.

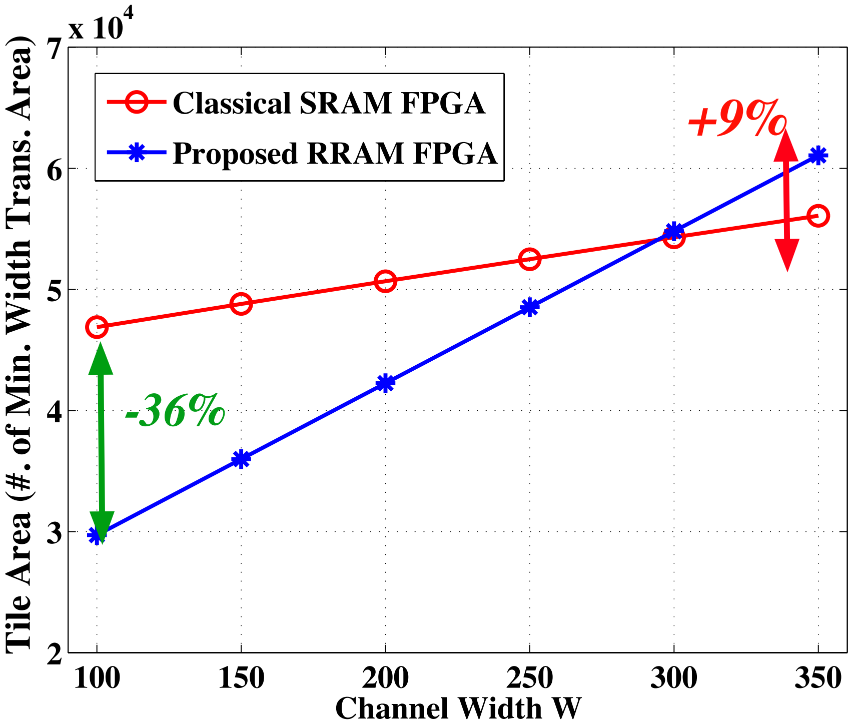


Figure 5.18 – Tile area comparison between a traditional FPGA architecture and the proposed RRAM FPGA architecture for different channel width W.

5.2.3 Increase Capacity of SB MUXes

Since RRAM-based multiplexer is more delay-efficient than SRAM-based multiplexer, the connection flexibility parameter of *Switch Block* (SB) F_s can be increased. Classical FPGA architectures typically set $F_s = 3$, where each routing track on one side of a SB can reach three other routing tracks on different sides of a SB. In SRAM-based FPGAs, $F_s = 3$ promises the best area-delay product [98]. Indeed, a larger F_s can improve the routability but it may produce area and delay overhead coming from the larger SB multiplexers to be used. However, considering RRAM-based routing architecture, the delay overhead is no longer a concern thanks to the advantage of RRAM multiplexers. Therefore, a larger F_s, i.e. = 6, can considered, where a routing track can drive six different tracks, as shown in Fig. 5.16 with $Track3$. Note that a large F_s significantly improves the routability of the proposed routing architecture. Take the example of Fig. 5.16 where $netA$ is routed through $Track3$. If $F_s = 3$, $Track3$ can only drive $Track0$, $Track4$ and $Track6$. If $Track0$ is not available, the output of $LUT0$ has to seek for another routing track by increasing the channel width. If $F_s = 6$, $Track3$ can reach both $Track0$ and $Track2$. When $Track0$ is occupied by another net, $Track3$ can easily use $Track2$ to route $netA$.

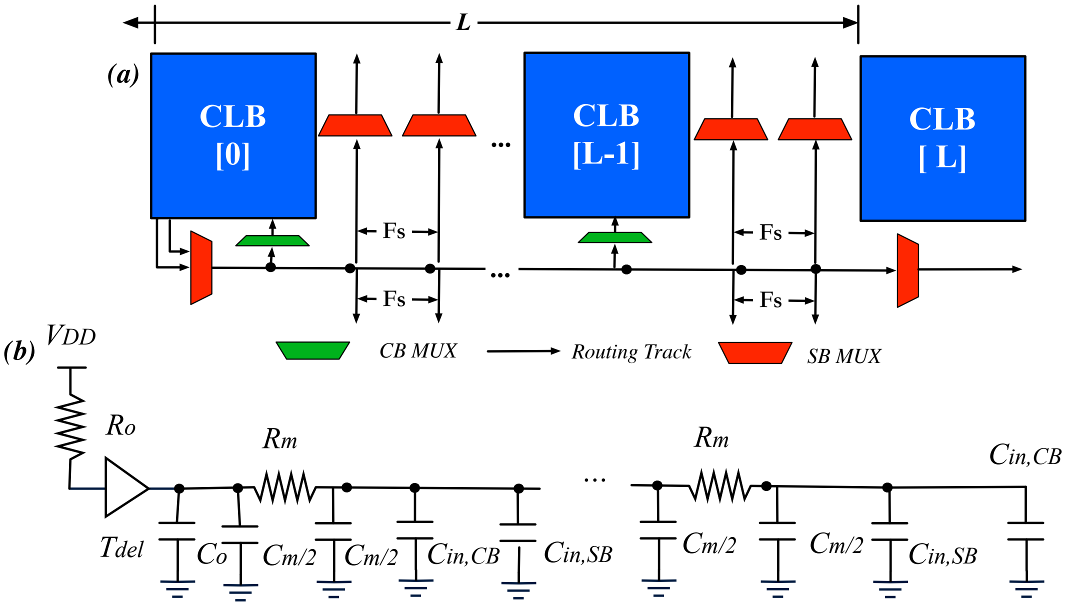
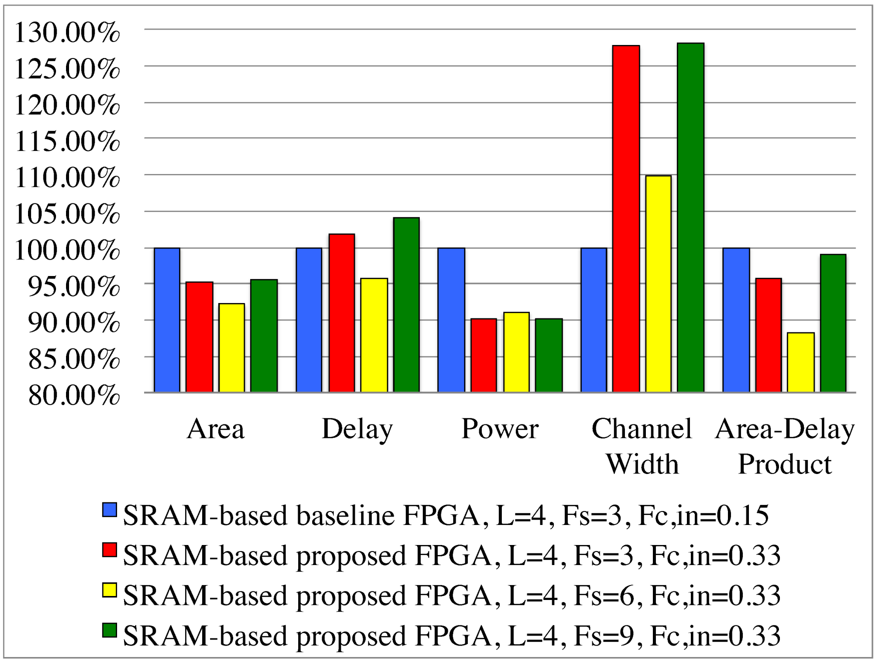
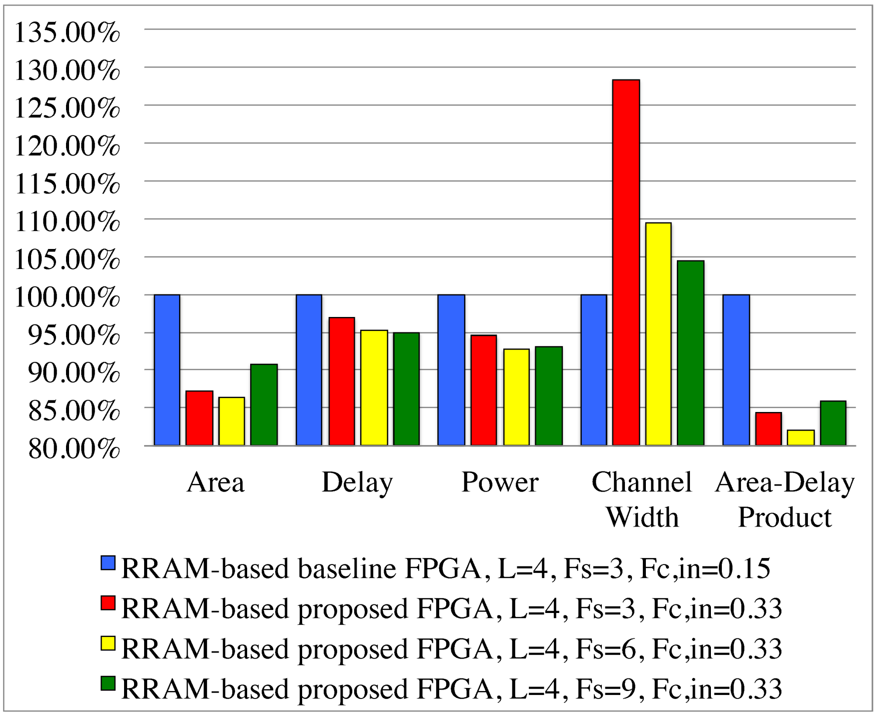


Figure 5.19 – (a) Driver multiplexer and fan-outs of a Length-L wire; (b) Equivalent RC model of a Length-L wire.

We sweep F_s to determine its best value for the proposed architecture. Fig. 5.20(a) and (b) show normalized average area, delay, power and channel width of SRAM-based and RRAM-based proposed architectures with $F_s = \{3, 6, 9\}$, when compared to the baseline architectures, respectively. The proposed RRAM-based architectures can benefit larger delay reduction (-7%) than SRAM-based (-4%), because RRAM-based multiplexers are more delay efficient for the



(a)



(b)

Figure 5.20 – Normalized average area, delay, power and channel width of baseline and proposed architectures by sweeping F_S: (a) SRAM-based architectures; (b) RRAM-based architectures.

unified connection block. However, $F_s > 3$ introduces larger SB multiplexers, which potentially increases the area of both SRAM-based and RRAM-based proposed architectures. On the other hand, larger SB multiplexers improve the flexibility of the routing architecture and reduce the number of necessary SB multiplexers, as explained in Fig. 5.16. In the end, the proposed architecture can maintain the same power efficiency as baseline SRAM one. Therefore, $F_s = 6$ produces the best area-delay-power product for both SRAM-based and RRAM-based proposed architectures. Note that, even when $F_s = 9$, RRAM-based proposed architecture leads to a 8% delay reduction thanks to its RRAM-based multiplexer, while, the SRAM-based proposed architecture has a 5% delay overhead. As a large F_s boosts the routability, a 20% channel width reduction is achieved in both SRAM-based and RRAM-based proposed architectures, as compared to those with $F_s = 3$. In terms of the best overall performance, we consider $F_s = 6$ for the proposed FPGA architectures in the rest of this part.

5.2.4 Smaller Best Length Wire < 4

In FPGA architectures, a length-L wire is a wire that spans across L CLBs [4]. As illustrated in Fig. 5.19(a), a length-L wire is driven by an output of $CLB[0]$ and ends at $CLB[L-1]$. All the CLBs and SBs along the length-L wire can be directly routed from the driving output of $CLB[0]$. When only one type of wires is allowed to be used in an FPGA, the type of length-L wires that produces best area-delay product is called *best single wire length*. Commercial FPGAs typically provide different types of wires, i.e. length-1 for short connections and length-8 for long connections. However, best single wire length is useful in deciding which type of wires should be predominant within the architecture.

Length-4 wires are the best choice for classical SRAM-based FPGA architectures ($F_{c,in} = 0.15, F_s = 3$) [4]. V. Betz *et al.* show that a length-4 wire is faster than shorter wires in terms of delay per logic block ($= T_{delay,wire}/Length$). In other words, for a routing path spanning X CLBs, length-4 wires promise the best average delay. Indeed, when there is a routing path with $X < 4$, shorter wires such as length-1 or length-2 will give better delay. However, for a routing path with $X \geq 4$, multiple cascaded length-4 wires are faster than not only any length-X ($X > 4$) wire but also multiple cascaded length-1 or length-2 wires. Therefore, on average, length-4 wires provide the best trade-off between short and long connections.

In SRAM-based FPGAs, why long length wires, such as length-4 wires, are preferred is established on the fact that the delay of a SB multiplexer is larger than a long metal wire across a logic block. However, RRAM-based multiplexers are more delay efficient and can be even faster than a long metal wire. Therefore, as the cost function between a SB multiplexer and a long metal wire has been twisted, the best single wire length L should be revisited. Fig. 5.19(a) illustrates the different elements composing a length-L wire, while Fig. 5.19(b) shows the extracted RC model. We use Elmore delay [104] to estimate the delay per logic block of a

Length-L wire:

$$\begin{aligned}
T_{delay,wire}/L &= \frac{1}{L}\sum_{i=0}^{L-1} R_i \sum_{j=i}^{L-1} C_j \\
&= L\cdot\frac{R_m C_m}{2} + \frac{1}{L}\cdot(T_{del} + R_o C_o - 2R_m C_{SB} - 2R_m C_{CB}) \\
&+ R_m(C_{SB} + C_{CB} - C_m) + R_o(C_m + C_{SB} + C_{CB})
\end{aligned} \tag{5.4}$$

where R_m and C_m are the resistance and capacitance of a metal wire spanning a logic block, respectively, T_{del} represents the intrinsic delay of a SB multiplexer, R_o and C_o denote the equivalent resistance and capacitance of the tapered buffer that drives the metal wire, respectively, C_{SB} and C_{CB} are the equivalent input capacitance of each SB and CB, respectively. According to (5.4), there exists a $L_{optimal}$ which guarantees the minimum $T_{delay,wire}/L$:

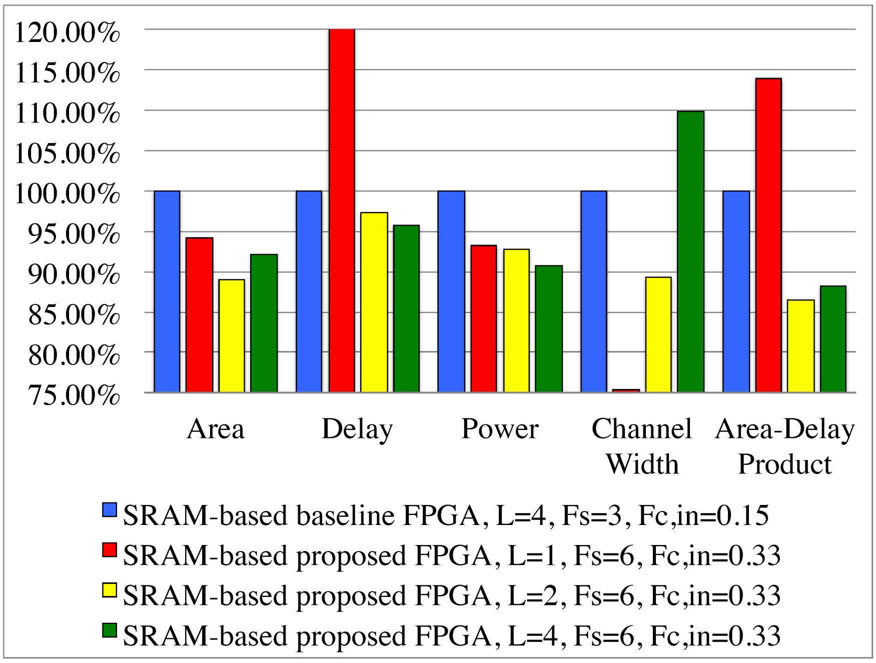
$$L_{optimal} = \frac{(T_{del} + R_o C_o - 2R_m C_{SB} - 2R_m C_{CB})}{2R_m C_m} \tag{5.5}$$

Note that C_{SB} and C_{CB} are related to F_s and $F_{c,in}$ respectively:

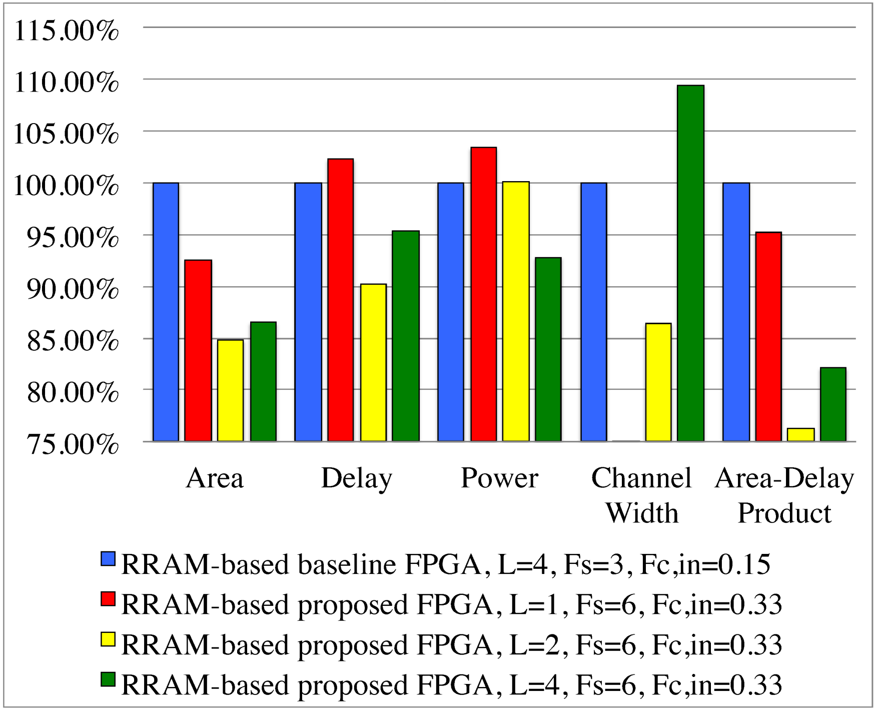
$$\begin{aligned}
C_{SB} &= F_s \cdot C_{in} \\
C_{CB} &= W \cdot F_{c,in} \cdot C_{in}
\end{aligned} \tag{5.6}$$

In the proposed RRAM-based routing architecture, where both F_s and $F_{c,in}$ increased and T_{del} decreased thanks to RRAM-based multiplexer, $L_{optimal}$ will definitely decrease. In addition, the tile area of the proposed architecture may be slightly larger than the classical architecture because of the F_s and $F_{c,in}$ increases, leading to an increased R_m and C_m. This would further decrease the $L_{optimal}$. Therefore, the best single wire length of the proposed routing architecture will be smaller than 4. When a smaller L (< 4) is used, previous work [4] show that the routability is improved significantly. Therefore, the proposed RRAM-based routing architecture can achieve routability improvement without delay overhead.

We sweep L to determine its best value for the proposed architecture. Fig. 5.21(a) and (b) show normalized average area, delay, power and channel width of SRAM-based and RRAM-based proposed architectures with $L = \{1, 2, 4\}$, when compared to the baseline architectures, respectively. In SRAM-based architectures, whatever F_s is, length-4 wires achieve the best delays and area-delay-power products. However, the proposed RRAM-based architecture with length-2 wires promises the best delay (-11%) and also the best area-delay-power product (-24%), thanks to its better routability and lower routing congestion. As L is reduced from 4 to 2, we see a 26% channel width reduction because short wires are more flexible. Conversely, length-1 wires have the smallest channel width but more SB multiplexers have to be used in long routing paths. Therefore, we see significant area and power overhead. Length-4 wires guarantee the best power results since less multiplexers are required in a SB compared to the case where length-2 and length-1 wires are used. In terms of the best overall performance, $L = 2$ is the best single wire length for the proposed FPGA architecture.



(a)



(b)

Figure 5.21 – Normalized average area, delay, power and channel width of baseline and proposed architectures by sweeping L: (a) SRAM-based architectures; (b) RRAM-based architectures.

5.2.5 RRAM-based FPGAs vs. SRAM-based FPGAs

In Section 5.2.2, Section 5.2.3 and Section 5.2.4, we have determined that $F_{c,in} = 0.33, F_s = 6$ and $L = 2$ produce the best performances for the proposed FPGA architecture. In this section, we make a general comparison between SRAM-based and RRAM-based FPGAs architectures. Fig. 5.22 shows the area, delay, power and channel width of three FPGA architectures: (1) SRAM-based FPGA with classical architecture; (2) RRAM-based FPGA with classical architecture; (3) RRAM-based FPGA with architectural optimizations. When implemented with classical architecture, RRAM-based FPGAs improve the delay by 32% and the area by 15%, as compared to SRAM-based FPGAs, thanks to the delay efficiency of the RRAM-based routing elements. By properly optimizing the architecture, RRAM-based FPGAs can further reduce the area by 15%, the delay by 10% and the channel width by 13%, leading to a total improvement of 38% in delay and 43% in area compared to an SRAM-based FPGA architecture. In terms of *Area-Delay Product* (ADP) and *Delay-Power Product* (PDP), the proposed RRAM-based FPGA architecture brings a reduction of 57% and 38% respectively.

As explained in Chapter 3, the resistance of RRAMs is only impacted by programming voltage and therefore a near-V_t working voltage leads to less performance degradation for RRAM-based circuits, when compared to pure CMOS implementations. Such outstanding feature strongly motivates us to evaluate the potential of the proposed RRAM-based FPGA architecture in the near-V_t regime. In this section, we consider the SRAM-based FPGA with classical architecture operating at nominal working voltage ($V_{DD} = 0.9$) as the baseline. We investigate the area, delay and power of the RRAM-based FPGAs with architectural optimizations operating at both nominal ($V_{DD} = 0.9$) and near-V_t ($V_{DD} = 0.7$ and $V_{DD} = 0.8$) working voltages. As shown in Fig. 5.23, when operated in the near-V_t regime, the proposed RRAM-based FPGA at $V_{DD} = 0.7$ can achieve 42% and 5× improvement on *Area-Delay Product* and *Power-Delay Product* respectively, as compared to a classical SRAM-based FPGA running at a nominal voltage. Note that such significant power reduction is achieved with zero delay overhead and such feature can not be achievable by any SRAM-based FPGA.

5.3 Summary

This chapter combines the efforts from 4T1R-based multiplexers (introduced in Chapter 3) and FPGA-SPICE (introduced in Chapter 4), in studying RRAM-based FPGA architectures. We first presented a generic RRAM-based FPGA architecture exploiting the 4T1R-based multiplexers and BL/WL sharing strategy, whose functionality has been verified by FPGA-SPICE. With layout-level implementation and accurate electrical simulator, we analyze the area breakdown and power characteristics of the proposed RRAM-based FPGA architecture and compare to its SRAM-based counterpart. Thanks to the 4T1R-based multiplexers, the propose RRAM-based FPGA can be as area efficient as SRAM-based FPGA, and meanwhile achieve non-volatility. Electrical simulations show that to guarantee power efficiency, R_{HRS} of RRAMs does not need be as large as the *off*-resistance of a transistor, but should be at least $20M\Omega$. To further leverage

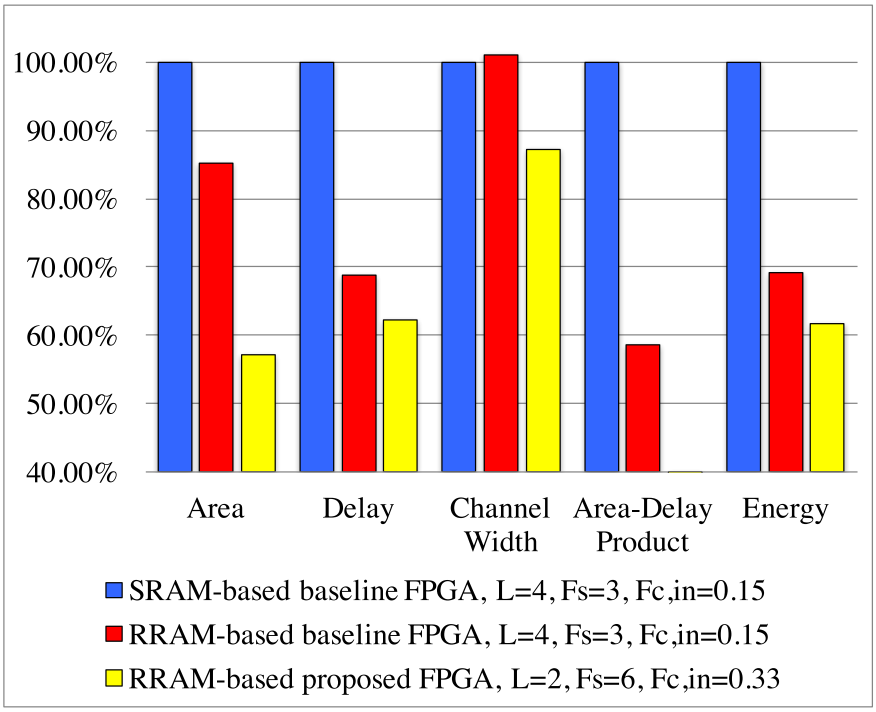


Figure 5.22 – Normalized average area, delay, energy and channel width of baseline and proposed architectures: (a) baseline SRAM-based architectures; (b) baseline RRAM-based architectures; (c) proposed RRAM-based architectures

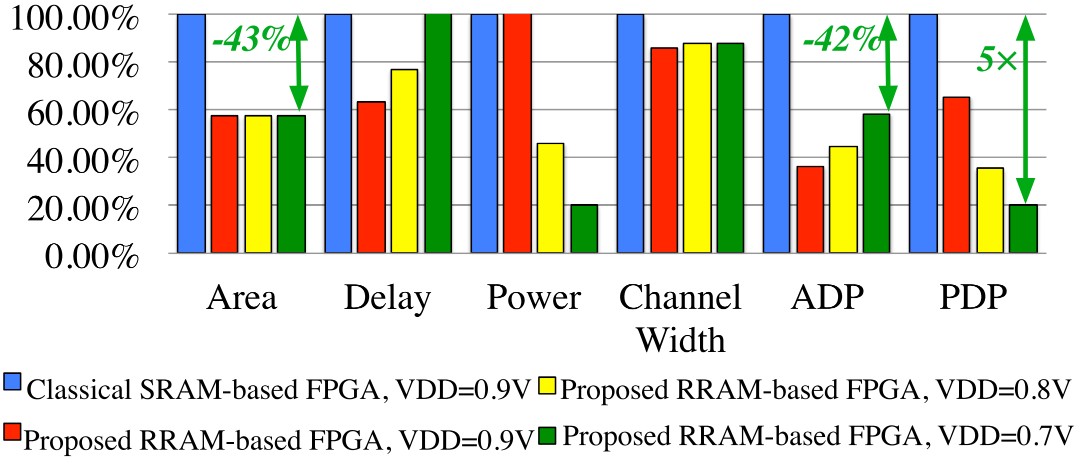


Figure 5.23 – Normalized average area, delay, power, channel width, ADP and PDP of classical SRAM-based and proposed RRAM-based architectures.

the potential of 4T1R-based multiplexers, we propose three architecture optimizations: (a) The traditional CB and local routing are replaced with a unified CB, leading to ultra-fast interconnection from routing tracks to LUT inputs; (b) The CB connectivity parameter $F_{c,in}$ should be at least 0.33 to ensure routability, while the SB connectivity parameter F_s can be increased to achieve routability improvements without delay overhead; (c) The best single wire length L is reduced, leading to better routability. We study the best values of $F_{c,in}$, F_s and L in terms of area, delay, power and channel width. Experimental results show that a RRAM-based FPGA properly optimized should employ ($F_{c,in} = 0.33$, $F_s = 6$ and $L = 2$) to achieve optimal performances. Compared to best SRAM-based FPGAs, a optimized RRAM-based FPGA architecture brings a reduction of 57% on *Area-Delay Product* (ADP) and 38% on *Delay-Power Product* (PDP) respectively. In particular, when operating at near-V_t regime, RRAM-based FPGAs demonstrate a 5× improvement on the power with zero delay overhead as compared to optimized SRAM-based FPGA operating at nominal working voltage.

6 Conclusion and Future Work

Before this thesis, merits of RRAM-based FPGAs, i.e., area, delay and power, were predicted without solid circuit-level studies nor specialized CAD tools, which caused architecture-level conclusions to be less meaningful. In this thesis, we have provided a systematic study on RRAM-based FPGAs by considering realistic device modelling, circuit designs under physical design considerations and accurate architecture-level simulations. The major principle of our works is to leverage the potential of RRAMs in FPGA architectures by integrating RRAMs and programming structures into the datapaths, replacing the classical SRAM-based routing elements. In order to achieve the research goal, our contributions involve three related research fields: circuit designs (Chapter 3), CAD tool (Chapter 4) and architecture-level optimizations (Chapter 5). From a circuit design perspective, we investigated the fundamental of RRAM-based programming structure, proposed a high-current-density 4T1R programming structures and 4T1R-based multiplexer designs. Compared to best CMOS implementations, the proposed RRAM-based circuits significantly reduce the area, delay and power. From a CAD perspective, we propose a simulation-based architecture exploration tool suite for FPGAs, which is called FPGA-SPICE. Compared to the existing VTR tool suite, FPGA-SPICE enables more accurate and realistic area and power analysis for both SRAM-based and RRAM-based FPGAs, From an architecture perspective, we present a generic RRAM-based FPGA architecture, quantified the minimum requirements for R_{HRS} of RRAM devices and proposed architecture-level optimizations. Accurate experimental results show that the proposed RRAM-based FPGAs improve *Area-Delay Product* (ADP) by 57% and *Power-Delay Product* (PDP) by 38% when compared to well-optimized SRAM-based FPGAs.

The rest of this chapter is divided into two parts. Section 6.1 highlights our contributions in each research fields. Section 6.2 envisages the future work.

6.1 Summary of Contributions

Table 6.1 summarizes our contributions in three research fields: circuit designs, CAD tool and architecture-level optimizations.

Table 6.1 – Summary of Contributions in Differnt Research Fields.

Research field	Contributions
Circuit designs	• Analysis of 2T1R programming structure.
	• Proposition of 4T1R programming structure.
	• Proposition of boosting methodologies for improving driving current density of programming structures.
	• Proposition of one-level, two-level and tree-like 4T1R-based multiplexer designs with physical design details.
	• Proposition of programming transistor sizing technique.
	• Proposition of optimal physical location of RRAMs.
	• Investigation of the excellence on delay and power of RRAM-based circuits at near-V_t regime.
	• Investigation of the robustness of 4T1R-based multiplexers to process variations of RRAMs.
CAD	• Proposition of FPGA-SPICE enables automatic generation of SPICE and synthesizable Verilog netlists for full FPGA fabric.
	• Extension of FPGA architecture description language to support modelling transistor-level circuit designs, the physical structure of I/O circuits and different types of configuration circuits.
	• Proposition of netlist splitting strategies to better trade-off between simulation runtime and accuracy.
	• Study on the accuracy of analytical power model VersaPower, with respect to simulation results.
	• Study on the area characteristics of SRAM-based FPGAs with different configuration circuits.
FPGA architecture	• Proposition of novel RRAM-based FPGA architecture with efficient BL and WL sharing strategy.
	• Determining the lower bound of R_{HRS} to be 20Ω for a power efficient RRAM-based FPGA.
	• Study on the area characteristics of SRAM-based and RRAM-based FPGAs.
	• Proposition of three architecture-level optimizations for RRAM-based FPGAs: (1) unified connection blocks; (2) increase capacity of SB multiplexers; (3) smaller best length wire.
	• Investigation of the performance and power efficiency of near-V_t RRAM-based FPGA.

In addition, the contributions of this thesis include the novel and general approaches that we developed to study RRAM-based circuits and FPGA architectures:

1. Previous works typically bound their circuit designs and FPGA architectures tightly to a specific RRAM technology. Differently, this thesis selects another angle: we target generic RRAM technologies and quantify the minimum requirements on the RRAM devices, such as R_{LRS} and R_{HRS}, which can guarantee good circuit-level and architecture-level

performance. In other words, we determine the specifications for RRAM devices which can guarantee efficient circuit designs and FPGA architectures.

2. Previous works typically ignored physical design details of RRAM-based circuits, such as the parasitics and physical location of RRAMs. However, this thesis considers both resistive and capacitive characteristics of RRAMs and also parasitics of programming transistors when evaluating RRAM-based circuit designs and FPGAs. In particular, we propose two general optimizing techniques for RRAM-based circuits: programming transistor sizing (See Section 3.7.3) and optimal physical location of RRAMs (See Section 3.7.2), derived from RC modeling and Elmore delay model. Both optimizing techniques have demonstrated significant performance improvement on RRAM-based circuits.

3. Previous works mainly depended on analytical models when evaluating FPGA architectures, strongly limiting the accuracy of the analysis and probably leading to misleading conclusions especially for FPGAs based on emerging technology, e.g., RRAMs. In this thesis, we develop FPGA-SPICE and used electrical simulations and semi-custom P&R flows to accurately capture the difference in area and power characteristics of both SRAM-based and RRAM-based FPGA architectures. Note that the methodology provides accurate results and can be generalized to studying more generic FPGA architectures which are not limited to SRAM and RRAM technologies.

Novel research approaches leads to more realistic conclusions than previous works:

1. Previous works [26, 113, 9, 27, 8, 110, 6, 114, 111] commonly insisted that a low R_{LRS} is the guarantee for the high-performance of RRAM-based circuits and FPGA architectures. In some extreme case [113, 9], researches employ a R_{LRS} as low as 100Ω. However, the experimental results in Section 3.8 overturn these stereotypes: in terms of best performance, a proper R_{LRS} should ranges from $2k\Omega$ to $6k\Omega$ in the considered 40nm technology, which is similar to the equivalent resistance of a transmission gate. Actually, a low R_{LRS} do not guarantee the best performance for RRAM-based circuits in most cases. To achieve a low R_{LRS}, large programming transistors have to be used, which introduce large parasitic capacitances. Consequently, the performance of RRAM circuits with a low R_{LRS} is even worse than a moderate R_{LRS}. The high-performance of RRAM-based circuit actually comes from the efficient circuit design topology rather than R_{LRS}. As explained in Section 3.6, the delay and power efficiency of 4T1R-based multiplexers is owing to the smaller parasitic capacitances in the datapath.

2. Previous works [26, 113, 9, 27, 8, 110, 6, 114, 111] commonly assumed that RRAMs should be programmed by transistors operating in saturation region, and n-type transistors are preferred because of their high saturation current. However, the analysis and experimental results in Section 3.4 overturn these stereotypes once again: a pair of p-type and n-type transistors performs best in the driving current density. Even in the most efficient programming structure, i.e., 4T1R, the programming transistors usually

operate in linear region. In practice, since saturation current may never be reached, programming efficiency should be boosted through increasing programming voltage V_{prog} and sizes of programming transistors W_{prog}.

3. Previous works [26, 113, 9, 27, 8, 110, 6, 114, 111] typically concluded a remarkable area reduction (15%-50%) for RRAM-based FPGAs. However, the layout-level results in Section 5.1.4 overturn these stereotypes: area saving of RRAM-based FPGAs is in general up to 15%, and the area of RRAM-based FPGAs can be slightly larger than SRAM-based FPGAs when channel width is small. In fact, programming transistors occupy similar transistor area as transmission gates (See Chapter 3). Indeed, the transistor area contributed by SRAMs in multiplexers can be saved. But, considering their contribution is below 30% in the total area, the overall area reduction is limited.

4. Previous works [26, 113, 9, 27, 8, 110, 6, 114, 111] usually focused on RRAM-based FPGAs operating at nominal working voltage. However, this thesis intensively investigates the opportunity of RRAM-based circuits and FPGAs operating at near-V_t regime. Experimental results in Section 3.8 and Section 5.2.5 reveal that near-V_t regime may be the golden working voltage for RRAM-based circuits and FPGAs, because of the outstanding energy efficiency. Since the resistance of RRAMs is only impacted by programming voltage, a near-V_t working voltage leads to less performance degradation for RRAM-based circuits and FPGAs, when compared to pure CMOS implementations. Hence, RRAM-based circuit and FPGAs operating at near-V_t working voltage can remain as performant as they are in nominal working voltage. Note that RRAM-based circuits and FPGAs can still benefit from significant power reduction as their CMOS counterparts do in near-V_t regime.

5. Previous works [26, 113, 9, 27, 8, 110, 6, 111] commonly assumed a large R_{RHS} of RRAMs in order to avoid serious leakage power overhead. In some extreme case [113, 9], researches employ a R_{HRS} as large as $1G\Omega$. However, the experimental results in Section 5.1.5 overturn these stereotypes: R_{HRS} can be as low as $20M\Omega$ without causing power overhead. The reduction on R_{HRS} is owing to the different operating mechanism of RRAM-based FPGAs: non-volatility allows them to be simply powered-off during long idle period, consuming zero leakage power. Therefore, leakage power of RRAM-based FPGAs only occurs during standard operation time, which is typically along with high dynamic power consumption. In addition, there are other factors alleviating the effect of R_{HRS} on the power consumption of RRAM-based FPGAs: the use of CMOS circuits (such as LUTs), smaller dynamic power consumption of RRAM-based multiplexers and the reduced usage of SRAMs in FPGA architecture. As a result, considering the context of FPGA architectures, R_{HRS} can indeed be smaller than the *off*-resistance of a transistor, without leading to any power overhead.

In short, the benefits of integrating RRAMs into FPGAs can be summarized as follows:

1. A smaller area footprint. The total area of a full FPGA fabric can be reduced up to 15%.

2. High performance at both nominal and near V_t working voltages. The performance of multiplexers can be improved by up to 3.7×. The performance of FPGA can be improved by up to 39%.

3. Low power achieved without performance loss beyond the limitation of SRAM-based FPGAs. The energy efficiency of multiplexers can be improved by up to 4.7×. The energy efficiency of FPGA can be improved by up to 5×.

4. Non-volatility. FPGAs can be normally powered off and instantly powered on without losing configurations.

6.2 Future Work

As this thesis contributes to three research fields: circuit designs, CAD and FPGA architectures, the future works can also be split into the three categories:

1. **Circuit-level**: Considering the generality and efficiency of the 4T1R programming structure, we can investigate their opportunities in other RRAM-based applications, such as neuromorphic computing. In addition, we can also extend the use of the 4T1R programming structure to other emerging non-volatile memory technologies, such as *Phase Change Memory*. In Section 5.1.4, the NV SRAMs have a significant impact of the area of RRAM-based FPGAs. To further improve area efficiency, more compact NV SRAM design well worth an investigation. In Section 3.9, we have investigated the impact of process variations of V_{set} and V_{reset} on the 4T1R-based multiplexers. Such study can be extended to more RRAM device parameters such as writing speed. In addition, more robust RRAM-based circuit designs can be proposed to resist the process variations.

2. **CAD**: FPGA-SPICE has been developed to provide accurate area and power analysis for full FPGA fabric. Note that the area and power results can also be used to evaluate the effectiveness of CAD algorithms, such as packing, placement and routing algorithms for FPGAs. In addition, area and power results can serve as baselines for the purpose of examining the accuracy of analytical area and power models. For instance, accurate leakage power models can be developed for 4T1R-based multiplexers and examined with FPGA-SPICE. We believe that as an open-source tool suite, FPGA-SPICE can motivate more creative works in this research field.

3. **FPGA architecture**: In Section 5.1.4, we saw that the current BL/WL sharing strategy leads to larger configuration circuits for RRAM-based FPGAs than their SRAM-based counterpart. Therefore, it is necessary to study more efficient BL/WL sharing strategy or even novel configuration circuits for RRAM-based FPGAs. The architecture-level optimizations proposed in this thesis still is confined to the principles of SRAM-based FPGA architectures. To further leverage the potential of RRAM-based circuits, we believe that future work on RRAM-based FPGA architecture should break the routing topology

of conventional FPGA architectures. For instance, the routing architecture can be fully re-designed to leverage the high-performance of RRAM-based multiplexers. In addition, the proposed LB architecture in Chapter 5 eliminates the complex routing efforts during packing stage, which is required for the local routing in a classical architecture. But the default packer in VPR still performs full routing efforts, leading to an increase in the overall routing (local and global) runtime by 2.4× on average. We believe that the runtime of EDA flow can be significantly reduced by developing a lighter packer.

A An appendix

A.1 Examples of FPGA-SPICE Architecture Modeling

The following XML description models a representative homogeneous SRAM-based FPGA architecture featured by $K = 6$, $N = 10$, $I = 40$, $L = 4$, $F_{c,in} = 0.15$ and $F_{c,in} = 0.1$. Note that all the SRAMs are configured by BL/WL decoders, as shown in Fig. 5.1.

```
<architecture>
 <models>
  <model name="io">
   <input_ports>
    <port name="outpad"/>
   </input_ports>
   <output_ports>
    <port name="inpad"/>
   </output_ports>
  </model>
 </models>
 <!- Physical descriptions begin ->
 <layout auto="1.0"/>
 <spice_settings>
```

```
  <parameters>
   <options sim_temp="25" post="on" captab="off" fast="on"/>
   <measure sim_num_clock_cycle="auto" accuracy="1e-13" accuracy_type="abs">
    <slew>
     <rise upper_thres_pct="0.95" lower_thres_pct="0.05"/>
     <fall upper_thres_pct="0.05" lower_thres_pct="0.95"/>
    </slew>
    <delay>
     <rise input_thres_pct="0.5" output_thres_pct="0.5"/>
     <fall input_thres_pct="0.5" output_thres_pct="0.5"/>
    </delay>
   </measure>
   <stimulate>
    <clock op_freq="auto" sim_slack="0.2" prog_freq="2.5e6">
     <rise slew_time="20e-12" slew_type="abs"/>
     <fall slew_time="20e-12" slew_type="abs"/>
    </clock>
    <input>
     <rise slew_time="100e-12" slew_type="abs"/>
     <fall slew_time="100e-12" slew_type="abs"/>
    </input>
   </stimulate>
  </parameters>
  <tech_lib lib_type="industry" transistor_type="TOP_TT" lib_path="commercial_40nm_tech.l"
nominal_vdd="0.9" io_vdd="2.5"/>
  <transistors pn_ratio="2" model_ref="M">
```

```
   <nmos model_name="nch" chan_length="40e-9" min_width="140e-9"/>

   <pmos model_name="pch" chan_length="40e-9" min_width="140e-9"/>

   <io_nmos model_name="nch_25" chan_length="270e-9" min_width="320e-9"/>

   <io_pmos model_name="pch_25" chan_length="270e-9" min_width="320e-9"/>

  </transistors>

  <module_spice_models>

   <spice_model type="inv_buf" name="INVTX1" prefix="INVTX1" is_default="1">

    <design_technology type="cmos" topology="inverter" size="1" tapered="off"/>

    <port type="input" prefix="in" size="1"/>

    <port type="output" prefix="out" size="1"/>

   </spice_model>

   <spice_model type="inv_buf" name="buf4" prefix="buf4" is_default="1">

    <design_technology type="cmos" topology="buffer" size="4" tapered="off"/>

    <port type="input" prefix="in" size="1"/>

    <port type="output" prefix="out" size="1"/>

   </spice_model>

   <spice_model type="inv_buf" name="tap_buf4" prefix="tap_buf4" is_default="1">

    <design_technology type="cmos" topology="buffer" size="1" tapered="on"
tap_buf_level="2" f_per_stage="4"/>

    <port type="input" prefix="in" size="1"/>

    <port type="output" prefix="out" size="1"/>

   </spice_model>

   <spice_model type="pass_gate" name="TGATE" prefix="TGATE" is_default="1">

    <design_technology type="cmos" topology="transmission_gate" nmos_size="1"
pmos_size="2"/>

    <input_buffer exist="off"/>

    <output_buffer exist="off"/>
    <port type="input" prefix="in" size="1"/>

    <port type="input" prefix="sel" size="1"/>

    <port type="input" prefix="selb" size="1"/>

    <port type="output" prefix="out" size="1"/>

   </spice_model>

   <spice_model type="chan_wire" name="chan_segment" prefix="track_seg" is_default="1">

    <design_technology type="cmos"/>

    <input_buffer exist="off"/>

    <output_buffer exist="off"/>

    <port type="input" prefix="in" size="1"/>

    <port type="output" prefix="out" size="1"/>

    <wire_param model_type="pie" res_val="0" cap_val="0" level="1"/>

   </spice_model>

   <spice_model type="wire" name="direct_interc" prefix="direct_interc" is_default="1">

    <design_technology type="cmos"/>

    <input_buffer exist="off"/>

    <output_buffer exist="off"/>

    <port type="input" prefix="in" size="1"/>

    <port type="output" prefix="out" size="1"/>

    <wire_param model_type="pie" res_val="0" cap_val="0" level="1"/>

   </spice_model>

   <spice_model type="mux" name="mux_2level" prefix="mux_2level" is_default="1"
dump_structural_verilog="true">

    <design_technology type="cmos" structure="multi-level" num_level="2"/>

    <input_buffer exist="on" spice_model_name="INVTX1"/>

    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="input" prefix="in" size="1"/>
    <port type="output" prefix="out" size="1"/>
    <port type="sram" prefix="sram" size="1"/>
   </spice_model>
   <spice_model type="mux" name="mux_1level" prefix="mux_1level" dump_structural_verilo
    <design_technology type="cmos" structure="one-level"/>
    <input_buffer exist="on" spice_model_name="INVTX1"/>
    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="input" prefix="in" size="1"/>
    <port type="output" prefix="out" size="1"/>
    <port type="sram" prefix="sram" size="1"/>
   </spice_model>
   <spice_model type="ff" name="static_dff" prefix="dff" spice_netlist="ff.sp"
verilog_netlist="ff.v">
    <design_technology type="cmos"/>
    <input_buffer exist="on" spice_model_name="INVTX1"/>
    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="input" prefix="D" size="1"/>
    <port type="input" prefix="Set" size="1" is_global"true" default_val="0"
is_set="true"/>
    <port type="input" prefix="Reset" size="1" is_global="true" default_val="0"
is_reset="true"/>
    <port type="output" prefix="Q" size="1"/>
    <port type="clock" prefix="clk" size="1" is_global="true" default_val="0"
```

```
/>
   </spice_model>
   <spice_model type="lut" name="lut6" prefix="lut6" dump_structural_verilog="true">
    <design_technology type="cmos"/>
    <input_buffer exist="on" spice_model_name="INVTX1"/>
    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <lut_input_buffer exist="on" spice_model_name="tap_buf4"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="input" prefix="in" size="6"/>
    <port type="output" prefix="out" size="1"/>
    <port type="sram" prefix="sram" size="64"/>
   </spice_model>
   <spice_model type="sram" name="sram6T_blwl" prefix="sram_blwl" spice_netlist="sram.sp"
verilog_netlist="sram.v">
    <design_technology type="cmos"/>
    <input_buffer exist="on" spice_model_name="INVTX1"/>
    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="input" prefix="in" size="1"/>
    <port type="output" prefix="out" size="2"/>
    <port type="bl" prefix="bl" size="1" default_val="0" inv_spice_model_name="INVTX1"/>
    <port type="blb" prefix="blb" size="1" default_val="1" inv_spice_model_name="INVTX1"/>
    <port type="wl" prefix="wl" size="1" default_val="0" inv_spice_model_name="INVTX1"/>
   </spice_model>
   <spice_model type="iopad" name="iopad" prefix="iopad" spice_netlist="io.sp"
verilog_netlist="io.v">
    <design_technology type="cmos"/>
    <input_buffer exist="on" spice_model_name="INVTX1"/>
    <output_buffer exist="on" spice_model_name="INVTX1"/>
    <pass_gate_logic spice_model_name="TGATE"/>
    <port type="inout" prefix="pad" size="1"/>
    <port type="sram" prefix="en" size="1" mode_select="true" spice_model_name="sram6T_
default_val="1"/>
    <port type="input" prefix="outpad" size="1"/>
    <port type="input" prefix="zin" size="1" is_global="true" default_val="0"
/>
    <port type="output" prefix="inpad" size="1"/>
   </spice_model>
  </module_spice_models>
 </spice_settings>
 <device>
  <sizing R_minW_nmos="8926" R_minW_pmos="16067" ipin_mux_trans_size="1.222260"/>
  <timing C_ipin_cblock="1.47e-15" T_ipin_cblock="7.247000e-11"/>
  <area grid_logic_tile_area="0"/>
  <sram area="6">
   <verilog organization="memory_bank" spice_model_name="sram6T_blwl"/>
   <spice organization="standalone" spice_model_name="sram6T" />
  </sram>
  <chan_width_distr>
   <io width="1.000000"/>
   <x distr="uniform" peak="1.000000"/>
   <y distr="uniform" peak="1.000000"/>
  </chan_width_distr>
  <switch_block type="wilton" fs="6"/>
 </device>
 <cblocks>
  <switch type="mux" name="cb_mux" R="0" Cin="1.47e-15" Cout="0" Tdel="7.247e-11"
mux_trans_size="2.630740" buf_size="4" spice_model_name="mux_1level" structure="multi-level"
num_level="1">
  </switch>
 </cblocks>
 <switchlist>
  <switch type="mux" name="0" R="551" Cin=".77e-15" Cout="4e-15" Tdel="58e-12"
mux_trans_size="2.630740" buf_size="27.645901" spice_model_name="mux_1level"
structure="one-level" num_level="1">
  </switch>
 </switchlist>
 <segmentlist>
  <segment freq="1" length="4" type="unidir" Rmetal="101" Cmetal="22.5e-15"
spice_model_name="chan_segment">
   <mux name="0"/>
   <sb type="pattern">1 1 1 1 1</sb>
   <cb type="pattern">1 1 1 1</cb>
  </segment>
 </segmentlist>
 <complexblocklist>
  <!- Define I/O pads begin ->
  <pb_type name="io" capacity="8" area="0" idle_mode_name="inpad" physical_mode_name="io_phy'
   <input name="outpad" num_pins="1"/>
   <output name="inpad" num_pins="1"/>
   <!- physical design description ->
   <mode name="io_phy" available_in_packing="false">
```

```
    <pb_type name="iopad" blif_model=".subckt io" num_pb="1" spice_model_name="iopad">
     <input name="outpad" num_pins="1"/>
     <output name="inpad" num_pins="1"/>
    </pb_type>
    <interconnect>
     <direct name="inpad" input="iopad.inpad" output="io.inpad">
      <delay_constant max="4.243e-11" in_port="iopad.inpad" out_port="io.inpad"/>
     </direct>
     <direct name="outpad" input="io.outpad" output="iopad.outpad">
      <delay_constant max="1.394e-11" in_port="io.outpad" out_port="iopad.outpad"/>
     </direct>
    </interconnect>
   </mode>
   <!- IOs can operate as either inputs or outputs.
   Delays below come from Ian Kuon.  They are small, so they should be interpreted
as
   the delays to and from registers in the I/O (and generally I/Os are registered
   today and that is when you timing analyze them.
   ->
   <mode name="inpad">
    <pb_type name="inpad" blif_model=".input" num_pb="1" spice_model_name="iopad"
mode_bits="1">
     <output name="inpad" num_pins="1"/>
    </pb_type>
    <interconnect>
     <direct name="inpad" input="inpad.inpad" output="io.inpad">
      <delay_constant max="4.243e-11" in_port="inpad.inpad" out_port="io.inpad"/>
     </direct>
    </interconnect>
   </mode>
   <mode name="outpad">
    <pb_type name="outpad" blif_model=".output" num_pb="1" spice_model_name="iopad"
mode_bits="0">
     <input name="outpad" num_pins="1"/>
    </pb_type>
    <interconnect>
     <direct name="outpad" input="io.outpad" output="outpad.outpad">
      <delay_constant max="1.394e-11" in_port="io.outpad" out_port="outpad.outpad"/>
     </direct>
    </interconnect>
   </mode>
   <fc default_in_type="frac" default_in_val="0.15" default_out_type="frac"
default_out_val="0.10"/>
   <pinlocations pattern="custom">
    <loc side="left">io.outpad io.inpad</loc>
    <loc side="top">io.outpad io.inpad</loc>
    <loc side="right">io.outpad io.inpad</loc>
    <loc side="bottom">io.outpad io.inpad</loc>
   </pinlocations>
   <gridlocations>
    <loc type="perimeter" priority="10"/>
   </gridlocations>
   <power method="ignore"/>
  </pb_type>
```

```
<!- Define I/O pads ends ->
<pb_type name="clb" area="53894" opin_to_cb="false">
 <pin_equivalence_auto_detect input_ports ="off" output_ports="off"/>
 <input name="I" num_pins="40" equivalent="true"/>
 <output name="O" num_pins="10" equivalent="false"/>
 <clock name="clk" num_pins="1"/>
 <pb_type name="fle" num_pb="10" idle_mode_name="n1_lut6" physical_mode_name="n1_lut6
  <input name="in" num_pins="6"/>
  <output name="out" num_pins="1"/>
  <clock name="clk" num_pins="1"/>
  <mode name="n1_lut6">
   <pb_type name="ble6" num_pb="1">
    <input name="in" num_pins="6"/>
    <output name="out" num_pins="1"/>
    <clock name="clk" num_pins="1"/>
    <!- Define LUT ->
    <pb_type name="lut6" blif_model=".names" num_pb="1" class="lut" spice_model_name=
     <input name="in" num_pins="6" port_class="lut_in"/>
     <output name="out" num_pins="1" port_class="lut_out"/>
     <delay_matrix type="max" in_port="lut6.in" out_port="lut6.out">
      261e-12
      261e-12
      261e-12
      261e-12
      261e-12
      261e-12
```

```
       </delay_matrix>
      </pb_type>
      <!- Define flip-flop ->
      <pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop" spice_model_name="st
       <input name="D" num_pins="1" port_class="D"/>
       <output name="Q" num_pins="1" port_class="Q"/>
       <clock name="clk" num_pins="1" port_class="clock"/>
       <T_setup value="66e-12" port="ff.D" clock="clk"/>
       <T_clock_to_Q max="124e-12" port="ff.Q" clock="clk"/>
      </pb_type>
      <interconnect>
       <direct name="direct1" input="ble6.in" output="lut6[0:0].in"/>
       <direct name="direct2" input="lut6.out" output="ff.D">
        <pack_pattern name="ble6" in_port="lut6.out" out_port="ff.D"/>
       </direct>
       <direct name="direct3" input="ble6.clk" output="ff.clk"/>
       <mux name="mux1" input="ff.Q lut6.out" output="ble6.out" spice_model_name="mux_1level"
        <!- LUT to output is faster than FF to output on a Stratix IV ->
        <delay_constant max="25e-12" in_port="lut6.out" out_port="ble6.out"
/>
        <delay_constant max="45e-12" in_port="ff.Q" out_port="ble6.out" />
       </mux>
      </interconnect>
     </pb_type>
    <interconnect>
     <direct name="direct1" input="fle.in" output="ble6.in"/>
```

```
      <direct name="direct2" input="ble6.out" output="fle.out[0:0]"/>
      <direct name="direct3" input="fle.clk" output="ble6.clk"/>
     </interconnect>
    </mode>
   </pb_type>
   <interconnect>
    <complete name="crossbar" input="clb.I fle[9:0].out" output="fle[9:0].in"
spice_model_name="mux_2level">
     <delay_constant max="95e-12" in_port="clb.I" out_port="fle[9:0].in" />
     <delay_constant max="75e-12" in_port="fle[9:0].out" out_port="fle[9:0].in"
/>
    </complete>
    <complete name="clks" input="clb.clk" output="fle[9:0].clk">
    </complete>
    <direct name="clbouts1" input="fle[9:0].out[0:0]" output="clb.O[9:0]"/>
   </interconnect>
   <fc default_in_type="frac" default_in_val="0.15" default_out_type="frac"
default_out_val="0.10"/>
   <pinlocations pattern="spread"/>
   <!- Place this general purpose logic block in any unspecified column ->
   <gridlocations>
    <loc type="fill" priority="1"/>
   </gridlocations>
  </pb_type>
 <!- Define general purpose logic block (CLB) ends ->
 </complexblocklist>
</architecture>
```

Bibliography

[1] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai, “Metal-Oxide RRAM,” *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, June 2012.

[2] E. Lee, G. Lemieux, and S. Mirabbasi, “Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays,” in *2006 IEEE International Conference on Field Programmable Technology*, Dec 2006, pp. 89–96.

[3] X. Tang, E. Giacomin, G. D. Micheli, and P. E. Gaillardon, “Circuit Designs of High-Performance and Low-Power RRAM-Based Multiplexers Based on 4T(ransistor)1R(RAM) Programming Structure,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 5, pp. 1173–1186, May 2017.

[4] J. R. V. Betz and A. Marquardt, *Architecture and CAD for Deep-Sub-micro FPGAs.* Kluwer Academic Publishers Norwell, MA, USA, 1999.

[5] I. Kazi, P. Meinerzhagen, P. E. Gaillardon, D. Sacchetto, Y. Leblebici, A. Burg, and G. D. Micheli, “Energy/Reliability Trade-Offs in Low-Voltage ReRAM-Based Non-Volatile Flip-Flop Design,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 11, pp. 3155–3164, Nov 2014.

[6] X. Tang, P. E. Gaillardon, and G. D. Micheli, “A High-Performance Low-Power Near-V_t RRAM-based FPGA,” in *2014 International Conference on Field-Programmable Technology (FPT)*, Dec 2014, pp. 207–214.

[7] J. Greene, S. Kaptanoglu, W. Feng, V. Hecht, J. Landry, F. Li, A. Krouglyanskiy, M. Morosan, and V. Pevzner, “A 65nm Flash-based FPGA Fabric Optimized for Low Cost and Power,” in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA ’11).* New York, NY, USA: ACM, 2011, pp. 87–96.

[8] P. E. Gaillardon, D. Sacchetto, G. B. Beneventi, M. H. B. Jamaa, L. Perniola, F. Clermidy, I. O’Connor, and G. D. Micheli, “Design and Architectural Assessment of 3-D Resistive Memory Technologies in FPGAs,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 1, pp. 40–50, Jan 2013.

[9] J. Cong and B. Xiao, “FPGA-RPI: A Novel FPGA Architecture With RRAM-Based Programmable Interconnects,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 864–877, April 2014.

[10] P. Friess, *Internet of Things - Global Technological and Societal Trends From Smart Environments and Spaces to Green ICT*, ser. River Publishers Series in Communications. River Publishers, 2011. [Online]. Available: https://books.google.ch/books?id=Eug-RvslW30C

[11] Evans and Dave, “The Internet of Things: How the Next Evolution of the Internet Is Changing Everything,” Cisco, Tech. Rep., April 2011.

[12] L. D. Xu, W. He, and S. Li, “Internet of Things in Industries: A Survey,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, November 2014.

[13] K. Srinidhi, D. John, and W. Oliver, “BLAS Comparison on FPGA, CPU and GPU,” in *Proceedings of the IEEE Annual Symposium on VLSI (ISVLSI)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 288–293.

[14] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, “A Cloud-Scale Acceleration Architecture,” in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[15] GrandView Research, “Field Programmable Gate Array (FPGA) Market Analysis By Technology (SRAM, EEPROM, Antifuse, Flash), By Application (Consumer Electronics, Automotive, Industrial, Data Processing, Military & Aerospace, Telecom), And Segment Forecasts, 2014 - 2024,” GrandView Research Inc, Tech. Rep., December 2016. [Online]. Available: http://www.grandviewresearch.com/industry-analysis/fpga-market/segmentation

[16] P. Dillien. And the Winner of Best FPGA of 2016 is. [Online]. Available: http://www.eetimes.com/author.asp?section_id=36&doc_id=1331443

[17] Y. Zhou, W. Wang, and X. Huang, “FPGA Design for PCANet Deep Learning Network,” in *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FFCM)*, May 2015, p. 232.

[18] I. Kuon and J. Rose, *Quantifying and Exploring the Gap Between FPGAs and ASICs*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[19] D. S. Jeong, R. Thomas, R. S. Katiyar, J. F. 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.

[20] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, “Overview of Candidate Device Technologies for Storage-Class Memory,” *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, July 2008.

[21] J. R. Stephen D. Brown, Robert J. Francis and Z. G. Vranesic, *Field-Programmable Gate Arrays.* Springer US, 1992, vol. 180.

[22] Y. C. Chen, H. Li, W. Zhang, and R. E. Pino, “The 3-D Stacking Bipolar RRAM for High Density,” *IEEE Transactions on Nanotechnology*, vol. 11, no. 5, pp. 948–956, September 2012.

[23] S. Ambrogio, S. Balatti, V. Milo, R. Carboni, Z. Q. Wang, A. Calderoni, N. Ramaswamy, and D. Ielmini, “Neuromorphic Learning and Recognition With One-Transistor-One-Resistor Synapses and Bistable Metal Oxide RRAM,” *IEEE Transactions on Electron Devices*, vol. 63, no. 4, pp. 1508–1515, April 2016.

[24] A. Chen, “Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications,” in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 10.7.1–10.7.4.

[25] O. Turkyilmaz, S. Onkaraiah, M. Reyboz, F. Clermidy, C. A. Hraziia, J. Portal, and M. Bocquet, “RRAM-based FPGA for "Normally off, Instantly on" Applications,” in *2012 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2012, pp. 101–108.

[26] 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, Nov 2011.

[27] P.-E. Gaillardon, D. Sacchetto, S. Bobba, Y. Leblebici, and G. D. Micheli, “GMS: Generic Memristive Structure for Non-Volatile FPGAs,” in *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*, October 2012, pp. 94–98.

[28] X. Tang, G. D. Micheli, and P. E. Gaillardon, “A High-performance FPGA Architecture Using One-Level RRAM-based Multiplexers,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 1–12, 2016.

[29] I. G. Baek, M. S. Lee, S. Seo, M. J. Lee, D. H. Seo, D. S. Suh, J. C. Park, S. O. Park, H. S. Kim, I. K. Yoo, U. I. Chung, and J. T. Moon, “Highly Scalable Nonvolatile Resistive Memory Using Simple Binary Oxide Driven by Asymmetric Unipolar Voltage Pulses,” in *IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004.*, Dec 2004, pp. 587–590.

[30] P. E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. D. Micheli, “The Programmable Logic-in-Memory (PLiM) Computer,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 427–432.

[31] Y. Zha and J. Li, “Reconfigurable In-Memory Computing with Resistive Memory Crossbar,” in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8.

[32] S. Shirinzadeh, M. Soeken, P. E. Gaillardon, and R. Drechsler, “Fast Logic Synthesis for RRAM-based In-Memory Computing Using Majority-Inverter Graphs,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 948–953.

[33] J. F. Kang, B. Gao, P. Huang, L. F. Liu, X. Y. Liu, H. Y. Yu, S. Yu, and H. S. P. Wong, “RRAM based Synaptic Devices for Neuromorphic Visual Systems,” in *2015 IEEE International Conference on Digital Signal Processing (DSP)*, July 2015, pp. 1219–1222.

[34] G. Indiveri, E. Linn, and S. Ambrogio, “ReRAM-Based Neuromorphic Computing,” *Resistive Switching: From Fundamentals of Nanoionic Redox Processes to Memristive Device Applications*, pp. 715–736, 2016.

[35] R. Liu, H. Wu, Y. Pang, H. Qian, and S. Yu, “A Highly Reliable and Tamper-Resistant RRAM PUF: Design and Experimental Validation,” in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2016, pp. 13–18.

[36] Y. Pang, H. Wu, B. Gao, N. Deng, D. Wu, R. Liu, S. Yu, A. Chen, and H. Qian, “Optimization of RRAM-Based Physical Unclonable Function With a Novel Differential Read-Out Method,” *IEEE Electron Device Letters*, vol. 38, no. 2, pp. 168–171, Feb 2017.

[37] T. Breuer, L. Nielen, B. Roesgen, R. Waser, V. Rana, and E. Linn, “Realization of Minimum and Maximum Gate Function in $T_{a2}O_5$-based Memristive Devices,” *Scientific reports*, vol. 6, 2016.

[38] 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, Sept 2015.

[39] D. Apalkov, B. Dieny, and J. M. Slaughter, “Magnetoresistive Random Access Memory,” *Proceedings of the IEEE*, vol. 104, no. 10, pp. 1796–1830, Oct 2016.

[40] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, “Phase Change Memory,” *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[41] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, “Power Modeling and Characteristics of Field Programmable Gate Arrays,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1712–1724, Nov 2005.

[42] B. H. Calhoun, J. F. Ryan, S. Khanna, M. Putic, and J. Lach, “Flexible Circuits and Architectures for Ultralow Power,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 267–282, Feb 2010.

[43] L. Cheng, P. Wong, F. Li, Y. Lin, and L. He, "Device and Architecture Co-optimization for FPGA Power Reduction," in *Proceedings. 42nd Design Automation Conference, 2005.*, June 2005, pp. 915–920.

[44] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145708

[45] Cadence Design Systems Inc. (2016) Virtuoso Layout Suite. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/custom-ic-analog-rf-design/virtuoso-layout-suite-gxl-ds.pdf

[46] J. B. Goeders and S. J. E. Wilton, "VersaPower: Power Estimation for Diverse FPGA Architectures," in *2012 International Conference on Field-Programmable Technology*, Dec 2012, pp. 229–234.

[47] Synoposys Inc. (2010) HSPICE: The Gold Standard for Accurate Circuit Simulation. [Online]. Available: https://www.synopsys.com/content/dam/synopsys/verification/datasheets/hspice-ds.pdf

[48] J. Luu, J. H. Anderson, and J. S. Rose, "Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 227–236. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950457

[49] W. Kim, S. I. Park, Z. Zhang, Y. Yang-Liauw, D. Sekar, H. S. P. Wong, and S. S. Wong, "Forming-free Nitrogen-doped AlO_X RRAM with Sub-μA Programming Current," in *2011 Symposium on VLSI Technology - Digest of Technical Papers*, June 2011, pp. 22–23.

[50] Z. Fang, H. Y. Yu, X. Li, N. Singh, G. Q. Lo, and D. L. Kwong, "$H_fO_x/T_iO_x/H_fO_x/T_iO_x$ Multilayer-Based Forming-Free RRAM Devices With Excellent Uniformity," *IEEE Electron Device Letters*, vol. 32, no. 4, pp. 566–568, April 2011.

[51] M.-J. Lee, S. Han, S. H. Jeon, B. H. Park, B. S. Kang, S.-E. Ahn, K. H. Kim, C. B. Lee, C. J. Kim, I.-K. Yoo *et al.*, "Electrical Manipulation of Nanofilaments in Transition-Metal Oxides for Resistance-based Memory," *Nano letters*, vol. 9, no. 4, pp. 1476–1481, 2009.

[52] C. Y. Mei, W. C. Shen, Y. D. Chih, Y.-C. King, and C. J. Lin, "28nm High-k Metal Gate RRAM with Fully Compatible CMOS Logic Processes," in *2013 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, April 2013, pp. 1–2.

[53] D. Pramanik, T. Chiang, and D. Lazovsky, "Creating an Embedded Reram Memory from a High-K Metal Gate Transistor Structure," Aug. 12 2014, uS Patent 8,803,124. [Online]. Available: https://www.google.com/patents/US8803124

[54] J. Liang and H. S. P. Wong, “Cross-Point Memory Array Without Cell Selectors — Device Characteristics and Data Storage Pattern Dependencies,” *IEEE Transactions on Electron Devices*, vol. 57, no. 10, pp. 2531–2538, Oct 2010.

[55] 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, Y. H. Kim, C. S. Lee, J. B. Park, I. G. Baek, and I. K. Yoo, “2-stack 1D-1R Cross-point Structure with Oxide Diodes as Switch Elements for High Density Resistance RAM Applications,” in *2007 IEEE International Electron Devices Meeting*, Dec 2007, pp. 771–774.

[56] Z. Wei, Y. Kanzawa, K. Arita, Y. Katoh, K. Kawai, S. Muraoka, S. Mitani, S. Fujii, K. Katayama, M. Iijima, T. Mikawa, T. Ninomiya, R. Miyanaga, Y. Kawashima, K. Tsuji, A. Himeno, T. Okada, R. Azuma, K. Shimakawa, H. Sugaya, T. Takagi, R. Yasuhara, K. Horiba, H. Kumigashira, and M. Oshima, “Highly Reliable T_aO_x ReRAM and Direct Evidence of Redox Reaction Mechanism,” in *2008 IEEE International Electron Devices Meeting*, Dec 2008, pp. 1–4.

[57] W. Guan, S. Long, Q. Liu, M. Liu, and W. Wang, “Nonpolar Nonvolatile Resistive Switching in C_u Doped Z_rO_2,” *IEEE Electron Device Letters*, vol. 29, no. 5, pp. 434–437, May 2008.

[58] Y. S. Chen, H. Y. Lee, P. S. Chen, P. Y. Gu, C. W. Chen, W. P. Lin, W. H. Liu, Y. Y. Hsu, S. S. Sheu, P. C. Chiang, W. S. Chen, F. T. Chen, C. H. Lien, and M. J. Tsai, “Highly Scalable Hafnium Oxide Memory with Improvements of Resistive Distribution and Read Disturb Immunity,” in *2009 IEEE International Electron Devices Meeting (IEDM)*, Dec 2009, pp. 1–4.

[59] J. Sandrini, M. Thammasack, T. Demirci, P.-E. Gaillardon, D. Sacchetto, G. De Micheli, and Y. Leblebici, “Heterogeneous Integration of ReRAM Crossbars in 180nm CMOS BEoL process,” *Microelectronic Engineering*, vol. 145, pp. 62–65, 2015.

[60] B. Govoreanu, A. Redolfi, L. Zhang, C. Adelmann, M. Popovici, S. Clima, H. Hody, V. Paraschiv, I. P. Radu, A. Franquet, J. C. Liu, J. Swerts, O. Richard, H. Bender, L. Altimime, and M. Jurczak, “Vacancy-Modulated Conductive Oxide Resistive RAM (VMCO-RRAM): An Area-Scalable Switching Current, Self-Compliant, Highly Nonlinear and Wide On/Off-Window Resistive Switching Cell,” in *2013 IEEE International Electron Devices Meeting*, Dec 2013, pp. 10.2.1–10.2.4.

[61] A. Schönhals, J. Mohr, D. J. Wouters, R. Waser, and S. Menzel, “3-bit Resistive RAM Write-Read Scheme Based on Complementary Switching Mechanism,” *IEEE Electron Device Letters*, vol. 38, no. 4, pp. 449–452, April 2017.

[62] B. Govoreanu, G. S. Kar, Y. Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. J. Wouters, J. A. Kittl, and M. Jurczak, “$10 \times 10 nm^2$ H_f/H_fO_x Crossbar Resistive RAM with Excellent Performance, Reliability

and Low-energy Operation," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 31.6.1–31.6.4.

[63] H. W. Pan, K. P. Huang, S. Y. Chen, P. C. Peng, Z. S. Yang, C. H. Kuo, Y. D. Chih, Y. C. King, and C. J. Lin, "1Kbit FinFET Dielectric (FIND) RRAM in Pure 16nm FinFET CMOS Logic Process," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 10.5.1–10.5.4.

[64] S. G. Kim, T. J. Ha, S. Kim, J. Y. Lee, K. W. Kim, J. H. Shin, Y. T. Park, S. P. Song, B. Y. Kim, W. G. Kim, J. C. Lee, H. S. Lee, J. H. Song, E. R. Hwang, S. H. Cho, J. C. Ku, J. I. Kim, K. S. Kim, J. H. Yoo, H. J. Kim, H. G. Jung, K. J. Lee, S. Chung, J. H. Kang, J. H. Lee, H. S. Kim, S. J. Hong, G. Gibson, and Y. Jeon, "Improvement of Characteristics of N_bO_2 Selector and Full Integration of $4F^2$ 2x-nm Tech 1S1R ReRAM," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 10.3.1–10.3.4.

[65] S. Yu, X. Guan, and H. S. P. Wong, "On the Stochastic Nature of Resistive Switching in Metal Oxide RRAM: Physical Modeling, Monte Carlo Simulation, and Experimental Characterization," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 17.3.1–17.3.4.

[66] D. Kim, M. Lee, S. Ahn, S. Seo, J. Park, I. Yoo, I. Baek, H. Kim, E. Yim, J. Lee *et al.*, "Improvement of Resistive Memory Switching in N_iO using I_rO_2," *Applied physics letters*, vol. 88, no. 23, p. 232106, 2006.

[67] S. Yu, B. Gao, H. Dai, B. Sun, L. Liu, X. Liu, R. Han, J. Kang, and B. Yu, "Improved Uniformity of Resistive Switching Behaviors in H_fO_2 Thin Films with Embedded Al Layers," *Electrochemical and Solid-State Letters*, vol. 13, no. 2, pp. H36–H38, 2010.

[68] Q. Liu, M. Liu, S. Long, W. Wang, M. Zhang, Q. Wang, and J. Chen, "Improvement of Resistive Switching Properties in Z_rO_2 based ReRAM with Implanted Metal Ions," in *2009 Proceedings of the European Solid State Device Research Conference*, Sept 2009, pp. 221–224.

[69] W.-Y. Chang, K.-J. Cheng, J.-M. Tsai, H.-J. Chen, F. Chen *et al.*, "Improvement of Resistive Switching Characteristics in T_iO_2 Thin Films with Embedded Pt Nanocrystals," *Applied Physics Letters*, vol. 95, no. 4, p. 042104, 2009.

[70] B. Lee and H. S. P. Wong, "N_iO Resistance Change Memory with a Novel Structure for 3D Integration and Improved Confinement of Conduction Path," in *2009 Symposium on VLSI Technology*, June 2009, pp. 28–29.

[71] J. Lee, J. Shin, D. Lee, W. Lee, S. Jung, M. Jo, J. Park, K. P. Biju, S. Kim, S. Park, and H. Hwang, "Diode-less Nano-scale Z_rO_x/H_fO_x RRAM Device with Excellent Switching Uniformity and Reliability for High-density Cross-point Memory Applications," in *2010 International Electron Devices Meeting*, Dec 2010, pp. 19.5.1–19.5.4.

[72] Y. Wu, J. Liang, S. Yu, X. Guan, and H. S. P. Wong, “Resistive Switching Random Access Memory - Materials, Device, Interconnects, and Scaling Considerations,” in *2012 IEEE International Integrated Reliability Workshop Final Report*, Oct 2012, pp. 16–21.

[73] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H. S. P. Wong, “A Neuromorphic Visual System using RRAM Synaptic Devices with Sub-pJ Energy and Tolerance to Variability: Experimental Characterization and Large-scale Modeling,” in *2012 International Electron Devices Meeting*, Dec 2012, pp. 10.4.1–10.4.4.

[74] F. M. Puglisi, P. Pavan, L. Larcher, and A. Padovani, “Analysis of RTN and Cycling Variability in H_fO_2 RRAM Devices in LRS,” in *2014 44th European Solid State Device Research Conference (ESSDERC)*, Sept 2014, pp. 246–249.

[75] A. Levisse, B. Giraud, J. P. Noël, M. Moreau, and J. M. Portal, “SneakPath Compensation Circuit for Programming and Read Operations in RRAM-based CrossPoint Architectures,” in *2015 15th Non-Volatile Memory Technology Symposium (NVMTS)*, Oct 2015, pp. 1–4.

[76] F. Puglisi, C. Wenger, and P. Pavan, “A Novel Program-Verify Algorithm for Multi-Bit Operation in H_fO_2 RRAM,” *IEEE Electron Device Letters*, vol. 36, no. 10, pp. 1030–1032, 2015.

[77] H. Aziza, M. Bocquet, M. Moreau, and J.-M. Portal, “A Built-In Self-Test Structure (BIST) for Resistive RAMs Characterization: Application to Bipolar OxRRAM,” *Solid-State Electronics*, vol. 103, pp. 73–78, 2015.

[78] Altera Corporation. (2014, December) MAX 10 FPGA Device Overview. [Online]. Available: http://www.altera.com/literature/hb/max-10/m10overview.pdf

[79] B. Gao, J. F. Kang, Y. S. Chen, F. F. Zhang, B. Chen, P. Huang, L. F. Liu, X. Y. Liu, Y. Y. Wang, X. A. Tran, Z. R. Wang, H. Y. Yu, and A. Chin, “Oxide-based RRAM: Unified Microscopic Principle for Both Unipolar and Bipolar Switching,” in *2011 International Electron Devices Meeting*, Dec 2011, pp. 17.4.1–17.4.4.

[80] C. Clos, “A Study of Non-Blocking Switching Networks,” *Bell Labs Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[81] Xilinx Inc. (1997) XC4000E and XC4000X Series Field-Programmable Gate Arrays.

[82] Xilinx Inc. (2017) All Programmable 7 Series Product Selection Guide (XMP101). [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf

[83] S. Mühlbach and A. Koch, “A Dynamically Reconfigured Multi-FPGA Network Platform for High-speed Malware Collection,” *International Journal of Reconfigurable Computing - Special issue on Selected Papers from the International Conference on Reconfigurable Computing and FPGAs*, vol. 2012, pp. 4:4–4:4, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/342625

[84] M. Stepniewska, A. Luczak, and J. Siast, “Network-on-Multi-Chip (NoMC) for Multi-FPGA Multimedia Systems,” in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Sept 2010, pp. 475–481.

[85] Intel Corporation. (2017) Stratix 10 GX/SX Device Overview. [Online]. Available: https://www.altera.com/documentation/joc1442261161666.html#joc1443027925492

[86] D. Tavana, W. Yee, and V. Holen, “FPGA Architecture with Repeatable Tiles Including Routing Matrices and Logic Matrices,” Oct. 28 1997, uS Patent 5,682,107. [Online]. Available: http://www.google.com/patents/US5682107

[87] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, “The stratix ii logic and routing architecture,” in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, ser. FPGA ’05. New York, NY, USA: ACM, 2005, pp. 14–20. [Online]. Available: http://doi.acm.org/10.1145/1046192.1046195

[88] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, “Architectural Enhancements in Stratix-III™and Stratix-IV™,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’09. New York, NY, USA: ACM, 2009, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508135

[89] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, “Architectural Enhancements in Stratix V™,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’13. New York, NY, USA: ACM, 2013, pp. 147–156. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435292

[90] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, and J. Van Dyken, “The Stratix™10 Highly Pipelined FPGA Architecture,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’16. New York, NY, USA: ACM, 2016, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847267

[91] Xilinx Inc. (2017) All Programmable SoC with Hardware and Software Programmability. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[92] J. H. Kim and J. H. Anderson, “Synthesizable FPGA Fabrics Targetable by the Verilog-to-Routing (VTR) CAD Flow,” in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015, pp. 1–8.

[93] J. Luu, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "On Hard Adders and Carry Chains in FPGAs," in *Proceedings of the 2014 IEEE 22Nd International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 52–59. [Online]. Available: http://dx.doi.org/10.1109/.23

[94] A. Petkovska, G. Zgheib, D. Novo, M. Owaida, A. Mishchenko, and P. Ienne, "Improved Carry Chain Mapping for the VTR Flow," in *2015 International Conference on Field Programmable Technology (FPT)*, Dec 2015, pp. 80–87.

[95] Z. Chu, X. Tang, M. Soeken, A. Petkovska, G. Zgheib, L. Amarù, Y. Xia, P. Ienne, G. De Micheli, and P.-E. Gaillardon, "Improving circuit mapping performance through mig-based synthesis for carry chains," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, ser. GLSVLSI '17. New York, NY, USA: ACM, 2017, pp. 131–136. [Online]. Available: http://doi.acm.org/10.1145/3060403.3060432

[96] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving Synthesis of Compressor Trees on FPGAs via Integer Linear Programming," in *Design, Automation and Test in Europe, 2008. DATE'08.* IEEE, 2008, pp. 1256–1261.

[97] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault *et al.*, "Improving FPGA Performance and Area Using an Adaptive Logic Module," *Field Programmable Logic and Application*, pp. 135–144, 2004.

[98] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and Single-Driver Wires in FPGA Interconnect," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, Dec 2004, pp. 41–48.

[99] J. Tyhach, M. Hutton, S. Atsatt, A. Rahman, B. Vest, D. Lewis, M. Langhammer, S. Shumarayev, T. Hoang, A. Chan, D. M. Choi, D. Oh, H. C. Lee, J. Chui, K. C. Sia, E. Kok, W. Y. Koay, and B. J. Ang, "Arria ™10 Device architecture," in *2015 IEEE Custom Integrated Circuits Conference (CICC)*, Sept 2015, pp. 1–8.

[100] G. Lemieux and D. Lewis, "Using sparse crossbars within lut," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, ser. FPGA '01. New York, NY, USA: ACM, 2001, pp. 59–68. [Online]. Available: http://doi.acm.org/10.1145/360276.360299

[101] G. Lemieux, P. Leventis, and D. Lewis, "Generating Highly-Routable Sparse Crossbars for PLDs," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, ser. FPGA '00. New York, NY, USA: ACM, 2000, pp. 155–164. [Online]. Available: http://doi.acm.org/10.1145/329166.329199

[102] X. Tang, P. E. Gaillardon, and G. D. Micheli, "Pattern-based FPGA Logic Block and Clustering Algorithm," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2014, pp. 1–4.

[103] X. Tang, P.-E. Gaillardon, and G. De Micheli, “A Full-Capacity Local RoutingArchitecture for FPGAs (Abstract Only),” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’16. New York, NY, USA: ACM, 2016, pp. 281–281. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847314

[104] W. C. Elmore, “The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers,” *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[105] T. Tuan and B. Lai, “Leakage Power Analysis of a 90nm FPGA,” in *Proceedings of the IEEE 2003 Custom Integrated Circuits Conference, 2003.*, Sept 2003, pp. 57–60.

[106] J. J. Wang, S. Samiee, H. S. Chen, C. K. Huang, M. Cheung, J. Borillo, S. N. Sun, B. Cronquist, and J. McCollum, “Total ionizing dose effects on flash-based field programmable gate array,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3759–3766, Dec 2004.

[107] W. D. Brown, J. E. Brewer *et al.*, “Nonvolatile Semiconductor Memory Technology,” *IEEE, New York*, 1998.

[108] M. Zangeneh and A. Joshi, “Performance and Energy Models for Memristor-based 1T1R RRAM Cell,” in *Proceedings of the Great Lakes Symposium on VLSI*, ser. GLSVLSI ’12. New York, NY, USA: ACM, 2012, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/2206781.2206786

[109] Y. C. Chen, W. Wang, H. Li, and W. Zhang, “Non-volatile 3D stacking RRAM-based FPGA,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 367–372.

[110] 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 *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, Dec 2010, pp. 62–65.

[111] X. Tang, S. R. Omam, P. Meinerzhagen, P.-E. Gaillardon, and G. De Micheli, “Low Power FPGAs Based on Resistive Memories,” CRC Press, Tech. Rep., 2015.

[112] K. H. et al., “A Low Active Leakage and High Reliability Phase Change Memory (PCM) based Non-Volatile FPGA Storage Element,” *IEEE TCAS I*, vol. 61, no. 9, pp. 2605–2613, 2014.

[113] J. Cong and B. Xiao, “mrFPGA: A Novel FPGA Architecture with Memristor-based Reconfiguration,” in *Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures.* IEEE Computer Society, 2011, pp. 1–8.

[114] X. Tang, P. E. Gaillardon, and G. D. Micheli, "Accurate Power Analysis for Near-Vt RRAM-based FPGA," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015, pp. 1–4.

[115] N. Jovanović, O. Thomas, E. Vianello, J. M. Portal, B. Nikolić, and L. Naviner, "OxRAM-based Non Volatile Flip-Flop in 28nm FDSOI," in *2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS)*, June 2014, pp. 141–144.

[116] J.-M. Portal, M. Bocquet, M. Moreau, H. Aziza, D. Deleruyelle, Y. Zhang, W. Kang, J.-O. Klein, Y. Zhang, C. Chappert *et al.*, "An Overview of Non-Volatile Flip-Flops based on Emerging Memory Technologies," *J. Electron. Sci. Technol.*, vol. 12, no. 2, pp. 173–181, 2014.

[117] Y. Y. Liauw, Z. Zhang, W. Kim, A. El Gamal, and S. S. Wong, "Nonvolatile 3D-FPGA with Monolithically Stacked RRAM-based Configuration Memory," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International.* IEEE, 2012, pp. 406–408.

[118] K. Huang, R. Zhao, W. He, and Y. Lian, "High-Density and High-Reliability Nonvolatile Field-Programmable Gate Array With Stacked 1D2R RRAM Array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 139–150, Jan 2016.

[119] P. E. Gaillardon, M. H. Ben-Jamaa, M. Reyboz, G. B. Beneventi, F. Clermidy, L. Perniola, and I. O'Connor, "Phase-change-memory-based Storage Elements for Configurable Logic," in *2010 International Conference on Field-Programmable Technology*, Dec 2010, pp. 17–20.

[120] 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.

[121] 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.

[122] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-Path Constraints in Memristor Crossbar Arrays," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on.* IEEE, 2013, pp. 156–160.

[123] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification.

[124] J. Lamoureux and S. J. E. Wilton, "Activity Estimation for Field-Programmable Gate Arrays," in *2006 International Conference on Field Programmable Logic and Applications*, Aug 2006, pp. 1–8.

[125] C. Chiasson and V. Betz, "COFFE: Fully-Automated Transistor Sizing for FPGAs," in *2013 International Conference on Field-Programmable Technology (FPT)*, Dec 2013, pp. 34–41.

[126] F. N. Najm, “A Survey of Power Estimation Techniques in VLSI Circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 446–455, 1994.

[127] J. H. Anderson and F. N. Najm, “Power estimation techniques for fpgas,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, pp. 1015–1027, Oct 2004.

[128] K. K. Poon, S. J. Wilton, and A. Yan, “A Detailed Power Model for Field-Programmable Gate Arrays,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 2, pp. 279–302, 2005.

[129] S. R. Vemuru and N. Scheinberg, “Short-Circuit Power Dissipation Estimation for CMOS Logic Gates,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 41, no. 11, pp. 762–765, Nov 1994.

[130] Z. Jiang, S. Yu, Y. Wu, J. H. Engel, X. Guan, and H.-S. P. Wong, “Verilog-A Compact Model for Oxide-based Resistive Random Access Memory (RRAM),” in *Simulation of Semiconductor Processes and Devices (SISPAD), 2014 International Conference on.* IEEE, 2014, pp. 41–44.

[131] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H. S. P. Wong, “A Compact Model for Metal-Oxide Resistive Random Access Memory With Experiment Verification,” *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, May 2016.

[132] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, “Digital Integrated Circuits,” 2002.

[133] X. Tang, G. Kim, P.-E. Gaillardon, and G. De Micheli, “A Study on the Programming Structures for RRAM-based FPGA Architectures,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 4, pp. 503–516, 2016.

[134] L. Benini and G. De Micheli, “Networks on Chips: A New SoC Paradigm,” *IEEE Computer*, pp. 70–78, 2002.

[135] Mentor Graphics. (2017) ModelSim. [Online]. Available: https://www.mentor.com/products/fv/modelsim/

[136] Laboratory of Integrated Systems (LSI) of EPFL . (2011) FPGA-SPICE Introduction Webpage. [Online]. Available: http://lsi.epfl.ch/downloads

[137] Cadence Design Systems Inc. (2017) Innovus Implementation System: Meet PPA and TAT Requirements At Advanced Nodes. [Online]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/hierarchical-design-and-floorplanning/innovus-implementation-system.html

[138] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0.* Microelectronics Center of North Carolina (MCNC), 1991.

[139] Nanoscale Integration and Modeling (NIMO) Group at Arizona State University (ASU). (2011) Predictive Technology Model (PTM). [Online]. Available: http://ptm.asu.edu/

[140] B. Hoefflinger, *ITRS: The International Technology Roadmap for Semiconductors*. Springer, 2011.

[141] M. Lin, A. E. 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, Feb 2007.

[142] W. Feng and S. Kaptanoglu, "Designing Efficient Input Interconnect Blocks for LUT Clusters Using Counting and Entropy," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, pp. 6:1–6:28, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1331897.1331902

Xifan Tang

tangxifan@gmail.com
+41 78 943 6628
EPFL-IC-LSI
Chemin du Bochet 18, Nr. 17, Ecublens CH-1024, Vaud, Switzerland

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland 09/2013-Present
PhD Candidate
École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland 09/2011-08/2013
Master in Electrical Engineering GPA: 5.23/6.0
Fudan University Shanghai, China 09/2007- 07/2011
Bachelor in Science, Concentration in Micro-Electronics GPA: 3.38/4.0

RESEARCH EXPERIENCE

Supervisor: Prof. Pierre-Emmanuel Gaillardon and Prof. G. De Micheli
Laboratory: LSI, EPFL **01/2013-Present**

- **Circuit Design, architecture exploration and EDA for FPGAs (Focus on RRAM-based FPGAs)**
- **Reconfigurable architecture with ambipolar Logic**

Supervisor: Lecturer Vasileios F.Pavlidis and Prof. G. De Micheli
Laboratory: LSI, EPFL 09/2011-09/2012

- Resonant Clock Tree Network
- Clock and Power Distribution Network on 3-D ICs
- Accurate Power Analysis on LUTs

Supervisor: Prof. Lingli WANG
Laboratory: State Key Lab of ASIC & System, Fudan University 08/2009-07/2011

- RABBIT (Routing Automation of Breadboard Integrated Tools)
- Power Estimation in FPGA
- The Effect of LUT size on Nanometer FPGA Architecture

Wangdao Project Funded by Fudan's Undergraduate Research Opportunity Program (FDUROP)

- Team member in Error Check and Correct System (ECC) on FPGA
- Team member in RAM-BIST (Built In-Self Test) in FPGA

AWARDS AND HONORS

Chinese Government Award for Outstanding Self-Financed Students Abroad	2015
Best paper award nomination at ICFPT 2014 conference	2014
EPFL EDIC Fellowship	2013
Wangdao Scholar honored by FDUROP	2010-2011
Third Prize of Excellent Students at Fudan University	2010-2011
Third Prize of Excellent Students at Fudan University	2007-2008

BOOK CHAPTERS

[1] **Xifan Tang**, S. Rahimian Omam, P. Meinerzhagen, P.-E. Gaillardon and G. De Micheli, *"Low Power FPGAs based on Resistive Memories"* in P.-E. Gaillardon, Editor, "*Reconfigurable Logic: Architecture, Tools and Applications*," CRC press, 28th October 2015, pp. 399-432. 215

JOURNAL PUBLICATIONS (fully refereed)

[1] **Xifan Tang,** E. Giacomin, G. De Micheli and P.-E. Gaillardon, "*Circuit Designs of High-performance and Low-power RRAM-based Multiplexers based on 4T(ransistor)1R(RAM) Programming Structure*", IEEE Transaction on Circuits and Systems I: Regular Papers (TCAS-I), Vol. 64, No. 5, 2017, pp. 1173-1186. (In the list of top 50 most popular papers in May 2017)

[2] **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*A High-performance FPGA Architecture Using One-level RRAM-based Multiplexers*", IEEE Transaction on Emerging Topics in Computing (TETC), Vol. 5, No. 2, pp. 210-222. (In the list of top 50 most popular papers in June and July 2017)

[3] **Xifan Tang,** K. Gain, P.-E. Gaillardon and G. De Micheli, "*A Study on the Programming Structures for RRAM-based FPGA Architectures*", IEEE Transaction on Circuits and Systems I: Regular Papers (TCAS-I), Vol. 63, No. 4, 2016, pp. 503-516. (In the list of top 50 most popular papers in April 2016, and top 10 most popular papers in May 2016)

[4] P.-E. Gaillardon, **Xifan Tang**, G. Kim and G. De Micheli, "*A Novel FPGA Architecture Based on Ultrafine Grain Reconfigurable Logic Cells*", IEEE Transactions on VLSI (Very Large Scale Integration) Systems (TVLSI), Vol. 23, No. 10, pp. 1063-8210, 2015.

[5] J. Zhang, **Xifan Tang**, P.-E. Gaillardon and G. De Micheli, "*Configurable Circuits Featuring Dual-Threshold-Voltage Design With Three-Independent-Gate Silicon Nanowire FETs*", IEEE Transaction on Circuit And Systems Part 1: Regular Papers (TCAS-I), Vol. 61, No. 10, pp. 2851-2861. 2014.

[6] Hu Xu, V. F.Pavlidis, **Xifan Tang**, Wayne P. Burleson, G. De Micheli, "*Timing Uncertainty in 3-D Clock Trees due to Process Variations and Power Supply Noise*", IEEE Transactions on VLSI (Very Large Scale Integration) Systems (TVLSI), Vol. 21, No. 12, pp. 2226-2239, 2013.

[7] S. Rahimian Omam, V. F.Pavlidis, **Xifan Tang** and G. De Micheli, "*An Enhanced Design Methodology for Resonant Clock Trees*", Journal of Low Power Electronics, Vol. 9, No. 2, pp. 198-206, 2013.

CONFERENCE PUBLICATIONS (fully refereed)

[1]. **Xifan Tang,** G. De Micheli and P.-E. Gaillardon, "*Optimization Opportunities in RRAM-based FPGA Architectures*", IEEE Latin American Symposium on Circuits and Systems (LASCAS), 2017, pp. 281-284.

[2]. **Xifan Tang,** E. Giacomin, G. De Micheli and P.-E. Gaillardon, "*Physical Design Considerations of One-level RRAM-based Routing Multiplexers*", ACM/SIGDA International Symposium on Physical Design (ISPD), 2017, accepted for publication.

[3]. **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*A Full-capacity Local Routing Architecture for FPGAs*", International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, U.S.A, 2016, pp. 281-281.

[4]. **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*FPGA-SPICE: A Simulation-based Power Estimation Framework for FPGAs*", International Conference on Computer Design (ICCD), New York, U.S.A., 2015, pp. 696-703.

[5]. **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*Accurate Power Analysis for Near-Vt RRAM-based FPGA*", Field Programmable Logic and Applications (FPL), London, United Kingdom, 2015, pp. 1-4.

[6]. **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*A High-performance Low-power Near-Vt RRAM-based FPGA*", Field Programmable Technology (FPT), Shanghai, China, 2014, pp. 207-214. (**Best paper nomination**)

[7]. **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*Pattern-base Logic Block and Clustering Algorithm*", Field Programmable Logic and Applications (FPL), Munich, Germany, 2014, pp.1-4.

[8]. **Xifan Tang,** J. Zhang, P.-E. Gaillardon and G. De Micheli, "*TSPC Flip-flop Circuit Design with Three-Independent-Gate Silicon Nanowire FETs*", International Symposium on Circuit And Systems (ISCAS), Melbourne, Australia, 2014, pp. 1660-1663.

[9]. **Xifan Tang**, L. Wang, "*The Effect of LUT Size on Nanometer FPGA Architecture*", IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Xi'An, China, 2012, pp. 1-3.

[10]. **Xifan Tang**, L, Wang and H. Xu, *"An Accurate Dynamic Power Model on FPGA Routing Resources"*, IEEE IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT),, Xi'An, China, 2012, pp. 1-4.

[11]. Z. Chu**, Xifan Tang**, *et al.*, "*Improving Circuit Mapping Performance Through MIG-based Synthesis for Carry* Chains", accepted to 27th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2017.

[12]. P.-E. Gaillardon**, Xifan Tang**, J. Sandrini, M. Thammasack, S. Rahimian Omam, D. Sacchetto, Y. Leblebici and G. De Micheli, "*A Ultra-low-power FPGA based on Monolithically Integrated RRAMs*", Design, Automation and Test in Europe Conference and Exhibition (DATE), Grenoble, France, 2015, pp. 1203-1208. (**Invited Paper**)

[13]. P.-E. Gaillardon**,** G. Kim, **Xifan Tang**, L. Amaru and G. De Micheli, "*Towards More Efficent Logic Blocks By Exploiting Biconditional Expansion*", International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, U.S.A, 2015, pp. 262-262.

[14]. S. Rahimian Omam, **Xifan Tang,** P.-E. Gaillardon and G. De Micheli, "*A Study on Buffer Distribution for RRAM-based FPGA Routing Structures*", IEEE Latin American Symposium on Circuit And Systems (LASCAS), Montevideo, Uruguay, 2015, pp. 1-4.

[15]. P.-E. Gaillardon, **Xifan Tang** and G. De Micheli, "*Novel Configurable Logic Block Architecture Exploiting Controllable-Polarity Transistors*", IEEE International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, France, 2014, pp. 1-3. (**Invited Paper**)

INVITED TALKS

[1]. **Xifan Tang**, P.-E. Gaillardon, G. De Micheli, "*High Performance Near-Vt RRAM-based FPGA: Opportunities for Low-Power Versatile Computing,*" HiPEAC (European Network on High Performance and Embedded Architecture and Compilation), Athens, Oct. 8th, 2014.

PATENTS

[1]. **Xifan Tang**, P.-E. Gaillardon, G. De Micheli, "*Pattern-based FPGA Logic Block and Clustering Algorithm,*" Application, US 14/808,506, 26 August 2014.

[2]. P.-E. Gaillardon, **X. Tang**, G. De Micheli, "*A High-Performance Low-Power Near-Vt RRAM-based FPGA,*" Application, US 14/444,422, 28 July 2014, **granted**.

PROFESSIONAL SERVICE

Reviewer for the IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)
Reviewer for the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)
Reviewer for the ACM Computing Surveys (CSUR)
Reviewer for the IEEE Transactions on Nanotechnology (TNANO)
Reviewer for the IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)
Reviewer for the ACM Journal Transactions on Design Automation of Electronic Systems (TODAES)
Reviewer for the 2017 IEEE International Symposium on Circuits And Systems (ISCAS)
Reviewer for the 2017 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)
Reviewer for the Journal of Circuits, Systems and Computers (JCSC)

TEACHING ACTIVITY

Design Technology for Integrated System, Master/PhD course at EPFL 09/2016-12/2016
Responsible for exercise/homework, laboratory sessions and projects.
Design Technology for Integrated System, Master/PhD course at EPFL 09/2015-12/2015
Responsible for exercise/homework, laboratory sessions and projects.
Design Technology for Integrated System, Master/PhD course at EPFL 09/2014-12/2014

Responsible for exercise/homework, laboratory sessions and projects.

INTERNSHIP

Melexis	Bevaix, Switzerland	10/2012-12/2012

Supervisor: Christophe Guillaume-Gentil

Internship Project: Modeling a Near-Field Communication (NFC) Chip with Verilog-A

EXTRACURRICULAR ACTIVITIES

Volunteer of World EXPO 2010	Shanghai	05/2010

COMPUTING SKILLS AND OTHERS

Computer skills:	Linux, C, Perl, VHDL, VHDL-AMS, Verilog-A, HSPICE, Matlab, ModelSim, Design Compiler, Virtuoso, Quartus II, LabView NI, Visual Basic, Qt, Hadoop
Languages:	Chinese(Native), English (Fluent)