EMERGING RUN-TIME RECONFIGURABLE FPGA AND CAD TOOLS

by

Yi-Chung Chen

B.S. in Electrical Engineering, Yuan Ze University, Taiwan, 2006

M.S. in Electrical Engineering,

Polytechnic Institute of New York University, 2011

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Yi-Chung Chen

It was defended on

September 20th 2014

and approved by

Hai Li, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Yiran Chen, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Alex K. Jones, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Kartik Mohanram, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Jun Yang, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Wei Zhang, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering, Hong Kong University of Science and Technology

Dissertation Director: Hai Li, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

EMERGING RUN-TIME RECONFIGURABLE FPGA AND CAD TOOLS

Yi-Chung Chen, PhD

University of Pittsburgh, 2014

Field-programmable gate array (FPGA) is a post fabrication reconfigurable device to accelerate domain specific computing systems. It offers offer high operation speed and low power consumption. However, the design flexibility and performance of FPGAs are severely constrained by the costly on-chip memories, *e.g. static random access memory* (SRAM) and FLASH memory. The objective of my dissertation is to explore the opportunity and enable the use of the emerging *resistance random access memory* (ReRAM) in FPGA design.

The emerging ReRAM technology features high storage density, low access power consumption, and CMOS compatibility, making it a promising candidate for FPGA implementation. In particular, ReRAM has advantages of the fast access and nonvolatility, enabling the on-chip storage and access of configuration data. In this dissertation, I first propose a novel three-dimensional stacking scheme, namely, *high-density interleaved memory* (HIM). The structure improves the density of ReRAM meanwhile effectively reducing the signal interference induced by sneak paths in crossbar arrays. To further enhance the access speed and design reliability, a fast sensing circuit is also presented which includes a new sense amplifier scheme and reference cell configuration.

The proposed ReRAM FPGA leverages a similar architecture as conventional SRAM based FPGAs but utilizes ReRAM technology in all component designs. First, HIM is used to implement *look-up table* (LUT) and *block random access memories* (BRAMs) for functionality process. Second, a 2R1T, two ReRAM cells and one transistor, nonvolatile switch design is applied to construct *connection blocks* (CBs) and *switch blocks* (SBs) for signal transition. Furthermore, *unified BRAM* (uBRAM) based on the current BRAM architecture

is introduced, offering both configuration and temporary data storage. The uBRAMs provides extremely high density effectively and enlarges the FPGA capacity, potentially saving multiple contexts of configuration. The fast configuration scheme from uBRAM to logic and routing components also makes fast run-time *partial reconfiguration* (PR) much easier, improving the flexibility and performance of the entire FPGA system.

Finally, modern place and route tools are designed for homogeneous fabric of FPGA. The PR feature, however, requires the support of heterogeneous logic modules in order to differentiate PR modules from static ones and therefore maintain the signal integration. The existing approaches still reply on designers' manual effort, which significantly prolongs design time and lowers design efficiency. In this dissertation, I integrate PR support into VPR – an academic place and route tool by introducing a *B*-tree modular placer* (BMP) and PR-aware router. As such, users are able to explore new architectures or map PR applications to a variety of FPGAs. More importantly, this enhanced feature can also support fast design automation, *e.g.* mapping IP core, loading pre-synthesizing logic modules, *etc.*

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 MOTIVATION 1

1.1.1 Challenge 1: High Density On-chip Memory System 2

1.1.2 Challenge 2: Components of Nonvolatile FPGA and Large Local Memory 3

1.1.3 Challenge 3: CAD tools for Partial Reconfiguration 4

1.2 Research Approach and Dissertation Outline 4

2.0 PREVIOUS WORK 7

2.1 Resistive Random Access Memory (ReRAM) 7

2.1.1 Resistive Random Access Memory 7

2.1.2 Crossbar Array 9

2.1.3 Complementary ReRAM 10

2.1.4 Stacking of ReRAM 11

2.2 Novel FPGA Components 12

2.3 Place and Route for FPGA 13

3.0 HIGH DENSITY RERAM 16

3.1 3D High-Density Interleaved Memory 16

3.1.1 The Proposed 3D-HIM Structure 16

3.1.2 Memory Density Improvement 17

3.1.3 Memory Accesses in 3D-HIM 18

3.1.3.1 Bi-Group Operation Scheme 18

3.1.3.2 Read Operation 18

3.1.3.3 Write Operation 20

3.2 Simulation result & discussion . . . 21
3.2.1 Impact of Data Pattern and Cell Location . . . 22
3.2.1.1 Impact of Data Patterns . . . 22
3.2.1.2 Impact of Cell Location . . . 23
3.2.2 Read operation . . . 25
3.2.3 Write operation . . . 26
4.0 RERAM FPGA . . . 29
4.1 ReRAM Look-up Table . . . 29
4.1.1 Conventional LUT in FPGAs . . . 29
4.1.2 Design Concept and Overview . . . 30
4.1.3 Read Scheme Design . . . 32
4.1.3.1 ReRAM Crossbar Access Control . . . 32
4.1.3.2 Expanded ReRAM Crossbar Structure . . . 34
4.1.3.3 Sense Amplifier . . . 36
4.1.4 Write Scheme Design . . . 38
4.1.5 Memory Configuration . . . 41
4.1.6 Flexible Configurations . . . 41
4.1.7 Simulation Results . . . 43
4.1.7.1 Sense Amplifier . . . 43
4.1.7.2 Area and Delay . . . 46
4.1.7.3 System-level Exploration: ReRAM LUT . . . 48
4.2 ReRAM Routing Components . . . 51
4.2.1 Connection Blocks . . . 51
4.2.2 Switch Blocks . . . 52
4.2.3 Discussion – ReRAM FPGA . . . 53
4.2.3.1 Delay and Area of the Proposed CB and SB . . . 54
4.2.3.2 Power Consumption of Components . . . 55
4.2.3.3 Various Comparison of Compositions in One Tile . . . 55
4.2.3.4 Benchmark Mapping . . . 56
4.2.4 Power Estimation and Comparison . . . 59

4.2.5 Miscellaneous Discussion . . . 62
4.3 uBRAM and Reconfiguration . . . 63
4.3.1 Architecture of Unified Block Memory . . . 63
4.3.2 Reconfiguration with uBRAM . . . 67
4.3.2.1 Initialization . . . 67
4.3.2.2 Runtime Partial Reconfiguration . . . 67
4.3.3 Case Studies – FPGA with uBRAM . . . 69
4.3.3.1 Case Study–Dynamic Loading Pre-Synthesized Configurations 71
4.3.3.2 Case Study–Runtime Calculating Configurations . . . 73
5.0 PLACE AND ROUTE OF PR FPGA – PR-AWARE ROUTER AND LOCAL APR . . . 76
5.1 Mapping Flow . . . 76
5.2 Logic Synthesis . . . 78
5.2.1 Preliminary Logic Library . . . 78
5.2.2 Netlist Combination Tool . . . 80
5.3 Place and Route of PR . . . 83
5.3.1 Preliminary - BMP . . . 83
5.3.2 BMP supporting PR . . . 84
5.3.3 Exploring Types of PR Routes and Port-Over-Track . . . 87
5.3.4 PR-aware Global Routing . . . 90
5.3.5 Local Place and Route . . . 95
5.3.5.1 Local Place . . . 95
5.3.5.2 Local Route . . . 96
5.3.6 Results and Demonstration . . . 96
5.3.7 Verification of the Work . . . 99
6.0 CONCLUSION AND FUTURE WORKS . . . 103
6.1 Dissertation Summary . . . 103
6.2 Future Work . . . 104
6.2.1 Device and Circuit studies of ReRAM . . . 104
6.2.2 System and Application . . . 105

6.2.3 CAD for FPGA 105
BIBLIOGRAPHY 106

LIST OF TABLES

1 Driving Conditions of Writing Operations 21
2 Parameters of ReRAM and 3D-HIM 22
3 Comparison of Various Sense Amplifier Designs 45
4 Delay of the SB and the CB 54
5 Power Comparison: SRAM and ReRAM 55
6 Mapping results of OFDM modulations 71

LIST OF FIGURES

1 (a) Structure of Cu-$Ge_{0.3}Se_{0.7}$-SiO_2-Pt [28]. (b) The complementary cell structure [20]. 8

2 3X3 crossbar array. 9

3 ReRAM crossbar array and the sneak path. 10

4 Stacking crossbar array. 11

5 3D-HIM structure. 17

6 Layout of 3D-HIM design. 18

7 Selected WL_1GC in read operation. 19

8 Selected WL_1GC in SET operation. 20

9 Three catalogs of cells used for data patterns analysis. 23

10 Worst case and best case scenario of the cell locations. 24

11 SM difference by the cell locations and data patterns 24

12 SM of 3D-HIM 25

13 Portion of sneak path conducting current in sensing current through R_{sense} and SM of four-layer 3D-HIM with various R_{on} 27

14 Proper programming voltage 27

15 (a) A SRAM LUT. (b) The proposed ReRAM LUT. 30

16 The proposed read scheme. 33

17 The impact of data pattern on (a) $R_{14,eq}$ and $R_{ref1,eq}$, and (b) $\Delta R = R_{14,eq} - R_{ref1,eq}$. 35

18 The proposed sense amplifier in ReRAM LUT. 36

19 Write scheme of 6-input LUT. The function decoding and driving source are shared by four LUTs within one LB. The thick lines represent multiple interconnection signals. 38
20 This is an example of the write scheme design. The green block is unshared *isolation* part. The purple block is *driving* part, which can be shared by multiple LUTs. 40
21 Address of the proposed 6-input LUT. 42
22 (a) Simulated waveforms of sense amplifier in a 4-input LUT under critical sensing condition of data pattern. (b) An enlarged view of sense amplifier's inputs. 44
23 ReRAM resistance varies 5% to 30% from its design value. (a) The waveforms of sense amplifier inputs under the worst case data pattern. (b) The corresponding sensing result. 45
24 Timing diagram of 6-input LUT with 2 outputs at 500Mhz. 46
25 (a) Area comparison of various LUTs. Area estimations normalized to conventional 6-input LUT. (b) Delay of LUT with proposed sensing scheme for different operation frequency. 47
26 (a) Delay comparison. (b) Area estimation. 49
27 Area and delay of different sizes of LUT. 50
28 The CB is designed with pass-gates and buffers. The green lines show connections of ReRAM cells in CRS and pass-gates. The ReRAM cells in CRS can open or close the pass-gate based on resistance difference. 52
29 ReRAM pass gate switch: a) program short. b) program open. c) short in normal operation. d) open in normal operation. 53
30 The SB is designed with pass-gates and buffers. The green lines show connections of ReRAM cells in CRS and pass-gates. The ReRAM cells in CRS can open or close the pass-gate based on resistance difference. 54
31 Power distribution of components in one tile. 56
32 CAD and corresponding file in this work. 57
33 MCNC Comparison: Area (μm^2) 59

34 MCNC Comparison: Delay of Critical Path (ns) 60
35 Power estimation of 45nm SRAM and ReRAM FPGAs. 62
36 This is a RU of the proposed architecture composed of eight tiles. The blue area is a tile comprising LB, SB, and CB. 64
37 The proposed ReRAM uBRAM with eight memory islands and a set of shared address lines. 64
38 The interconnections of uBRAM for data/configuration mode. 65
39 Special tracks from uBRAM to each tile. 66
40 The rough flow for optimizing, controlling, and monitoring reconfigurable logic. 70
41 Signal flow of OFDM. 71
42 The internal logic generates weighting to uBRAM for later configuring. 73
43 The internal runtime logic is generated to uBRAM for later configuring. . . . 75
44 Mapping flow for PR logic. 77
45 (a) Various shape of the same module. (b) Solid and dash red line shows wirelength estimation from two modules. 80
46 A general flow of NCT. 82
47 A placement and corresponding B*-tree representation. 84
48 Boundtop floorplan. 85
49 Boundtop placement. 86
50 Types of connections. 88
51 General. 93
52 PR-aware. 94
53 Global placement of PR boundtop. 97
54 Global routing of PR boundtop. 98
55 Local placement of a new context of the PR region. 100
56 Local routing of a new context of the PR region. 101

PREFACE

This dissertation is submitted in partial fulfillment of the requirements for Yi-Chung Chen's degree of Doctor of Philosophy in Electrical and Computer Engineering. It contains the work done from January 2010 to August 2014. My advisor is Hai (Helen) Li, Polytechnics of New York University, 2009–2012 and University of Pittsburgh, 2012 – present. My co-advisor is Wei Zhang, Nanyang Technological University, 2010 – 2013 and Hong Kong University of Science and Technology, 2013 – present.

The work is to the best of my knowledge original, except where acknowledgement and reference are made to the previous work. There is no similar dissertation has been submitted for any other degree at any other university.

Part of the work has been published in the publications:

1. F. Mao, **Y.-C. Chen**, W. Zhang, and H. Li, 'BMP: A Fast B*-Tree based Modular Placer for FPGAs,' ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2014, pp. 248-248.
2. H. Liang, **Y.-C. Chen**, W. Zhang, and H. Li, 'Hierarchical Library Based Power Estimation for Versatile FPGAs,' ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2014, pp. 243-243.
3. **Y.-C. Chen**, W. Wang, W. Zhang, and H. Li, 'uBRAM-based Run-time Reconfigurable FPGA and Corresponding Reconfiguration Methodology,' in IEEE International Conference on Field-Programmable Technology (FPT), 2012, pp. 80-86.
4. **Y.-C. Chen**, W. Wang, H. Li, and W. Zhang, 'Non-volatile 3D stacking RRAM-based FPGA,' in IEEE International Conference on Field Programmable Logic and Applications

(FPL), 2012, pp. 367-372.

5. **Y.-C. Chen**, H. Li, and W. Zhang, 'A Novel Peripheral Circuit for RRAM-based LUT,' in IEEE International Symposium on Circuits and Systems (ISCAS), 2012, pp. 1811-1814.

6. **Y.-C. Chen**, W. Zhang, and H. Li, 'A Look Up Table Design with 3D Bipolar RRAMs,' in IEEE Asia and South Pacific Design Automation Conference (ASPDAC), 2012, pp. 7378.

7. **Y.-C. Chen**, H. Li, W. Zhang, and R. E. Pino, 'The 3D Stacking Bipolar RRAM for High Density,' IEEE Transactions on Nanotechnology (TNANO), pp. 948-956, 2012.

8. **Y.-C. Chen**, H. Li, W. Zhang, and R. E. Pino, 'A RRAM-based Memory System and Applications,' in Non-volatile Memory Workshop (NVMW), 2012.

9. **Y.-C. Chen**, H. Li, W. Zhang, and R. E. Pino, '3D-HIM: A 3D High-density Interleaved Memory for Bipolar RRAM Design,' in IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), 2011, pp. 59-64.

10. **Y.-C. Chen**, H. Li, Y. Chen, and R. Pino, '3D-ICML: A 3D Bipolar ReRAM Design with Interleaved Complementary Memory Layers,' in IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, pp. 1-4.

Part of the work has been submitted to journal publication and under review:

1. **Y.-C. Chen**, H. Li, and W. Zhang, 'RRAM-based Lookup Table Design for Nonvolatile FPGAs.'

2. **Y.-C. Chen**, W. Wang, H. Liang, W. Zhang, and H. Li, 'RRAM-based FPGA with uBRAM for Run-time Reconfigurable Computing.'

3. H. Liang, **Y.-C. Chen**, W. Zhang, and H. Li, 'Hierarchical Library Based Power Estimator for Versatile FPGAs.'

4. F. Mao, **Y.-C. Chen**, W. Zhang, and H. Li, 'Design Automation for PR FPGAs - Part I: B*-tree based Modular Placer.'

5. **Y.-C. Chen**, F. Mao, W. Zhang, and H. Li, 'Design Automation for PR FPGAs – Part II: PR-aware Router.'

ACKNOWLEDGEMENTS

I would never be able to finish my dissertation without the guidance of my committee members and help from friends. I'd like to thank Professor Hai (Helen) Li, and Professor Wei Zhang for their excellent guidance during the research. Professor Hai (Helen) Li gives me guidance of nonvolatile memory design fro device to circuit. Professor Wei Zhang gives me guidance of FPGA design, architecture, to CAD. Special thanks go to Professor Yiran Chen, Professor Alex K. Jones, Professor Kartik Mohanram, and Professor Jun Yang for being my committee members.

I'd like to thank Fubing Mao, Nanyang Technological University, for his work 'B*-tree based Modular Placer.' I'd like to thank Hao Liang, Nanyang Technological University and Hong Kong University of Science and Technology, for his work 'Hierarchical Library Based Power Estimator.' I'd like to thank Wenhua Wang, Polytechnics of New York University and Standard & Poor's, for her collaboration work of ReRAM FPGA. Finally, I'd like to thank my friends, Xiuyuan Bi, Miao hu, and Zhenyu Sun for their help and discussion during my research.

1.0 INTRODUCTION

1.1 MOTIVATION

Field-programmable gate array (FPGA) has been widely used in system designs and prototype developments for its flexible post-fabrication reconfigurability, low development risk, short time to market, and low cost of small volume. In modern FPGAs, configuration data including both logic and routing elements can be stored in either a volatile memory, *e.g.*, *static random access memory* (SRAM) [1, 2], or nonvolatile media, such as antifuse or Flash memory [3, 4]. Since data in SRAM FPGA cannot be retained without power supply, an external nonvolatile memory is needed to store configuration while powering off the system [5]. It also needs an initialization stage after powering on the system. In contrast, external storage is not necessary for an antifuse-based or Flash-based FPGA because a logic configuration can be retained within the system. For an antifuse FPGA, it can be programmed only one time [5]. An FLASH-based design needs a non-standard CMOS process to fabricate the chip [5]. Thus, we need a new on-chip memory system to support on-chip storage of configuration without adding large design complexity. Meanwhile, density of the memory should be as higher as possible for improving performance of FPGA [5].

Due to slowing down of Moore's Law, FPGA has also attracted attentions as accelerators in computing systems because of its high performance, low power, high design flexibility, and low cost [6][7][8]. For example, FPGAs are widely used in domain-specific computing to accelerate the critical and intensive computation, and to implement non-general functions [9]. Some adaptive systems is even required to modify the function based on run-time requirements. To support such requirements, modern FPGAs have enabled the function of dynamic *partial reconfiguration* (PR). PR is an advanced type of reconfiguration for FP-

GAs to support online replacement of partial logic according to run-time requirements. The operation requires separation of PR and static logic, *i.e.* the logic kept unchanged during operation, because connections among the static logic can not be routed in PR region to prevent signal interruption caused by PR [10]. Decomposing a design into logic modules is a well adopted method to differentiate PR and static logic [11][12]. The modules of various logic functions may be quite different in area, shape, delay, *etc.* Therefore, to place modules to a FPGA is different from the conventional fin-grain manner that has homogeneous unit, *e.g. configurable logic block*, for considerations of optimization[13]. There are commercial PR *computer-aided design* (CAD) tools, such as PlanAhead [11] and TransFR [14] to support modular placement. However, mapping steps still mainly rely on manual efforts. For academic CAD tools, few works considered module-based placement and routing to support PR. The commonly used placer and router, *e.g.* VPR, is based on homogeneous tiles [15, 16] for optimization. Thus, there is a missing link between modern CAD tools and requirements of PR including the modular placement and PR region-aware routing.

1.1.1 Challenge 1: High Density On-chip Memory System

To improve modern on-chip memory systems, which includes SRAM, FLASH, anti-fuse, in FPGA, it should be high density, low power, CMOS-compatible process, or even nonvolatility. Among emerging nonvolatile memory devices, *resistive random access memory* (ReRAM) is a promising candidate with above features. To improve memory density, 3D stacking that builds up multiple memory layers vertically is an efficient way. Conventionally, there is an isolation layer between two adjacent memory layers in order to avoid the malfunctions caused by the signal interference when simultaneously accessing multiple memory layers [17]. However, manufacturing the isolation layers introduces potential reliability issues, such as the melting (or even destruction) of metal interconnects during the annealing step. To prevent this from happening, a low thermal budget process, *e.g.*, undoped Methylsilsesquioxane (MSQ) Spin-on-Glass (SOG) technology [18], is required, which could significantly increase the process complexity and the fabrication cost [19].

We introduced *high-density interleave memory* (HIM) structure for ReRAM. HIM is a monolithic stacking crossbar structure and there is no isolation between memory layers. ReRAM cells are located at cross point of crossbar arrays. Then, it stacks multiple layers as a 3D structure by sharing wordlines and bitlines. Meanwhile, HIM controls memory access with a bi-group operation scheme. Area cost of a ReRAM cell in a crossbar structure is $4F^2$, where F is feature size of technologies. From the layout point of view, $4F^2$ is the smallest area cost for a memory cell in one layer memory array. For HIM, it further improves area cost to $4F^2/N$, where N is numbers of stacking layers.

1.1.2 Challenge 2: Components of Nonvolatile FPGA and Large Local Memory

As changing the memory system of FPGA to ReRAM, we have to modify components' circuits to adapt ReRAM in FPGA. Design of a ReRAM LUT is different from a conventional SRAM LUT. It has decoders to access the configuration bits while the SRAM LUT uses a multiplexer. ReRAM needs a *sense amplifier* (SA) to read out a cell. It is a huge area overhead by using a multiplexer since each cell needs its local SA. To reduce numbers of SA, ReRAM cells of a LUT share a SA with decoders to access the targeting cell. To construct nonvolatile routing system, we also introduced a 2R1T switch with ReRAM cells as the main component of *connection block* (CB) and *switch block* (SB). With forward and reverse biasing, it programs the switch open and close, respectively. To further reduce area cost, we aggregated switches into a crossbar array for sharing peripheral circuits, *e.g.* programming drivers, *etc.* With ReRAM, the FPGA only needs one time initialization compared to initialization after every power resuming of SRAM FPGAs.

We introduced *unified block random access memory* (uBRAM) with ReRAM for enhancing FPGA's performance and functionality. The main difference is that uBRAM can acts as both temporary data storage and configuration memory. As a temporary data storage, uBRAM is similar to a conventional BRAM as a local and wide-distributed memory, however, it has much higher density than a conventional BRAM. Large data can be stored in a local memory rather than an external RAMs. It reduces traffics through IO, which degrades system performance with IO latency and increases extra power consumption on buses. As a

local configuration memory, it configures logic and routing components through special tracks to corresponding ReRAM cells. Therefore, configuring logic is much faster than conventional FPGAs that loads configuration through external resource. For a PR operation, it reduces stall time between loading configurations. Meanwhile, it can store multiple configurations in FPGA because of its large capacity.

1.1.3 Challenge 3: CAD tools for Partial Reconfiguration

Modern CAD tools were developed to optimize synthesis, place, and route of homogeneous logic tiles of FPGAs. For PR, circuits between static and reconfigurable logic regions need to be partitioned from *hardware description language* (HDL) to floorplan. After partition, basic elements are logic modules, which should have their own preserved regions in placement. These modules of various functions are quite different in area, shape, delay, *etc.* Connections between the modules should be carefully designed to prevent computation from interruption during PR. All these resource management steps still mainly rely on manual efforts, which prolong design time and significantly lower efficiency. Thus, there is a missing link between modern CAD tools and requirements of PR applications. The missing link should compose of CAD tools for modular placer and reconfiguration region-aware router for PR. We introduced *B*-Tree modular placer* (BMP) and PR-aware router, which are modified from academic CAD, *Versatile Place and Route* (VPR), to support PR. BMP automatically generates floorplan of modules, which is partitioned by designer from HDL source or reference library of logic modules. Meanwhile, it kicks in a context-selecting algorithm to guarantee that all contexts of a module can be fitted in the designed PR region. PR-aware router then guides routes based on the placement and maintains signal integrity for PR operations.

1.2 RESEARCH APPROACH AND DISSERTATION OUTLINE

There are three parts to this research: high density ReRAM memory, design of ReRAM FPGA with dedicated PR, and CAD tool for PR FPGA. At the Chapter 2, we go through previous works related to the research. We discuss design idea and results in the rest chapters.

We introduce high density ReRAM in Chapter 3. Building up ReRAM in a 3D stacking structure boosts its advantage in array density. Conventionally, multiple bipolar ReRAM layers are piled up vertically by with isolation material to prevent signal interference between the adjacent memory layers. The process of the isolation material increases the fabrication cost and brings in the potential reliability issue. To alleviate the situation, we introduce a novel 3D stacking structures built upon bipolar ReRAM crossbars that eliminate the isolation layers. The bi-group operation scheme dedicated for the proposed designs to enable multi-layer accesses while avoiding the overwriting induced by the cross-layer disturbance is also presented. Our simulation results show that the proposed designs can increase the capacity of a memory island to 8K-bits (*i.e.*, 8 layers of 32 $\times$ 32 crossbar array) while maintaining the sense margin in the worst-case configuration greater than 20% of the maximal sensing voltage.

In Chapter 4, we demonstrate ReRAM crossbar structure in FPGA to develop LUT and uBRAM. A nonvolatile pass gate switch (2R1T) in *complementary resistive switches* (CRS) structure [20] is introduced as a basic component of nonvolatile SB and CB. The memory cells of switches are further integrated in a crossbar structure for reducing area cost. Compared to a conventional 6-input SRAM LUT [1], the ReRAM LUT cuts off 60.4% of layout area for a 180 nm technology node. Maximal operating frequency reaches 1 GHz at 10 mV input difference. The SB and CB can save 58.6% and 67% area cost for the 180 nm technology node. Our simulation results demonstrate that the FPGA achieves 62.7% area reduction and 34% access latency improvement compared with the conventional SRAM FPGA. The introduction of uBRAM enables runtime reconfiguration in a few μs. The case study on a *orthogonal frequency-division multiplexing* (OFDM) [21] module shows 34.5% and 17% saving in chip area and power consumption, respectively. Overall, the ReRAM FPGA demonstrates advantages such as a eliminating initialization stage, a fast runtime configuration scheme, and power saving with a deep sleep mode.

Place and route tool to support PR on FPGA is illustrated in Chapter 5. PR demonstrates significance in high performance systems, *e.g.* reconfigurable processor, customized computing system, etc. However, there is a lack of placer and router to support PR on FPGA. Conventional implementation of reconfigurable applications usually relies on man-

ual partition and floorplan, which requires huge efforts and long development period. To address this problem, we introduce a PR mapping flow, BMP, and PR-aware router based on an academic place and route tool, VPR [16]. The major function of the PR mapping flow is to support of PR logic for design automation. Moreover, it supports general logic by loading logic modules from a logic library, *e.g.* IP-core, pre-synthesized modules, *etc.* The flow helps designers to explore future architectures of PR FPGA and evaluate performance, cost, *etc.*, of a PR logic. BMP is a modular placer that takes static and reconfigurable functions as modules and performs modular placement to minimize total area and delay of the application. The modular information is represented in B*-tree structure [22] to allow fast searching of solution space. We amend the operations of B*-Tree to fit hardware characteristic of FPGAs. Different aspect ratios of the modules are explored during simulated annealing to achieve area-delay product optimization. The PR-aware router differentiates routing resource of static and PR logic while searching solution space. We introduce pseudo ports, which is based on idea of *port-over-track*, as connecting abutments between PR and static logic. Together of the two schemes, it guarantees signal integrity of the static logic during a PR operation. After the global place and route, it performs local place and route for modules of multi-context to complete place and route of the PR logic.

2.0 PREVIOUS WORK

2.1 RESISTIVE RANDOM ACCESS MEMORY (RERAM)

2.1.1 Resistive Random Access Memory

ReRAM is a two terminal memory device based on difference of resistance to store logic binary data "1" or "0." It can be realized by various materials of widely storage mechanisms [23]. Generally, all of these materials fall into only two operation types – *unipolar* switching and *bipolar* switching. Within this context, unipolar operation executes the programming/erasing by using short/long pulses, or by using high/low voltage with the same voltage polarity. In contrast, bipolar switching is implemented by short pulses with opposite voltage polarity for programming and erasing [24]. Popular bipolar ReRAM devices has an oxide layer as an insulator [25], which is ,by natural, high resistance state after fabrication. Redox, reduction and oxidation, is main switch mechanism of the type of devices. For major ReRAM devices today, redox is forming/disforming small conducting path, which known as filament, within oxide layer [26]. With filament, electronics pass though oxide layer so we can observe low resistance of he device, comparing to the one without filament. For modern applications, we only take ReRAM as two states device, high and low, device regardless unstable intermediate states.

In this work, we target mainly on 3D structures with bipolar ReRAM devices for their fast switching speed and the less power consumption in *RESET* (that is, erase) operation [27]. For demonstration purpose, we use the material Cu-$Ge_{0.3}Se_{0.7}$-SiO_2-Pt [28] as example. However, the proposed design concept can be easily extended to the other bipolar ReRAM devices.

Fig. 1(a) illustrates the structure of Cu-$Ge_{0.3}Se_{0.7}$-SiO_2-Pt [28]. It is a programmable metallization cell device formed in a sandwich structure with heterogeneous solid metal electrodes at two poles. One pole is relatively inert Pt (called as the *bottom electrode*, or BE), the other is electro-chemically active Cu (called as the *top electrode*, or TE). A thin electrolyte film composed of ternary glass $Ge_{0.3}Se_{0.7}$ with added dissolved active metal Cu is placed between the two electrodes. The SiO_2 is used as a buffer layer to improve the endurance in the electrolyte [29]. The $Ge_{0.3}Se_{0.7}$ and SiO_2 are the places where the resistance changing happens.

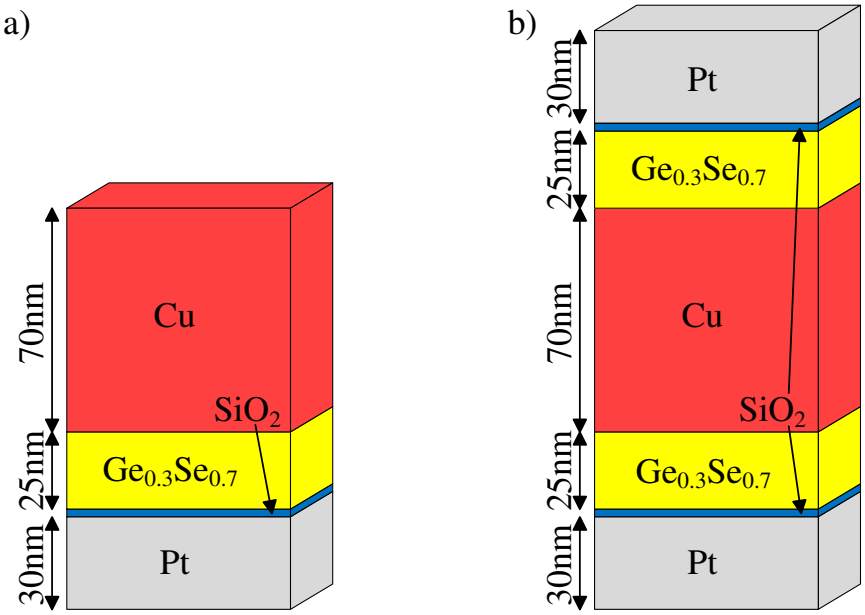


Figure 1: (a) Structure of Cu-$Ge_{0.3}Se_{0.7}$-SiO_2-Pt [28]. (b) The complementary cell structure [20].

For convenience, we define R_{on} and R_{off} as the resistance value at the *low resistance state* (LRS) and at the *high resistance state* (HRS), respectively. The R_{off}/R_{on} is an important device parameter representing the difference between HRS and LRS. In general, a high R_{off}/R_{on} is more preferable.

When a negative bias is applied to the BE during a *SET* operation (that is, the device changes to the LRS), the dissolving Cu reacts with Se in electrolyte compound to generate cation conductors which forms a filament between two electrodes for electron transportation.

As a result, the resistance between two electrodes is dramatically reduced. To *RESET* a cell (to change the device to the HRS), a positive bias can be applied on the BE and remove the random dissolving Cu from Cu-Ge-Se compound filament. The resistance becomes relatively high once the filament disappears in the electrolyte [29].

2.1.2 Crossbar Array

Crossbar array is widely used in ReRAM design for its simple structure and high density. Crossbar was firstly initiated and demonstrated in a telecommunication switching system, which contained two sets of wires and switches at cross points. Signal routing is controlled by properly selecting switches. In the nanometer-scale high-density memory design, the similar structure is maintained – a storage element is placed at each cross point of two sets of metal wires [30]. Generally, the two sets of wires are called wordlines and bitlines. Fig. 2 shows a 3X3 crossbar array. Blue wires and red wires are wordlines and bitlines, respectively. The storage devices are showed in yellow pillars at cross points. Theoretically, using crossbar array structure can achieve the smallest memory cell area $4F^2$, where F is the minimum feature size [24].

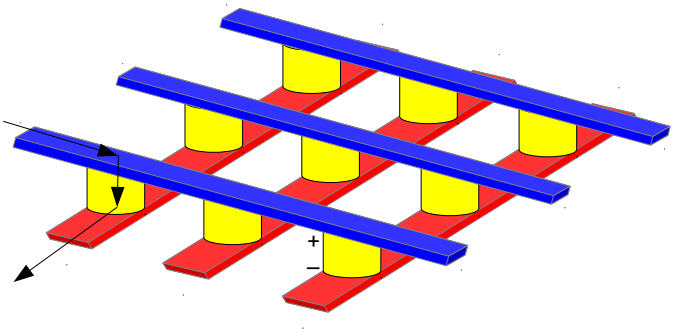


Figure 2: 3X3 crossbar array.

However, the crossbar array also results in sneak path in which three or more cells are connected in series as shown in Fig. 3. To guarantee the proper functionality in both write and read operations, the voltage/current across the selected memory cell must be much higher than the overall amount of current absorbed by the unselected cells [24]. On the

other hand, the voltage across an unselected cell must be smaller than the threshold of the SET/RESET operation to avoid the unwanted resistance change. To control the impacts of sneak paths within an acceptable range, the size and hence the capacity of a crossbar array is limited.

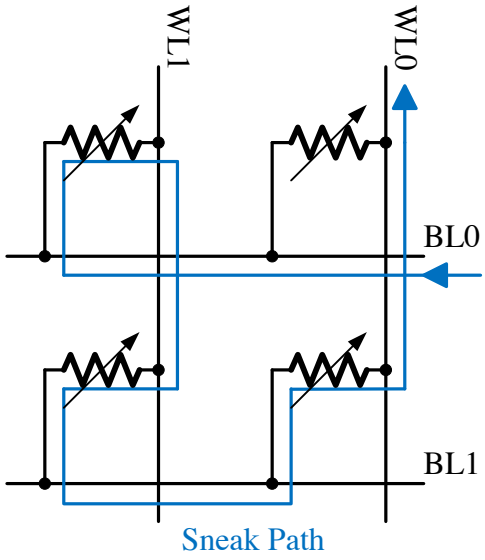


Figure 3: ReRAM crossbar array and the sneak path.

2.1.3 Complementary ReRAM

Recently, E. Linn, *et al.* proposed a complementary ReRAM cell structure, which is made of two anti-serial ReRAM devices as illustrated in Fig. 1(b) [20]. Under all the possible operation conditions, at least one of the two ReRAM devices in this complementary cell exhibits the HRS state, which can dramatically reduce the impact of sneak paths. However, any single data recording has to be associated with a multi-step write procedure which requires a careful and complex operation configuration. This design also brings in severe issues in terms of the high power consumption and the degraded device reliability. Moreover, considering that each memory cell includes two complementary ReRAM devices, the memory capacity is only half of a conventional 3D ReRAM design.

2.1.4 Stacking of ReRAM

Simply stacking multiple memory layers vertically is a common way to construct 3D design with bipolar ReRAM devices [31]. Fig. 4 illustrates the structure. Each memory layer has its own set of storage elements and interconnects. An isolation layer is inserted between two neighboring layers to prevent the signal interference. Recently, an improved design was proposed by Kugeler *et al.*, which the *word lines* (WLs) between two memory layers can be shared [32]. The two memory layers sharing the same WLs can be accessed and programmed simultaneously. However, *bit lines* (BLs) cannot be shared, and hence, the manufacturing of isolation layers are still needed.

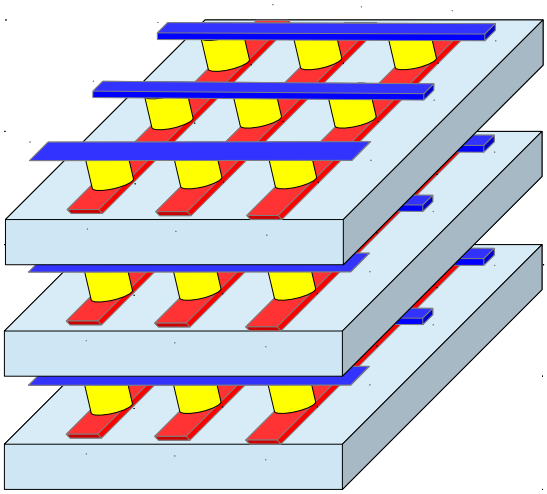


Figure 4: Stacking crossbar array.

SOG with MSQ *etc.* materials can be used to form isolation layers. However, there are some critical difficulties from a process development point of view, including device degradation due to thermal processing [33], misalignment of vias due to SOG [34], poor adhesion of SOG material [18], and heat accumulation because of the low conductivity of the isolation material [33][35]. Consequently, a 3D memory design excluding isolation process is preferred for lower fabrication cost and process complexity. Previously, Jonson *et al.* presented a bipolar multi-layered conductive metal oxide memory without isolation layer, but it can be applied only to one-time programming ROM applications [36].

2.2 NOVEL FPGA COMPONENTS

With emerging non-volatile memory technologies, such as *phase change random access memory* (PCM), *spin-transfer torque random access memory* (STT-RAM) [37] [38], ReRAM [26], *etc.,* configuration can be retained in emerging FPGAs without power supply.

New opportunities are provided to improve the reconfigurable system with nonvolatile memory. Different hybrid FPGA architectures were introduced by substituting computation and storage elements with PCM, STT-RAM, or ReRAM. J. Cong and B. Xiao introduced memristor devices into routing components design that usually are built with SRAM cells and multiplexers [39]. Memristor acts as a switch to connect and disconnect a route by low resistance and high resistance state of memristor, respectively. C.-Y. Wen, *et al.* proposed a PCM LUT to improve logic density compared to modern LUTs [40]. However, the large power consumption in PCM programming cannot be avoided. Meanwhile, H. Yan, *et al.* propose a unconventional LUT which is a programmable nanowire circuit [41] similar to nanoFPGA/nanoPLA [42]. Main structure of nanoFPGA/nanoPLA is a mesh structure. It contains programmable logic place and restore plane. A nanoFPGA/nanoPLA has back-to-back *NOR* logic planes to realize any logic function as conventional *AND-OR programmable logic array* (PLA). A *nanowire field effect transistor* (NWFET) is introduce at each cross point to restore the signal in the restore plane. Inputs traverse though the nanoPLA/nanoFPGA between programmable logic and restore plane to generate corresponding logic at output. It acts like a conventional LUT and is capable multi-layer stacking to increase logic density. However, fabrication problems of NEWFET have not yet been well addressed.

Y. Chen, *et al.* demonstrated 3D architecture for FPGA with MLC PCM [43]. The work has separated layers for routing and logic elements. TSV is applied for connections between the two layers. However, MLC PCM needs extra logic for controlling programming and sensing of the cell. On the other hand, dense logic requiring high density TSV and heavy traffic incurring huge power consumption on TSV are not well addressed in the work. S. Paul, *et al.* illustrated STTRAM FPGA with novel components' design [44]. STTRAM LUT has a decoder to share sense amplifier for saving area cost. Switch of CB and SB is composed of complementary STTRAM cells and a NMOS. Design difficulty of STTRAM

in FPGA is small sensing margin. Meanwhile, area saving with STTRAM is not obvious since the STTRAM still needs NMOS under each cell as well as complex peripheral circuit of STTRAM. S. Tanachutiwat, *et al.* demonstrated ReRAM FPGA with novel 2T2R and 2T1R switch as basic components for logic and routing elements [45]. They further estimated die to die stacking for increasing density of ReRAM FPGA. However, the ReRAM FPGA of a simple architecture is hard to compare performance, area, and power with modern sophisticated SRAM FPGA. Furthermore, 2T2R or 2T1R for all elements still has large CMOS portion indicating less efficiency of reducing area cost with ReRAM. Y. Y. Liauw, *et al.* showed a device level 3D stacking ReRAM FPGA in fabrication. The components are based on the 1T2R switch [46]. The work demonstrated possibility of stacking ReRAM FPGA with distinct reduction of power and area. It consisted of basic and important design concepts of 3D ReRAM FPGA. W. Zhang, *et al.* developed a nano/CMOS hybrid dynamically reconfigurable architecture based on nano memory and CMOS logic that supports the runtime reconfiguration and logic folding [47]. However, SRAM cells are still needed in the architecture limiting design capacity.

Our work further reduces area cost of ReRAM FPGA from other works. We introduce a new LUT design and the ReRAM switch with a sharing peripheral circuit. A high density ReRAM block memory is also introduced to accelerate computation by reducing IO access to external memory.

2.3 PLACE AND ROUTE FOR FPGA

Commercial place and route tools for FPGA are based on homogeneous units to map the logic functions [48][49]. VTR [16], as an academic tool, gives a complete flow from *hardware description language* (HDL) to physical mapping on FPGAs of various hardware architectures, where VPR is used inside to perform the tile-based placement and routing.

Ideas of PR was introduced and evaluated on reconfigurable computing system by Compton and Hauck [50]. Vendors of modern FPGA provide CAD tools for supporting PR applications [11, 12]. Most of the tools were designed for specific products provided by vendors.

Users had to follow the tools' flow and manually input information, such as floorplan, contexts, *etc.*, requiring knowledge of FPGA's hardware for designing PR applications. Xilinx *Early-Access* (EA) PR design flow [11] [51] [52] is commonly used in PR applications. It requires that PR regions are manually defined in terms of shape, size, and physical location. Hence, it needs extensive knowledge on the PR design flow and low-level architecture from the designer to produce a good quality design. Xilinx PlanAhead [51] is the tool behind the EA flow creating floorplan for a PR design with users' specification of PR regions' information and allocation. In order to reduce the manual efforts, various works introduced with automatic floorplanning for modules. In an earlier work [53], each PR module is modeled as a fixed-size block and floorplanning of a PR logic is formulated as a 3D template placement problem. However, the assumption is hard to be applied in practical applications. Later works [54, 55] developed an automatic flow for PR floorplan based on Xilinx process and special bus is needed to connect modules and support PR. S. Yousuf, *et al.* [54] introduced DAPR, a partial reconfiguration design flow which automates Xilinx intricate design process for PR designs. The work of C. Beckhoff, *et al.* [55] focused more on interface generation for PR after floorplanning. Other than the flows provided by white papers and tutorials from vendors, D. Koch further introduced optimization of the flow for decreasing designers' effort [56]. However, the techniques were still based on the vendors' tools for specific products. A broad survey of PR operation and performance analysis with various techniques on commercial tools were summarized by K. Papadimitriou, *et al.* [57].

Recently, researchers introduced new tools and flows of PR logic based on a academic tool, VPR [16], to explore future architectures and to map PR applications. Generally, the works were from two aspects, PR placer and PR router. R. He *et al.* introduced a new placer for PR applications on VPR [52]. The logic was decomposed into modules by designers and automatically optimized placement in terms of size, shape, and location for all modules, including non-PR and PR modules. It costed huge effort for searching solution space and needs a better flow for reducing the space. For example, size of each module could be estimated by synthesis rather than run-time calculation. B. A. Farisi, *et al.* introduced a novel router with an extended routing resource graph [58]. Switch nodes were inserted in the graph to identify dynamic or static routes. By applying the graph and the corresponding cost

function, the router generates connections of PR logic. On the other hand, N. Shah and J. Rose illustrated ping-to-wire and its difficulty of routing for PR applications on FPGA [59]. Routes through abutments around PR modules had various scenarios of routing resource, which results difference of critical oath delay, number of tracks, *etc.*

3.0 HIGH DENSITY RERAM

3.1 3D HIGH-DENSITY INTERLEAVED MEMORY

3.1.1 The Proposed 3D-HIM Structure

Figure 5 illustrates the proposed 3D-HIM structure. For simplicity, only six memory layers are demonstrated. A crossbar array is utilized in each layer. The basic design concept of 3D-HIM is to employ complementary material stack structures, *i.e.*, the regular memory stack and the one with a reversed deposition order, to the memory cells in neighboring layers. For instance, all the memory cells of Layer 1 in Figure 5 use the regular deposition process (purple pillars), and those of Layer 2 are made by reversing the deposition sequence (yellow pillars). The two types of memory stacks are applied to the odd and even layers alternatively. This process has been successfully demonstrated by Linn *et al.* and the memory cells made with regular and reversed depositions present the same device properties [20].

In the proposed design, memory devices and metal wires form a memory island without isolation layers. Any two adjacent memory cells at the same $x - y$ location are connected back to back, and hence, share the metal wire in between.

Some terms are defined to help understand the proposed structure and corresponding operations.

- Bitlines (BLs): A set of metal wires connected to TEs of ReRAM devices, which route along the $y-$axis as shown in Figure 5.
- Wordlines (WLs): A set of metal wires connected to BEs of ReRAM devices, which route along the $x-$axis. There are two sets of WLs, names as WL1 and WL2.
- WL1s and WL2s: We number the WL layer at the bottom of the 3D-HIM structure as

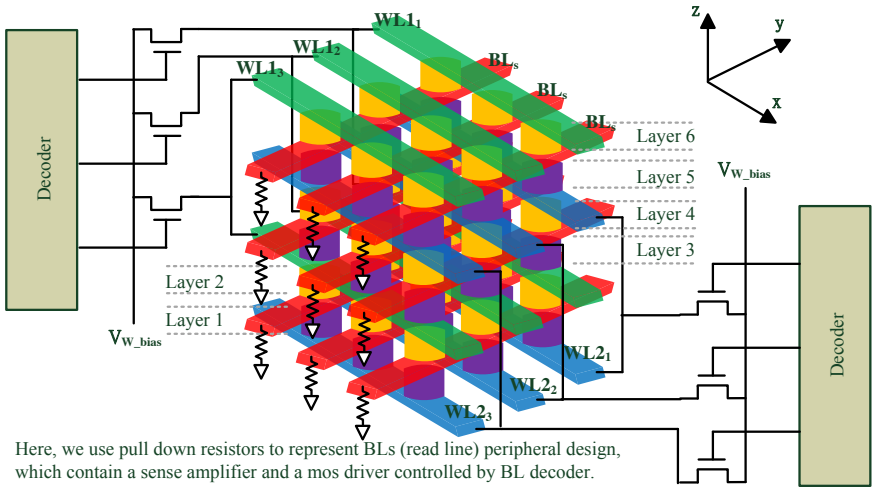


Figure 5: 3D-HIM structure.

'0' and continue counting the other WL layers from bottom to top. We define WL1s (WL2s) as those WL layers with odd (even) numbers.

- WL1$_i$GC and WL2$_j$GC: we name the group of memory cells connected to a given WL1$_i$ or WL2$_j$ as WL1$_i$GC (WL1$_i$ group cells) or WL2$_j$GC (WL2$_j$ group cells), respectively.

Totally, three sets of control signals, *i.e.*, BL, WL1 and WL2, are utilized. Each of them is responsible to the related operations to the memory layers above and below it.

3.1.2 Memory Density Improvement

Figure 6 illustrates the layout of a 3D-HIM from top view. The cell area is $A_{3D-HIM} = 4F^2$, which is the same as the cell size of the conventional crossbar array ($A_{conv} = 4F^2$). Note that for a 3D memory, its density is determined not only by the single memory cell area, but also by the the allowable number of memory layers. By sharing BEs and TEs between neighboring layers, 3D-HIM can reduce the overall number of conduction layers and remove isolation layers. For a given height of a 3D structure, which usually is a major limitation in fabrication process, more memory layers can be stacked up vertically. Thus, the memory capacity increases.

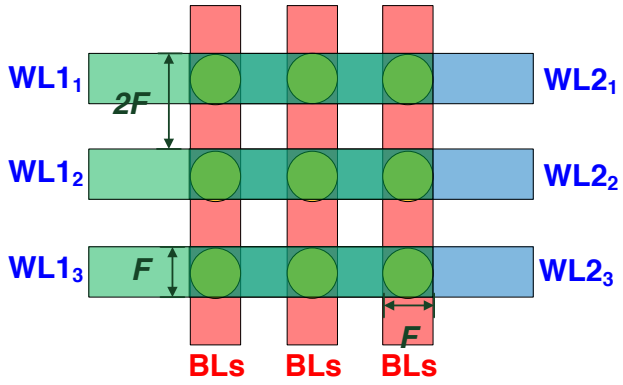


Figure 6: Layout of 3D-HIM design.

3.1.3 Memory Accesses in 3D-HIM

3.1.3.1 Bi-Group Operation Scheme In 3D-HIM, there are two sets of group cells – WL1$_i$GC and WL2$_j$GC. Only one of them can be accessed at once during read or write operations. We called it as "Bi-Group Operation Scheme."

This scheme has several advantages: (1) It increases throughput by simultaneously accessing multiple memory cells within either WL1$_i$GC or WL2$_j$GC. (2) The unselected groups can be biased to ground and taken as the signal isolators. Thus, we can avoid the unexpected overwriting caused by the write operations on different memory layers. (3) The BLs are shared by the ReRAM layers above the BLs, and below BLs. The peripheral circuitry connected to the BLs are also shared by two ReRAM layers to reduce area cost. Furthermore, WL1 and WL2 can be driven from the opposite sides of the memory island as shown in Fig. 5 to distribute the layouts of peripheral circuitry.

3.1.3.2 Read Operation To read out the stored data in a ReRAM cell, we provide a sense voltage (V_{sense}) to the corresponding WL, and measure the current through the cell. To prevent the unexpected overwriting, V_{sense} should be much smaller than the threshold voltage of ReRAM device. A sense amplifier is connected to the BL and shared by two group's cells WL1$_i$GC and WL2$_j$GC. Based on the Bi-Group Operation Scheme, only one group's cells can be sensed out at one time.

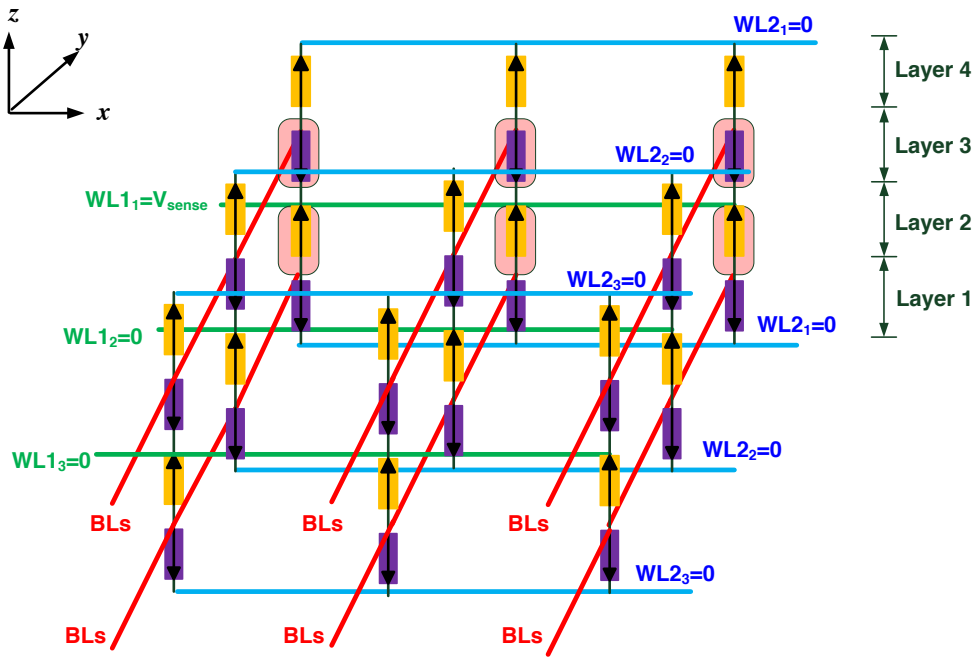


Figure 7: Selected WL_1GC in read operation.

Fig. 7 shows an example of reading out the cells in $WL1_1GC$. Accordingly, $WL1_1$ is raised to V_{sense} and all the other $WL1_i$ are tied to 0 V. To prevent the disturbance from/to WL2 groups, all the WL2s are forced to 0 V. Similarly, the read operation of $WL2_jGC$ on the $x-y$ plane can be accessed simultaneously. It sends a V_{sense} to selected wordline of WL2 groups while tying all wordlines of WL1 and rest of the WL2 to ground.

An active load (R_{sense}) is used at the end of BL to transfer current through the memory device to the input voltage of a sense amplifier $V_{R-sense}$. To simplify the evaluation of the read operation in this work, we apply a 100 Ω resistance (R_{sense}) as the input resistance of sense amplifier, and define the sensing margin (SM) by normalizing $V_{R-sense}$ with V_{sense}.

Assume that a 3D-HIM memory island has H memory layers, and each of them includes a $N \times N$ array. The capacity of such a memory island is $MC = N^2 \cdot H$ and the read bandwidth is $BW_{Read} = N \cdot H/2$.

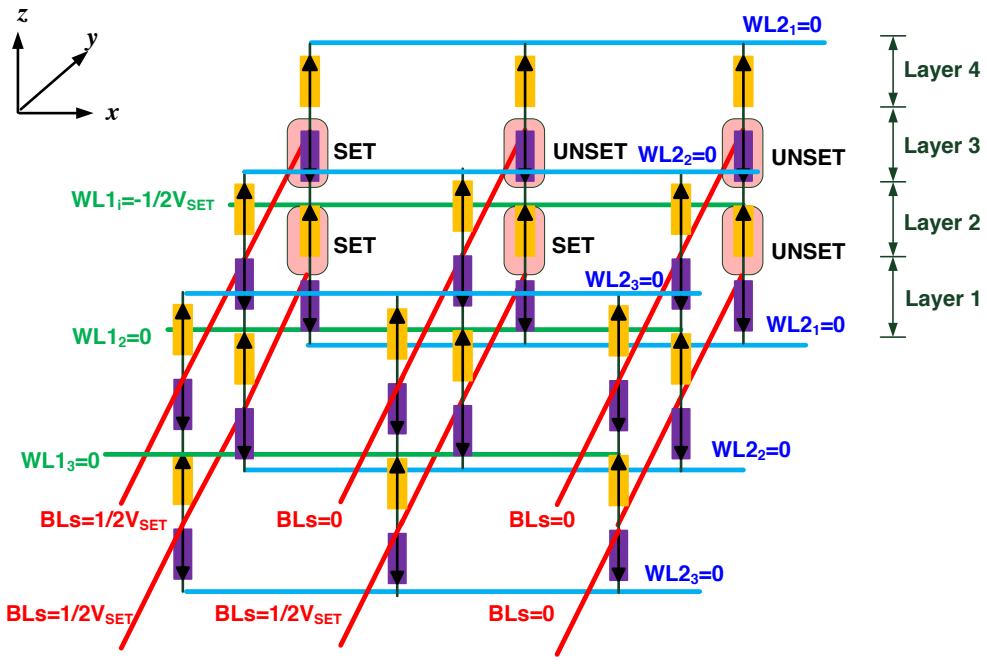


Figure 8: Selected WL_1GC in SET operation.

3.1.3.3 Write Operation As we state above, Bi-Group Operation Scheme needs to be used in write operation to increase throughput and prevent unexpected overwriting. There are two possible write procedures – SET and RESET. Like all the bipolar ReRAM crossbar design, these two procedures have to be separated because they require the opposite driving polarities. In 3D-HIM, the cells that programmed at the same time must locate in the same group and have the same incoming value.

The driving conditions need to be carefully controlled to avoid unexpected overwriting caused by sneak paths and to minimize the total write current. It has to maintain programming voltage across non-targeted cell smaller than threshold voltage of SET and RESET. The ideal bias voltages when performing SET and RESET are summarized in Table 1 [60]. All the other WL1s, WL2s and BLs that are not related to the writing operation are forced to 0 V.

Table 1: Driving Conditions of Writing Operations

Data	Cell Group	Driving Conditions
LRS	$WL1_i GC$	WL1:$-0.5V_{SET}$, BL:$0.5V_{SET}$
LRS	$WL2_j GC$	WL2:$-0.5V_{SET}$, BL:$0.5V_{SET}$
HRS	$WL1_i GC$	WL1:$-0.5V_{RESET}$, BL:$0.5V_{RESET}$
HRS	$WL2_j GC$	WL2:$-0.5V_{RESET}$, BL:$0.5V_{RESET}$

Fig. 8 illustrates an example of $WL1_1GC$ during a SET operation. For illustration purpose, we assume half of the cells in $WL1_1GC$ are in the SET procedure. $WL1_1$ are forced to $-0.5V_{SET}$, the BLs connected to the cells to be programmed are forced to $0.5V_{SET}$, and the unrelated control signals are set to 0 V. As shown in Fig. 8, a unselected cell within $WL1_1GC$ have only $0.5V_{SET}$ voltage drop across the cell, which is not big enough to change its content. The RESET procedure is similar to the SET in the example. The corresponding bias voltages are summaries in Table 1 [60]. To program $WL2_jGC$ on the $x-y$ plane, it has the similar setup requirement. It biases the selected cells corresponding $WL2_j$ to $-0.5V_{SET}$ and corresponding BL_s to $0.5V_{SET}$.

The average write bandwidth of 3D-HIM is $BW = N \cdot H/4$, while the maximal write bandwidth could be $N \cdot H/2$ when all the cells in the given group are programmed to the same content.

3.2 SIMULATION RESULT & DISCUSSION

We did simulations for the proposed 3D-HIM structure by using Spectre on Cadence CAD platform. The characteristic parameters of ReRAM are summarized in Table 2 [61]. To be more realistic, we embedded interconnect resistance (IR) in the simulation model. The existence of IR on control signals can result in voltage drop and decreases the real driving voltage delivered to the target memory cells. Based on the DRAM interconnect requirement

at $22nm$ technology node from ITRS 2009, we set IR per memory cell $R_{Interconnect} = 2.5\Omega$ [62]. Considering the limitations of back-end process, we assume up to eight memory layers can be stacked up in 3D-HIM.

Table 2: Parameters of ReRAM and 3D-HIM

Parameters	Value	Parameters	Value
V_{SET}	1.5 V	$R_{Interconnect}$	2.5 Ω
V_{RESET}	-1 V	V_{sense}	0.1 V
R_{off}(HRS)	1 MΩ	R_{sense}	100 Ω
R_{on}(LRS)	5 kΩ		

3.2.1 Impact of Data Pattern and Cell Location

3.2.1.1 Impact of Data Patterns The effectiveness of read and write operations in 3D-HIM depends on the memory data pattern. To investigate the impact of data pattern, we divide all the cells in a memory island into three catalogs: the target cell, the cells along the driving path (i.e. WL1 or WL2), and all the other cells. Figure 9 shows an example of the $WL1_1GC$ in the sensing operation: the target cell highlighted in PURPLE, the cells along the driving path (WL1) highlighted in BLUE, and all the other cells not highlighted. Four data patterns can be introduced – "LL", "LH", "HH" and "HL". Here, the first letter stands for the status of the target cell ('L'=LRS, 'H'=HRS), and the second letter stands for the cells along driving path, either WL1s or WL2s.

Our work shows that the cells along the driving path and the rest cells dominate the SM rather than the target cell. The worst case happens when the driving path cells and the rest cells are all at LRS. In such situation, the conducted current from the sneak paths and leakage current are maximized. The corresponding data pattern are "LL" or "HL". The impact of data patterns to the SM in our design is further discussed in the next section.

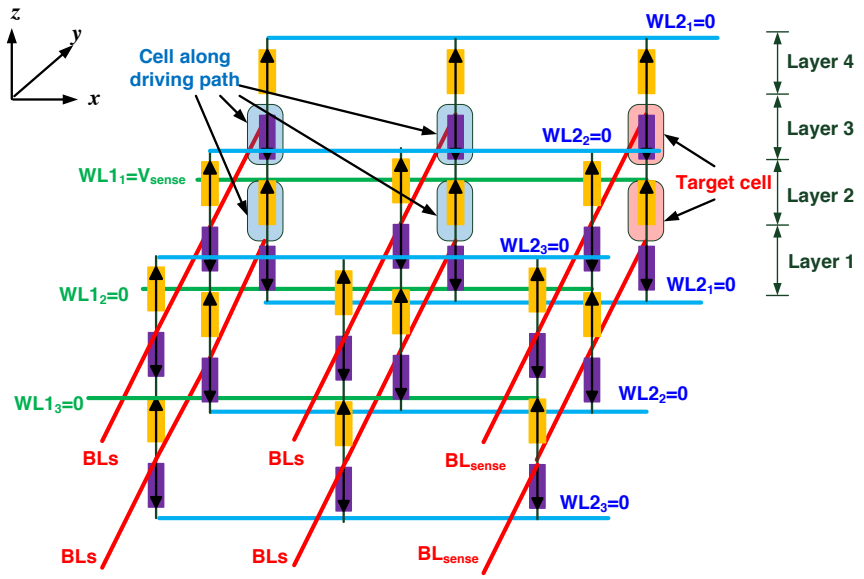


Figure 9: Three catalogs of cells used for data patterns analysis.

3.2.1.2 Impact of Cell Location The physical location of a cell could affect its operation scenario. For example, Fig. 10 illustrate a two-layer 3D-HIM in read operation of $WL1_1GC$. The driving current flows from the leftmost side of $WL1_1$ to the rightmost of the array along the x-axis. Due to interconnect resistance, the real voltages applied on the cells along $WL1_1$ are not same. The worst case scenario happens at the cell in the right corner (highlighted in RED) because it goes through the longest path from the driver to the sense circuitry. In the contrast, the cell located in the left corner (highlighted in BLUE) is affected least by the interconnect resistance, and hence, becomes the best situation.

Fig. 11 shows the SM difference between the worst scenario and the best scenario of cell locations with different data patterns in a four-layer 3D-HIM. As shown in the results, the impact of array size to the 'LL' pattern is much larger than the other patterns: the location difference incurs more than 10% SM difference. This is because the target cell at LRS suffers from high interconnect resistance and the other cells on sneak path at LRS sink a big portion of currents. To ease impact of location difference, ReRAM of high R_{on} which suffers less impact of IR is promising for large scale array.

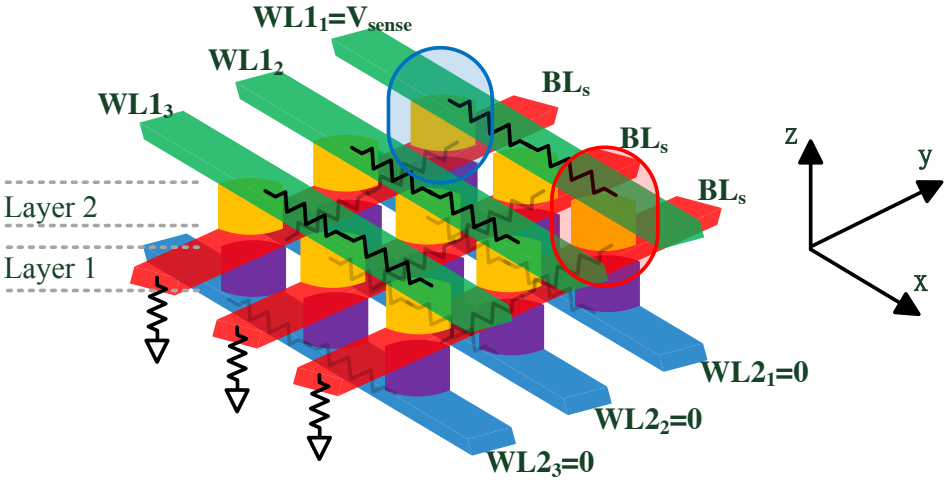


Figure 10: Worst case and best case scenario of the cell locations.

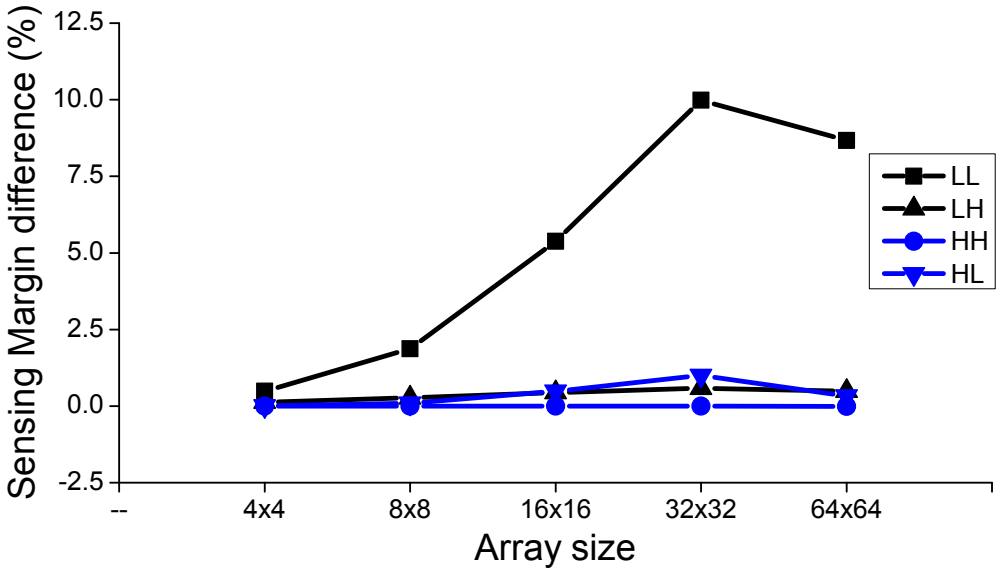


Figure 11: SM difference by the cell locations and data patterns

3.2.2 Read operation

Fig. 12 compares the SMs of conventional 3D ReRAM and 3D-HIM under different memory parameters. The worst case scenario of cell location and data pattern is assumed in the simulation.

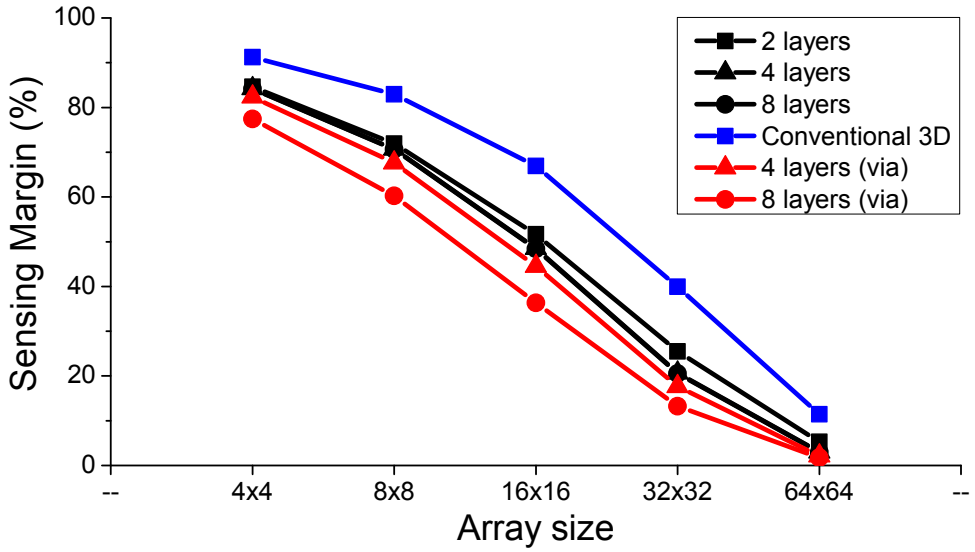


Figure 12: SM of 3D-HIM

The SMs of a conventional 3D ReRAM at different array sizes are shown in Fig. 12 by the BLUE curve. The BLACK curves demonstrate the SMs of a 3D-HIM with different layer numbers. The curves for 4-layers and 8-layers merged together. Compared to the conventional 3D ReRAM, 3D-HIM loses 10 $\sim$ 20% in SM. This is because that the conventional 3D ReRAM inserts a isolation layer between any two memory layers, hence, the SM is determined only by one crossbar array. The control signals (e.g., WLs) in 3D-HIM have to drive twice number of memory cells than the conventional design, which introduces more sneak paths. However, because of the interleaved design, stacking more layers in 3D-HIM only induces slight degradation on SM. The SM decreases obviously as the array size increases. For example, in a four-layer 32×32 3D-HIM, the SM is about 20%. When array size increases to 64×64, the SM significantly reduces to 3%, which means the status of memory cells are hard to be detected.

The small footprint of via could introduce a large resistance on the driving path, which incurs performance degradation in the upper layers. Based on ITRS 2009 – DRAM Interconnect technology requirements for $22nm$ technology [62], we approximate the via resistance as 14Ω. The RED lines in Fig. 12 shows the simulation results of 3D-HIM after including the impact of via resistance in small footprint.

Fig. 13 shows the composition of the sensing current under 'HL' pattern. As the array size increases, the percentage of the sneak path conducting current raises. In a four-layer 64×64 3D-HIM, the conducting current in the sneak path contributes 99% of the sensing current in 'HL' data pattern, which makes it hard to detect the correct memory status and reduces the SM significantly. To further increase the SM in 3D-HIM, we have to suppress sneak path current. By increasing R_{on} at LRS, the impact of sneak paths can be dramatically relieved. In Fig. 13, we compare the simulation results of R_{on}=10kΩ with the results of using the original value R_{on}=5kΩ. Increasing R_{on} to 10kΩ can eliminate 35% and 5% of the sneak path conducting current in a 16×16 and 64×64 array, respectively. Correspondingly, the sense margin of the 64×64 and 16×16 array improves to 11% and 20%, which increases the margin of the sense amplifier design of 3D-HIM. We can conclude the increasing sensing voltage of HL data pattern has significant impact to SM degradation. To suppress sneak path conducting current is a promising method to improve SM degradation.

3.2.3 Write operation

The cell location and data pattern also affect the write operations. The worst case happens at the same location and with the same data pattern as in the read operation. Due to space limitation, we only discuss the worst case scenario and follow the explanation for read operation.

Enlarging array size of 3D-HIM increases the total IR in a driving path. To compensate the impact of the increasing voltage drop on IR and properly program the target cells, a higher bias between WLs and BLs (V_{SET} or V_{RESET}) is required. The corresponding simulation for a four-layer 3D-HIM with various array sizes is shown in Fig. 14.

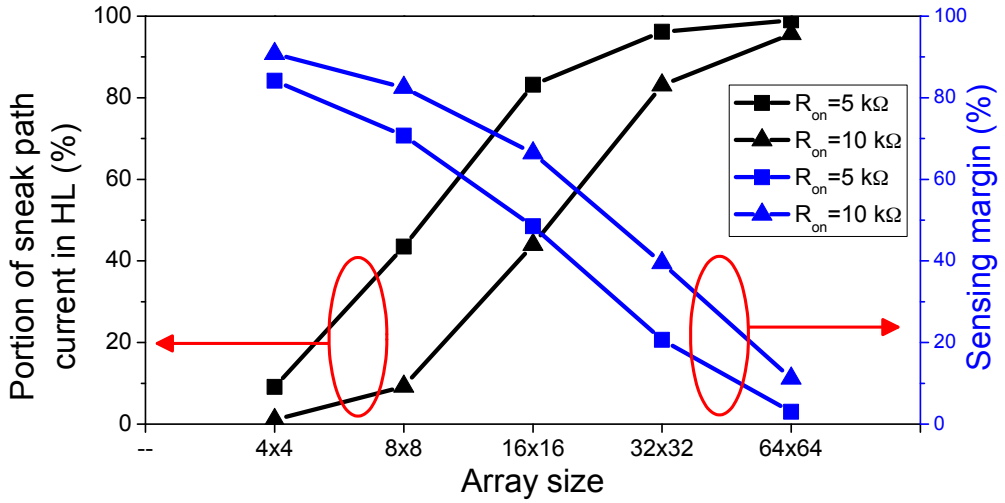


Figure 13: Portion of sneak path conducting current in sensing current through R_{sense} and SM of four-layer 3D-HIM with various R_{on}

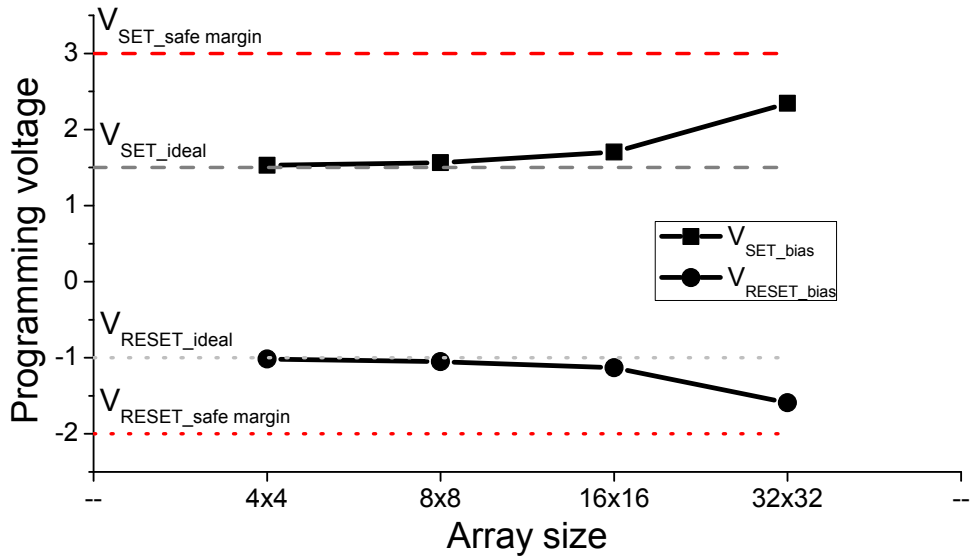


Figure 14: Proper programming voltage

The two dotted GRAY lines are the required SET and RESET voltages across a ReRAM cell, which are exactly $V_{SET-bias}$ and $V_{RESET-bias}$ in the ideal condition without IR. However, the impact of IR cannot be ignored in a real design and it results in the increase of programming voltages as array size increases as demonstrated by the BLACK curves. The dotted RED lines constrain the safe margins of programming voltages, which double the range of the GRAY curves. If V_{SET} or V_{RESET} exceeds the safe margins, some unselected cells may be overwritten since their voltage drop are higher than the threshold. As a result, the proper programming voltage (BLACK curves) and safe programming margins (RED lines) confine the array size. Our simulation shows that the maximal allowable array size of 3D-HIM is 32×32 to satisfy the constraints in write operations.

4.0 RERAM FPGA

4.1 RERAM LOOK-UP TABLE

4.1.1 Conventional LUT in FPGAs

A LUT is a memory-based logic element applied in modern FPGAs, which is usually constructed by memory cells to store logic configuration in boolean logic. As shown in Fig. 15(a), a *multiplexer e.g.*, 16:1, is demonstrated to access the target with inputs of the LUT acting as a address. Popular design of the LUT contains 6T SRAM cells, and pass gate based multiplexer to select configured logic in a reconfigurable system [15]. Each 6T SRAM cell works independently in the LUT with a buffer in the read scheme. SRAM cells of LUT occupy more than 50% of LUT's area. Based on the white paper of the Xilinx and the Altera, 6-input LUT is a trade-off design between delay and area of the LUT [63][64]. The LUTs in FPGAs are sequentially programmed every time when the chip powers up. This procedure is called as *initialization*. Conventionally, the LUT does not support local addressing for dynamic programming. The whole chip shares one SRAM programming circuit in the initialization. In modern FPGAs, vendors provide flexible configurations for LUTs. Some of LUTs can be configured as a *Distributed RAM* (D-RAM), which uses LUTs as a RAM to store temporary data. To realize the D-RAM function, address decoder and programming circuit for dynamically configuring SRAM cells are added in LUTs [65]. However, the extra circuits induce significant area overhead. Therefore, only a portion of a FPGA chip can adopt the dynamic configurable LUTs.

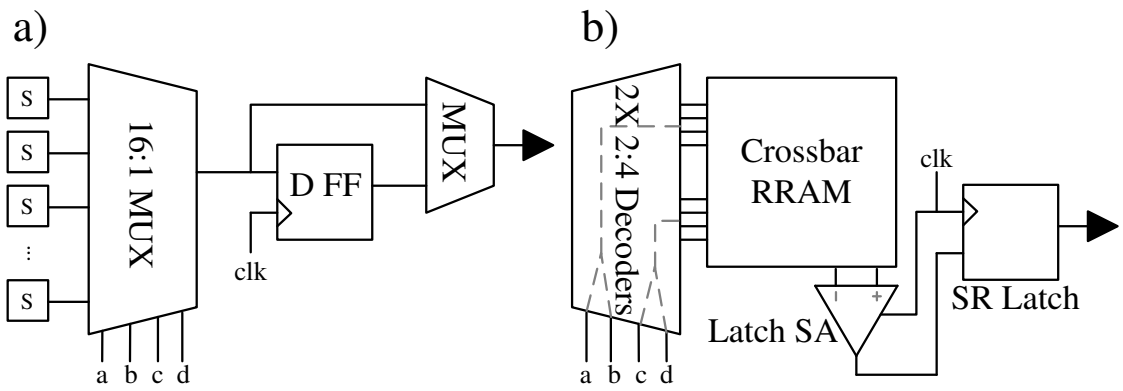


Figure 15: (a) A SRAM LUT. (b) The proposed ReRAM LUT.

4.1.2 Design Concept and Overview

Fig. 15(b) shows the proposed ReRAM LUT to substitute various LUTs in modern FPGAs. It composes of four main parts: memory cells, a decoder, a *sense amplifier* (SA), and a SR latch. Comparing designs in Fig. 15(a) and (b), main difference is that the proposed one has ReRAM cells rather than SRAM cells. Therefore, component for inputs as a address selection changes from a multiplexer to a decoder. Because the ReRAM needs a sense amplifier to sense the data in each cell, the design of the decoder helps all of the ReRAM cells share one sense amplifier for saving area. The proposed LUT has no *D flip-flop* (DFF), but, there is a combination of a latch comparator as sense amplifier and a SR latch as the same function as DFF. In the section, we would first introduce a 4-input LUT to explain the basic operation of the proposed LUT. Then, we extend the design to a 6-input LUT with the HIM structure mentioned in the previous section. Area cost of a ReRAM 6-input LUT in HIM structure does not linearly increase much area compared with a 4-input LUT as LUTs with SRAM cells. It is because the ReRAM cell is in monolithic stacking crossbar indicating that there is no area cost of memory cells on substrate. Only the peripheral circuit occupies area in the design. Therefore, it saves total area cost of a large LUT.

The proposed LUT has inputs, outputs, and control pins as introduced in the following:

- Logic block has six inputs (A-F) and two outputs ($O1$ and $O2$).

- *clk* is provided within CLB for timing control.
- *RE* and *WE* are read enable and write enable signals, respectively.
- *Din* is the data to be written into the memory. The location of the target cell is determined by *A*-*F*.
- *Vprg* is the required programming voltage of ReRAM memory. Its level is determined by device characteristics.
- *Vt* is supplied to the unselected cells in write operations to prevent undesired overwriting.
- *tbias* is connected to sense amplifier inside LUT to fine tune the performance at different operating frequencies.
- *vdd* and *gnd* represent power supply and ground, respectively.

The LUT has three operation modes: *read*, *write* (dynamic program), and *sleep*. For the read operation, it pulls the RE to high and pulls low the WE. The LUT acts as a conventional LUT to read out logic configuration from memory cells. The address is based on inputs to select the target cell as the output. In the write (dynamic program) operation, it pulls the RE low and pulls the WE to high. Since it is bit-addressable in the design, the proposed LUT can be used like D-RAM in the modern FPGA. D-RAM is an internal memory, which uses LUT to store temporary data. Inputs are the address bits and Din is the data input we attempt to store in the LUT. With the proposed LUT, all LUTs on a FPGA are dynamic programmable without adding extra cost. In modern SRAM FPGA, only half of LUTs can be used as D-RAM [66]. The sleep mode is a specific operation with the proposed ReRAM LUT by pulling the RE and the WE to low. We can cut off power supplying on idling LUTs. Once the resource is needed, it works immediately by turning on the LUTs.

Compared to a conventional SRAM LUT, our design has three advantages: (1) *"initialization free"*: since the ReRAM is nonvolatile, configured logic can be retained after powering off the system. Once the power is resumed, the system starts immediately. Thus, an external Flash memory for the initialization used in the conventional FPGA is not necessary. (2) *"deep-sleep mode"*: when a SRAM FPGA is confined by critical energy environment, *e.g.* mobile applications, it can switch unused LUTs into sleep mode for saving static power consumption. To prevent data loss on the SRAM cells, power supplies of those LUTs have to maintain at a certain level [67]. With advanced feature of ReRAM cells' nonvolatility, a

FPGA with the proposed LUTs can shut down unused LUTs completely and eliminate the corresponding static energy consumption. We name the operation to "deep-sleep mode". (3) *"high density"*: significant area reduction can be achieved by applying ReRAM cells to replace SRAM cells. So, a FPGA with the proposed ReRAM LUT has more LUTs than a SRAM FPGA.

Compared to the SRAM, programming the ReRAM is relatively slow. Very recently, Samsung reported $10ns$ device switching [68], which meets the prediction of ITRS 2011 [69]. However, the write performance in a FPGA is not that critical especially considering that the proposed ReRAM LUT supports bit-addressable programming and it does not need initialization when powering on. In this work, we focus on improving the speed of read operations. Our target operating frequency is 500MHz, which is compatible to modern FPGAs' operating frequency. The proposed peripheral circuit functions well up to 1GHz.

4.1.3 Read Scheme Design

To simply demonstrate the proposed sensing scheme, here we demonstrate a 4-input LUT which has one layer crossbar rather than four layers crossbar of a 6-input LUT. The 6-input LUT can be constructed in a similar method. We propose a expanded crossbar array with an extra column of reference cells to enhance sensing speed and noise margin. A corresponding sense amplifier is demonstrated to sense difference between data cells and reference cells.

4.1.3.1 ReRAM Crossbar Access Control Fig. 16 is a detailed of Fig. 15(b) showing that the proposed array access control scheme includes decoders, wordline drivers, and bitline selectors. In the the Fig. 16, a yellow block and a green block show wordline drivers and bitline selectors, respectively. A wordline driver can be constructed by two inverters. Transistors MP_{DR} and MN_{DR} are shared by all wordline drivers. The *read enable signal* (RE) is applied to turn on drivers or to force it into power save mode. A bitline selector is a NMOS transistor. It passes selected sensing current to a NMOS load.

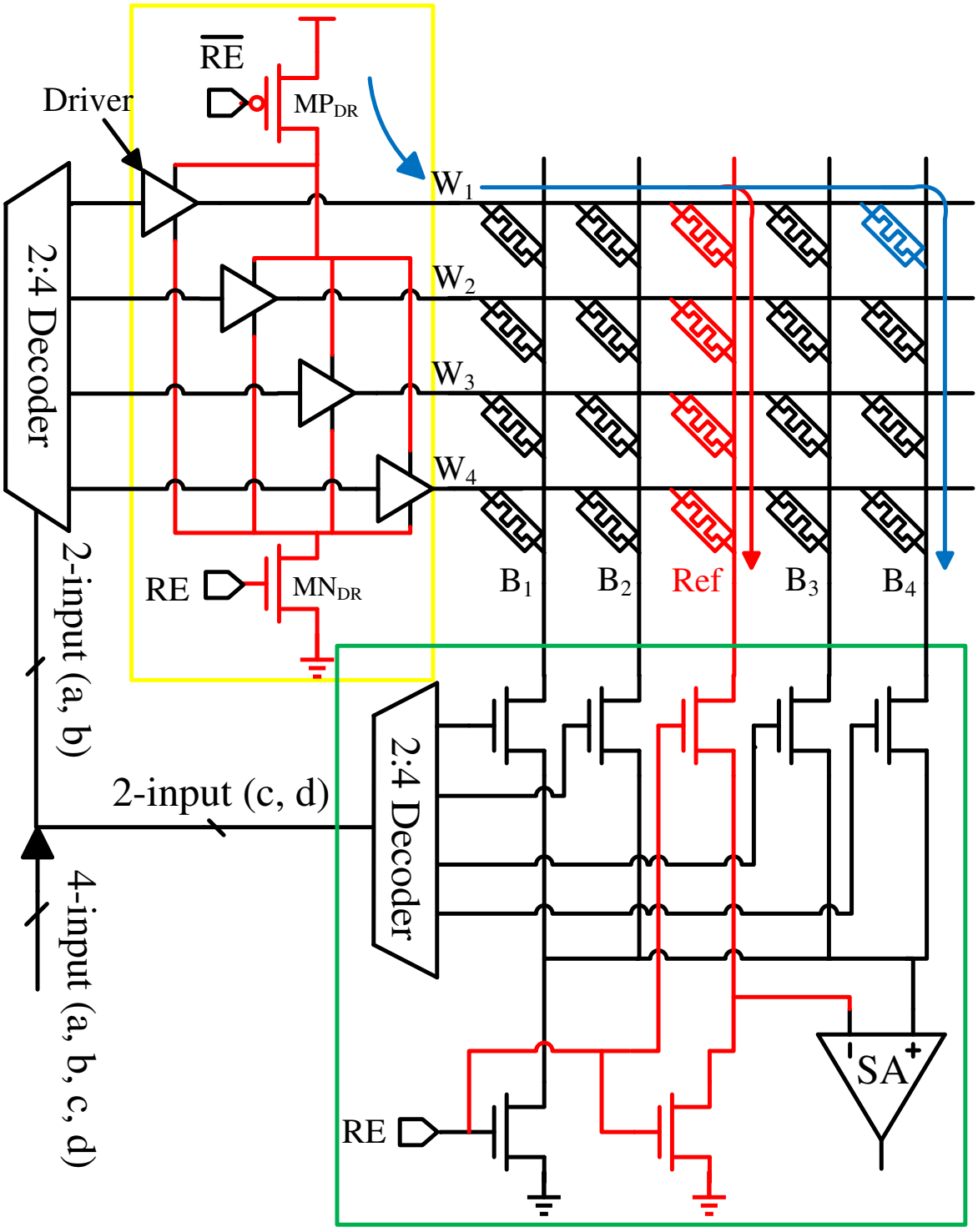


Figure 16: The proposed read scheme.

During a read operation, only a cell is accessed based on inputs of the LUT as an address. For example, to access the blue cell showed in Fig. 16, it has to access the wordline W_1 and the bitline B_4. The wordline decoder turns on the wordline driver at the W_1 based on the inputs of a and b. The selected wordline driver raises up to a sensing voltage. The unselected wordlines are forced to ground. Meanwhile, the bitline decoder turns on the bitline selector at the B_4 based on the inputs of c and d. The unselected bitlines are floating. Since only one bitline is activated, the sensing current concentrates along the driving path, highlighted in blue, and conducts a larger voltage across the NMOS load.

4.1.3.2 Expanded ReRAM Crossbar Structure Data patterns of a crossbar array have a great impact on sensing margin because of sneak path conducting current [70]. For example, when sensing data from the target cell R_{14}, highlighted in blue, in Fig. 16, corresponding equivalent resistance between wordline W_1 and bitline B_4 is defined as $R_{14,eq}$. Lines of hollowed symbols in Fig. 17(a), whose labels start with "D", represent $R_{14,eq}$ under different data patterns. The second letter in the label is resistance state of R_{14} – the LRS (L) or the HRS (H). The third letter represents resistance state of the other cells along the driving path, highlighted in the blue path of Fig. 16, are all in L or H. $X-axis$ is numbers of memory cells in LRS in the crossbar excluding the target data cell and the cells along the driving path. We name these cells are rest cells.

For data patterns of *DHH* and *DLL*, there is always a gap of equivalent resistance. It indicates no matter how many cells of LRS among the rest cells, we still has a margin to find a reference resistance for sensing HRS or LRS of the target cell. *DLH* or *DHL* represent content of the target cell in HRS or LRS, respectively. Considering data patterns of *DHL* and *DLH* in Fig. 17(a), there is a cross between the two lines stating that equivalent resistance might be the same under these two data patterns. So, there is no resistance as reference to differentiates *DLH* and *DHL* under various numbers of the rest cells in LRS. In other words, even if we know the equivalent resistance of the target cell, there is no way to determine whether the target cell's state is LRS or HRS.

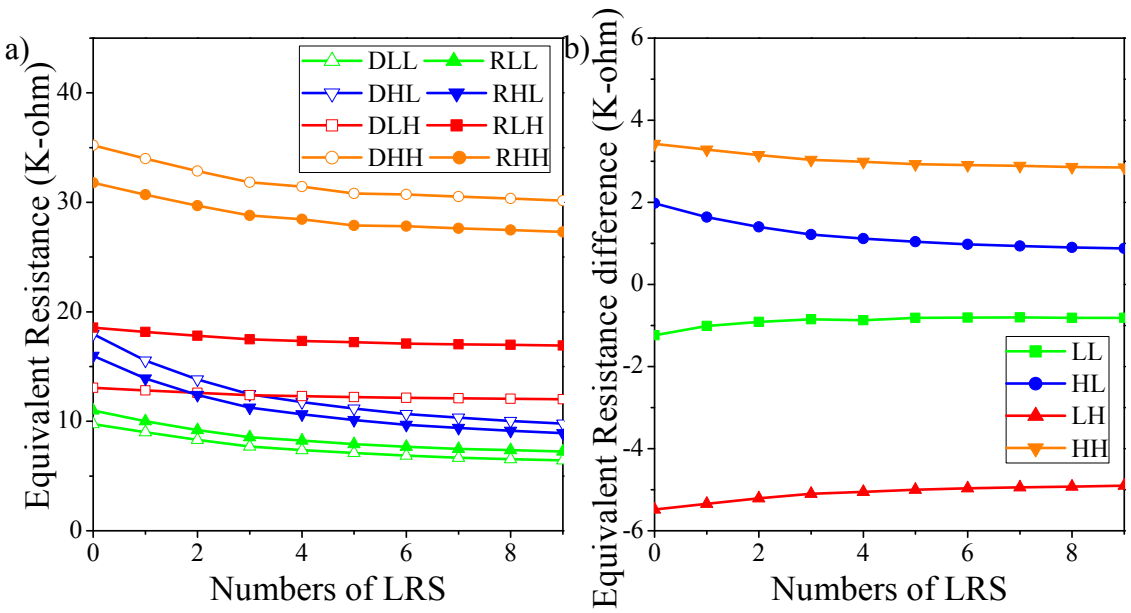


Figure 17: The impact of data pattern on (a) $R_{14,eq}$ and $R_{ref1,eq}$, and (b) $\Delta R = R_{14,eq} - R_{ref1,eq}$.

To solved the problem illustrated in the previous paragraph, some previous designs have a specific current or voltage as reference to detect the stored data [71]. The extra circuitry requires complicated control and results in high area cost. Therefore, it is not a proper design for the LUT.

We propose a expanded crossbar array with a reference cell as shown in Fig. 16. Resistance of reference cells should be between LRS and HRS of data cells to be applied as reference for differentiating states of the data cell. In this work, the resistance of reference cells are set to $R_H/2$. Based on a characteristics of ReRAM cells, resistance of a ReRAM cell is inversely linear proportion to footprint [72][73][74]. Thus, reference cells can be fabricated by doubling the footprint of the data cell and set to HRS.

For a read operation, the driving current from the same wordline driver flows through the data cell and the reference cell to loads as an example showed in Fig. 16. The current flows through reference cell is highlighted in red. We also shows equivalent resistance of the reference cell under various data patterns in the solid symbols in Fig. 17(a). Labels "R**"

represent $R_{ref1,eq}$. The second and the third labels have the same definition as the data cell's. Simulation results show that $R_{ref1,eq}$ follows similar trends as $R_{14,eq}$, that is, data patterns have similar impact on both the target cell and its corresponding reference cell. There is always a gap and no cross between the data cell and the reference cell under any data patterns. So, we can always sense a resistance difference between the two cells.

Fig. 17(b) shows $\Delta R = R_{14,eq} - R_{ref1,eq}$ under different data patterns. The first letter of labels represents resistance state of the target cell. The second letter is resistance state of cells along the driving path. $X - axis$ indicates numbers of the rest cells in LRS. When the target cell is at HRS (LRS), ΔR is always positive (negative) and maintains enough difference for sensing scheme.

4.1.3.3 Sense Amplifier Fig. 18 shows the proposed sense amplifier scheme with three main parts: a latch comparator [75], a nose cancellation, and a SR latch. The following section introduces functions of these three parts.

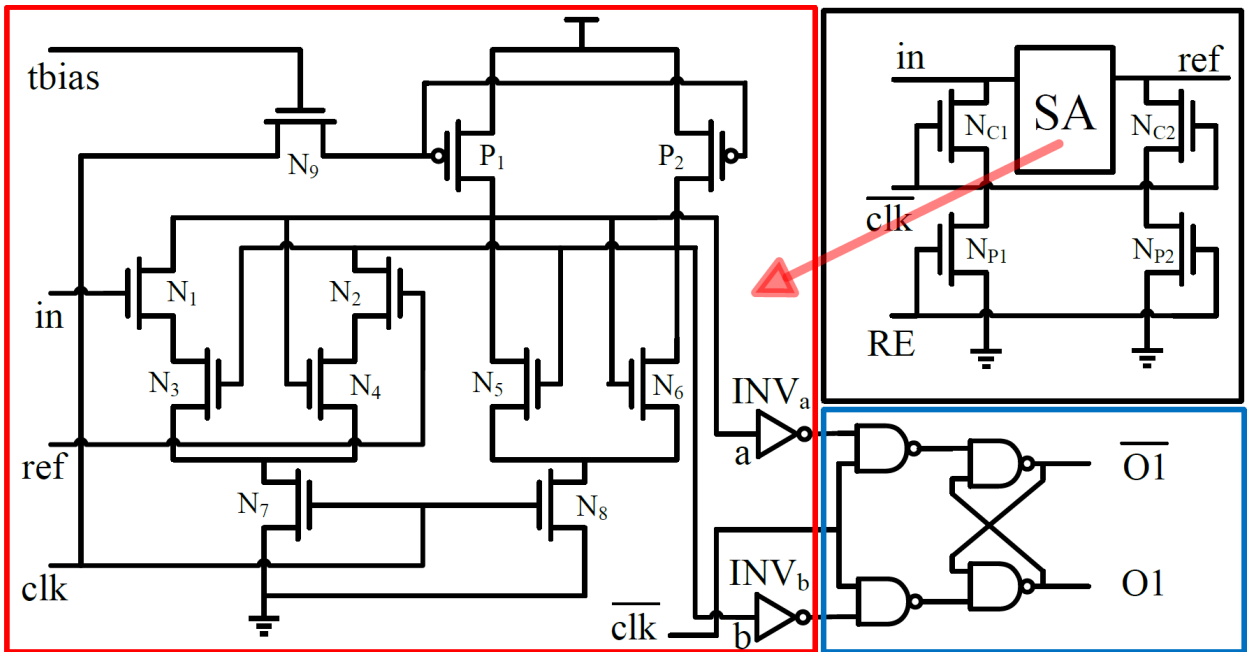


Figure 18: The proposed sense amplifier in ReRAM LUT.

Voltages across the loads of the data and the reference cells' sensing paths generate a voltage difference reflecting the ΔR. We apply a latch comparator to sense voltage difference at loads. The detailed schematic of the latch comparator is shown in the red block of Fig. 18. The latch comparator has sensing and non-sensing phases in the operation. NMOS transistors, N_7 and N_8, are in charge of phase control: when clock signal, CLK, is '1', the circuit is in the sensing phase; otherwise, the latch comparator enters the non-sensing phase.

In the sensing phase, N_1 and N_2 capture two input signals, *i.e.*, in and ref generated by the data and the reference cells, respectively. Decreasing widths of the N_1 and the N_2 result in smaller parasitic capacitance and fast operating speed. N_3, N_4, N_5, and N_6 together form a feedback loop. PMOS transistors P_1 and P_2 are active loads to amplify latched signals. External voltage bias V_{tbias} controls the resistance of N_9 and hence tunes active loads – P_1 and P_2. We can optimize sensing latency by controlling V_{tbias}. Although the latch comparator generate rail-to-rail output signals, we add two inverters INV_A and INV_B at complementary outputs to reduce sensing latency, improve rising/falling slope and increase the driving ability.

Noise interference has significant impact on the latch comparator's operation. Charge capacitance in the sensing phase may induce kick-back noise in the non-sensing phase. Thus, we built a noise cancellation circuit surrounding inputs of the latch comparator. N_{C1} and N_{C2} are added to cancel the noise. In the non-sensing phase, we turn on both N_{C1} and N_{C2} to tie in and ref to ground. So, the kick-back noise from the charge capacitance of N_1 and N_2 are also ground. We also introduce a power save mode by deactivating N_{P1} and N_{P2} with RE.

A modern LUT usually has a DFF for output synchronization. Since the proposed sense amplifier includes a latch comparator, we can simply utilize a SR latch at outputs of the latch comparator to realize the DFF. The SR latch is controlled by an inverted clock $\overline{clk}$. This design can balance capacitive loads at the outputs of the latch comparator and reduce layout area compared to the DFF.

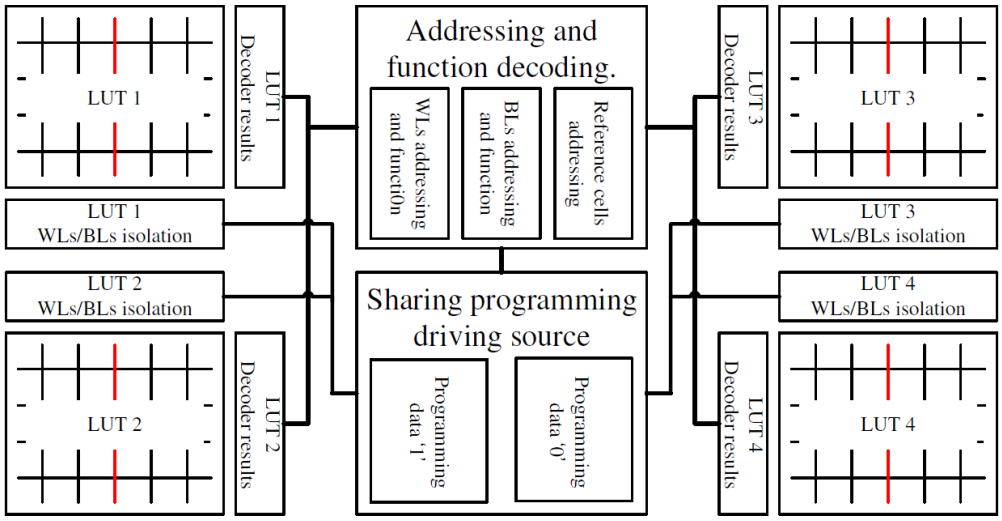


Figure 19: Write scheme of 6-input LUT. The function decoding and driving source are shared by four LUTs within one LB. The thick lines represent multiple interconnection signals.

4.1.4 Write Scheme Design

Fig. 19 shows a conceptual diagram of the proposed write scheme. It contains local address decoders in each LUT and a shared driving source by four LUTs. The proposed LUT has decoders to access ReRAM cells in the read operation. In the write scheme, these decoders address the target cell and trigger programming drivers based on input of data, logic '0' or '1'. ReRAM cells are integrated in the 3D-HIM structure which is based on a crossbar array without selecting devices. To have a better driving ability and low area cost, the proposed LUT can only program one bit in each program cycle. Thus, there is an isolation at every wordlines and bitlines. The target cell related wordline and bitline would be connected to programming drivers while programming. Meanwhile, it provides a bias Vt to unselected *wordlines* (WLs) and *bitlines* (BLs). So, voltage and current across nontarget cells are not large enough to program cell. The accident fault programming on nontarget cells incurred by programming bias is known as unexpected overwriting.

To program a ReRAM cell, we provide a forward and a reverse currents from wordline to bitline for programming logic '0' and '1', respectively. Since the ReRAM cell is integrated in a crossbar, numbers and data patterns of all the other ReRAM cells in the crossbar may influence the driving current. Because of leakage and sneak path current incurred by the other ReRAM cells, large transistors are required in a write circuit to provide proper voltage or current to program the target ReRAM cells. We propose a shared write scheme as shown in Fig. 19. The large transistors for programming are shared by four LUTs in one *logic block* (LB). The reason we have such a design is because write operation of a LUT is fewer than read operation. Sharing scheme may prolong write time, but it is an acceptable trade-off for the LUT. Although it needs extra controlling circuit, it still saves 33% area compared to a non-shared design in a 6-input LUT.

Fig. 20 shows the write scheme of one programming example. One crossbar array layer is shown in the figure for simplifying illustration. The 6-input LUT should has four layers crossbar array. All interconnections of the 3D-HIM should route to corresponding layers. There are three kinds of write circuits in the proposed write scheme: WLs' drivers, BLs' drivers for data cells, and BLs' drivers for reference cells. They only have slightly difference showed in a green block of Fig. 20. They all have transmission gates as isolation [60]. The isolation circuit needs large size of transistors because it guarantees a sufficient voltage and current supply across the ReRAM cell during a write operation. For WLs' and BLs' drivers, the drivers' source are different. To program logic '1' into a ReRAM cell, the WL and the BL drivers should be ground and V_{prg}, respectively. On the other hand, programming logic '0' needs the WL and the BL drivers to be V_{prg} and ground, respectively. Since we set reference cells HRS, we also program logic '0' to the reference cell shared the same WL of the target cell. Programming scheme of the reference cell shares the WL driver with data cell, but it requires a separated BL driver. The operation makes sure reference cells are always HRS which act as a precise reference for sensing. Control of the above three drivers are local which is showed in the green block of Fig. 20. All of the WLs and the BLs need these control transistors.

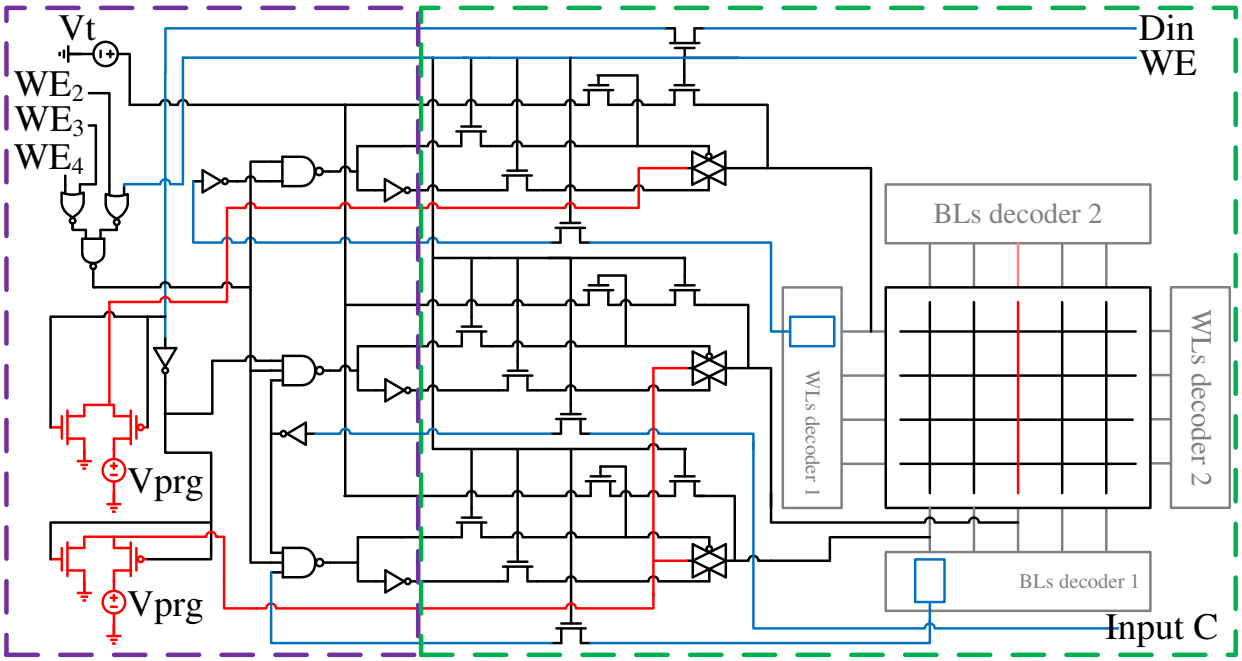


Figure 20: This is an example of the write scheme design. The green block is unshared *isolation* part. The purple block is *driving* part, which can be shared by multiple LUTs.

A purple block in Fig. 20 shows the shared drivers. They are shared by four LUTs in a LB. Example driving paths are highlighted in red lines. The drivers has a ∼10X transistor than a unit transistor to provide voltage and current for programming. With isolation of the transmission gates, the driver provides proper voltage or current to program a cell. *WE* represents write enable. A OR4 gate with *WE* signals from four LUTs controls activation of the write scheme. *Din* is an 1 bit input data which controls forward or reverse bias to program cells. The blue lines are a address from decoders, WL decoders and BL decoders (see Fig. 16). Input C, which is a selection of determining the layer using bitlines, is included in the drivers' control because of the sharing bitlines structure of the 3D-HIM.

4.1.5 Memory Configuration

Fig. 21 illustrates memory configuration of the proposed design. We take a 6-input LUT as an example. A LUT with 6 inputs and 2 outputs can be implemented with a 4-layer 3D-HIM structure. Layer 1 & 2 shares the same BL's (bitlines) and outputs *O1*. Similarly, *O2* comes from Layer 3 & 4. In each sensing cycle, the 6-input LUT generates 1 bit data to *O1* and *O2* from cells in layer 1 (2) and cells in layer 4 (3), respectively.

Address inputs *A*-*F* is applied to select data cells in the read and the write operations. Fig. 21 explains decoding definitions of the LUT. We use *A*, *B*, and *C* to control selection of BLs, while drivers of WLs are determined by *D*, *E*, and *F*. As illustration in Fig. 21, combinations of *C* and *F* decide a selected layer of the 3D-HIM. For example, when *C=0* and *F=0*, the BLs shared by Layer 1 & 2 and the WLs shared by Layer 2 & 3 are activated. Therefore, the target cell should be in the crossbar array on Layer 2 because both the WL and the BL are activated. Its exact location is determined by *A*, *B*, *D*, and *E*.

4.1.6 Flexible Configurations

The proposed LUT can support various configurations as a conventional LUT design does. For example, possible configurations of a LUT with 6 inputs and 2 outputs include:

- A 5-input LUT with two outputs: Two sets of independent logic functions are controlled by five common inputs *A*, *B*, *D*, *E*, and *F*. It's similar to the conventional LUT [63].

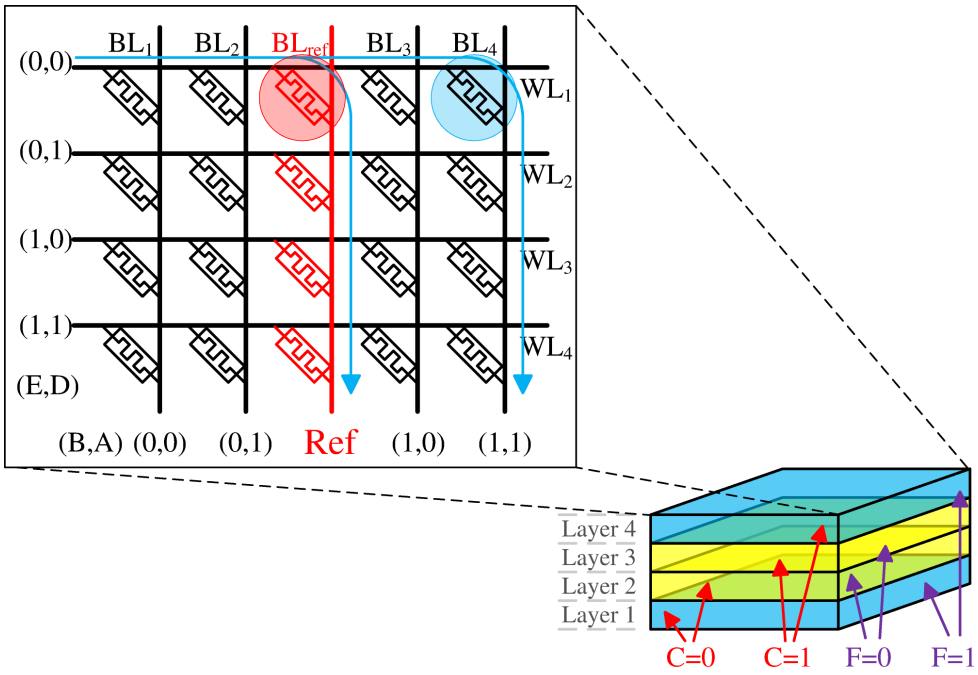


Figure 21: Address of the proposed 6-input LUT.

- A 6-input LUT with one output: Similar to the first configuration. But input C is in charge of a multiplexer to select *O1* or *O2* as a final output.
- LUT with less than 5 inputs: The first configuration can be downsized by fixing or discard usage of some of the inputs.
- Two small LUTs, each with no more than 3 inputs: Two outputs come from two sets of independent logic functions.

The above configurations are based on the decoding definition in Section 4.1.5. If the decoding definition is changed, configurations of corresponding inputs need to be modified. Various configurations provide design flexibility to utilize resource on a FPGA.

4.1.7 Simulation Results

In this section, we first evaluate the proposed sense amplifier for sensing ReRAM cells in a crossbar array with TSMC 0.18μm technology. Area, delay, and impact of process variation are included in discussions. Second, we compare the proposed ReRAM LUT with a conventional SRAM design. A system level exploration with benchmarks are introduced to evaluate area and delay. We show comparison results for 4-, 6- and 8-input LUTs.

4.1.7.1 Sense Amplifier Fig. 22(a) shows waveforms of the proposed sense amplifier at 500 MHz clock frequency. SA and SR represent sense amplifier and SR-latch, respectively. Nodes are corresponding to definitions in the Fig. 18. SA_out is the output of INV_b. SR_out is the same as $O1$. To demonstrate effectiveness of the design, we assume the configuration in a critical data pattern: the target cell and all the other cells are in LRS resulting in the smallest sensing margin. Signal difference at the inputs of the sense amplifier is enlarged and shown in Fig. 22(b). Under the worst scenario, there is 18 mV difference at 500 MHz operating frequency. SA_a and SA_b are two complementary results by the sense amplifier. After INV_a and INV_b, latency of the signals are improved showed in SA_out. With the SR latch, it shows logic '1' at SR_out which shows results of the sensing phase are correct.

Process variation induced resistance shifting of ReRAM cells is a severe issue. We conduct an evaluation on impacts of the process variation to the ReRAM cells. To verify the design, the critical data pattern is applied in the process variation's simulation which is similar to the previous simulations. We set extreme conditions by increasing resistance of the data cells and decreasing resistance of the reference cells. Thus, sensing margin of the data cells and the reference cells reduces.

Fig. 23(a) shows sense amplifier inputs SA_in and SA_ref with 5% to 30% variation of ReRAM resistance from its mean value at a step of 5%. Corresponding output results of the sense amplifier are shown in Fig. 23(b). Simulation results show that the proposed scheme can successfully read out stored data with up to 20% variation on the ReRAM cells' resistance. When variations are more than 20%, readout data are errant. The results in Fig. 23 demonstrates the proposed sense amplifier has a sensing margin limitation on

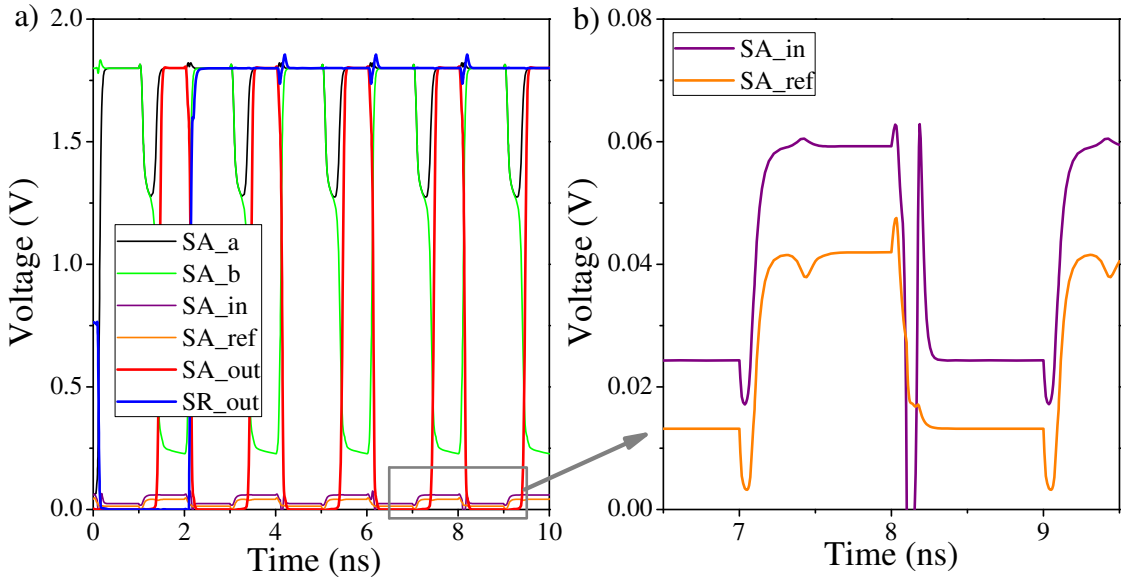


Figure 22: (a) Simulated waveforms of sense amplifier in a 4-input LUT under critical sensing condition of data pattern. (b) An enlarged view of sense amplifier's inputs.

$\sim$ 10 mV difference. When the signal difference between the two inputs is smaller than 10 mV, *i.e.*, the two fail cases which ReRAM resistance deviates 25% or 30% from its designed value, the sensing operation fails. A better noise cancellation circuit can further improve the sensing margin with price of area.

At last, we compare various sense amplifier designs [60] applied in a ReRAM crossbar array in terms of sense speed and design area. Summarized results in TABLE 3 show that the latch comparator and the expanded array in this work over performs the other sense amplifier designs. *Voltage divider* and *TIA* schemes generate reference signals by using a reference resistor, which costs a large area in modern CMOS technology. *Sigma delta* designs use an internal capacitor for integrating generated current that significantly slows down the sensing speed and increases the design area. Besides the latch comparator, the proposed design also benefits from the expanded crossbar array structure, which can better trace the impact of data patterns with a negligible area overhead.

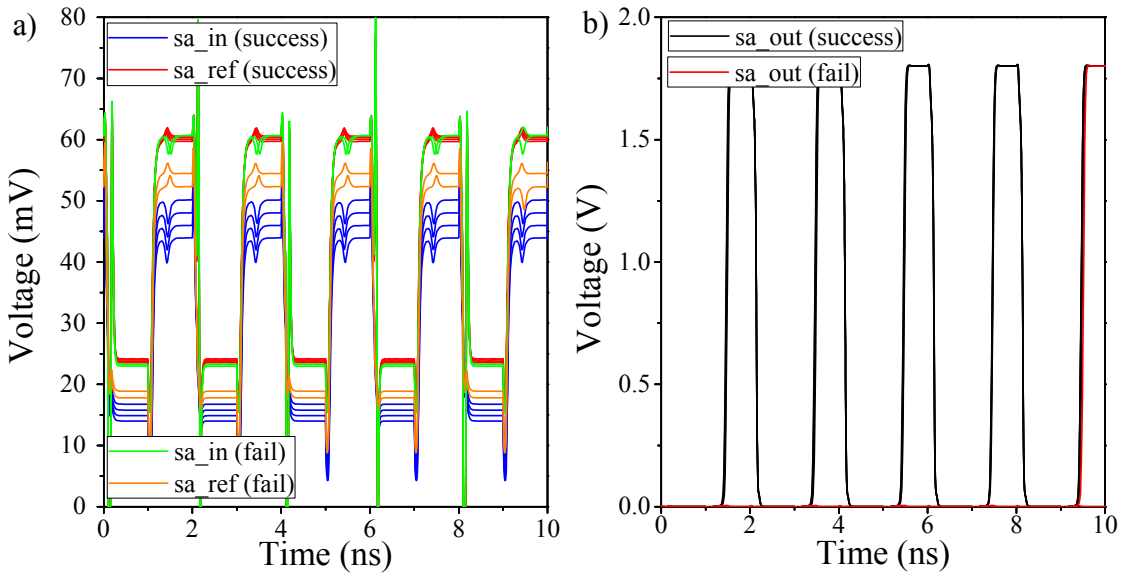


Figure 23: ReRAM resistance varies 5% to 30% from its design value. (a) The waveforms of sense amplifier inputs under the worst case data pattern. (b) The corresponding sensing result.

Table 3: Comparison of Various Sense Amplifier Designs

	Sense Speed	Area (normalized)
Latch Comparator (this work)	<4 ns	1.00
Voltage Divider [60]	<50 ns	1.78
TIA-based [60]	<100 ns	1.78
Sigma Delta w/ Buffer [60]	<10 us	7.29
Sigma Delta w/o Buffer [60]	<50 us	8.51

4.1.7.2 Area and Delay First, to verify functionality of the proposed LUT, Fig. 24 shows a simulated timing diagram of the 6-input LUT in read operations. Clock frequency is set to 500 MHz. Input signals change every clock cycle. It reads a serial of data from "000X00" to "000X11" and repeats it. Since it generates 2 outputs in each clock cycle, input C is not in usage. Data stored in addresses "000000" to "000011" and "000100" to "000111" are "1110" and "1011", respectively. Before 12 ns, output signals are based on content stored in the memory cells. The data are successfully read out and latched on the flipflops. After 12 ns, RE is tied to ground and the LUT is powered down. So, there is no signal after 14 ns. Similarly, write operation can be performed successfully at the frequency of 500 MHz.

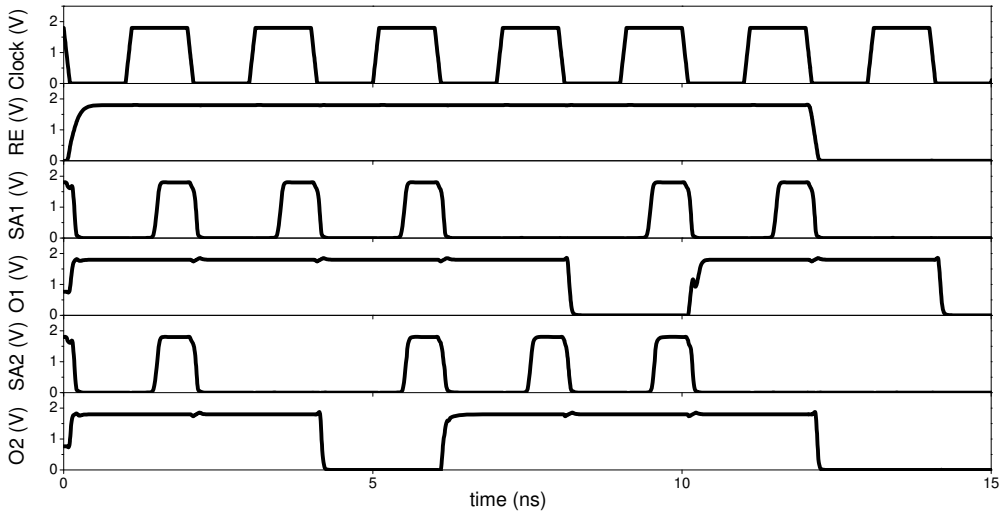


Figure 24: Timing diagram of 6-input LUT with 2 outputs at 500Mhz.

Area cost is one of major concerns in a LUT design. Fig. 25(a) compares area estimations of the proposed ReRAM LUT to a conventional SRAM LUTs, and a nanoPLA/nanoFPGA with various input numbers. Here, 4×5, 4-layer 4×5, and 4-layer 8×9 ReRAM crossbar arrays are used to construct 4-, 6-, and 8-input LUTs, respectively. The conventional SRAM LUT has corresponding 16, 64, and 256 SRAM cells. The nanoPLA/nanoFPGA has 4-input, 6-input, and 8-input PLA array [42].

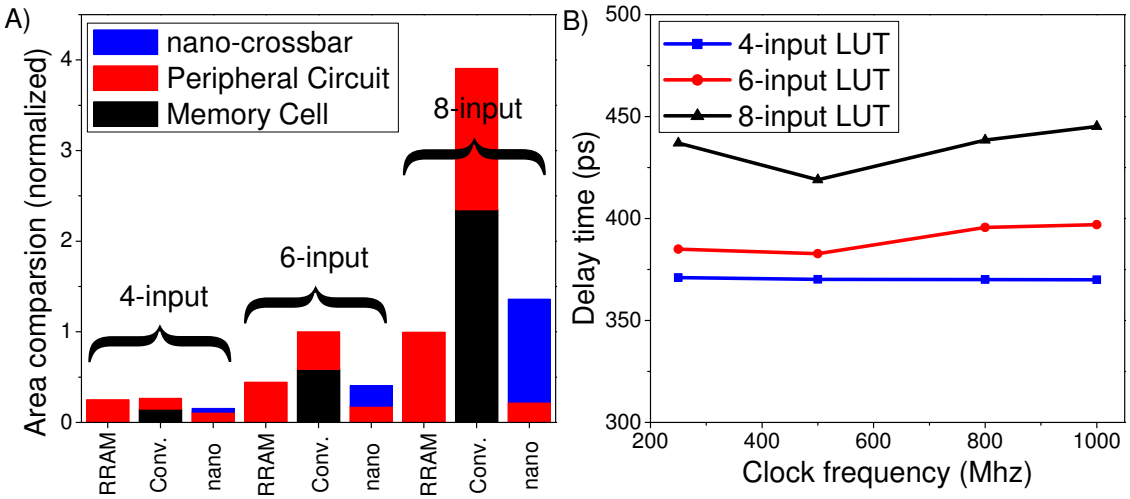


Figure 25: (a) Area comparison of various LUTs. Area estimations normalized to conventional 6-input LUT. (b) Delay of LUT with proposed sensing scheme for different operation frequency.

From Fig. 25(a), area of the proposed LUT is fully dominated by peripheral circuits because ReRAM crossbar array can be made in *back end of line* (BEOL) process which has no area cost on substrate. In the SRAM LUT, SRAM cells occupy more than 50% of overall area. The 4-input ReRAM LUT, which includes decoders for dynamic configuration, achieves 23.6% area saving compared to the SRAM one without supporting dynamic configuration. The SRAM LUT with decoders for dynamic configuration costs extra $\sim$ 23% of the area. When input number of the LUT increases from 4 to 6, the area of the ReRAM LUT is only 39.6% of that of the SRAM one. And the area saving grows to 75% in the 8-input LUT. Obviously, the area increases with increasing input numbers, however, the LUT with more inputs is able to realize larger function. Compared to the nanoPLA/nanoFPGA, the

ReRAM LUT has larger area when input number is 4 or 6 because of small area cost on the peripheral circuit of the nanoPLA/nanoFPGA. The area of the 8-input ReRAM LUT is slightly better than that of the nanoPLA/nanoFPGA. It is because the area of the nano-crossbar in the nanoPLA/nanoFPGA increases exponentially as increasing input numbers. Though the area of the proposed LUT is larger than that of the nanoPLA/nanoFPGA for the small LUT, the ReRAM is already close to fabrication while the device at cross points of the nanoPLA/nanoFPGA in still under development.

Fig. 25(b) shows access latency of the proposed read scheme when varying operating frequency. Delay time is measured from a rising edge of *clk* to a rising edge of the SA_outs. The LUT functions well at up to 1 GHz clock frequency. The given sense amplifier design was fine tuned at 500 MHz. So, the best performance at this frequency is observed. Increasing input numbers from 4 to 8, read delay grows from $\sim$ 370ps to $\sim$ 450ps. As increasing crossbar array size, sensing currents decrease which incurs degradation of sensing margins. So, it takes longer time to sense the difference. The access latency of the sense amplifier at the other operating frequencies can be improved by carefully adjusting V_{tbias}. Through simulations, the SRAM LUT delay is around $500ps - 700ps$ for $0.18\mu m$ technology. It can be seen that our LUT can achieve even faster speed than the conventional LUT at the same technology node. Compared to the delay of the nanoPLA/nanoFPGA, a fulladder's delay is 491ps [41]. The fulladder can be realized by using the 4-input proposed LUTs with the delay of 370ps. The proposed one is comparable to the nanoFPGA or slightly better.

To summarize, the proposed LUT achieves delay improvement, 2X-4X area reduction compared to the conventional LUT, as well as realization of the dynamic programming. By introducing the power saving mode for non-usage resource, it saves static power. Meanwhile, with less NMOS in the design, leakage power of the LUT is further reduced than that of the SRAM LUT. Compared to the nanoPLA/nanoFPGA, our design can achieve comparable or better area-delay results in addition to advantage of a mature fabrication.

4.1.7.3 System-level Exploration: ReRAM LUT Various LUT sizes, *i.e.*, input numbers, have significant impact on FPGA's architectures in terms of logic area efficiency. With increasing numbers of input, area and interconnections of the LUT grow fast [76]. Since

the LUT with more inputs can implement a larger function in one LUT, if total number of LUTs decreases for a logic function, overall area of the logic function's implementation may reduce. Many logic functions do not need to be implemented by large LUTs. In other words, utilization rates of the large size LUTs may be low. Area of a conventional SRAM LUT increases exponentially with more input numbers. Modern FPGAs usually use a 6-input LUT as a fundamental logic element to achieve the best logic utilization. In this section, we will explore trade-off for different LUT sizes of the proposed LUT design in a system level view. Design space exploration is needed to determine a optimal size for the proposed LUT.

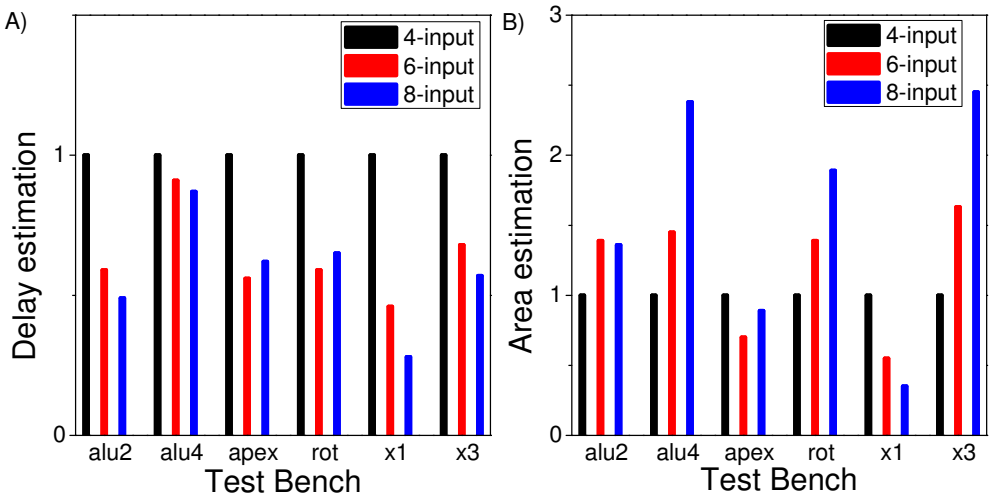


Figure 26: (a) Delay comparison. (b) Area estimation.

Fig. 26(a) and (b) show delay and area estimations of the proposed LUT on part of MCNC benchmarks [77], respectively. It also evaluates 4-, 6-, and 8-input of the proposed LUTs. From the previous section, we know that the area of the proposed LUT increases linearly with input numbers. We assume similar interconnections of modern FPGAs are used to connect the proposed LUTs. To demonstrate real area and delay of the proposed LUT with various input numbers, we set interconnections has no area and delay. So, the estimations of the area and the delay are only related to the LUTs. Meanwhile, corresponding results of the area and the delay are normalized to the 4-input LUT, which is a baseline for comparisons. Since the proposed LUT has two outputs, two small functions can share one big LUT by using the flexible configuration as described in Section 4.1.6.

From Fig. 26, some benchmarks, such as “alu2” and “x1”, can efficiently utilize the large LUT to reduce mapping the area and the delay. For benchmarks of ‘alu4” and “x3”, small LUTs are better because the large LUTs cannot be fully utilized and idle inputs result in resource waste. For benchmarks “apex” and “rot”, the 6-input LUT is the most efficient. We can see that efficiency of LUTs’ usage is determined by whether LUTs can effectively reduce numbers of the LUT in need and critical depth in a network. From these benchmarks, the system-level explorations and the preliminary mapping results introduce that the 6-input LUT is a promising design.

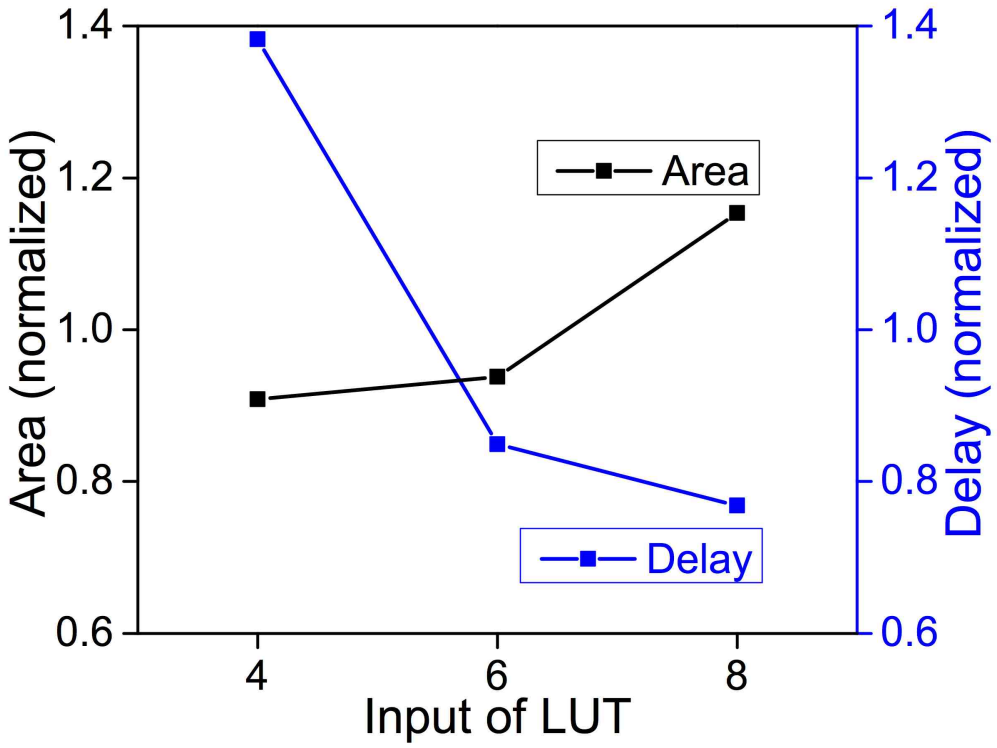


Figure 27: Area and delay of different sizes of LUT.

We explore the impact of different sizes of the proposed RRAM-based LUT at system level. Fig. 27 demonstrates comparisons of delay and area of MCNC benchmarks [77] and sample applications provided by VPR suite [16]. The results are normalized to the averages of each benchmarks and different sizes of LUT. To make fair comparison of all designs, numbers of track are fixed for all simulations and the routing system is designed of SRAM. The simulations focus on impact of sizes of LUT. The black and blue line are area and delay

of benchmarks in 4, 6, and 8-input LUT, respectively. The simulation results show that the trends of area and delay are similar to SRAM-based LUT that is larger LUT has larger area cost and better delay [63]. However, since area costs of 4-input and 6-input LUT are not linear increment, the area trend is relatively flat compared with SRAM-based LUT. From performance and area perspective, 6-input LUT is an optimized design for RRAM-based LUT. It has acceptable area increment and huge delay improvement of the applications.

4.2 RERAM ROUTING COMPONENTS

4.2.1 Connection Blocks

CB is applied in the FPGA to select inputs of a LB (CLB) from tracks and output of the LB to the target tracks. The modern CB has two types of design: memory cells with a multiplexer or memory cells with pass gates. The CB with the multiplexer has less memory cells for a configuration. However, the multiplexer itself needs more MOSFETs than pass gates to control signal. And, the pass gate design has better performance in delay. Considering tradeoff between cost and performance, the multiplexer design is popular in the modern SRAM FPGA.

Fig. 28 shows the proposed connection block with ReRAM cells for LB inputs. The CB for LB outputs can be built in a similar fabric. Since the ReRAM cells do not cost as SRAM cells, the proposed CB is designed with pass gate to save area of a mosfet and to have better performance. Moreover, the power consumption incurred by leakage current of mosfets is improved.

Fig. 29 demonstrates a ReRAM pass gate switch used in the CB. To program the switch short/open, it programs complementary cells to LRS+HRS or HRS+LRS by supplying a voltage pulse more than double threshold voltage, V_{th}, to the target cells. Fig. 29.(a) and (b) show the proper bias voltage to short or open the pass gate, respectively. For normal operations, operation voltage, V_{op}, across complementary ReRAM cells gives a voltage bias at gate of the NMOS to short or open the pass gate showed in Fig. 29.(c) and (d), respectively.

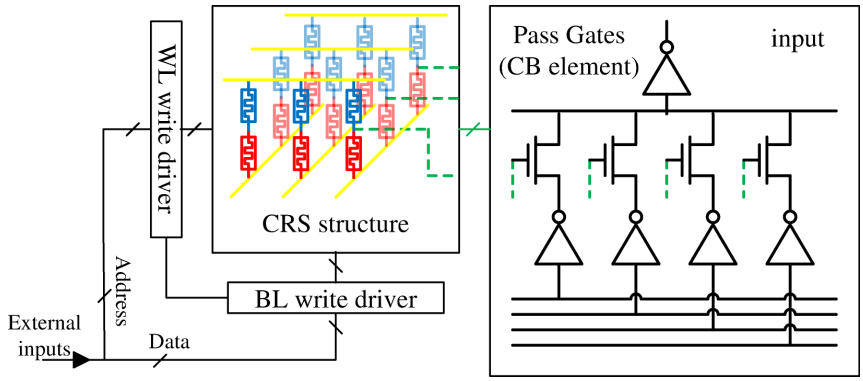


Figure 28: The CB is designed with pass-gates and buffers. The green lines show connections of ReRAM cells in CRS and pass-gates. The ReRAM cells in CRS can open or close the pass-gate based on resistance difference.

The data pattern in the complementary cell is LRS+HRS or HRS+LRS to properly control the NMOS pass gate. The LRS+LRS [20] is not a proper operation in the switch application which makes the pass gate unknown status. The V_{op} should not be larger than V_{th} to avoid programming complementary cells into LRS+LRS.

We gather multiple switches into a CRS structure [20] to further improve area cost and simplify control. Cells of the same wordline or bitline can share peripheral circuit to save area cost. For the normal operation, we can simply drive all wordlines to V_{op} and all bitlines to ground to make switches work properly. We do not read out data from bitlines as sensing data in the crossbar array, so controlling voltage at gate of the mosfets has no problem of interference. To program switches, it still has to program complementary cells one by one selecting the wordline and the bitline of the target cell [20].

4.2.2 Switch Blocks

SB links connections between tracks which is set to be the commonly used Wilton switch [78] for efficiency. It has multiple switch macros to bridge tracks to and from different directions. Switch macro has design of memory cells with multiplexers or memory cells with pass gates

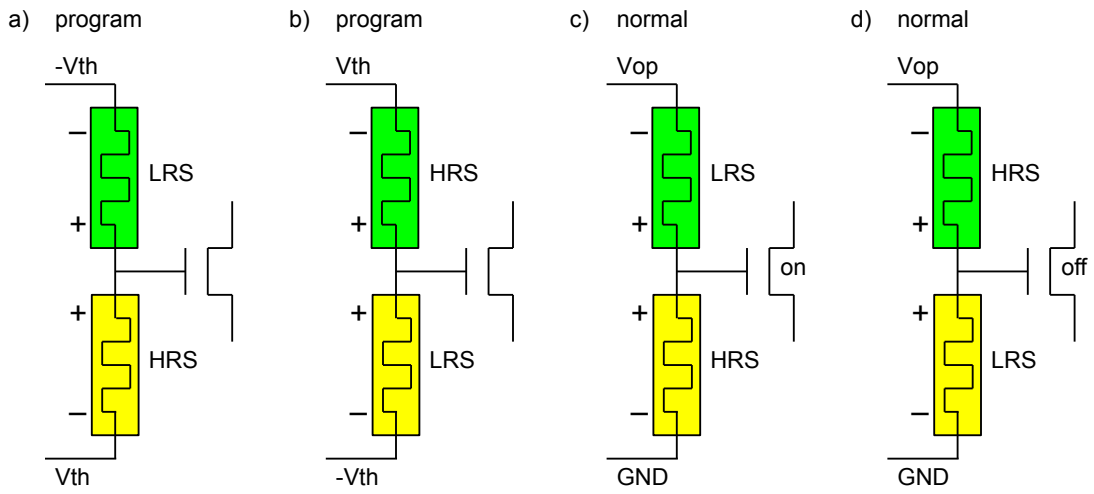


Figure 29: ReRAM pass gate switch: a) program short. b) program open. c) short in normal operation. d) open in normal operation.

to configure connections [79]. The pass gate design needs more memory cells but it has better performance [79]. Meanwhile, a pass gate itself has fewer mosfets comparing to multiplexer. Fig. 30 illustrates a proposed switch macro of the SB architecture. It has 10 ReRAM pass gate switches with complementary cells to control connections. The pass gate switch with complementary cells is the same as we mentioned in the previous section. Because the ReRAM has no area cost, ReRAM pass gate in a switch macro has advantage of area and performance comparing to design of multiplexer. Meanwhile, it can also be integrated into CRS structure for further saving cost of peripheral circuits and improving controllability which is similar as we mentioned in the previous section.

4.2.3 Discussion – ReRAM FPGA

In this section, we demonstrate characteristics of the proposed ReRAM FPGA with various technologies. We introduce performance of the purposed CB and SB with complementary memory cells in the CRS structure. The power consumption of the proposed components, LB, CB, and SB, are compared to SRAM design. We compare our work to other FPGA

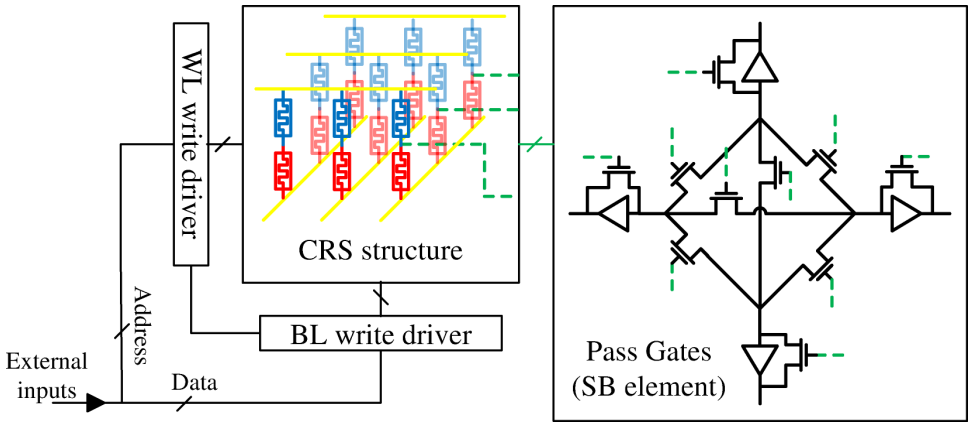


Figure 30: The SB is designed with pass-gates and buffers. The green lines show connections of ReRAM cells in CRS and pass-gates. The ReRAM cells in CRS can open or close the pass-gate based on resistance difference.

hardware with emerging memory technologies. Percentage of compositions would be included in discussion among various designs. Area and delay comparisons based on benchmark mappings from MCNC are included for analysis [77]. Finally we discuss pros and cons of the proposed design and propose possible methods and devices for future improvement.

Table 4: Delay of the SB and the CB

	180 nm (TSMC)	90 nm (PTM)	65 nm (PTM)	45 nm (PTM)	32 nm (PTM)
SB	164.9 ps	64.3 ps	67.7 ps	81.4 ps	144.6 ps
CB	69 ps	26.3 ps	23.3 ps	22.9 ps	24 ps

4.2.3.1 Delay and Area of the Proposed CB and SB Delay of the proposed CB and the SB with different technology nodes [80, 81] are given in Table 4. With shrinking of technology, it improves the delay time. The delay of the CB and the SB are similar to the conventional FPGA because we do not change the pass gate design. The main confinement

of the delay is still on the CMOS technology. Since we replace SRAM cells with ReRAM cells which are close to no area cost, 58.6% and 67% area cost on the CB and SB can be saved for a 180 nm technology node, respectively. For the other technology nodes, the ratio is similar because most of the CMOS devices are the smallest pass gates of various technology nodes. With advanced technology, area overhead of the buffer, which is a trade-off between area and delay time, could be alleviated because the delay time is smaller since the signal path is shorter.

4.2.3.2 Power Consumption of Components Table 5 lists power consumption of each components, LUT, CB, and SB, compared to the corresponding SRAM design at switching frequency of $500MHz$. We normalized the results to the SRAM FPGA. Significant saving from the ReRAM design can be observed from power consumption on tracks and the LUT. However, the CB and the SB have worse power consumption. They need static voltage across ReRAM cells of CRS to turn on/off the switches so static current leakage across the complementary cells consumes extra power.

Table 5: Power Comparison: SRAM and ReRAM

	LUT	SB	CB	Tracks
SRAM (45 nm)	1	1	1	1
ReRAM (45 nm)	0.90	1.68	1.11	0.62

4.2.3.3 Various Comparison of Compositions in One Tile Fig. 31 demonstrates the power distribution of components in each tile. The 2D CMOS and 2D rFPGA are from works of S. Tanachutiwat, *et al.* in 32nm technology [45]. The 2D CMOS is the conventional FPGA, and the 2D rFPGA is ReRAM FPGA with NOR ReRAM cells and 2T2R switch. The hardware architecture of the two designs are simple without complicated LBs. The ReRAM-32nm is the proposed design in the 32nm technology. The 2D CMOS, the 2D rFPGA, and the ReRAM-32nm are the same design as we mentioned before. The ReRAM-180nm is the tile of the proposed architecture in the TSMC 180nm technology.

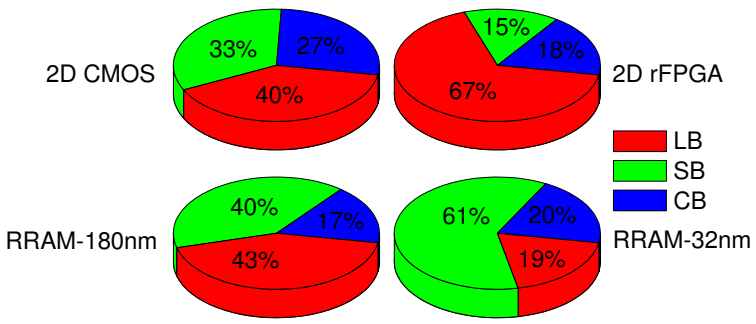


Figure 31: Power distribution of components in one tile.

There is a large difference on the SB between the proposed architecture and works of S. Tanachutiwat. It is because the proposed architecture has more tracks than the tile of the simpler design. The extra tracks causes more switches on the CB and the SB. It is resistive load to drive the ReRAM cells, so static current to maintain switches on/off is needed which causes larger power consumption. Comparing the 180 nm and the 32 nm technology, percentage of power distribution of the CB and the SB in the 32 nm technology increases since there is large amount of the CMOS devices in the the CB and the SB incurring more gate leakage than the LB does.

4.2.3.4 Benchmark Mapping Fig. 32 shows a software platform used for benchmark mapping and simulation. We apply a modified flow of CAD tools for the benchmark mapping and the simulation [16]. The flow begins from ODIN_II [82] to read in HDL description and translates into Boolean network. Then tool ABC synthesizes the Boolean network to a network of LUTs used in FPGA [83]. Following it, T-VPACK is used to group the LUTs into LB. Finally VPR [16] is used to accomplish placement and routing.

For interconnection hierarchy, we use more short wires in a mixed segment scheme and localize communication to fit size of the RU. We assume 96 tracks in each channel with setting of VPR: Fc-in = 0.15, Fc-out = 0.1, and Fs = 3 [16, 84]. The wire distribution is with 80% length-1 wires and 20% length-4 wires [16]. It is known that the interconnection parameters have great impact on area, delay of the architecture.

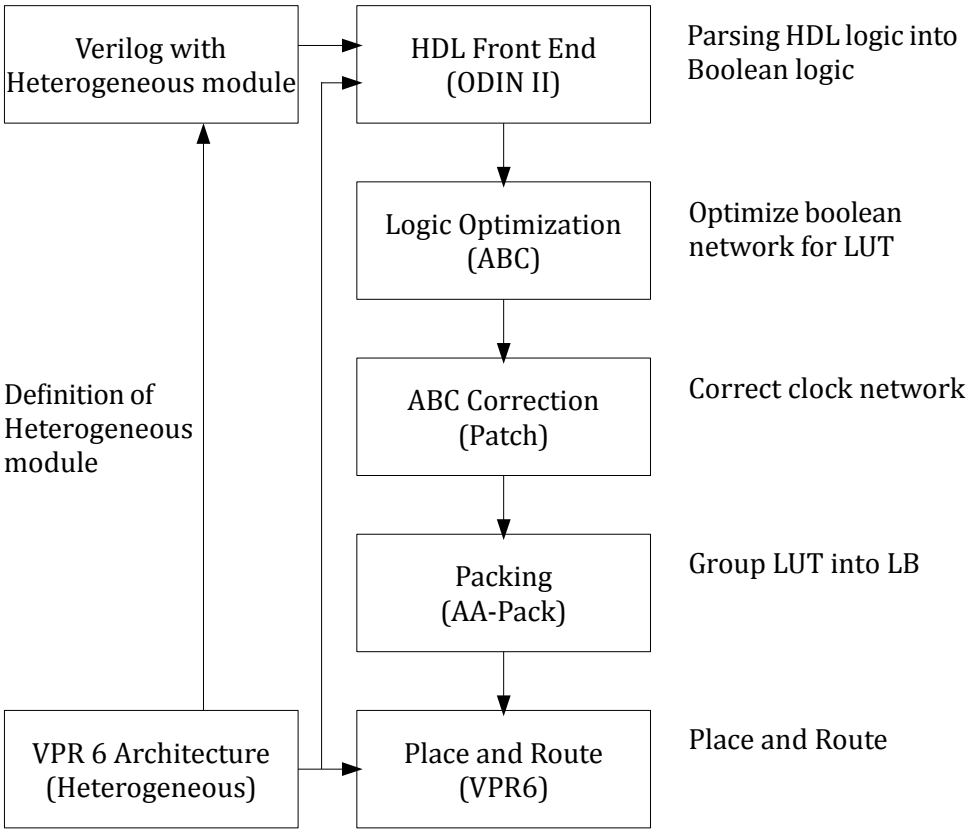


Figure 32: CAD and corresponding file in this work.

The proposed structure investigates 20 big benchmark mappings from MCNC [77] to show area, performance, and power of the ReRAM architecture. We assume the architecture instance with following parameters: LB, CB, and SB are the same as described in the previous discussion. The technology node is PTM 45nm [81]. Meanwhile, various architectures with either SRAM and other ReRAM memory systems are simulated for comparisons. Although architectures of each design are different, we can observe overall performance based on the MCNC benchmarks. Benchmarks are selected from 20 big in the MCNC benchmark suite [77] and mapped to the proposed architecture and the conventional SRAM architecture.

Fig. 33 shows results of area comparisons of various architectures including the SRAM and the ReRAM FPGA with 4-input and 6-input LUT. In general, the 4-input LUT has smaller area compared to 6-input LUT on both architectures. Comparing the 6-input ReRAM and SRAM FPGA, the ReRAM FPGA has 62.7% improvement in area because it has the ReRAM cells in a 3D stacking structure. By using the 6-input ReRAM FPGA, average of increasing area compared to 4-LUT is 40.4% rather than 60.9% of the SRAM FPGA. We can even compare the proposed ReRAM 6-input LUT to other works of the ReRAM 4-input FPGA, the 2D rFPGA [45]. The proposed one have extra 52.9% area. Our design has a bit-addressable LUT with more tracks, which incurs more area on the CB and the SB. However, it enables all LUTs for D-RAM usage. Meanwhile, from general idea, architecture of the 6-input LUT costs more area than that of the 4-input LUT. Overall, the ReRAM FPGA still has benefit in cost.

Fig. 34 demonstrates delay of critical path in MCNC benchmarks. The 6-input ReRAM FPGA has 34% improvement in the delay because the tracks of the ReRAM FPGA is smaller than the SRAM FPGA. The circuit delay is similar in two designs. The delay along the tracks dominates the results. The 6-input LUT has better performance than the 4-input LUT, because the routing path is smaller. The average delay of the critical path improves 57.2%. Comparing the 6-input ReRAM FPGA to the 2D rFPGA [45], the proposed design has extra 19.8% delay time. The delay results from longer tracks of the proposed FPGA.

By comparing average power consumption of the proposed FPGA to 2D rFPGA architecture [45] in 32nm technology, the proposed one is 5X power consumption than the 2D rFPGA in 32nm technology. However, by comparing to work of Y. Liauw *et al.* [46], the

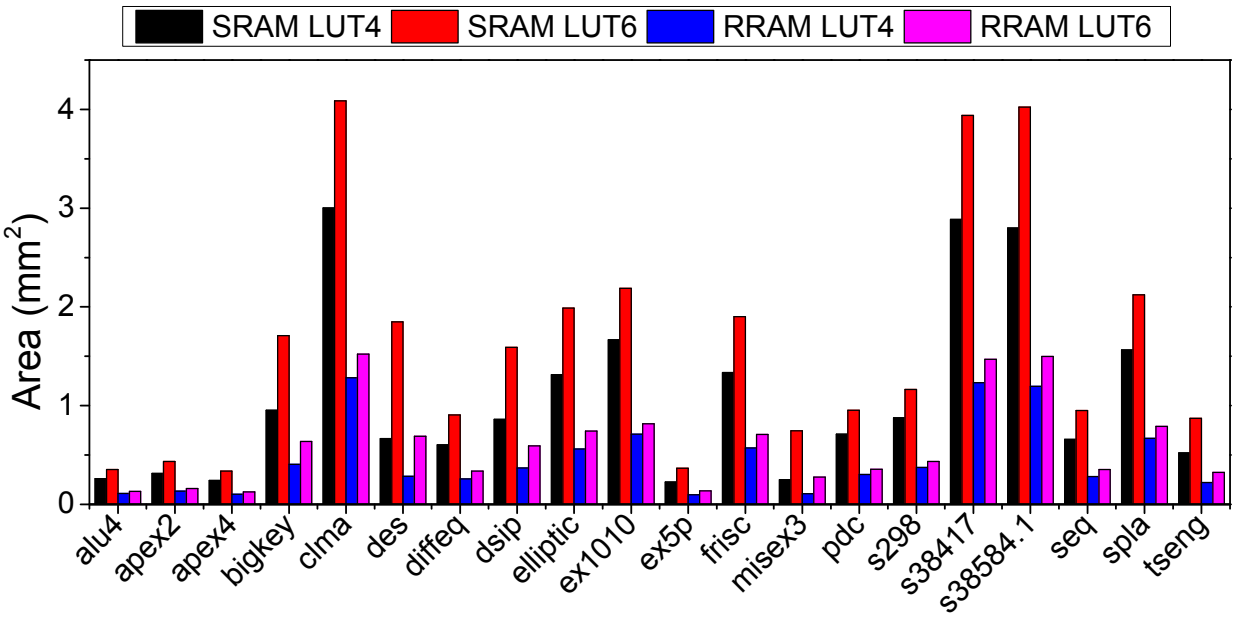


Figure 33: MCNC Comparison: Area (μm^2)

Energy-Delay-Product of the proposed architecture is only 17% of the Y. Liauw's work in 65nm technology. Apparently, power consumption of our work is between the two works. The difference is that we include a static analysis on the proposed model simulated by Spectre, Cadence. In the CB and the SB, static power is important since the complementary ReRAM cells have static leakage currents. Meanwhile, the LB has different design than the other works, which has smaller leakage power.

4.2.4 Power Estimation and Comparison

In the section, we demonstrated the power estimation between SRAM and ReRAM FPGA. The architecture of both FPGA are similar. The only difference is the memory device and corresponding peripheral circuit. As the method in the previous section, we have 20 big benchmarks from the MCNC suite [77] to show the results.

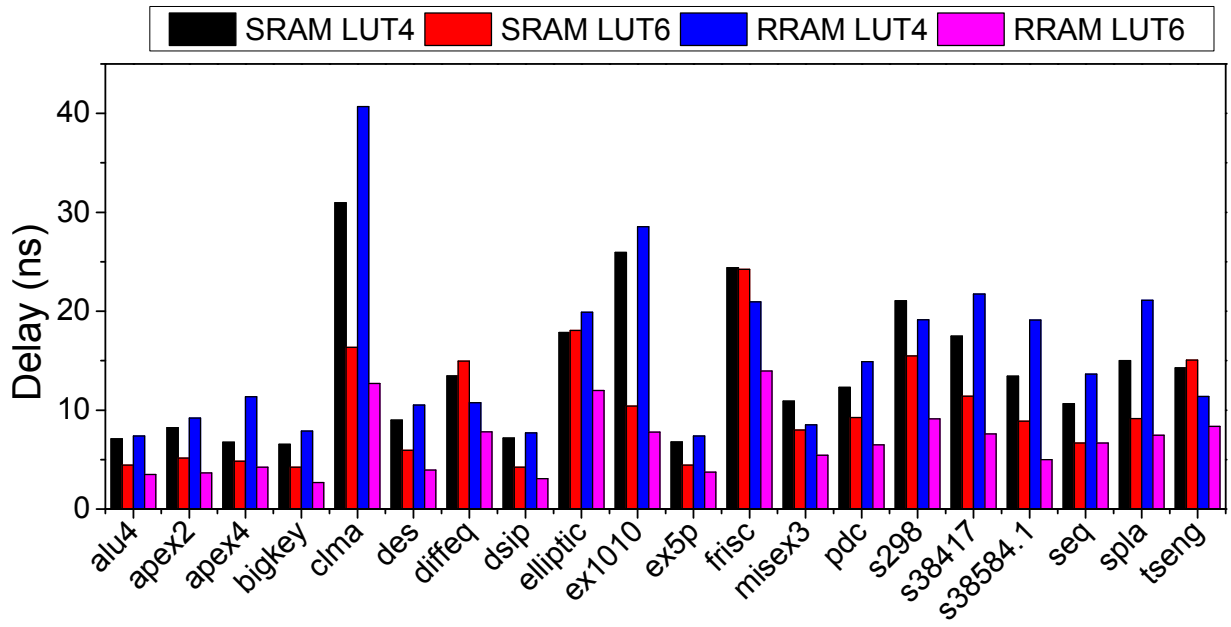


Figure 34: MCNC Comparison: Delay of Critical Path (ns)

First, we introduce the power estimation tool of the work. Since the components of the ReRAM FPGA is different from SRAM FPGA, an academic power estimation tool [85] is not proper to estimate the results. Thus, we proposed a power estimation tool which is based on trace results from the VPR 6 [16] and dynamic and static power information of the components provided by users. In general, FPGA power is divided into three parts, routing power, logic power and clock power demonstrating in Eq. 4.1. The power estimation of these three parts are further decomposed into multiple components in calculation. Eq. 4.2 to Eq. 4.4 explain the break down of these four components. The suffix $.D$ and $.S$ presents dynamic power and static power of a component respectively. Among them, clock network power is special that it was calculated separately with methods introduced by [85]. Things seem to be very easy if all the power information in the library are just absolute numbers. However, in order to preserve certain accuracy, we require more information from the library of power components. For example, for primary component routing track, the capacitance

of length-1 segments C_{wire} should be provided by the library or in the architecture file. Meanwhile, switching activity is needed information which can be acquired from the VPR. $P_{TRACK.D}$ is obtained by calculating Eq. 4.5 and $P_{TRACK.S}$ is usually zero.

$$P_{FPGA} = P_{routing} + P_{logic} + P_{clock} \tag{4.1}$$

$$\begin{aligned} P_{routing} = \Sigma P_{CB.D} + \Sigma P_{SB.D} + \Sigma P_{TRACK.D} \\ + \Sigma P_{CB.S} + \Sigma P_{SB.S} + \Sigma P_{TRACK.S} \end{aligned} \tag{4.2}$$

$$\begin{aligned} P_{logic} = \Sigma P_{Crossbar.D} + \Sigma P_{LUT.D} + \Sigma P_{DFF.D} \\ + \Sigma P_{Crossbar.S} + \Sigma P_{LUT.S} + \Sigma P_{DFF.S} \end{aligned} \tag{4.3}$$

$$P_{clock} = \Sigma P_{clock.D} + \Sigma P_{clock.S} \tag{4.4}$$

$$P_{TRACK.D} = 0.5 * activity * {V_{dd}}^2 * C_{wire} * length * f \tag{4.5}$$

Fig. 35 shows power estimation of the SRAMe and ReRAM FPGA. The architecture of both FPGAs are similar and designed in 45 nm technology node of PTM [81] as the same manner in the previous sections. We provide design information, which can be obtained in architecture files of FPGAs, and power estimation of components, including LUTs, CBs, SBs, tracks, *etc.*, based on Cadence EDA Tools [86]. Result shows that ReRAM FPGA consumes more power in all benchmarks. The track power is reduced by shorter segment length and the power of LUT is slightly smaller than SRAM LUT. However, the extra power is because of static bias to switch on/off of the pass gate switch. It can be improved by substituting a high resistance ReRAM device to reduce the static current across the ReRAM cells in CRS structure. Meanwhile, ReRAM FPGA can partially enter deep sleep mode to reduce power of idle logic.

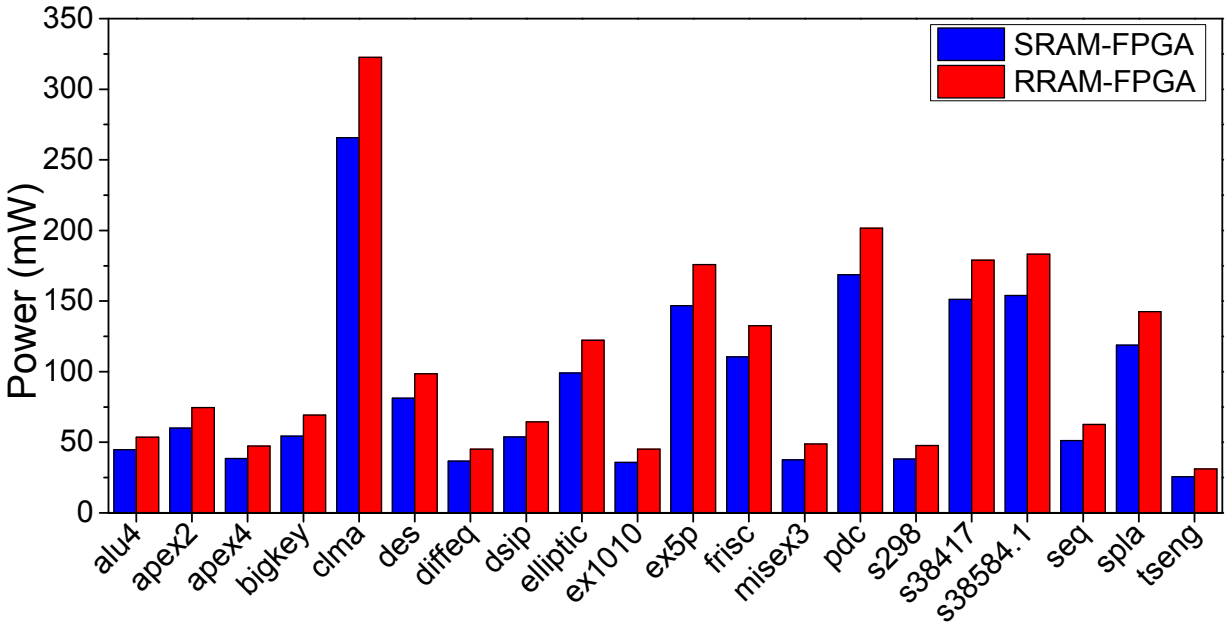


Figure 35: Power estimation of 45nm SRAM and ReRAM FPGAs.

4.2.5 Miscellaneous Discussion

We discuss design considerations of the ReRAM FPGA and propose possible methods for improvement. In the power comparison, the SB and the CB cost large power because of the switch design. However, we can change the ReRAM cells to a high resistance device such as Al_2O_3 to suppress the leakage. Meanwhile, a high R_{off}/R_{on} ReRAM is preferred in the design to control leakage of the pass-gate by giving gate voltage close to vdd or gnd to throttle the leakage. To reduce the delay of a LUT, the read scheme of the ReRAM cells should be improved in both the circuit and the device. A fast sensing device is promising for the proposed architecture since we already design the peripheral circuit to support it.

The proposed architecture is the bit-addressable design on every LUTs which is different to conventional FPGA. The design of the conventional FPGA needs extra area to achieve the function with a decoder and a local programming circuit. The bit-addressable LUT is not only for the D-RAM as we have in commercial FPGAs, but also for runtime configuring function.

Another improvement is nonvolatility of tiles since it substitutes all memory cells of the FPGA to the ReRAM cells. We can add a novel function which is called *deep-sleep mode* to the FPGA by partially powering off unused tiles. Unlike the conventional SRAM FPGA, there is no need to maintain any energy to keep data. Meanwhile, an initializing step can be omitted if a copy of configuration is available in the FPGA. The area and the cost are not only saved by the ReRAM cells within the FPGA, but it also saves design of an external FLASH. The FPGA can be standalone system after the first initialization.

4.3 UBRAM AND RECONFIGURATION

Fig. 36 gives an overview of a *reconfiguration unit* (RU), the basic building component in the proposed ReRAM FPGA. In our design, each RU is composed of eight *tiles* and one *unified block RAM* (uBRAM). As a basic functional unit, a tile comprises a LB, two CBs, a SB, and a set of segment-based interconnecting routing tracks. The uBRAM is a storage component saving configuration functions as well as temporary data. In the following subsections, we will describe design details of these building blocks and the operation of the ReRAM FPGA.

4.3.1 Architecture of Unified Block Memory

A modern SRAM FPGA needs an external FLASH to store configurations. Meanwhile, it contains SRAM embedded memory blocks for on-chip temporary data storage. For example, Virtex-7 contains more than one thousand 18Kb memory blocks [87]. We propose a uBRAM with the ReRAM cells to function as both a configuration memory and a data memory at different time.

Fig. 37 shows the proposed uBRAM architecture. The uBRAM contains n memory islands to increase its data bandwidth. The number of memory islands is set to be equal to the number of tiles in the RU to ease data transmission. Each memory island is composed of an 2^l-layer $2^m \times 2^m$ array in the 3D-HIM [70] structure. For a 8 islands uBRAM, each island includes 8k ReRAM cells, ratio of the uBRAM size to the tile size is estimated to be

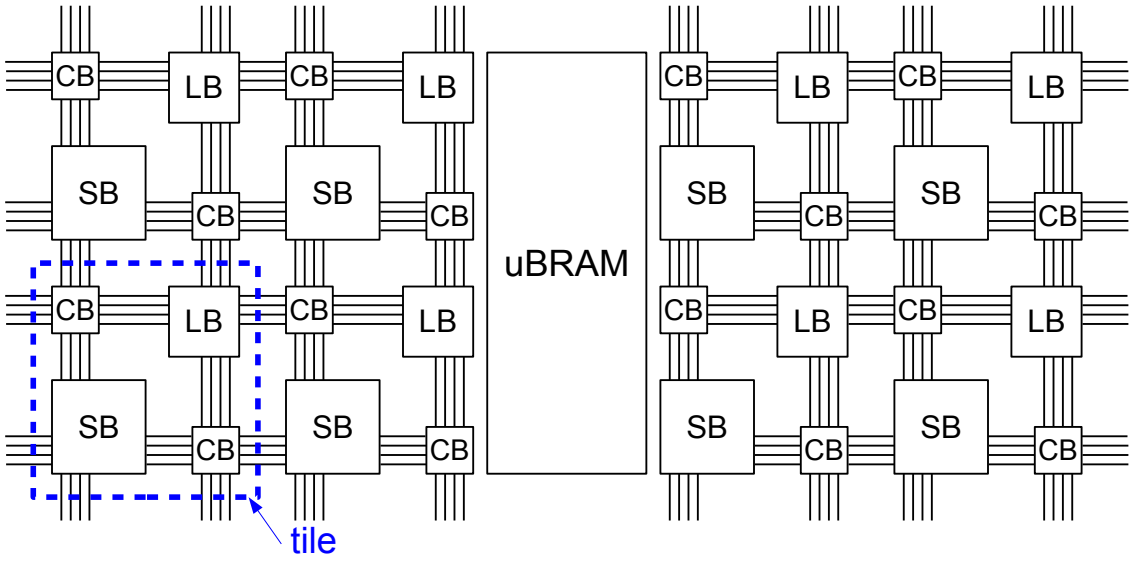


Figure 36: This is a RU of the proposed architecture composed of eight tiles. The blue area is a tile comprising LB, SB, and CB.

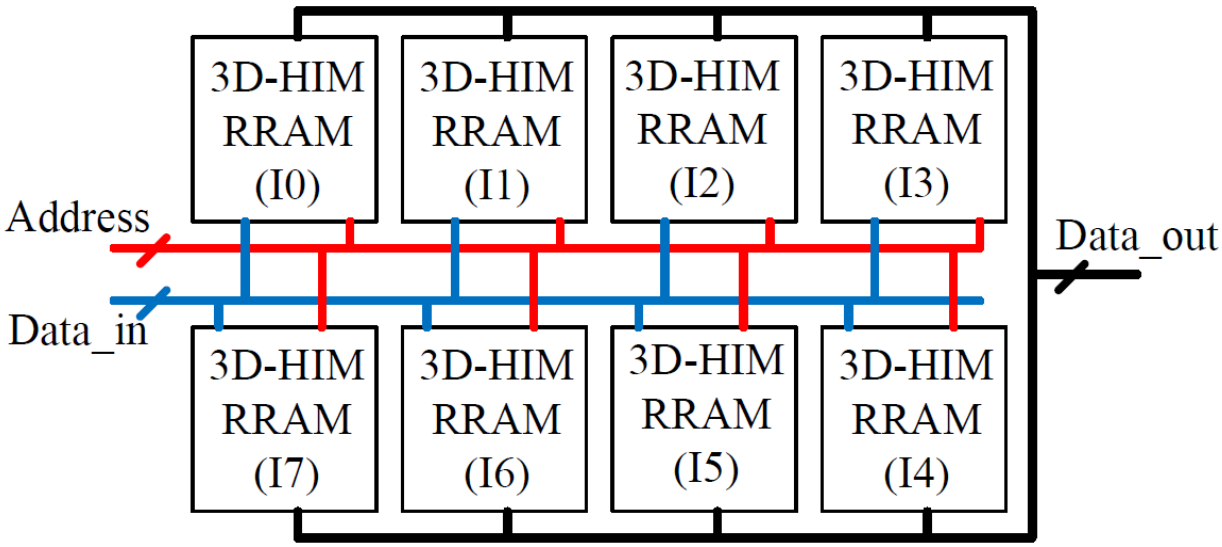


Figure 37: The proposed ReRAM uBRAM with eight memory islands and a set of shared address lines.

615%. All the memory islands will be accessed at the same time, hence, address decoders can be shared by all memory islands in the uBRAM. To address one bit in a island, $2m + l$ number of address lines are needed.

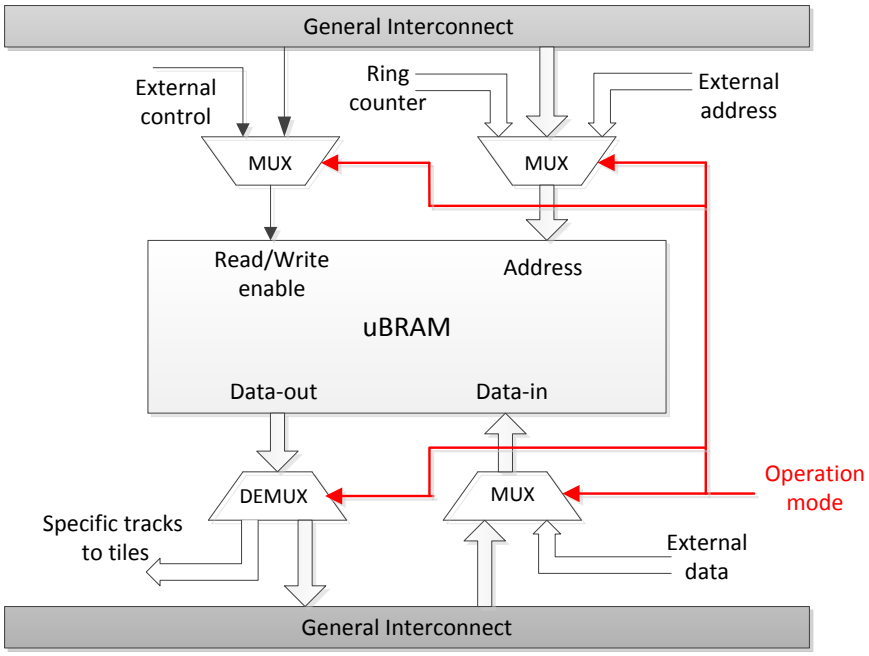


Figure 38: The interconnections of uBRAM for data/configuration mode.

Fig. 38 shows connections of the uBRAM. Data input has two sources, an external communication port or general tracks. Address is from the external communication port, an internal ring counter, or general tracks. Control signal is from the communication port or general tracks. The outputs could be special tracks to the tile or general tracks. The external communication port are special tracks and circuits needed to write configuration bits into uBRAM from the external communication port. It is managed by FPGA communication units.

When the uBRAM functions as a data memory in a logic operation, the inputs/outputs of the uBRAM have to connect general tracks to feed/fetch data from the LBs. Input bandwidth is related to numbers of memory islands in the RU since we program one bit per clock cycle to one memory island for area and power reduction. Output bandwidth of the uBRAM is flexible to configuration depending the bandwidth requirement and available

resource of general tracks. One island can output 1-2^{l-1} bits per cycle by adjusting number of address inputs. The uBRAM can generate maximal output of $n \times 2^{l-1}$ bits data per clock cycle which enables fast data accessing for high performance application. Hence, there are $2^{l-1}+1$ data lines (2^{l-1} for outputs and 1 for input) for one memory island.

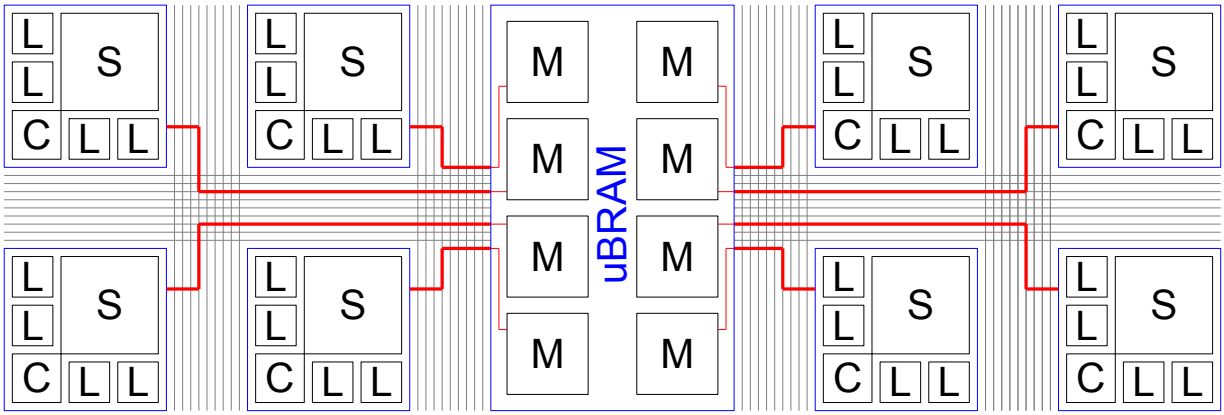


Figure 39: Special tracks from uBRAM to each tile.

To configure LB, CB, and SB from uBRAM, special tracks are used to connect the uBRAM to the ReRAM cells in the tile, *i.e.,* the ReRAM cells in the LB, the CB, and the SB. As demonstration in Fig. 39. Each blue line represents four tracks that connect four data lines of the uBRAM to each tile. The special tracks are small area overhead since they are direct connections from uBRAM to tiles. While in configuration mode, partial address lines are needed to select different configuration copies in the memory islands controlled by logic. Moreover, to read the selected configuration copy, bits can be read out sequentially and controlled by a simple ring counter for the address. The maximum number of bits, *i.e.,* 2^{l-1} bits per island, will be output per clock cycle to speed up a reconfiguration. Multiple bits output per island will need extra sensing amplifiers, however, it only costs small area increase. Taking an example of 8 layers, where $l = 3$, there is only 13% area increase. The design of the special tracks, the local addressing method, and the burst outputs has advantage in speeding up the configuration while avoiding interference routing or adding interconnection complexity.

We use an example to illustrate data mapping in the uBRAM and calculate the reconfiguration delay. It assumes 8 tiles are contained in one RU here. Hence, there are also 8 islands in the uBRAM. Assuming parameter setting for the island is $l = 3, m = 5$, then capacity of one island is $8K$ bits. A 11-bit ring counter is needed to supply the address to sequentially read out 4 bits per cycle from the $8K$ island. Based on the LB design, four 6-input LUTs need 256 configuration bits and a 15 to 24 fully populated crossbar needs 360 bits. As for the interconnection, 96 tracks, Fc-in = 0.15, Fc-out = 0.1, and Fs = 3 [16, 84], the configuration bits needed for the SB and the CB are 960 and 305, respectively. Hence, one configuration copy for the tile is 256+360+960+305 = 1881 bits. The address distribution is as follows: 0-255 for the LB, 256-615 for the crossbar, 616-1575 for the SB, and 1576-1880 for the CB. Since the burst mode has 4 bits output in one cycle, it takes 471 cycles to finish the programming. Assuming per cycle needs $4ns$ to program one ReRAM cell, total configuration time is 1884 ns. It is better than modern FPGAs which takes couple of seconds to finish programming.

4.3.2 Reconfiguration with uBRAM

4.3.2.1 Initialization The uBRAM introduces two stages reconfiguration. The first stage is to load the configuration uBRAM. Data and address are sent to the uBRAM through the communication interface of the FPGA under control of an external controller. Control signals specify the specific uBRAM to write to and provide write address. In the second stage, the control signal from the communication port triggers the configuration. It only takes couples of micro seconds for programming as we mentioned in the Section 4.3.1.

4.3.2.2 Runtime Partial Reconfiguration There are two types of runtime reconfiguration with the proposed FPGA: an external and an internal. The external configuration denotes the reconfiguration data which is from an external source in the first stage. Internal reconfiguration is defined as loading configuration copies from the uBRAM in the first stage. In both methods, the data is not directly programmed into logic and route elements. As a result, loading configuration to the uBRAM does not stall the system in the first stage.

It is only about 0.2% of the conventional partial reconfiguration delay, which is around milliseconds. The external runtime reconfiguration loads configuration from external communication port. We monitor the system and load the corresponding synthesized logic into the uBRAM. The data and address are packed into bitstream and sent to target uBRAMs. The step is similar to initialization of the proposed FPGA. The difference is that we only send data to the target uBRAMs rather than all of the uBRAMs on the FPGA. The procedure is similar as the partial reconfiguration in modern FPGAs. However, modern FPGA will directly program the SRAM cells. Hence, the computation has to be halted during reconfiguration. With uBRAM, the system loads the configuration in the first stage, then programming programming data in the second stage which only causes 2 μs.

The internal runtime reconfiguration loads configuration from the uBRAM. There is a synthesized monitor function to control the runtime reconfiguration internally. The 8k uBRAM island can store maximum four copies of configuration, bit (0-1880), (2048-3928), (4096-5976), and (6144-8024). The configuration stored in the uBRAM has two sources: pre-stored logic and runtime synthesized logic. The pre-stored logic is from the initialization or runtime external loading in the first stage. The uBRAM has multiple copies of the logic. While doing the internal runtime reconfiguration, it proceeds the second stage and selects the target logic from configurations in the uBRAM. The runtime synthesized logic is generated on chip. We configure the target RU's uBRAM as a data memory. The data of the first stage is generated by other logic function. Once, the monitor function decides to reconfigure the system, it directly proceeds the second stage to configure the tiles.

Comparing the external and the internal configuration, the external configuration provides an efficient way of supporting more possible configuration copies and fully utilizing all resource for logic function. With limited numbers of reconfigurable copies, the internal reconfiguration can form a standalone system by choosing copies from the uBRAM. It can be a self-adapting system while loading the runtime synthesized logic. However, the design costs large resource for generating the logic function. System designers can determine the proper configuration method based on the available resource and system's requirement.

In our design, the first stage acts as a buffering time for loading configurations into the uBRAM. There is no system stall in the first stage to disturbing concurrent computation. Through this approach, the loading delay is capable to be hidden. The real reconfiguration happens in the second stage which only takes less than 2 μs. The programming mechanism can be performed in parallel among RUs. With the uBRAM, eight 8k islands, maximum four copies of logic functions can be performed indicates that four times of more logic, 4×8(tiles per RU)= 32 tiles, can be implemented. Relative area saving obtained is $32/6.15 \approx 5.2$ times. Note that more reconfiguration copies stored, more reconfiguration can be performed and more area saving can be achieved.

4.3.3 Case Studies – FPGA with uBRAM

We evaluate two examples to demonstrate the flexibility of runtime reconfiguration and the benefits on area, delay and power. To set up the experiments, we assume the architecture instance with the following parameters: The LB, SB, CB, and uBRAM are the same as described in the work. CMOS technology node in the following simulations are based on the PTM 45 nm technology [81]. CAD tools are the same as Section 4.2.3.4.

Fig. 40 illustrates a synthesis flow for applying run-time reconfiguration in an application. System designers identify run-time reconfigurable modules on HDL by tracing through task graph or manual noting by designers. Logic function is then decomposed into modules of one and various contexts. All of the modules and their contexts are synthesized separately and given indices for their corresponding configuration contexts. For design automation consideration, each module has the specific context as a representation of the module in global place and route stage. We apply local place and route for rest of the contexts to their designed region from the result of global place and route. Configurations of contexts would be stored in the uBRAM of the designed region and then be loaded while in need. Corresponding control unit, such as micro-controller, would monitor logic function and schedule run-time reconfiguration. According to the required reconfiguration contexts and available uBRAM size, either the external configuration or the internal configuration can be determined. Both methods still apply two stages reconfiguration for reducing reconfiguring time and simplify the control of reconfiguration.

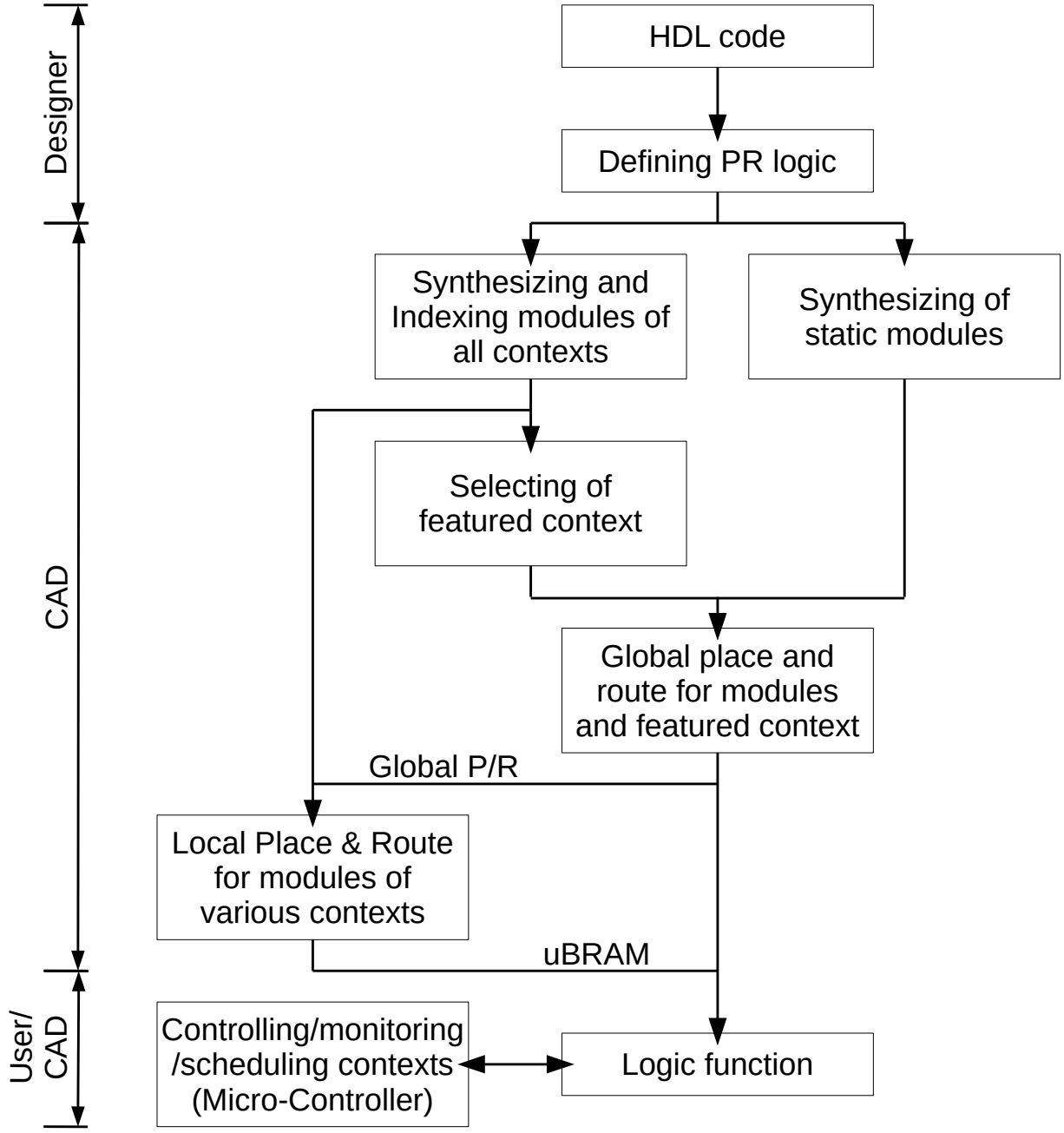


Figure 40: The rough flow for optimizing, controlling, and monitoring reconfigurable logic.

4.3.3.1 Case Study–Dynamic Loading Pre-Synthesized Configurations

We have a case study on software defined radio to evaluate the proposed design for the external configuration and the internal configuration with pre-synthesized configuration bits. The uBRAM stores multiple contexts which are runtime loaded to the corresponding tiles to change logic function and reuse the same resource of the FPGA.

In software defined radio and other wireless applications, it requires a dynamic loading of different modulations according to runtime conditions. We take dynamic switching of modulations in *Orthogonal frequency-division multiplexing* (OFDM) communication as an example. The system is illustrated in Fig. 41. According to different signal modulation requirements, three modules including QAM16, QPSK, and BPSK will be dynamically switched. First, if three configuration copies for three modules are already contained in the uBRAM, only internal configuration is needed. An on-chip controller can generate the control signals for configuration copy selection and a start/stop of the internal configuration.

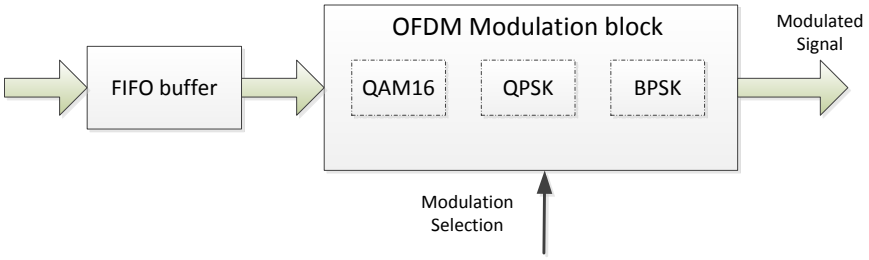


Figure 41: Signal flow of OFDM.

Table 6: Mapping results of OFDM modulations

	tiles	RUs	Area (normalized to tile)
QAM16	34	5	40 + 30.75(uBRAM)
QPSK	35	5	40 + 30.75(uBRAM)
BPSK	33	5	40 + 30.75(uBRAM)
non-reconfigurable	108	–	108

Table. 6 shows mapping results of OFDM modulations in the proposed architecture. Ac-

cording to mapping results, QAM16, QPSK, and BPSK need 34, 35 and 33 tiles, respectively. However, through reconfiguration, the tiles can be reused and shared between different modules. Five RUs, which includes 8 LBs in each RU, would be needed because we use RU as the smallest reconfigurable unit. In contrast to it, if no reconfiguration is allowed, total 108 tiles are needed for mapping the three modules. Since internal configuration delay is determined by the configuration time of a tile in the second stage configuration, switching time between modules takes 471 clock cycle as described in Section 4.3.1. Data rate for software defined radio is typically around $20ms$ [88]. Hence the switching time does not affect data processing. Since the uBRAM has 6.15X area to the size of a tile, if assuming no uBRAM contained in the non-reconfigurable architecture, 34.5% area saving is obtained without degrading the data processing throughput compared to the non-reconfiguration case. However, embedded memory are sometimes used even for the non-reconfigurable architecture for data storage. Comparing with the non-reconfigurable architecture with block memory, the area saving can be 2.7X. Considering extra area reduction because of replacing the SRAM with the ReRAM, the area saving achieved by the proposed architecture using dynamic reconfiguration compared to current FPGAs can be significant.

Through performing the dynamic reconfiguration, leakage power can be saved because of reduction of tiles in usage. However, at the same time, we need to consider uBRAM's read power and ReRAM cells' write power in RUs. Each reconfiguration in the ReRAM system is about 1.90X power consumption to the tile's leakage consumption. Experimental results show 17% power consumption can be reduced in the reconfigurable case. Compared to the SRAM FPGA with no reconfigurable application, it increases 4.3% power consumption because of larger power consumption of the CB and the SB. Since the power increase is small, it still can be considered comparable.

If modules exceeds the uBRAM capacity, the external configuration will use dedicated addressing circuit to select the uBRAM to load new reconfiguration bits under external control as discussion in Section 4.3.2. Taking the biggest reconfiguration copy of QAM64 as an example, it takes less than $4ms$ to finish loading in the first stage of configuration. Since the loading will not stop the current computation, the external reconfiguration delay can be hidden and area saving can be obtained while performing proper predicted prefetch.

4.3.3.2 Case Study–Runtime Calculating Configurations In some applications, *e.g.* weighting calculation, synapse, self-adapted circuit, *etc.*, runtime adjustment of the parameters or functions is needed. It would be good if the current configuration bits can be modified or generated according to a runtime calculation result. However, it is hard to achieve the function for previous works without the two stage configuration to store configuration in the internal memory and configure the logic function later. Otherwise, it takes system a large stall for the runtime calculation. The internal configuration with runtime configuration calculation supported by our proposed architecture makes it feasible. Next, we will use two examples to illustrate the runtime adjusting of parameters and functions.

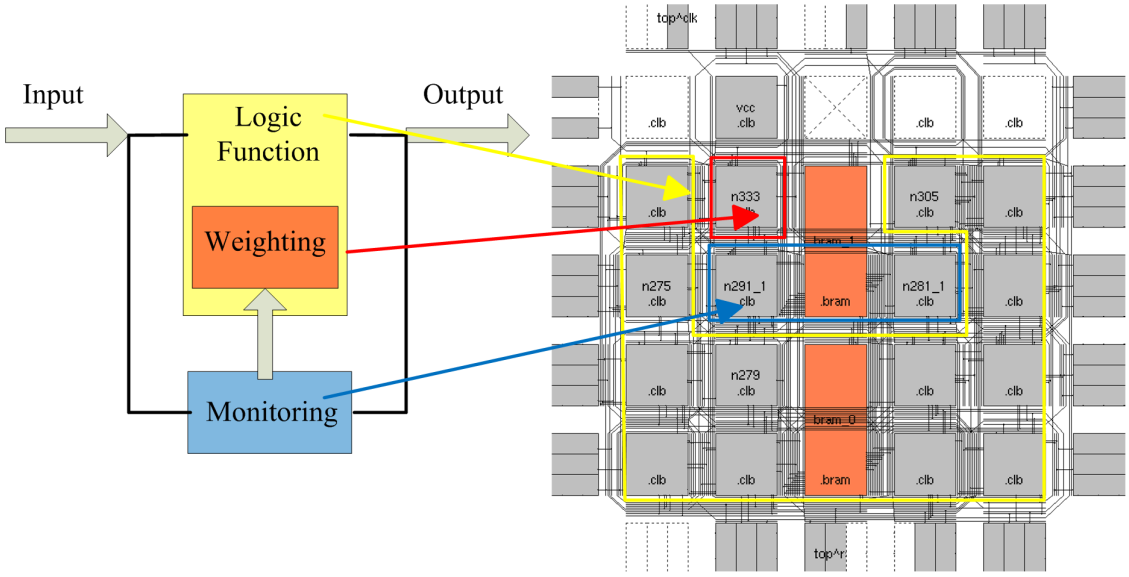


Figure 42: The internal logic generates weighting to uBRAM for later configuring.

The first example for weight calculation is shown in Fig. 42. The logic function varies slightly based on results of runtime weighting calculation. Such application is no need to do runtime synthesis for logic function, hence, it requires small resource. Here we simplify control signal as a input/output pin and emphasize the uBRAM of fetching and configuring. As shown in the figure, The LBs, highlighted in blue, function as monitoring and generating the weighting data to uBRAM. The LB which needs an adaptive weighting as configuration bits and stores the weighting related function is highlighted in red. The functions of the rest LBs are not affected. First, the logic function generates the weight and stores the data

into uBRAM. After the data is stored to the uBRAM, the monitor control the uBRAM to configure the logic. After the configuration, the system continues with the newly adapted weighting. The system functions as a self-adaptive system. The application can be used in large coefficient system, *e.g.* real-time compress sensing [89]. In the application, two dimensional local uncorrelated coefficients are needed for compress sensing. Previous works use DFF to locally store values which cost large area. With the proposed reconfiguration scheme, the new value can be self-calculated and stored in a fine-grained ReRAM system to generate a logic function.

Fig. 43 shows a mapping result of the internal configuration with the runtime function synthesis. Since the synthesis of logic function is relatively complicated, the resource within the FPGA are most occupied by the synthesis module rather than the targeted logic module. Only one tile highlighted with red is the logic function to be loaded and used there. The corresponding uBRAM showed in orange are accessed by the synthesis module for storing the runtime calculated configuration. Rest of them belong to lib-synthesizing modules which can fetch pre-synthesized modules to help on the runtime synthesis of functions. We can see that for function generation, large computation resources are needed. In this example, only 3.7% of the resource is for implementing the generated logic function. Similar reconfiguration procedure will need to be followed to accomplish the runtime functionality update. These two examples showed the different self-adaptive application mapping and demonstrate flexibility of the proposed architecture to support various reconfiguration requirement.

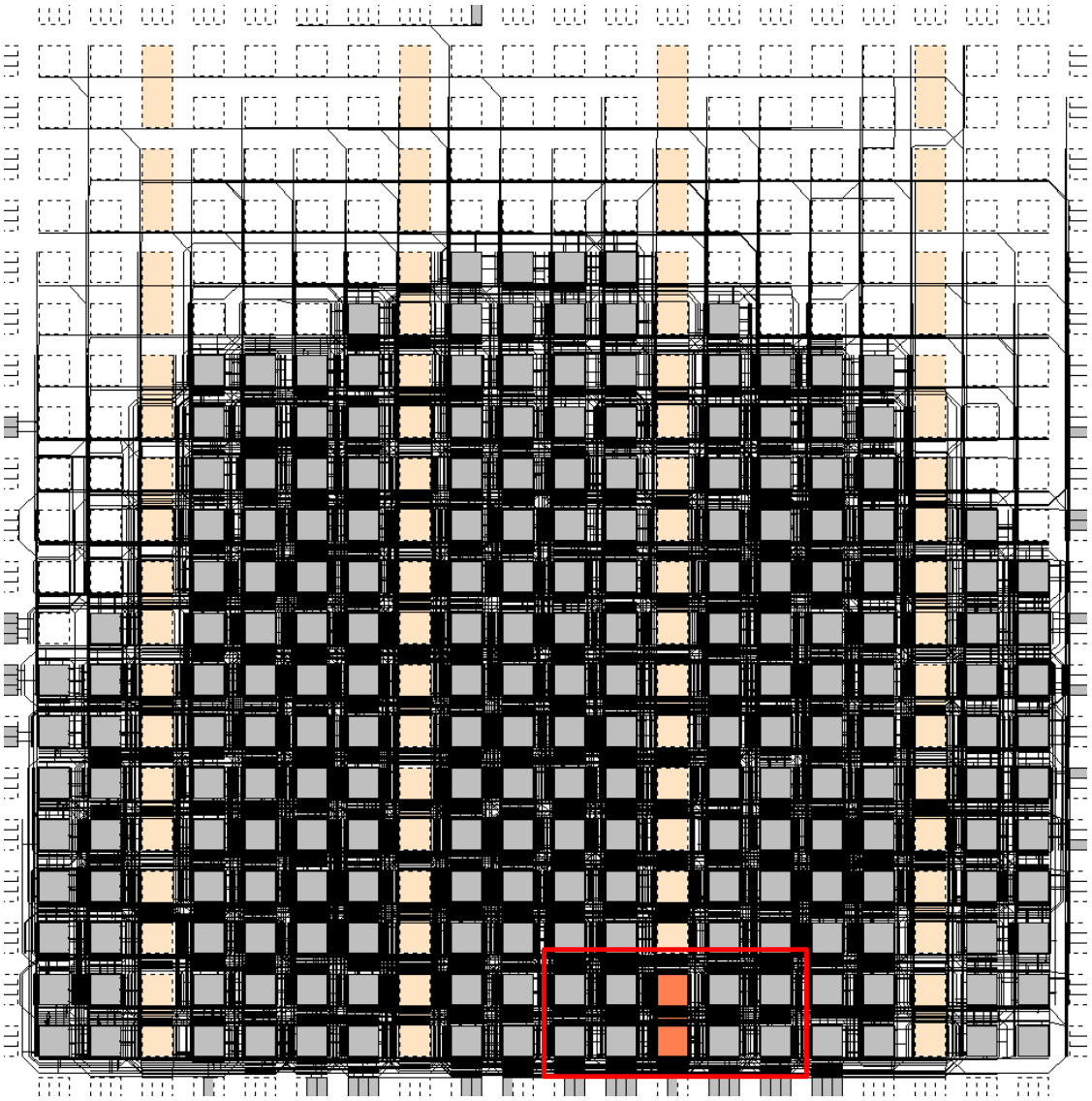


Figure 43: The internal runtime logic is generated to uBRAM for later configuring.

5.0 PLACE AND ROUTE OF PR FPGA – PR-AWARE ROUTER AND LOCAL APR

5.1 MAPPING FLOW

In the section, we introduce our CAD flow for PR logic in Fig. 44 [90]. The flow has two main parts: logic and place and route. Each part has further two stages in process. Generally, the flow is based on logic modules. Thus, we assume that designers decomposed PR logic into multiple modules at the beginning of logic synthesis. Since modern logic are composed of sub-modules, it does not need much extra effort by using the flow. Designers give a list of modules in usage and information of connection between modules to run the flow. For supporting PR logic, designers should also identify multi-context modules and provide corresponding contexts. Connection information between modules can be easily done by naming ports of the same connection into the same and the unique name. A useful tool of fast renaming can be found in *Verilog-Perl* [91].

At the stage one, the flow checks availability of physical information of each module in a logic library. The logic library contains pre-synthesized modules' information, *e.g.* delay, area, netlist, *etc.* The flow gathers information of modules in usage from the library to proceed to rest stages. Otherwise, designers should provide HDL of modules in usage, including various contexts, for a trial round to gather physical information. The second stage collects netlist files from modules and combines these netlist files into a new global netlist. Unlike general flow of VPR, there is no need of combining netlist from a top level HDL. It only takes physical information of connection rather than the whole logic.

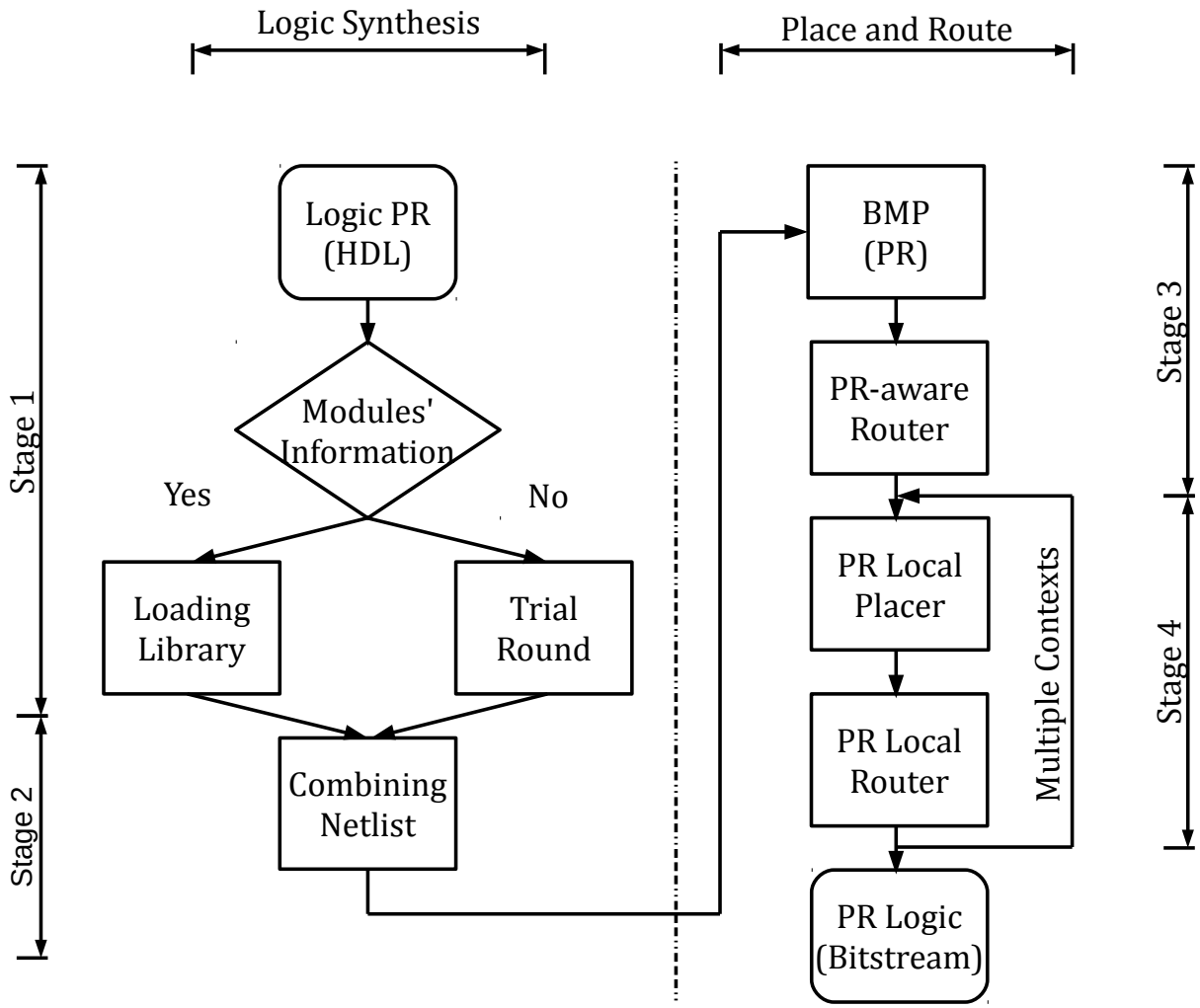


Figure 44: Mapping flow for PR logic.

In the stage three, it generates a global placement and routing of PR logic. A *B*-tree Modular Placer* (BMP) [90] is modified in the work for supporting multi-context modules. In the work, we add a context-selection mechanism, which identifies the featured context for a multi-context module. The mechanism chooses the largest area of the contexts as the featured context. By placing the modules with the featured context, it guarantees that all contexts can be placed in the designed region for the module. We also introduce *PR-aware router* to generates global connections. Comparing to general router of VPR, the main function of *PR-aware router* is to maintain signal integrity while system in PR operation. *PR-aware router* manages routing resource of the modules. Modules of one and multiple contexts has different routing resource and *PR-aware router* carefully avoid cross usage of resource while seeking solution from solution space for a routes. Thus, routes of a multi-context module would not affect routes of static modules in a PR operation, which guarantees signal integrity of the system. At the stage, it also extracts information of pseudo ports by using the technique, *port-over-track*, which are then used as source and sink for local routing. The forth stage is local place and route. To place and route a different context of a multi-context module into the PR FPGA, the only change is placement and routing of the designed region. Thus, to complete place and route, it has similar technique as we mentioned in the stage three by using resource management while seeking solution in solution space. Once we limited the resource, the stage is very similar to conventional/general place and route, however, the source and sink is from and to pseudo ports as source and sink, respectively. Then, the flow iteratively executes stage four until finishing all contexts of multi-context modules.

5.2 LOGIC SYNTHESIS

5.2.1 Preliminary Logic Library

As we mentioned in the previous section, the logic library stores physical information of various logic modules and we can reuse the information to fast generate place and route result for an application [90]. Of course, the information is for a specific architecture family

rather than a general architecture. For example, a synthesis result for an application on a 4-input LUT FPGA is definitely different from the application on a 6-input LUT FPGA. We have to re-synthesis all logic modules from HDL and acquire new physical information. Modern FPGA has architecture family and FPGA of the same family has similar fabric. They can share the same logic library. Fortunately, each FPGA company only has couples of family in the whole product lines. To create a whole library for all product lines is not a huge task.

Physical information for a logic module stored in the logic library is area, geometric shape, numbers of CLB, numbers of heterogeneous block, critical path delay, netlist, blif [92], position of IOs (ports), position of CLB, and position of heterogeneous block. Since our modules are all in rectangular shapes, area is different from CLB numbers because area of a logic module may include some waste area. However, we still need numbers of CLB while selecting featured context as a double check and make sure all context can be loaded into designed region decided by featured context. Critical path delay is used for fast estimation of delay cost in cost function while placing. As we showed in Fig. 45 (a) 1-4, modules of the same size (without no waste CLB) may have different geometric shapes. Through our exploration, we found out the same logic module with different shapes may have different delay and area cost [90]. To generate a placement as we would mention in Section 5.3.1, providing more selections of shape increases chances to have better area cost and delay. IO position is also needed for better delay estimation for fast estimation in cost function. Fig. 45 (b) illustrates wirelength estimation for delay with IO position in solid red line. Otherwise, we can only use Manhattan, center to center, for delay estimation in placement, which is fast but it is not very precise for a large module. Position of CLB is used for completing placement by mapping CLB on floorplan. Netlist and blif are for merging logic modules together and generating global connection among logic modules. In Section 5.2.2, we would detailed introduce merging technique. Numbers and position of heterogeneous blocks are needed to map heterogeneous blocks such as memory, DSP, *etc.*, from a logic module to global placement at specific locations. We only provide very limited function for mapping heterogeneous blocks by replacing all heterogeneous modules in the global placement regardless position information stored in the logic library.

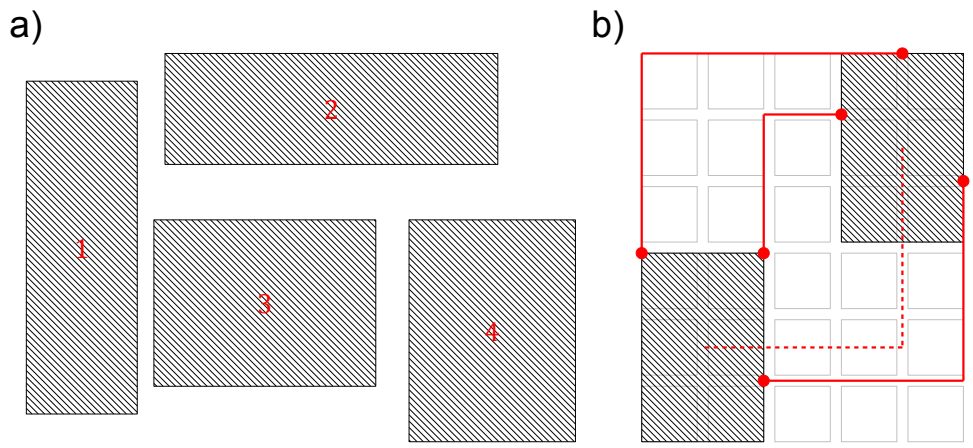


Figure 45: (a) Various shape of the same module. (b) Solid and dash red line shows wire-length estimation from two modules.

5.2.2 Netlist Combination Tool

With the stored information from logic library or a trial around, a complete netlist needs to be created to support global placement and routing by connecting all the modules. We introduce an auxiliary tool called *netlist combining tool* (NCT) to combine netlist from multiple logic modules to form a complete logic. Besides netlist or blif from each modules, the tool also needs a list of modules in usage and IO information of top-level to complete the task. Fig. 5.2.2 demonstrates a general flow of NCT. To create the top-level logic, the following steps are needed:

- S.1 Creating a list of modules.
- S.2 Collect and modify netlist or blif of modules.
- S.3 Identifying the IO of top-level logic.
- S.4 Combining modules' netlist.

S.1 is to identify which modules are needed for the function. The list of modules can be obtained by using the scripts VTR provided to parse the top-level logic file, where each module is a subfunction. Or, designers should manual input the list. To identify connection

between modules, name of IO (ports) of each module should be well designed in the stage since NCT would connect IOs with the same name among modules and recognize it as the only connection among the top-level logic. A fast renaming tool of IO for HDL, which is very helpful, is provided by *Verilog-Perl* [91]. Or, if the logic modules is loaded from logic library, swapping old IO name to new one in blif file is even easier. We collect and modify collected netlist or blif in S.2. The modification is needed because we have to give a specific name to each instance in netlist for avoiding repeated name among instances as well as fixing legacy naming rule of VPR. A patch is provided for blif in NCT. It checks all random names created by the VPR and traces corresponding nodes to name a specific and meaningful name. For some nodes as internal nodes without connection from source or to sink, it adds-on the module name after its original name for avoiding repeating name in top-level netlist. Here, it does not rename IO of logic modules since name of IO is linked for global connections. Top-level IO is introduced in S.3. The information should be provided by system designers. It should follows the same format of netlist. NCT would directly load it to the global netlist without modification. In S.4, NCT gathers all netlist of logic modules and IO definition to create the top-level netlist. NCT also support PR applications to create the new netlist after PR operation if global netlist is in need. Designers can simply swap netlist of new context to part netlist belong to the old context. Though the overall optimization is not performed in the step, the step is needed for giving the detail connection information for estimating wirelength in the next stage of placement.

Note that in the original flow of VPR, combining logic function needs integration of the modules verilog codes to get a complete logic function. Now, for a library based design, designers do not need to get the original verilog files for combining the netlist. It saves synthesis time and provides more flexibility. To design a new function, designers only needs to list modules in usage, provide connection port name and top module IO information, which are highlighted in red of Fig. 5.2.2. Then to finish the design, only results of modular place and route is needed. For example, designers can fast create a new function by merging multiple IP-cores into a new function. Generally, designers has no idea of detailed logic design of IP cores but know function of IP-core. Linking all IP-cores together is easier and faster than building logic from HDL in design consider point of view.

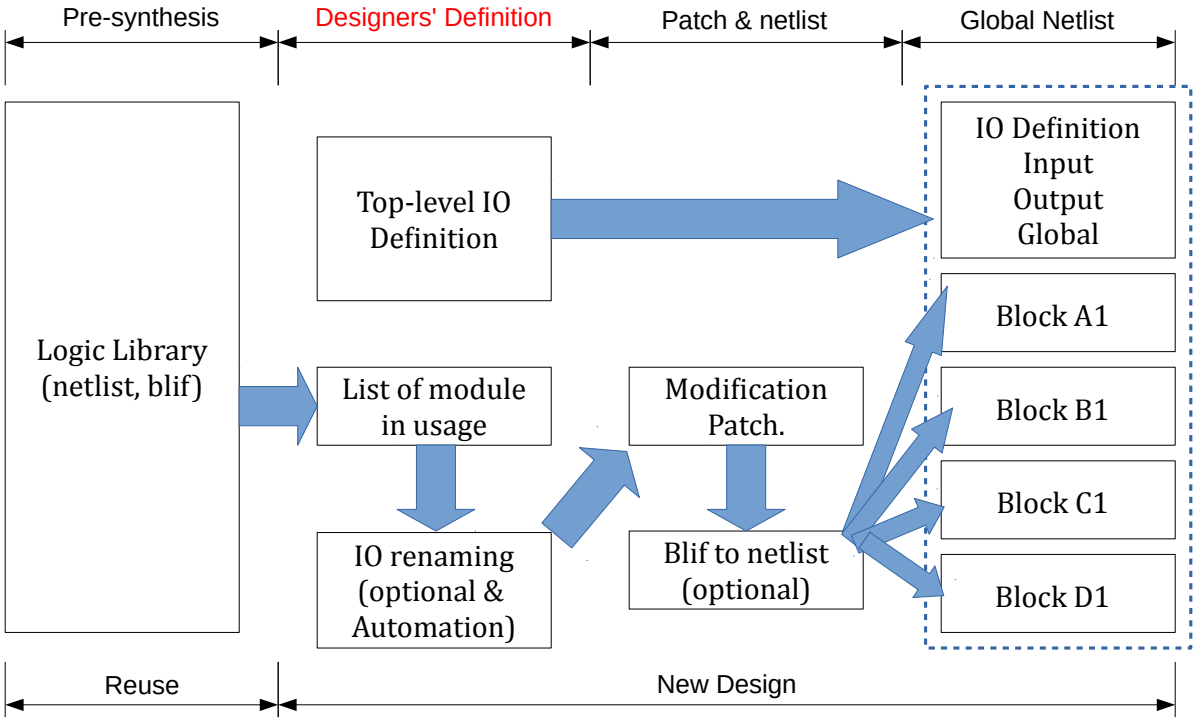


Figure 46: A general flow of NCT.

5.3 PLACE AND ROUTE OF PR

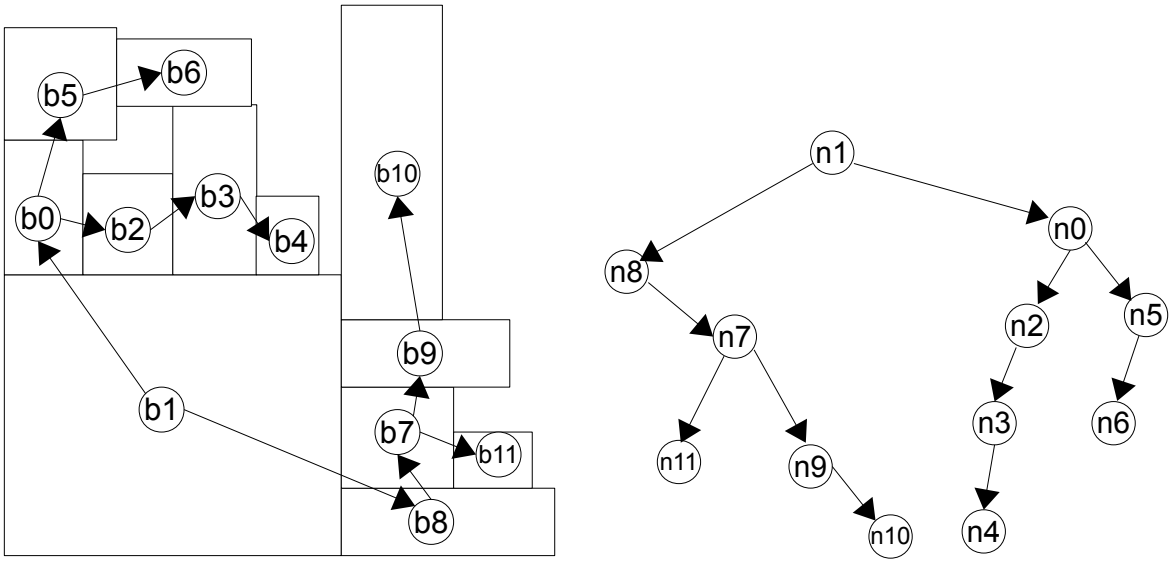
In the section, we would go through detailed design ideas of the place and route flow for mapping PR logic on VPR. Referring to Fig. 5.1, the place and route flow is from stage three to stage four. We start from modified version of BMP supporting PR, exploring types of PR routes, *PR-aware router*, to local place and route.

5.3.1 Preliminary - BMP

BMP was introduced as a new type placer for placing heterogeneous modules based on VPR [90] and logic library. It supports heterogeneous logic blocks while original VPR only supports homogeneous logic blocks for placement [16]. The heterogeneous logic blocks may have various shape. size, delay, *etc.* Different from previous works, BMP has B*-tree representation [22] for improving searching in solution space. Fig. 47(a) illustrates a pseudo floorplan result. Based on B*-tree representation, it can be converted into a binary tree demonstrated in Fig. 47(b). For a logic module in floorplan, which is represented as a node, may has an adjacent module at right side and the other adjacent module at up side representing as left and right son in the binary tree, respectively. For example, *b1* module has modules of *b8* at right side and *b0* at up side. In B*-tree, *n8* and *n0* are left and right son of node *n1*, respectively. By using the B*-tree representation and corresponding physical information from the logic library, and cost function [93], BMP completes floorplan.

Fig. 48 shows a floorplan of BMP with benchmark *boundtop* and Fig. 49 demonstrates the the mapping result from BMP to VPR. The position of CLB from each module is loaded back from logic library. As demonstration in Fig. 49, placement may contain some waste area. There are two type of area, blank tile of logic module and modular placement incurring waste area. One way to ease the penalty is through modifying parameters of cost function. It makes BMP run more iterations to cool down in simulated annealing [90].

Modular placement is important for a PR logic function since it helps identify PR regions, manage routing resource in advance, and maintain signal integrity while PR operation. With modular placement, it is easier for designers to swap a new context to the targeting



(a) A non-overlap placement.

(b) The (horizontal) B*-tree.

Figure 47: A placement and corresponding B*-tree representation.

area for PR operation. Signal interference happens at routes between modules of one and multi-contexts. While PR operation, if routes pass through one-context modules, the routes corresponding route components, *e.g.* switch and connection blocks, are also configured. Designers should avoid these configurations since we have maintain signal integrity of one-context modules. Therefore, modular placement is a solution to maintain signal integrity by constraining routes of multi-context modules in the designed regions. Once it needs connect to other modules, no matter modules of one or multiple contexts, it connects through pseudo ports that we would introduce in the following sections.

5.3.2 BMP supporting PR

As illustration in the Section 5.3.1, we know that BMP [90] supports placement of heterogeneous logic blocks, a.k.a. logic modules, on VPR [16]. To further support PR logic, BMP should access physical information, especially CLB numbers and area of logic blocks, of all contexts of all modules. For dealing modules of one and multi-context, we discussed them

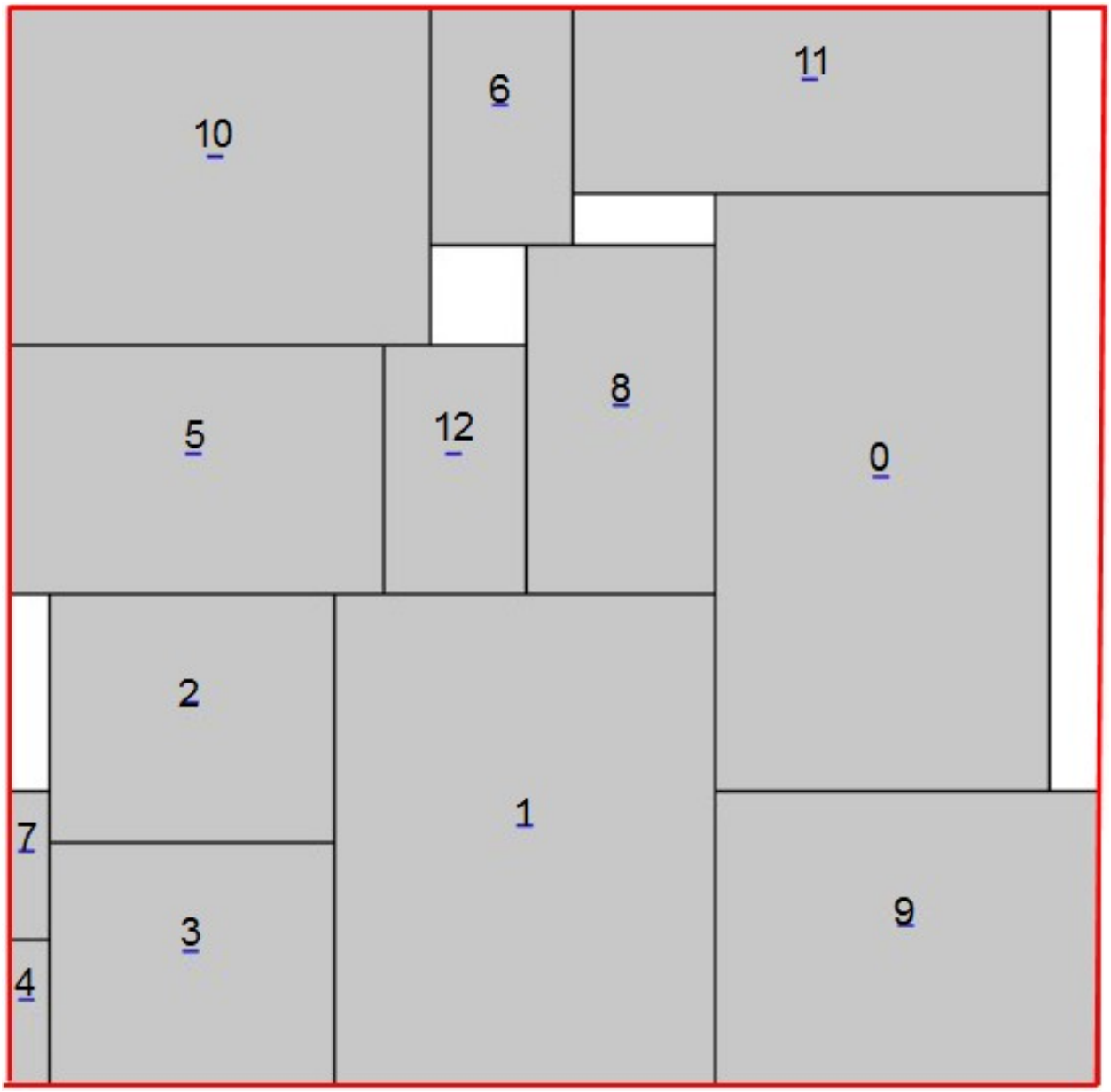


Figure 48: Boundtop floorplan.

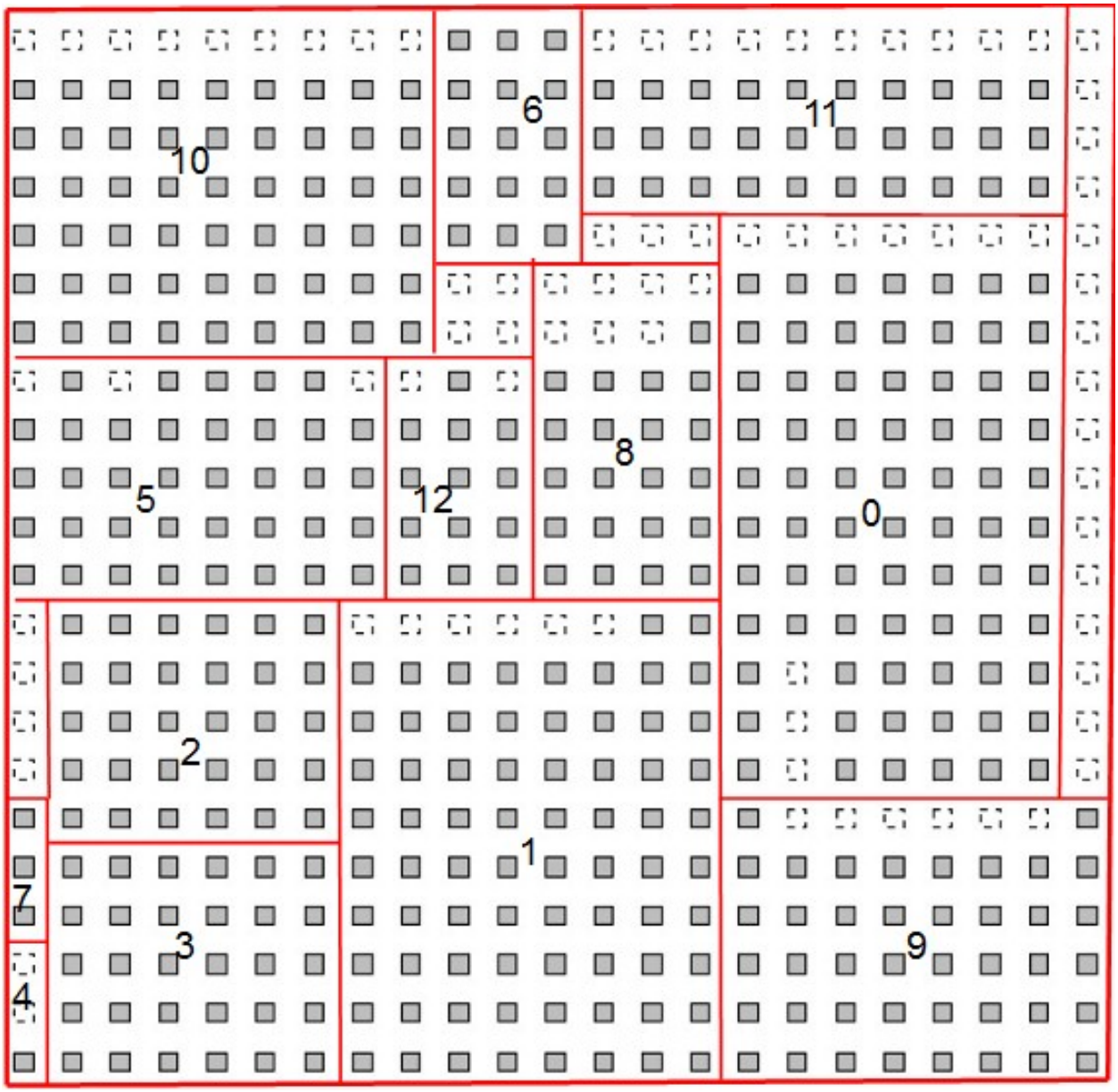


Figure 49: Boundtop placement.

separately and derive the work from the original work of BMP [90]. For modules of one contexts, it is the same as the work of BMP [90]. It loads physical information fro logic library to complete the work. For multi-context modules, an extra selecting mechanism is kicked in to determine the featured context of each module. The featured context represents the module with corresponding physical information while running global placement. Therefore, it should guarantee that all contexts is able to be loaded in the designed region after global placement. We applied an area-dominate mechanism to select the featured context among various context of the multi-context module. The mechanism takes modules' size rather than CLB numbers as the condition. It is because a logic module has the same number of CLBs but different area in various shapes caused by waste area. Therefore, the largest area guarantees all contexts with various shapes can be loaded in the designed region. Of course, there is a double check mechanism to make sure that CLB numbers of different contexts is smaller or equal to the designed region.

5.3.3 Exploring Types of PR Routes and Port-Over-Track

As we discussed in the previous section, signal integrity while PR is an issue for designing CAD supporting PR. Otherwise, it might accidentally change signal of static function and cause unknown, wrong, or unstable signal causing system failed. To maintain signal integrity while PR, we start to avoid the situation from very beginning stage of placement and routing. Modular placement completed by BMP helps identifying regions for multi-context modules. Thus, to manage routing resource becomes easier by restricting designed related resource. In the routing stage, we search the designed region related routing resource and manage usage of the resource under different connection condition. Routes may have different types of connections resulting different routing resource of solution space. In the section, we discuss the types of connections and provide port-over-track, which acts as quarantine or abutment for routing between different types of connection, to manage routing.

Fig. 50 shows a FPGA having 6x6 *logic element* (LE), routing tracks, and IOs. It has static logic regions and two PR regions. In the routing stage, even there is multiple regions of static logic, we discards the definition of static regions but take all static LEs as the

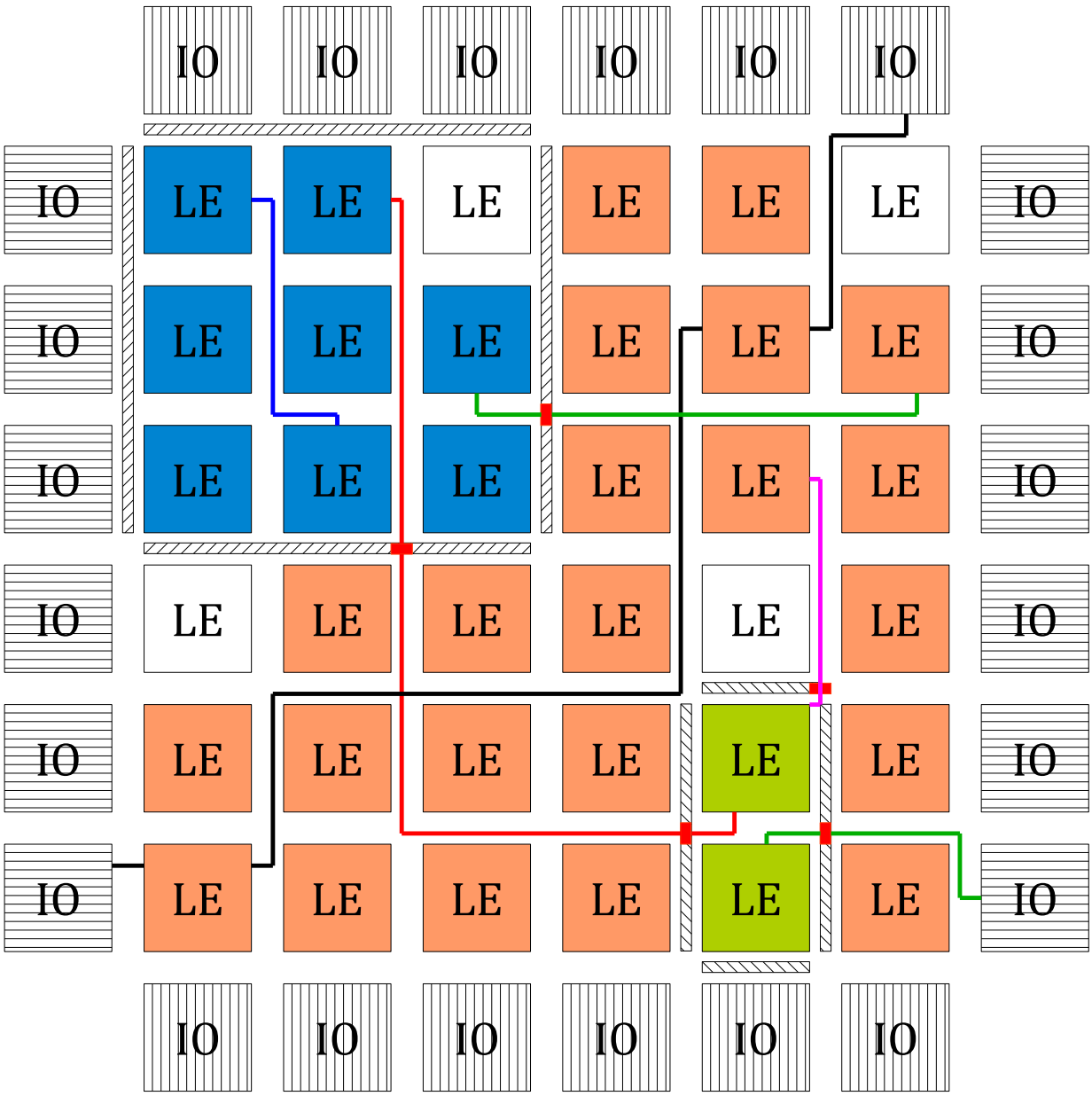


Figure 50: Types of connections.

static type to simplify the routing process. In Fig. 50, LE in orange color indicates that LE is in usage. For LE in PR regions, LE in blue and green indicate they are two different PR logic functions located in different regions. LE in white is blank LE, which have no configuration. Types of routes are classified into static to static, static to PR, PR to static, and PR to PR. A tile located in a static region and to be connected to another tile located in the same/another static region belongs to type of static to static connection. We illustrate routes of the type with black lines in Fig. 50. To route a connection of the type, it is similar to VPR with constrain of extra resource confinement. The routes have to avoid regions of PR, which are blue and green regions in the figure.

Routes of static to PR or PR to static are similar. They are showed as magenta and green lines from different PR regions in Fig. 50. While the source is a tile located in a static region, it is not directly routed to a tile of sink located in a PR region. At beginning. the route has resource of all of static regions for routing. Once it has to connect to any resource within the PR region, the routing resource would be limited to the PR region for maintaining signal integrity while PR. At the boundary between resource of static regions and the PR region, we introduce pseudo port as abutment by technique called *port-over-track*. We would detailed illustrate pseudo port later in the section. For PR to static routes, the scheme is quite similar. At beginning, resource is limited to the PR region. After it traverse though pseudo ports, it has full routing resource for all static regions.

PR to PR connection is further classified into two sub-type: two tiles in the same or different PR regions. For a routes of two tiles of the same PR region, the router works as VPR with resource limited to the same region. The blue route showed in Fig. 50 belongs to the type. It can only use the resource within the regions. Rest of the resource, including static regions and other PR regions, is forbidden in the routing. A net of two tiles from different PR regions needs two nearby pseudo ports as source and sink for the route. We showed it as the red route in Fig. 50. Again, pseudo ports are applied for connections from PR regions to static regions, or vice versa. For the part between two pseudo ports, it is the same as static to static connection. Resource confinement of PR regions still applies on it.

From the above discussion, we know any routes traversing through both PR and static regions need a quarantine to maintain signal integrity while run-time PR. We introduce pseudo ports as the quarantine. In Fig. 50, thick lines with hatching around the PR regions are quarantines and red points upon them are active pseudo ports. Pseudo ports separate static and PR regions. A route passing through two regions has to be connect at a pseudo port. From hardware point of view, we introduce technique *port-over-track* as pseudo ports. We use tracks as abutments for connecting two type of regions. To access the tracks, designers control connection blocks to access these pseudo ports. These ports can be used as either pseudo source or sink in the routing. While run-time configuring logic, it only programs tiles [15] of a PR region. Through well management of routing resource by avoiding tracks as pseudo ports, it guarantees signal integrity of static routes while run-time PR.

5.3.4 PR-aware Global Routing

We introduce a new router called *PR-aware router* with resource management for seeking solution in solution space. It is slightly different from the router of VPR by limiting resource of PR regions and adding a new type of tracks acting as pseudo ports. After BMP finishes floorplan and mapping the result back to VPR for generating placement, the flow kicks in *PR-aware router* to complete global routing. Input files are the top level netlist creating by NCT, the placement with featured contexts of multi-context modules, and the list of PR modules.

Algorithm of *PR-aware router* is summarized in the Alg. 1. For each net, it starts from the source of the net and searches its neighbor nodes to reach the proper sink of the net. Each neighbor node is picked up according to the evaluation of cost and *PR-aware router* checks whether the node is capable for the route. Here, *PR-aware router* kicks in the routing resource management for checking the routes. If it finds a proper route, it records the current node and search next node for another route. Otherwise, the tool discards the node and choose another one. Besides original definition of source and sink in VPR, *PR-aware router* also takes pseudo ports as sink or source in searching solution. The situation happens if it has a source node located in the static and seeks sink node inside the PR regions.

Then, it seeks the ports name of the PR region, by using pseudo port of port-over-track, as substitution solution. The reverse step of seeking sink from a node located in PR to static regions is similar. As we summarized in Section 5.3.3, *PR-aware router* generates a route based on routes' classification and rules. Since *PR-aware router* limits some resource, possibility of routing fail is larger than general routing. Moreover, *PR-aware router* has an extra checking step to check signal integrity after finishing routing stage. It simply checks if routes stay in the same type of region. For example, if a route is from a source to a sink within a PR region, the routes should not pass through any static regions nor other PR regions.

We demonstrates routing results of two extreme cases by *PR-aware router*. For routing considering no PR regions, which indicates all of the modules are static, is showed in Fig 51. The router operates in certain case is very similar to VPR. The difference is that the router supports the flow of modular placement. For the other extreme case, we assume that all the modules are PR regions demonstrating in Fig. 52. Once we preset all regions of logic modules are PR regions, the routing resource are very limited. Any resource inside of any regions are forbidden for global routing between modules. Therefore, available routing resource are tracks between modules, tracks closing to IO, and tracks of blank tiles. Tracks between modules are generally meant connection through pseudo ports here. They are connection abutment, which can be used for connecting routes going into modules or out of modules. Therefore, the abutments does not cost much resource on tracks compared to separated in and out special tracks. For modern FPGA architecture with direct connection [94][66] to adjacent logic blocks, resource confinement incurring congestion along tracks can further be improved. Even though tracks closing IO might also be adjacent to PR regions on the other side, they are like free tracks without resource confinement since IO are always taken as mutual type, not static nor PR. For tracks around blank tiles, here it is waste area caused by placement, the resource is like static region without any confinement. Through comparison of Fig 51 and Fig. 52, it shows that routes congest around modules, which is benchmark *boundtop* showed in Fig. 49. Since all connections between modules are crowded into limited routing tracks, time-driven algorithm for routing used in VPR may not be not applicable under the certain case. Congestion-driven algorithm to improve routability might be proper.

Algorithm 1 PR-aware routing Flow

1: **get** module-based placement result, reconfigurable regions information and top-level netlist
2: **for** current iteration number less than maximum iteration number **do**
3: **for each** net[i] of the net set **do**
4: **check** the source and sink of net[i]
5: **For case 1:** if source and sink in two reconfigurable regions A and B respectively
During route, the path, from source to sink of net[i], can go through PR region A and B and all other static regions, but can not go through any other reconfigurable regions
6: **For case 2:** if source and sink are all in a reconfigurable region A
During route, the path, from source to sink of net[i], can only go through PR region A, but can not go through any other regions.
7: **For case 3:** if source in the PR region A, but sink in static region C
During route, the path, from source to sink of net[i], can go through PR region A and all other static regions, but can not go through any other reconfigurable regions
8: **For case 4:** if source and sink all in the static regions
During route, the path, from source to sink of net[i], can go through all the static regions, but can not go through any other reconfigurable regions
9: **end for**
10: **check** whether route successful or fail (check resource overused or not)
11: **If** route successful and meet the requirement, stop
12: **end for**
13: **If** route successful, get the route result
14: **else** route fail

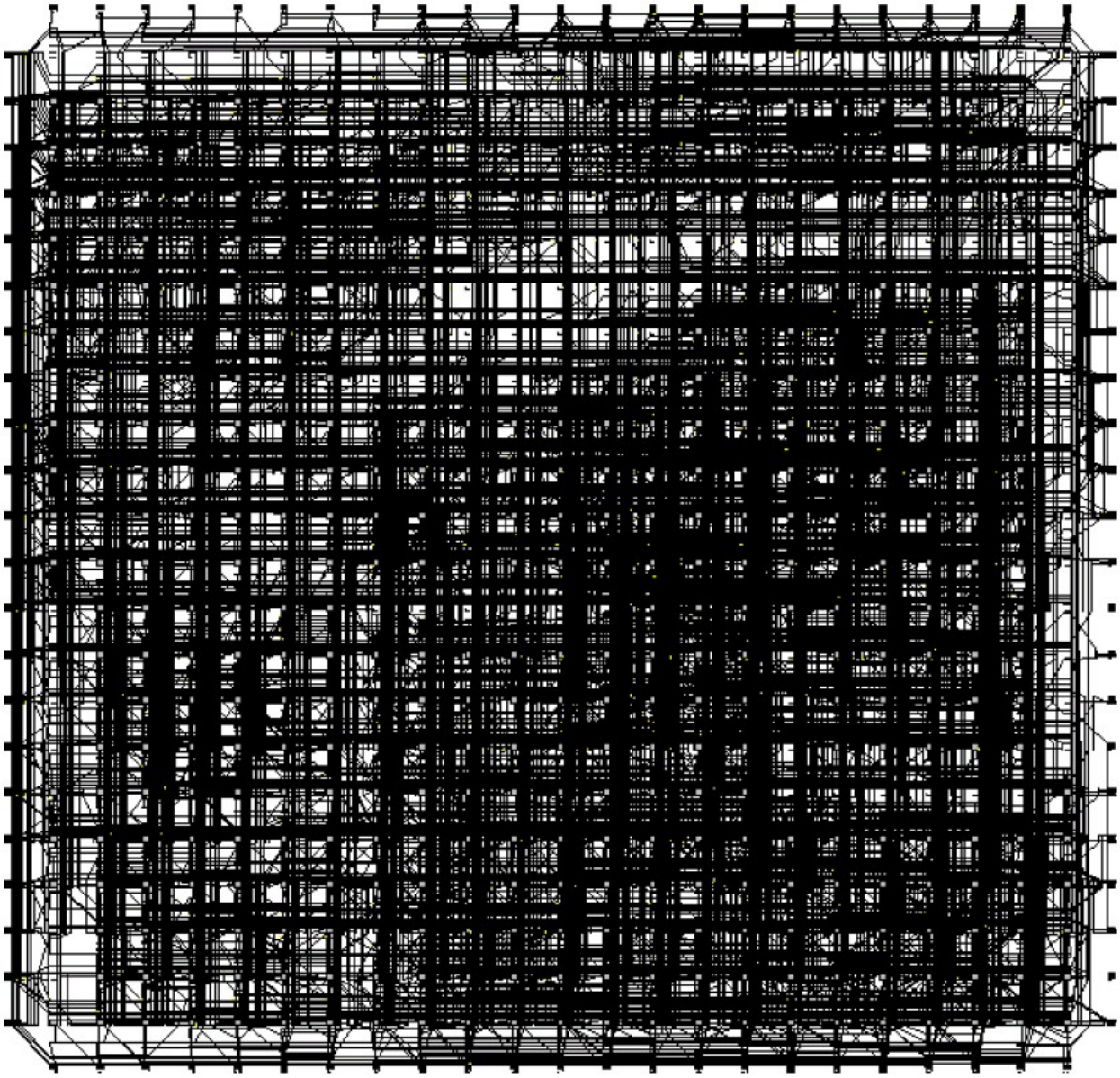


Figure 51: General.

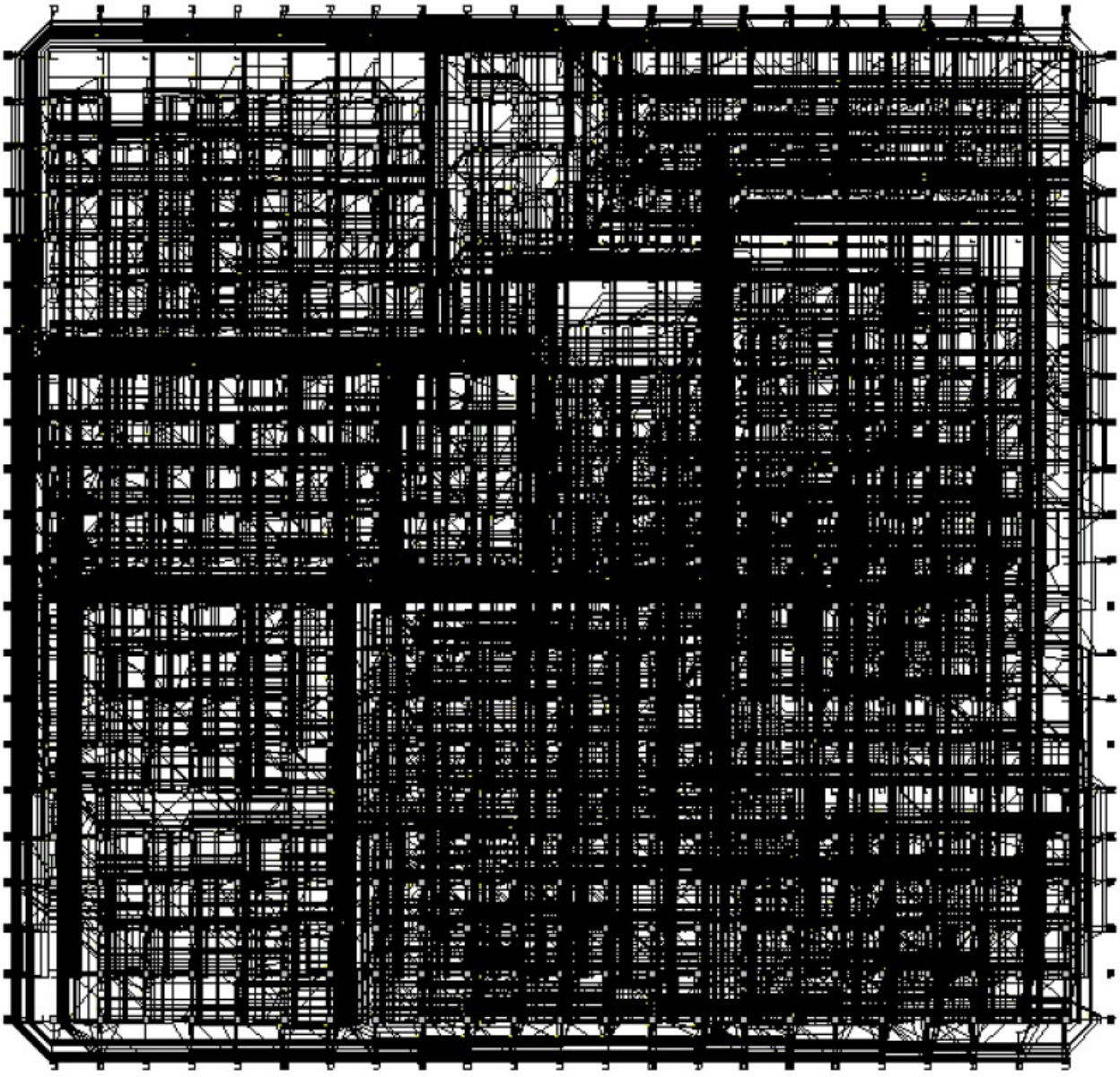


Figure 52: PR-aware.

5.3.5 Local Place and Route

After finishing the global placement and routing with static modules and PR modules of featured contexts, we then perform local place and route to optimize PR modules with various contexts in stage four. The tool select one of unfinished PR module, and, each time, it swap a new context into the designed region. Within the designed region, the tool place CLB first and then routes connections. After it finishes the current context, it processes next context until all contexts being done. Then, it iteratively go though every PR modules of the design. Information of pseudo ports from previous stage is applied here as source and sink for local place and route. The idea is different from conventional routing by searching source and sink from IO. The are all substituted by pseudo ports for local routing. From PR pointing of view, we only need the specific physical information and logic content, which can be loaded from logic library, for swapping a new context for an old one of a PR module. From system point of view, designers only need to send the piece of bitstream to complete PR operation.

5.3.5.1 Local Place To perform local place for PR regions, we need position of pseudo ports of a PR module and contexts' netlist or blif. Here we provide two methods for local placement: modified VPR and B*-tree. For the method of modified VPR, we create a VPR architecture with the same area and aspect ratio as the PR module. Position of the pseudo ports of the PR module is also mapped into VPR as IOs of the floorplan. Cost function and simulated annealing follows general VPR's setting for optimizing placement. Thus, VPR places CLBs as general flow with constraint of preset IOs position, aspect ratio, and netlist or blif for connection information. The second approach is to use BMP as a independent placer. It also loads information of IOs position and aspect ratio. However, the approach reuse BMP rather than VPR. It omits operations of reshape and rotation [90] since CLBs are homogeneous in FPGA. Aspect ratio confines available floorplan area and shape. Connection information are still needed by using netlist or blif. BMP optimizes the placement according wirelength, delay, *etc.* After the floorplan generated by BMP, it maps the result back to VPR as the new placement for the PR module.

5.3.5.2 Local Route After the flow finish local placement and maps it back to the top level placement, it then routes connections by a modified router of VPR. The difference of the modified router is that pseudo ports as source and sink of some nets are introduced to the resource management. Inputs and outputs of a PR module act as source and sink, respectively. Also, resource outside of the designed region, except tracks for pseudo ports, for the PR module is restricted because we do not want routes to traverse outside of the PR region. For each net, the router checks available routing resource of the PR module and seek solution for successful routing. Here, it generally applies time-driven algorithm [16] for all case as VPR.

5.3.6 Results and Demonstration

In the section, we show place and route of the PR mapping flow step by step. Benchmark *boundtop* is demonstrated with a PR logic module of two contexts. FPGA architecture is set to 45 nm technology with LE of four 6-input LUT set with 200 length-4 tracks [69][84]. To simplify demonstration, there is no heterogeneous block, such as memory and DSP block.

Fig. 53 shows global placement of benchmark *boundtop*. The placement is assembled from logic modules. In the figure, we can roughly identify each module's region. There are white CLB on the placement indicating waste resource, which is inevitable for modular placement. The region highlighted in red is the only PR region. The following steps focus on the region.

Fig. 54 is global routing of benchmark *boundtop*. The PR-aware router has routing resource confinement at the PR region. It first routes all connections outside of the PR region, which is known as static region, and then it routes the internal connections of the region. The blue region of the Fig. 54 is the PR region. The routing is not condense since only the PR logic could be routed inside the region. Between the red and blue lines, it is routing channel around the PR region. The routes are intense since they have to pass through PR region without go insides the region. Outside of region highlighted in red is similar to the non-PR routing and relatively relax comparing to channels around the region.

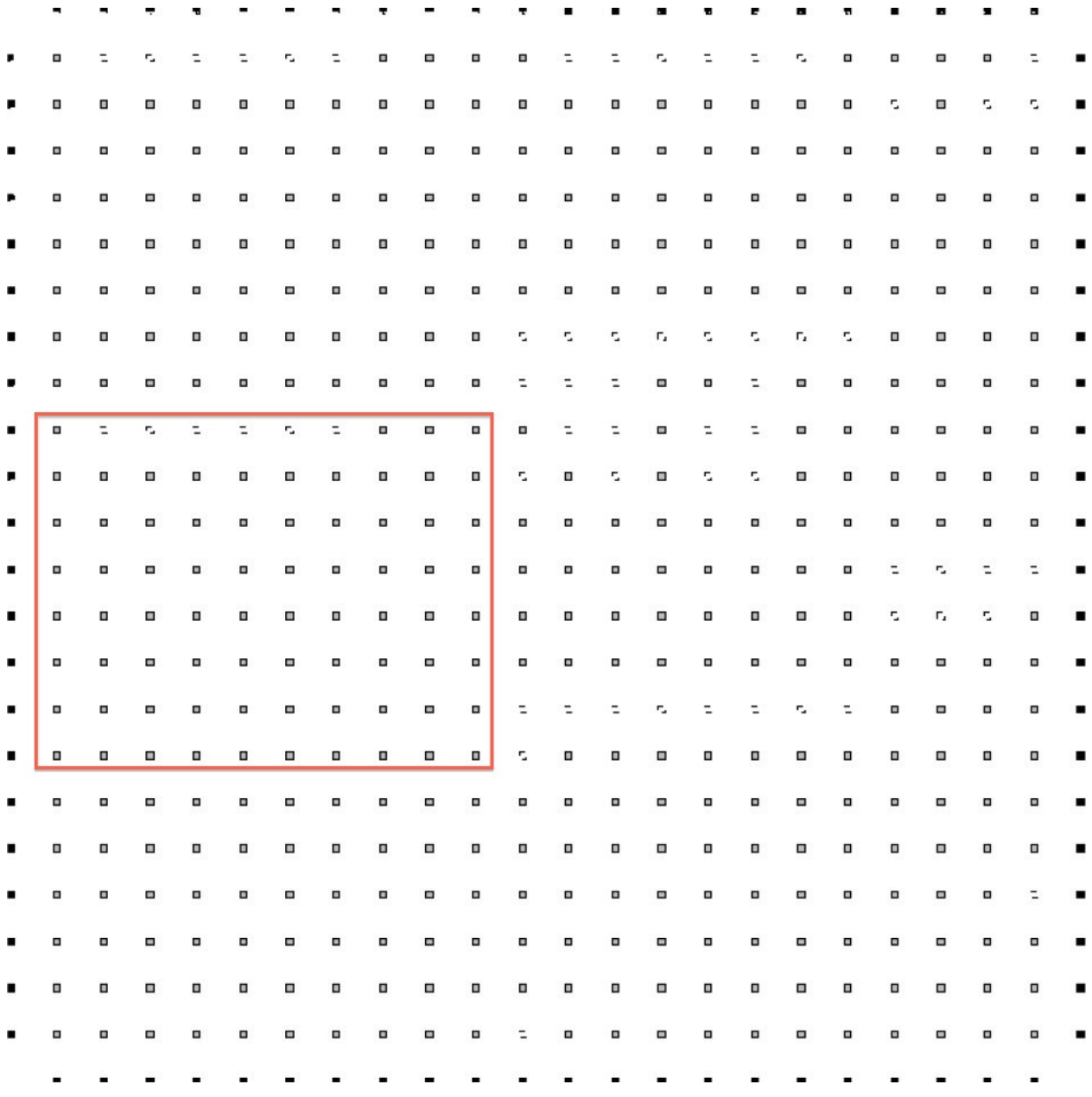


Figure 53: Global placement of PR boundtop.

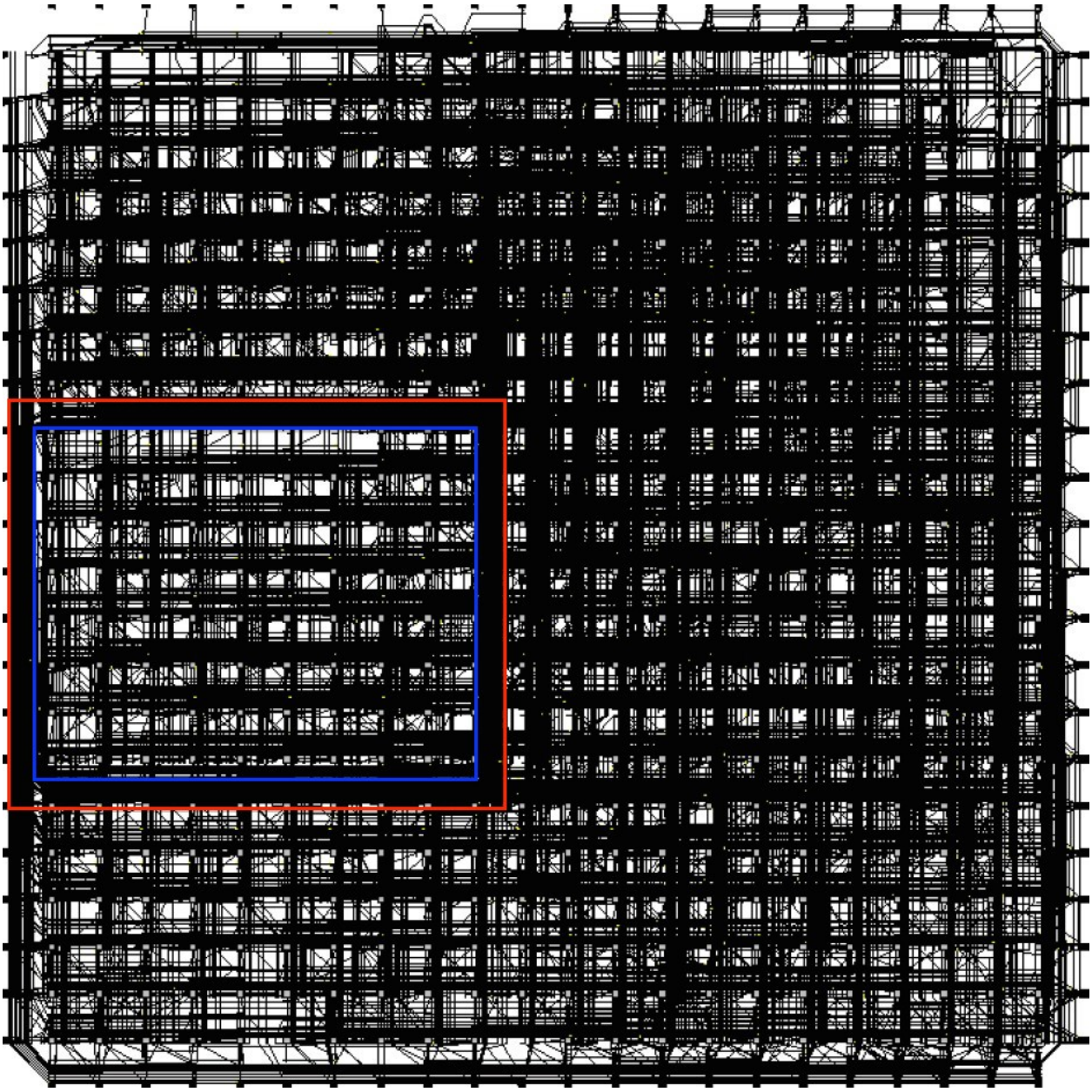


Figure 54: Global routing of PR boundtop.

Here we demonstrate to load a new context into the PR region. Fig. 55 shows the placement of the new context. Obviously, it only needs place CLB of the new context to the designed region rather than the whole FPGA. As demonstration, the CLB is close to boundary of the region. It has pseudo ports just around the boundary to connect source or sink out of the PR region.. So, CLB would be placed into available resource close to the boundary for better optimization. Fig. 56 demonstrates routing results of the current context. As we mentioned, there is no need to route the whole FPGA. For local routing, intense routing happens at tracks close to boundary. Because the flow has good optimization on placement, routes around boundary with shorter length are good enough for routing optimization. Meanwhile, because the PR region has limited resource for source and sink, which are pseudo ports, most connections go through these ports incurring intense routing around boundary.

5.3.7 Verification of the Work

The proposed scheme is based on VPR [16], where the netlist, placement, and routing files are done using VPR's rules, which can be briefly summarized as follows:

VPR first checks legality of the netlist by checking for duplicate declarations of blocks, block types, and IO status - such as floating inputs or outputs, multiple connections to a single output, duplications of an IO pad, *etc.* The corresponding rule definitions are located in the file at '*base**check_netlist.c*.' A similar check is performed on the placement results, with the rules stored in '*base**read_place.c*' used to read in and check the placement result for legality. For example, duplicate name of blocks, illegal subblocks based on the architecture, *etc.* would all fail the check. After this step, the placement results generated by BMP would be taken as nodes in the routing graph. VPR's router then checks legality of all routing resources before generating routes. Here, we provide some information regarding routing rules in VPR. The VPR routing rules check are stored in '*router**check_router.c*' and '*router**check_rr_graph.c*.' It reuses the netlist again to check legality of the routing resource. The placement result is verified in this stage to make sure that it matches the description in the netlist file. In VPR's source files, it mentions that the router checks two

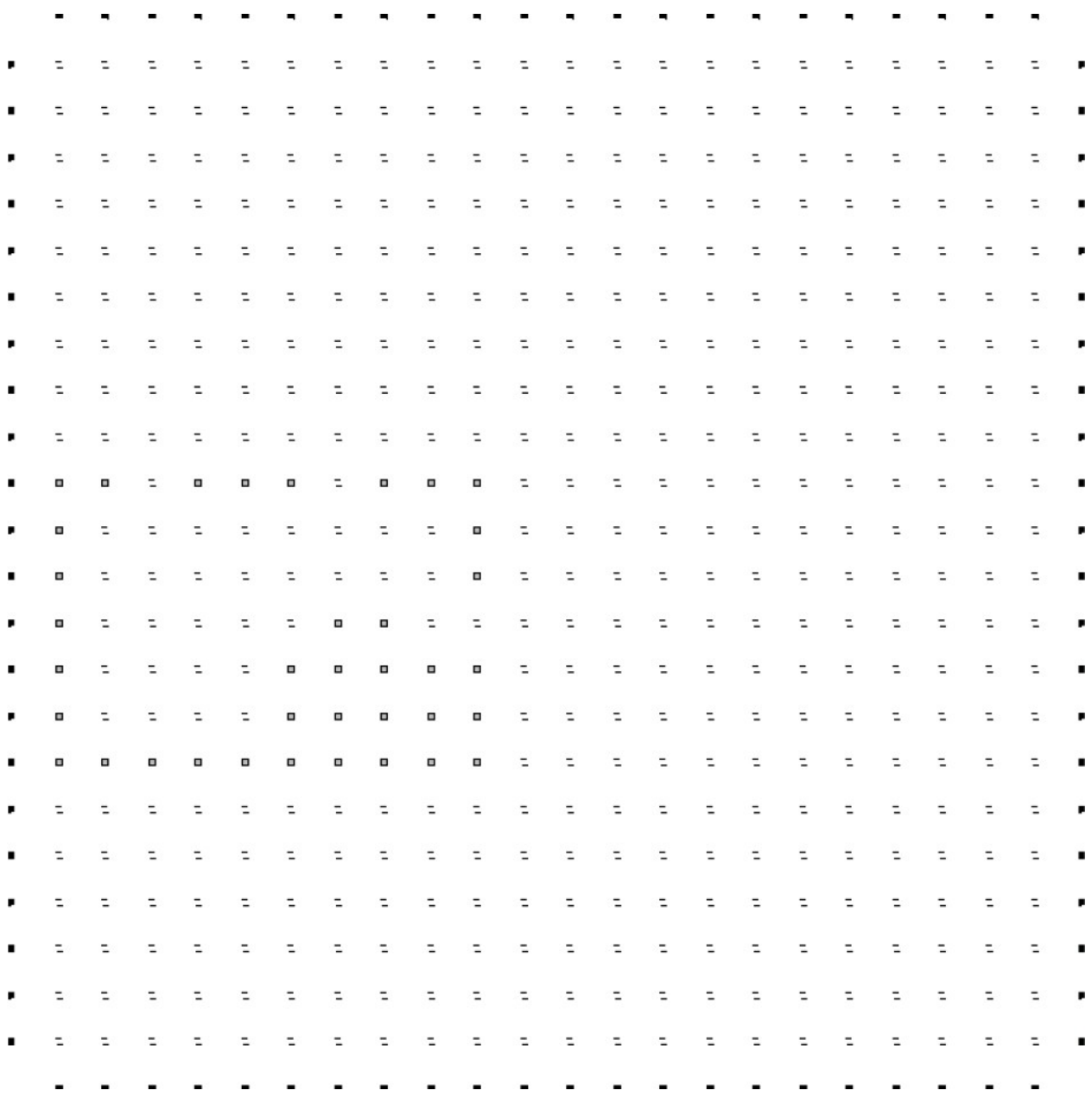


Figure 55: Local placement of a new context of the PR region.

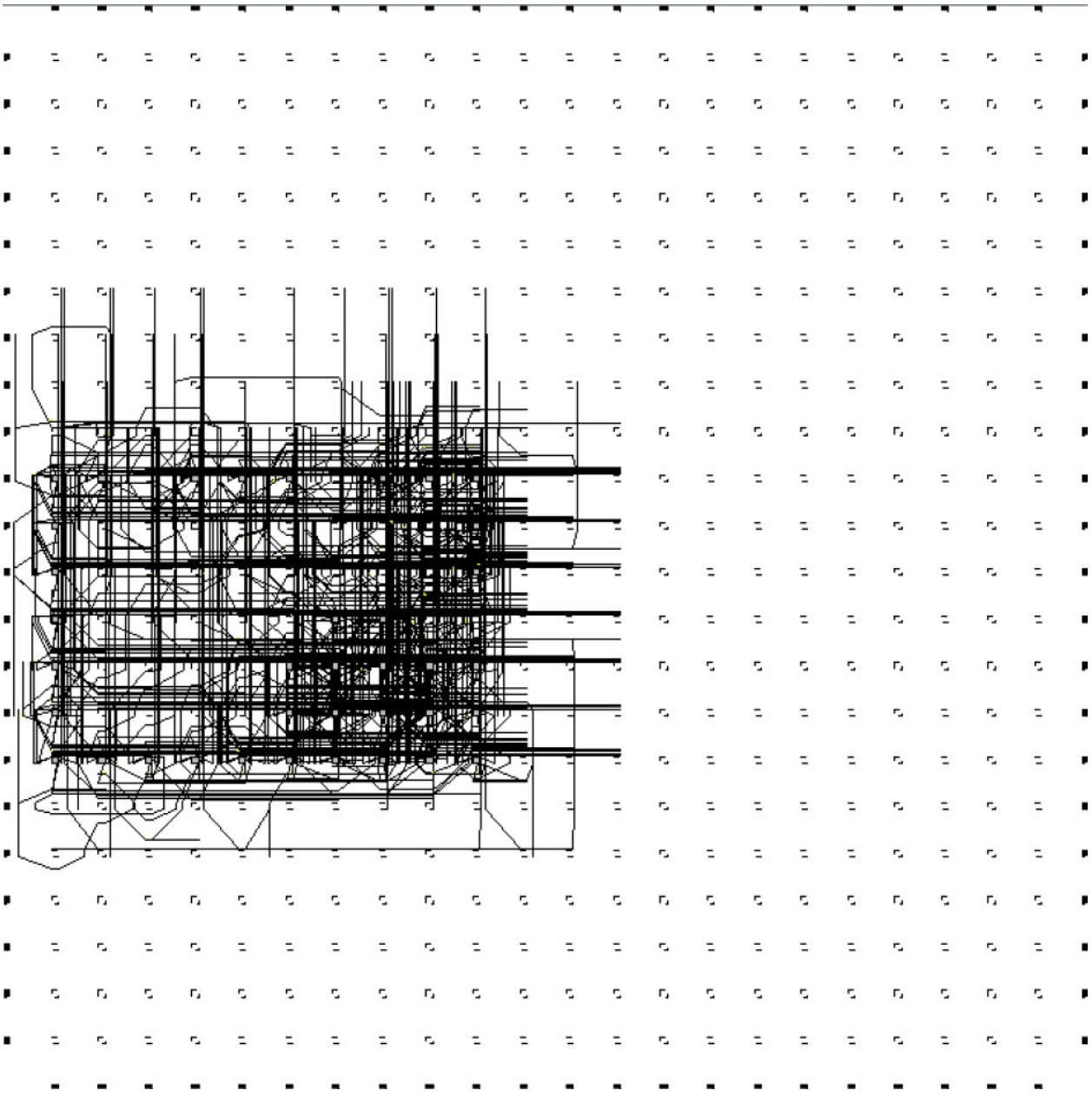


Figure 56: Local routing of a new context of the PR region.

major rules: proper description for each net in the netlist files and proper connections of pins spanned by the net. It checks a sink for two things: early termination of the sink without connection to IO pad and no sink for the net. For the source of a net, it checks if a source is valid for the net and if there is any fan in for the net. Similar to sink check, the source should be from IO pad at some initial point. With all above rules of netlist, VPR guarantees the netlist integrity before it actually generates routes. After the final routing, VPR checks if it has exhausted net and routing resources. The corresponding rules are stored in the '$\backslash router \backslash route\_common.c$'. The step guarantees that the routing matches description of netlist. Otherwise, it shows error to tell the user the resource is not empty indicating that it does not match the netlist. Since our routing result passed all checks from VPR, we can assume that the routing result is correct and matches the netlist.

6.0 CONCLUSION AND FUTURE WORKS

6.1 DISSERTATION SUMMARY

Modern FPGAs introduce various opportunity of post-fabrication reconfigurabiliy for system design. Our work demonstrates improvement of large area cost introduced by memory system of modern FPGAs. We start from a monolithic stacking memory with an emerging nonvolatile memory device, ReRAM. Then, we introduce peripheral circuit designs of FPGA's components, *i.e.* LUT, CB, and SB, as well as uBRAM for both configuration and temporary data storage. At the final part, we introduce CAD tools to support PR FPGA for evaluating architectures and mapping applications.

In Chapter. 3, we proposed a 3D stacking structures built upon bipolar ReRAM crossbar arrays, called 3D-HIM. The design is performed by alternating the deposition of ReRAM materials in forward and reverse sequences. As demonstration of the simulation results, the interleaved structure helps to maintain sensing margin and proper programming voltage while suppressing impact of sneak paths and leakage current. Compared with other ReRAM structures, the proposed designs have advantages of simple fabrication and higher memory capacity. Intuitively, 3D-HIM can be utilized in any bipolar ReRAM, especially those materials with a higher resistance of LRS are preferable.

Chapter. 4 demonstrates a ReRAM FPGA with uBRAM, which is a combination of FLASH of configuration storage and BRAM for temporary data storage. LUT and uBRAM are based on 3D-HIM structure to save area cost. CB and SB are based on CRS structure to maintain performance and share peripheral circuit in a crossbar for saving area cost. With uBRAM as internal memory, the two stage configuration scheme is introduced to perform fast PR. The data can be load into uBRAM in the first stage without stalling the current

computation. In the second stage, it configures the FPGA in a very short time. Compared with SRAM FPGA, ReRAM FPGA has 62.7% area reduction and 34% delay decreasing. We demonstrate external and internal PR, which shows greater design flexibility than previous manners.

Chapter. 5 demonstrates a PR mapping flow and corresponding tools for place and route of PR FPGA. The tools are BMP and PR-aware router to support modular place and route of FPGA. It is the first design automation PR flow and tools for academic usage. Meanwhile, we minimize designers' efforts of manually inputting information. The flow needs only a list of modules and connection information between modules to complete the design. Designers can use the flow to evaluate new architectures of FPGA and to map PR applications. Even though we discussed ReRAM FPGA through the dissertation, the PR flow and CAD tools are able to apply on modern FPGAs.

6.2 FUTURE WORK

The work presented here is cross multiple domains. It starts from structure of high density ReRAM, circuit with ReRAM, system of FPGA, application with ReRAM FPGA, to CAD tools for FPGA. Some works are still needed to refine the current work and make progress of ReRAM FPGA to be realized to real products and improve modern FPGAs. We discuss these topics from device and circuit level, system and application, to CAD tools in following paragraphs.

6.2.1 Device and Circuit studies of ReRAM

With more researches on ReRAM, many novel materials have better characteristics, such as power, latency, resistance of HRS and LRS, *etc.*, circuit design of ReRAM needs an improvement for these devices and advanced technology nodes. ReRAM with integrated selector is a new research topic since good selector helps to enlarge size of array for increasing memory density. The sense amplifier of the 3D-HIM should be improved to further reduce

area cost and improve delay. Since area cost of ReRAM are area of CMOS components, to save area of peripheral circuit, such as sharing drivers, reducing numbers of decoders, *etc.*, is very important.

6.2.2 System and Application

By using ReRAM as memory components in FPGA, architecture studies of FPGA should be involved to build FPGA of better performance or efficiency in design. In Chapter. 4, we have eight LE and an uBRAM as a RU for PR FPGA. Numbers of LE inside a RU and corresponding size of uBRAM need further discussion for efficient design. The discussion requires realistic applications to determine better architecture. Even though PR operation attracts many research works, a benchmark suite for verifying PR FPGA is still unavailable as well as academic version of HDL designs for PR applications. Scheduling and loading contexts to a PR FPGA is another study topic we might have in the future. Based on different applications and various architectures of PR FPGA, issues of scheduling becomes an optimization problem for specific occasions.

6.2.3 CAD for FPGA

We have done an initial work of CAD tools to support PR FPGA. However, optimization and algorithm behind the tools still need refinement. Even we integrated current work with VPR, however, a lot of temp files and patch are generated by our tools since VPR has no support of PR. Deep integration with VPR should be provided soon. In the near future, a research topic for CAD tools is to map result of place and route from the software platform to a real FPGA product. We successfully mapped result of modular placement to Xilinx board [95] based on the work from E. Hung, *et al.* [96] and our modifications. So, we can verify and fine tune the tools for better estimation in delay, area cost, and power dissipation. Next step, research work should map PR applications into commercial evaluation boards.

BIBLIOGRAPHY

[1] Xilinx, "7 Series FPGAs Overview," 2011, http://www.xilinx.com.

[2] Altera, "Logic Array Blocks and Adaptive Logic Modules in Stratix V Devices," 2011, http://www.altera.com.

[3] Microsemi, "Axcelerator Family FPGAs," 2012, http://www.actel.com.

[4] ——, "IGLOO Low Power Flash FPGAs," 2012, http://www.actel.com.

[5] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[6] N. Telle, W. Luk, and R. C. Cheung, "Customising Hardware Designs for Elliptic Curve Cryptography," in *Computer Systems: Architectures, Modeling, and Simulation*. Springer, 2004, pp. 274–283.

[7] G. Stitt, F. Vahid, and S. Nematbakhsh, "Energy Savings and Speedups from Partitioning Critical Software Loops to Hardware in Embedded Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 1, pp. 218–232, 2004.

[8] Vereen, L., "Soft FPGA Cores Attract Embedded Developers," April 2004, http://www.embedded.com//showArticle.jhtml?articleID=19200183.

[9] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable Domain-specific Computing," *IEEE Design & Test of Computers*, vol. 28, no. 2, pp. 6–15, 2011.

[10] P. Sedcole, B. Blodget, J. Anderson, P. Lysaghi, and T. Becker, "Modular Partial Reconfigurable in Virtex FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2005, pp. 211–216.

[11] Xilinx, "Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite," 2012, http://www.xilinx.com/.

[12] Altera, "Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs," 2010, http://www.altera.com.

[13] ALtera, "Best Practices for Incremental Compilation Partitions and Floorplan Assignments," 2012, http://www.altera.com.

[14] Lattice, "Field Update FPGAs While System Operates," 2005, http://www.latticesemi.com.

[15] P. Chow, S. Seo, J. Rose, K. Chung, G. Páez-Monzón, and I. Rahardja, "The Design of a SRAM-Based Field-Programmable Gate Array–Part II: Circuit Design and Layout," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 7, no. 3, pp. 321–330, 1999.

[16] 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 *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2012, pp. 77–86.

[17] D. Lewis and H. Lee, "Architectural Evaluation of 3D stacked RRAM Caches," in *IEEE International Conference on 3D System Integration (3DIC)*, 2009, pp. 1–4.

[18] M. Meier, R. Rosezin, S. Gilles, A. Rudiger, C. Kugeler, and R. Waser, "A Multilayer RRAM Nanoarchitecture with Resistively Switching Ag-doped Spin-on Glass," in *IEEE 10th International Conference on Ultimate Integration of Silicon (ULIS)*, 2009, pp. 143–146.

[19] A. Madayag and Z. Zhou, "Optimization of Spin-on-glass Process for Multilevel Metal Interconnects," in *University/Government/Industry Microelectronics Symposium, 2001. Proceedings of the Fourteenth Biennial.* IEEE, 2001, pp. 136–139.

[20] E. Linn, R. Rosezin, C. Kugeler, and R. Waser, "Complementary Resistive Switches for Passive Nanocrossbar Memories," *Nature Materials*, 2010.

[21] R. v. Nee and R. Prasad, *OFDM for Wireless Multimedia Communications.* Artech House, Inc., 2000.

[22] T.-C. Chen and Y.-W. Chang, "Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 25, no. 4, pp. 637–650, 2006.

[23] R. Waser and M. Aono, "Nanoionics-based Resistive Switching Memories," *Nature Materials*, vol. 6, no. 11, pp. 833–840, 2007.

[24] H. Li and Y. Chen, "An Overview of Non-volatile Memory Technology and the Implication for Tools and Architectures," in *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009, pp. 731–736.

[25] S. Ha and S. Ramanathan, "Adaptive Oxide Electronics: A Review," *Journal of Applied Physics*, vol. 110, no. 7, pp. 071 101–071 101, 2011.

[26] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories–Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[27] Y. Hosoi, Y. Tamai, T. Ohnishi, K. Ishihara, T. Shibuya, Y. Inoue, S. Yamazaki, T. Nakano, S. Ohnishi, N. Awaya, H. Inoue, H. Shima, H. Akinaga, H. Takagi, H. Akoh, and Y. Tokura, "High Speed Unipolar Switching Resistance RAM (RRAM) Technology," in *International Electron Devices Meeting (IEDM)*, 2006, pp. 1–4.

[28] R. Soni, P. Meuffels, H. Kohlstedt, C. Kugeler, and R. Waser, "Reliability Analysis of the Low rResistance State Stability of $Ge_{0.3}Se_{0.7}$ based Solid Electrolyte Nonvolatile Memory Cells," *Applied Physics Letters*, vol. 94, no. 12, p. 3503, 2009.

[29] R. Soni, M. Meier, A. Rudiger, B. Hollander, C. Kugeler, and R. Waser, "Integration of Ge_xSe_{1-x} in Crossbar Arrays for Non-volatile Memory Applications," *Microelectronic Engineering*, vol. 86, no. 4-6, pp. 1054–1056, 2009.

[30] S. Jo, K. Kim, and W. Lu, "High-density Crossbar Arrays based on a Si Memristive System," *Nano letters*, vol. 9, no. 2, pp. 870–874, 2009.

[31] D. Strukov and R. Williams, "Four-dimensional Address Topology for Circuits with Stacked Multilayer Crossbar Arrays," *Proceedings of the National Academy of Sciences*, vol. 106, no. 48, p. 20155, 2009.

[32] C. Kugeler, M. Meier, R. Rosezin, S. Gilles, and R. Waser, "High Density 3D Memory Architecture based on the Resistive Switching Effect," *Solid-State Electronics*, vol. 53, no. 12, pp. 1287–1292, 2009.

[33] Y.-C. Lu, "3D Technology based Circuit and Architecture Design," in *IEEE International Conference on Communications, Circuits and Systems*, 2009, pp. 1124–1128.

[34] T. Gao, B. Coenegrachts, J. Waeterloos, G. Beyer, H. Meynen, M. Van Hove, and K. Maex, "Integration of Unlanded Aia in a Non-etchback SOG Direct-on-metal Approach in 0.25 Micron CMOS Process," in *IEEE Proceedings of the IEEE International Interconnect Technology Conference*, 1998, pp. 45–47.

[35] K. Saraswat, K. Banerjee, A. Joshi, P. Kalavade, P. Kapur, and S. Souri, "3-D ICs: Motivation, Performance Analysis, and Technology," in *IEEE Proceedings of the 26th European Solid-State Circuits Conference (ESSCIRC)*, 2000, pp. 406–414.

[36] M. Johnson, A. Al-Shamma, D. Bosch, M. Crowley, M. Farmwald, L. Fasoli, A. Ilkbahar, B. Kleveland, T. Lee, T. Liu, *et al.*, "512-Mb PROM with a Three-dimensional Array of Diode/Antifuse Memory Cells," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1920–1928, 2003.

[37] Y. Chen, C. Rettner, S. Raoux, G. Burr, S. Chen, R. Shelby, M. Salinga, W. Risk, T. Happ, G. McClelland, *et al.*, "Ultra-Thin Phase-Change Bridge Memory Device Using GeSb," in *International Electron Devices Meeting (IEDM)*, 2006, pp. 1–4.

[38] X. Wang and Y. Chen, "Spintronic Memristor Devices and Application," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2010, pp. 667–672.

[39] J. Cong and B. Xiao, "mrFPGA: A Novel FPGA Architecture with Memristor-Based Reconfiguration," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2011, pp. 1–8.

[40] C. Wen, J. Li, S. Kim, M. Breitwisch, C. Lam, J. Paramesh, and L. Pileggi, "A Non-volatile Look-up Table Design Using PCM (Phase-Change Memory) Cells," in *IEEE Symposium on VLSI Circuits (VLSIC)*, 2011, pp. 302–303.

[41] H. Yan, H. Choe, S. Nam, Y. Hu, S. Das, J. Klemic, J. Ellenbogen, and C. Lieber, "Programmable Nanowire Circuits for Nanoprocessors," *Nature*, vol. 470, no. 7333, pp. 240–244, 2011.

[42] A. DeHon and M. Wilson, "Nanowire-Based Sublithographic Programmable Logic Arrays," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2004, pp. 123–132.

[43] Y. Chen, J. Zhao, and Y. Xie, "3D-nonFAR: Three-Dimensional Non-Volatile FPGA Architecture Using Phase Change Memory," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2010, pp. 55–60.

[44] S. Paul, S. Mukhopadhyay, and S. Bhunia, "A Circuit and Architecture Codesign Approach for a Hybrid CMOS–STTRAM Nonvolatile FPGA," *IEEE Transactions on Nanotechnology (TNANO)*, vol. 10, no. 3, pp. 385–394, 2011.

[45] 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. 99, pp. 2023–2032, 2011.

[46] Y. Liauw, Z. Zhang, W. Kim, A. Gamal, and S. Wong, "Nonvolatile 3D-FPGA with Monolithically Stacked RRAM-based Configuration Memory," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2012, pp. 406–408.

[47] W. Zhang, N. Jha, and L. Shang, "A Hybrid Nano/CMOS Dynamically Reconfigurable System–Part I: Architecture," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 5, no. 4, p. 16, 2009.

[48] Xilinx, "ISE Design Suite," 2014, http://www.xilinx.com.

[49] Altera, "Quartus II," 2014, http://www.altera.com.

[50] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[51] X. Inc., "Early Access PR User Guide," 2011, http://www.xilinx.com.

[52] R. He, G. Liang, Y. Ma, Y. Wang, and J. Bian, "PDPR: Fine-grained Placement for Dynamic Partially Reconfigurable FPGAs," in *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2012, pp. 350–356.

[53] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design & Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[54] S. Yousuf and A. Gordon-Ross, "DAPR: Design Automation for Partially Reconfigurable FPGAs," in *Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2010, pp. 97–103.

[55] C. Beckhoff, D. Koch, and J. Torreson, "Automatic Floorplanning and Interface Synthesis of Island Style Reconfigurable Systems with GOAHEAD," in *Architecture of Computing Systems (ARCS)*. Springer, 2013, pp. 303–316.

[56] D. Koch, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Springer, 2012, vol. 153.

[57] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, p. 36, 2011.

[58] B. A. Farisi, K. Bruneel, and D. Stroobandt, "Staticroute: A Novel Router for the Dynamic Partial Reconfiguration of FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–7.

[59] N. Shah and J. Rose, "On the Difficulty of Pin-to-Wire Routing in FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 83–90.

[60] M. Qureshi, M. Pickett, F. Miao, and J. Strachan, "CMOS Interface Circuits for Reading and Writing Memristor Crossbar Array," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011, pp. 2954–2957.

[61] J. Liang and H. 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, 2010.

[62] ITRS, "International Technology Roadmap for Semiconductors 2009 Edition," 2009, http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[63] Xilinx, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs," 2006, http://www.xilinx.com.

[64] Altera, "FPGA Architecture White Paper," 2006, http://www.altera.com.

[65] D. Lewis, P. Leventis, V. Betz, T. Wong, A. Lee, P. Pan, *et al.*, "Distributed Memory in Field-Programmable Gate Array Integrated Circuit Devices," Feb. 2 2010, US Patent 7,656,191.

[66] Xilinx, "Spartan-3 Generation FPGA User Guide," 2014, http://www.xilinx.com.

[67] ——, "Power Consumption at 40 and 45 nm," 2009, http://www.xilinx.com.

[68] M. Lee, C. Lee, D. Lee, S. Lee, M. Chang, J. Hur, Y. Kim, C. Kim, D. Seo, S. Seo, *et al.*, "A Fast, High-endurance and Scalable Non-volatile Memory Device Made from Asymmetric Ta_2O_{5-x}/TaO_{2-x} Bilayer Structures," *Nature Material*, vol. 10, no. 8, pp. 625–630, 2011.

[69] ITRS, "International Technology Roadmap for Semiconductors 2011 Edition," 2011, http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[70] Y.-C. Chen, H. Li, W. Zhang, and R. Pino, "3D-HIM: A 3D High-density Interleaved Memory for Bipolar RRAM Design," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2011, pp. 59–64.

[71] S. Sheu, M. Chang, K. Lin, C. Wu, Y. Chen, P. Chiu, C. Kuo, Y. Yang, P. Chiang, W. Lin, *et al.*, "A 4Fb Embedded SLC Resistive-RAM Macro with 7.2ns Read-Write Random-Access Time and 160ns MLC-Access Capability," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2011, pp. 200–202.

[72] B. Choi, A. Chen, X. Yang, and I. Chen, "Purely Electronic Switching with High Uniformity, Resistance Tunability, and Good Retention in Pt-Dispersed SiO2 Thin Films for ReRAM," *Advanced Materials*, vol. 23, no. 33, pp. 3847–3852, 2011.

[73] C. Ho, C. Hsu, C. Chen, J. Liu, C. Wu, C. Huang, C. Hu, and F. Yang, "9nm Half-Pitch Functional Resistive Memory Cell with $< 1\mu$A Programming Current Using Thermally Oxidized Sub-Stoichiometric WO_x Film," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2010, pp. 19–1.

[74] J. P. Strachan, D. B. Strukov, J. Borghetti, J. J. Yang, G. Medeiros-Ribeiro, and R. S. Williams, "The Switching Location of a Bipolar Memristor: Chemical, Thermal and Structural Mapping," *Nanotechnology*, vol. 22, p. 254015, 2011.

[75] B. Goll and H. Zimmermann, "A 0.12μm CMOS Comparator Requiring 0.5 V at 600MHz and 1.5 V at 6GHz," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 316–605.

[76] V. Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block," *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 10–15, 1998.

[77] K. Minkovich, "MCNC Benchmark," 2007, http://cadlab.cs.ucla.edu/~kirill/.

[78] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory," Ph.D. dissertation, University of Toronto, 1997.

[79] G. Lemieux and D. Lewis, "Circuit Design of Routing Switches," in *FPGA*, 2002, pp. 19–28.

[80] TSMC, "TSMC CMOS 0.18μm Technology," 2008, http://www.mosis.com/products/fab/vendors/tsmc/tsmc018/.

[81] W. Zhao and Y. Cao, "Predictive Technology Model (PTM)," 2008, http://ptm.asu.edu/.

[82] P. Jamieson, K. Kent, F. Gharibian, and L. Shannon, "Odin II-An Open-Source Verilog HDL Synthesis Tool for CAD Research," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 149–156.

[83] B. SYNTHESIS, "ABC: A System for Sequential Synthesis and Verification," *Berkeley Logic Synthesis and Verification Group*, 2011.

[84] K. I. and R. J., "intelligent FPGA Architecture Repository," 2008, http://www.eecg.utoronto.ca/vpr/architectures/.

[85] J. B. Goeders and S. J. Wilton, "VersaPower: Power Estimation for Diverse FPGA Architectures," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 229–234.

[86] Cadence, "Spectre - Cadence EDA Tool," 2013, http://www.cadence.com.

[87] Xilinx, "DS180 7 Series FPGAs Overview," 2011.

[88] K. Dimou, M. Wang, Y. Yang, M. Kazmi, A. Larmo, J. Pettersson, W. Muller, and Y. Timner, "Handover within 3gpp lte: Design principles and performance," in *IEEE Vehicular Technology Conference Fall (VTC)*, 2009, pp. 1–5.

[89] V. Majidzadeh, L. Jacques, A. Schmid, P. Vandergheynst, and Y. Leblebici, "A (256× 256) Pixel 76.7mW CMOS Imager/Compressor Based on Real-time In-Pixel Compressive Sensing," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 2956–2959.

[90] F. Mao, Y.-C. Chen, W. Zhang, and H. Li, "BMP: A Fast B*-Tree based Modular Placer for FPGAs," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2014, pp. 248–248.

[91] W. Snyder, "Introduction to Verilog-Perl," 2014, http://www.veripool.org/wiki/verilog-perl.

[92] U. Berkeley, "Berkeley Logic Interchange Format (BLIF)," *Oct Tools Distribution*, vol. 2, pp. 197–247, 1992.

[93] T.-C. Wang and D. Wong, "An Optimal Algorithm for Floorplan Area Optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 1991, pp. 180–186.

[94] Altera, "Logic Array Blocks and Adaptive Logic Modules in Arria V Devices," 2014, http://www.altera.com.

[95] Xilinx, "Virtex-6 Family Overview," 2014, http://www.xilinx.com.

[96] E. Hung, F. Eslami, and S. J. Wilton, "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices," in *IEEE Field-Programmable Custom Computing Machines (FCCM)*, 2013, pp. 45–52.