Sneak Path Free Reconfiguration and Fault Diagnosis for Via-Switch Crossbar Based FPGA

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2020

Ryutaro DOI

Publications

Academic Journal

[AJ1] **R. Doi**, J. Yu, and M. Hashimoto, "Sneak Path Free Reconfiguration with Minimized Programming Steps for Via-switch Crossbar Based FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (accepted).

[AJ2] H. Ochi, K. Yamaguchi, T. Fujimoto, J. Hotate, T. Kishimoto, T. Higashi, T. Imagawa, **R. Doi**, M. Tada, T. Sugibayashi, W. Takahashi, K. Wakabayashi, H. Onodera, Y. Mitsuyama, J. Yu, and M. Hashimoto, "Via-Switch FPGA: Highly Dense Mixed-Grained Reconfigurable Architecture With Overlay Via-Switch Crossbars," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2723–2736, Dec 2018.

[AJ3] **R. Doi**, M. Hashimoto, and T. Onoye, "An Analytic Evaluation on Soft Error Immunity Enhancement due to Temporal Triplication," *International Journal of Embedded Systems*, vol. 10, no. 1, pp. 22–31, January 2018.

International Conference

[IC1] **R. Doi**, X. Bai, T. Sakamoto, and M. Hashimoto, "Fault Diagnosis of Via-Switch Crossbar in Non-volatile FPGA," in *Design, Automation, and Test in Europe Conference and Exhibition (DATE)*, (accepted).

[IC2] M. Hashimoto, X. Bai, N. Banno, M. Tada, T. Sakamoto, J. Yu, **R. Doi**, Y. Araki, H. Onodera, T. Imagawa, H. Ochi, K. Wakabayashi, Y. Mitsuyama, and T. Sugibayashi, "Via-switch FPGA: First Implementation in 65-nm CMOS and Architecture Extension for AI Applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, (accepted).

[IC3] **R. Doi**, J. Yu, and M. Hashimoto, "Sneak Path Free Reconfiguration of Via-switch Crossbars Based FPGA," in *IEEE/ACM International Conference on*

Computer-Aided Design (ICCAD), Nov 2018, pp. 1–8.

[IC4] **R. Doi** and M. Hashimoto, “SAT Encoding-Based Verification of Sneak Path Problem in Via-Switch FPGA,” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2018, pp. 429–434.

[IC5] M. Hashimoto, Y. Nakazawa, **R. Doi**, and J. Yu, “Interconnect Delay Analysis for RRAM Crossbar Based FPGA (invited),” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2018, pp. 522–527.

[IC6] J. Hotate, T. Kishimoto, T. Higashi, H. Ochi, **R. Doi**, M. Tada, T. Sugibayashi, K. Wakabayashi, H. Onodera, Y. Mitsuyama, and M. Hashimoto, “A Highly-dense Mixed Grained Reconfigurable Architecture with Overlay Crossbar Interconnect using Via-switch,” in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[IC7] **R. Doi**, J. Hotate, T. Kishimoto, T. Higashi, H. Ochi, M. Tada, T. Sugibayashi, K. Wakabayashi, H. Onodera, Y. Mitsuyama, and M. Hashimoto, “Highly-dense Mixed Grained Reconfigurable Architecture with Via-switch,” in *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, March 2016.

[IC8] **R. Doi**, M. Hashimoto, and T. Onoye, “An Analytic Evaluation on Soft Error Immunity Enhancement due to Temporal Triplication,” in *IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, November 2015.

Domestic Conference

[DC1] 土井 龍太郎, 劉 載勲, 橋本 昌宜, “ビアスイッチFPGAの部分的再構成における書き換えスイッチ数の最小化,” 情報処理学会*DA*シンポジウム, August 2019.

[DC2] Y. Sun, **R. Doi**, and M. Hashimoto, “RC Extraction-free Wiring Delay Analysis Focusing on Number of On-state Switches for Via-switch FPGA,” in *IPSJ DA Symposium*, August 2019.

[DC3] 土井 龍太郎, 劉 載勲, 橋本 昌宜, “ビアスイッチFPGA再構成時のスニークパス問題を回避するプログラミング順決定手法,” 情報処理学会*DA*シンポジウム, August 2018.

[DC4] 中澤 祐希, 土井 龍太郎, 劉 載勲, 橋本 昌宜, “ビアスイッチFPGA向け配線遅延解析手法の検討,” 電子情報通信学会*VLSI*設計技術研究会, no. VLD2017-120, March 2018.

[DC5] 土井 龍太郎, 橋本 昌宜, “ビアスイッチFPGAにおけるスニークパス問題のSAT符号化を用いた検証,” 情報処理学会*DA*シンポジウム, August 2017.

[DC6] 土井 龍太郎, 橋本 昌宜, 尾上 孝雄, “時間的三重化によるソフトエラー耐性向上の解析的評価,” 電子情報通信学会 ディペンダブルコンピューティング研究会, November 2014.

Workshop

[WS1] J. Hotate, T. Kishimoto, T. Higashi, H. Ochi, **R. Doi**, M. Tada, T. Sugibayashi, K. Wakabayashi, H. Onodera, Y. Mitsuyama, and M. Hashimoto, “Highly-dense Mixed Grained Reconfigurable Architecture with Via-switch,” in *Work in Progress Session, Design Automation Conference (DAC)*, June 2016.

Summary

Reconfigurable computing hardware, such as field-programmable gate arrays (FPGAs), is gaining its popularity since the design and manufacturing cost of application-specific integrated circuits (ASICs) is elevating according to the device miniaturization and larger-scale integration. However, the performance and energy efficiency of conventional FPGAs are much lower than those of ASICs. These drawbacks originate from low area efficiency due to static random access memory (SRAM)-based programmable switches that consist of transistors and have a large area, resistance and capacitance.

To overcome the conventional FPGA drawbacks, FPGAs that exploit via-switches, which are a kind of Resistive RAM (RRAM), instead of SRAM-based switches are actively studied. The via-switch is a non-volatile switch with a small footprint and parasitic load, and it can be implemented and programmed without using transistors. Thanks to its characteristics, the via-switch is expected to boost the performance and energy efficiency of FPGA.

On the other hand, the via-switch FPGA is in an early stage of development and hence facing design, test, and programming challenges for practical application. First, in the design phase, there is no sufficient discussion on the interconnect structure that fully exploits the via-switch advantages. Previous studies on atom switch based FPGAs, where the atom switch is one of the component devices of the via-switch, presented a fundamental structure based on crossbars for higher integration density. Here, the crossbar is a circuit that has a switch device at every intersection of vertical and horizontal interconnections for signal routing. However, the appropriate interconnect structure with the via-switch crossbar is not discussed enough, and the performance improvement effect thanks to via-switches is not quantitatively evaluated. Next, the manufacturing phase requires the via-switch FPGA manufacturer to verify the crossbar functionality before the shipment for ensuring arbitrary routings at the FPGA user side. Therefore, fault testing is indispensable to check whether all the via-switches can be normally turned on/off. However, fault testing methodology for the via-switch crossbar has not been established so far. Meanwhile, in the user programming phase after the shipment, there is a possibility that the sneak path problem occurs due to the crossbar programming structure depending on configuration patterns of via-switches in the crossbar. The sneak path problem changes on/off state of non-target via-switches unintentionally due to programming signal detouring, and interferes the reconfiguration of FPGA. Therefore, it is crucially important to identify the occurrence conditions and develop countermeasures for this problem.

Solving the challenge at the design phase, this dissertation proposes an interconnect structure that can selectively insert repeaters to signal paths and achieve small interconnect delay and high energy efficiency. This work also clarifies the requirement of the programming structure for the connection switch between crossbars focusing on the sneak path problem. Transistor-level SPICE-based evaluation demonstrates that the proposed interconnect structure achieves a significant performance improvement compared with conventional SRAM-based FPGA. The proposed structure reaches 26X higher crossbar integration density and reduces interconnect delay and energy by 90% and 94% at 0.5 V operation.

For the challenge at the manufacturing phase, this dissertation proposes a fault testing methodology to verify the via-switch crossbar functionality for ensuring arbitrary routings. This work confirms that a general differential pair comparator successfully discriminates on/off-states of via-switches, where the comparator can be implemented with a small area at the peripheral part of the via-switch FPGA chip. This work also identifies fault modes of a via-switch using SPICE simulation that injects stuck-on/off faults to atom switch and varistor, which are components of via-switches. Then, this dissertation proposes a fault diagnosis method that identifies faulty via-switches in the crossbar according to the comparator response difference between normal and faulty cases. The proposed method attains 100% fault detection. As for the diagnosability, the successful ratios of the fault diagnosis are 100% and 79% in cases that the number of faulty components in a via-switch is up to one and up to two, respectively.

Aiming at overcoming the challenge at the user programming phase, this dissertation establishes a sneak path free reconfiguration methodology of via-switch crossbar. This work investigates the occurrence conditions of the sneak path problem and identifies the crossbar programming status that causes the sneak path. Based on the occurrence conditions, this dissertation proposes a sneak path free programming method that arranges the programming sequence of via-switches in a crossbar. This work devises an algorithm that effectively derives a sneak path free programming order by constructing the connection status of signal lines in a crossbar as a tree structure, which is called the connection tree. This dissertation also gives a proof that a sneak path free programming order necessarily exists for arbitrary on-off patterns in a crossbar as long as no loops exist. The simulation-based evaluation demonstrates that the proposed method significantly improves the routing flexibility of the via-switch crossbar. In a practically-sized 100x100 crossbar, the number of available configurations increases by over four orders of magnitude. The proposed method successfully solves the sneak path problem in any practical configurations of via-switch FPGA.

Furthermore, this dissertation extends the above programming method for partial reconfiguration that partially turns on and off via-switches in the already programmed crossbar. This work proposes a partial reconfiguration method that minimizes programming steps while avoiding the sneak path problem. The proposed method minimizes the number of programmed switches by arranging the root node of the connection tree. This work models the optimal root node selection as a set cover problem with cost minimization, and also proposes a low computational complexity method that obtains the same solution of the set cover problem without solving it. In a test case of simulation-based evaluation, the proposed method reduces the number of programmed switches by 77.4% compared to the conventional approach, which enables 4.4X more reconfigurations of the via-switch FPGA in its device lifetime and reduces reconfiguration time by 77.4%.

This dissertation covers challenges at all phases of via-switch FPGA development, i.e., design phase, manufacturing phase, and user programming phase. The proposed interconnect structure contributes to boosting the operating speed and energy efficiency. The proposed fault diagnosis method helps the manufacturer to inspect manufactured products and to prevent the shipment of defective products. The proposed sneak path solution enables to implement all the practical configuration patterns to the via-switch FPGA. The proposed partial reconfiguration method contributes to extending the lifetime of via-switches and speeding up the reconfiguration of the via-switch FPGA for both user programming and manufacturing test. At last, thanks to these contributions, FPGA users can enjoy implementing any systems on high-performance and defect-free via-switch FPGA.

Acknowledgments

First of all, I would like to express my deepest gratitude to Professor Masanori Hashimoto in Osaka University for providing an excellent research environment and a precious opportunity to have various experiences for seven years of bachelor's, master's, and doctoral courses. He comprehensively supported my research, and his advanced perspective and thoughtful advice led me to successful achievements. I have learned a lot from him and realized the growth of my skills and mind.

I would like to deeply express my appreciation to Associate Professor Jaehoon Yu in Tokyo Institute of Technology for valuable discussions and accurate advice. His thoughtful supports improved my research achievements.

I am grateful to Professor Tatsuhiro Tsuchiya, Associate Professor Ittetsu Taniguchi, and Associate Professor Hiromitsu Awano in Osaka University for detailed reviews and valuable suggestions.

I would like to express great gratitude to Professor Hidetoshi Onodera in Kyoto University, Professor Hiroyuki Ochi in Ritsumeikan University, Associate Professor Yukio Mitsuyama in Kochi University of Technology, Assistant Professor Takashi Imagawa in Ritsumeikan University, Dr. Kazutoshi Wakabayashi, Mr. Tadahiko Sugibayashi, Dr. Toshitsugu Sakamoto, and Dr. Munehiro Tada in NEC Corporation for precious discussions, insightful suggestions, and teaching me the fun of collaborative research.

I would like to express my appreciation to Professor Takao Onoye in Osaka University, Professor Yoshinori Takeuchi in Kindai University, Associate Professor Yuichi Itoh, and Assistant Professor Masahide Hatanaka in Osaka University for their advice and supports.

I would like to thank Assistant Professor Yutaka Masuda in Nagoya University, Assistant Professor Wang Liao in Kochi University of Technology, Dr. Koichi Mitsunari, Mr. Jun Chen, and Mr. Ryo Shirai for daily discussions and their supports. My sincere appreciation goes to Ms. Asako Murakami and Ms. Yuki Yoshida for their various supports throughout my student life. I also would like to thank all members of Integrated System Design Laboratory and Information Systems Synthesis Laboratory in Osaka University for having interesting and good time in the laboratory.

My heartfelt thanks go to my family for supporting my life and giving a delightful and relaxing time.

This work was supported by JSPS KAKENHI Grant Number JP17J10008 and JST CREST Grant Number JPMJCR1432, Japan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation addresses design, test, and programming issues of non-volatile via-switch FPGA that is an emerging FPGA with high energy efficiency. First, this chapter explains the background and objectives of this dissertation. Section 1.1 describes the research background. Then, Section 1.2 briefly explains the conventional FPGA and its drawbacks. As a solution to overcome conventional FPGA drawbacks, emerging non-volatile memory-based FPGAs are introduced in Section 1.3. After that, Section 1.4 explains via-switch FPGA that is mainly focused on in this dissertation, and also discusses issues to be solved. Finally, Section 1.5 states the research objectives of this dissertation.

1.1 Background

In 1965, Gordon Moore found that the number of transistors in an integrated circuit chip doubles every 18 months. This observation is called Moore's Law and has become a self-fulfilling prophecy [1]. Moore's Law has been maintained for recent decades by the effort of manufacturers. For example, the transistor count in Intel microprocessors has doubled every 26 months since the development of the 4004 microprocessor that had only 2,300 transistors in 1971 [2]. On the other hand, in 2019, Apple A13 Bionic microprocessor that contains 8.5 billion transistors was launched [3]. The continuation of Moore's Law over such a long term is mainly thanks to the scaling down of transistor size. The 4004 microprocessor was fabricated at a process node of 10 μm, while the Apple A13 Bionic used a process node of 7 nm. Manufacturers adopt a new process node every a few years with a 30% smaller transistor size to pack twice as many transistors in the same area [1, 4].

The successive scaling down of process node is beneficial to transistor performance, i.e., shrinking the transistor size brings faster operating speed and

Table 1.1: Comparison between ASICs and FPGAs for system implementations.

	ASICs	FPGAs
Programmability	No	Yes
Layout/Mask/Manufacturing	Necessary	Unnecessary
NRE Cost	Very High	Low
Time to Market	Long	Short
Design Change	Difficult	Easy
Design Flow	Comlex	Simple
Performance	High	Medium (Higher than CPU)

lower power consumption. On the other hand, especially in application-specific integrated circuits (ASICs), the development cost is increasing exponentially as the transistor size scales down [5]. This is because non-recurring engineering (NRE) costs, which include costs of mask-sets and engineering design efforts of layout and verification, are very expensive in smaller process nodes. For example, in 14 nm process, the total development cost of an ASIC reaches 270 million dollars [6]. Such expensive development costs of ASICs are unacceptable in many fields except for some fields that will sell in huge quantities or that have cutting-edge performance requirements, e.g., smartphone market, gaming hardware market, etc. In addition to high development costs, the long time to market due to complicated design and verification processes is another concern of ASIC implementations. It also takes a long time and high cost for design modifications or bug fixes.

In recent years, field-programmable gate arrays (FPGAs) are becoming popular since ASIC development cost is elevating and development time becomes longer due to process node scaling. Table 1.1 summarizes the advantages and disadvantages of ASICs and FPGAs for system implementations. FPGAs are computing devices that can reprogram the circuit functionality after manufacturing. We utilize a hardware description language (HDL) such as Verilog HDL and VHDL, which are similar to programming languages, to design the circuit with already manufactured FPGA, and hence layout, mask, manufacturing steps are unnecessary in FPGA implementations. Eliminating these steps significantly reduces NRE costs and shortens the development time. When we need to modify the circuit design or fix design errors, the modification can be achieved by just changing the description of HDL and reprogramming the FPGA. The design flow of FPGA implementations is also simpler than that of ASIC implementations.

Thanks to the above advantages, FPGAs are widely adopted not only for prototyping but also practical use in various fields despite lower performance compared to ASICs. For example, Microsoft, Amazon Web Services, and IBM introduced

FPGAs to their data centers of cloud computing services [7–10]. FPGAs often offer much higher performance than central processing units (CPUs), and this higher performance contributes to both speed up and energy reduction of the system. Besides, field-programmability, low NRE cost, and a short time to market of FPGAs are suitable for edge computing devices in the Internet of Things (IoT) era. Even in the fields of large variety and small quantity production, the adoption of FPGAs can improve profitability. Furthermore, hardware acceleration with FPGAs is drawing attention especially for artificial intelligence (AI) in recent years. For instance, Intel, Xilinx, and Microsoft corporations accelerate deep neural network (DNN) inference by using FPGAs [11–13]. In this way, FPGAs have a wide range of applications, and the FPGA market is expected to expand further in the future.

1.2 Conventional FPGA and Drawbacks

Section 1.2.1 briefly reviews the structure and operating principle of conventional FPGAs. Then, Section 1.2.2 takes a look at the drawbacks of conventional FPGAs.

1.2.1 SRAM-Based FPGA

Static random access memory (SRAM)-based FPGAs are the most popular FPGAs in many fields. These FPGAs typically have an island-style structure illustrated in Figure 1.1. The island-style structure consists of programmable logic elements and interconnects arranged in an array, where programmable logic elements are called logic blocks (LBs) and programmable interconnect wires are called connection blocks (CBs) and switch blocks (SBs). We can implement any circuit functionality by constructing arbitrary combinational or sequential circuits in LBs and routing arbitrary signal paths between LBs in CBs and SBs [14–16].

Figure 1.2 shows the most basic structure of an LB. A k-input look-up table (LUT) circuit has 2^k-bit SRAM cells and a 2^k-to-1 multiplexer (MUX) with k select terminals. To construct arbitrary combinational logic, we store a truth table of the desired function in k-input and 1-output, and give the input signals of LB to the select terminals of MUX. Thanks to the 2^k-bit SRAM cells, we can define the output values for all the combinations of k input signals. The LB also contains flip-flops (FFs) to implement a sequential circuit.

CB and SB have the structures depicted in Figure 1.3 and 1.4, respectively. CB is responsible for inputting to and outputting from the LB. MUX or pass transistors are used for controlling the connection between the LB and global interconnections. Here, the global interconnections are wiring resources that are

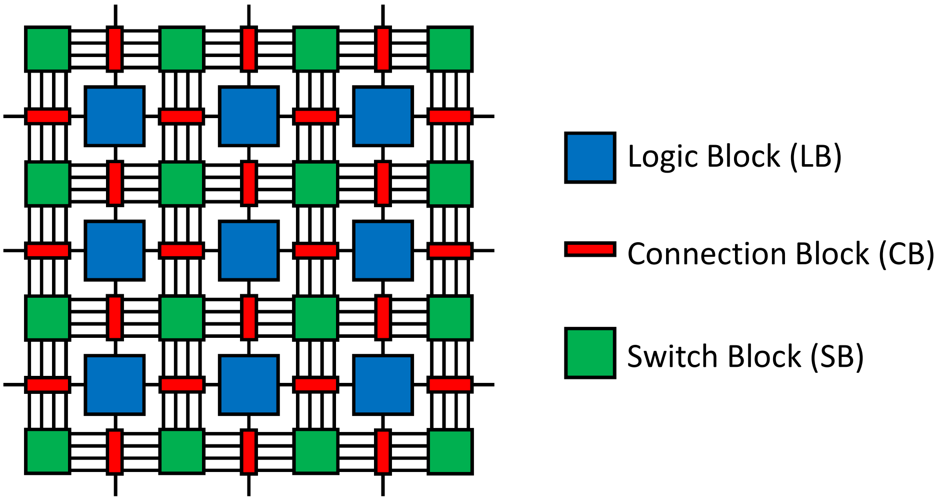


Figure 1.1: Isrand-style FPGA structure.

aligned outside LBs, while the local interconnections are arranged inside LBs. An SRAM cell is connected to the select terminal of MUX or gate terminal of the pass transistor. On the other hand, SB connects or disconnects the vertical interconnection and horizontal interconnection by pass transistors with SRAM cells. In both CB and SB, we program stored values in SRAM cells to construct the desired connection between LBs.

1.2.2 Drawbacks of SRAM-Based FPGA

Conventional SRAM-based FPGAs are inferior to ASICs regarding operating speed, power consumption, and implementation area [17]. For example, when we implement the same circuit function, Kuon et al. report that FPGA implementation has four times slower operating speed, fourteen times more dynamic power, 87 times more static power, and 35 times larger circuit area than ASIC implementation [18]. These drawbacks originate from a tremendous amount of programmable switches that are equipped in SRAM-based FPGAs to acquire reconfigurability as mentioned in the previous subsection. Lin et al. say that 80% of the circuit delay comes from programmable interconnections with SRAM-based switches [19].

Typical SRAM-based programmable switch is composed of a switch gate, such as transmission gate or MUX, and an SRAM cell to hold the on/off states of switch gate. The switch gate consists of transistors, and hence it has high resistance and large capacitance, which degrade the circuit speed and increase power dissipation. The SRAM cell using six transistors consumes a large area. This increases not only circuit area but also wiring length, and the consequent

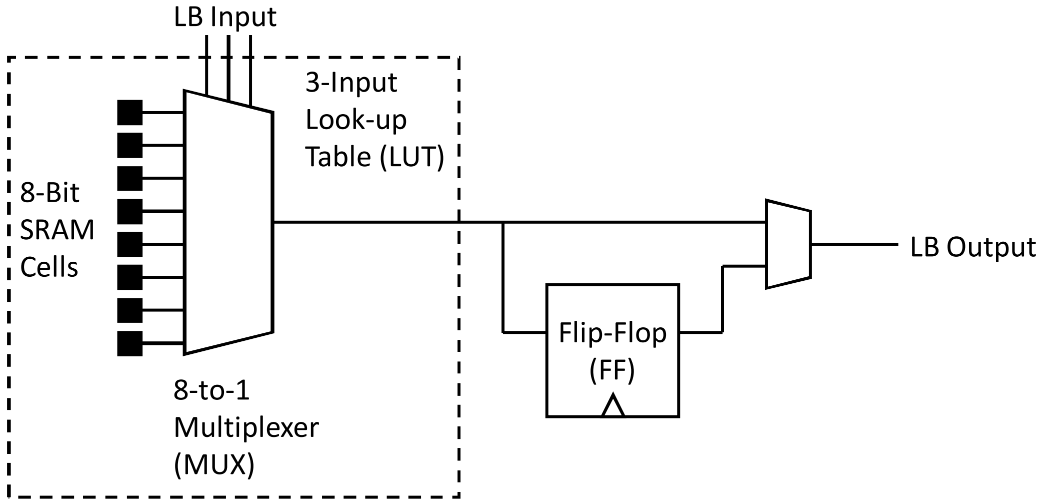


Figure 1.2: SRAM-based logic block structure.

long wire imposes a bad impact on the speed and power. Besides, the volatility of SRAM cells is also a concern. We need to keep supplying power to SRAM cells even during standby, and the leakage current in a large amount of SRAM cells causes enormous static power consumption. Furthermore, conventional FPGA structure where programming resources and logic circuits share the same layer, i.e., the transistor layer, is also undesirable. This is because SRAM cells dominate the FPGA chip area due to the large area and amount of them, and consequently logic circuits can only use a small area. This makes it difficult to put computing units that have high computing performance, e.g., arithmetic unit and multiply-accumulator, on the chip. Ref. [19] reports that programmable routing resources including SRAM-based switches occupy 78% of the chip area, whereas computing logic circuits account for only 14%.

From the above reasons, SRAM-based programmable switches lead to the degradation of interconnect performance and area efficiency of the FPGA. Therefore, miniaturizing the programmable switch, which is the performance bottleneck of conventional FPGAs, directly contributes to an area reduction of the entire FPGA. According to the area reduction, wiring length becomes shorter, and it reduces signal transmission delay and power consumption. Also, exploiting low resistance and small capacitance of programmable switches contributes to boosting the FPGA performance. In recent years, various FPGAs using non-volatile memories (NVM) are widely developed to overcome the drawbacks of conventional SRAM-based FPGAs. The details of such FPGAs are discussed in the following sections.

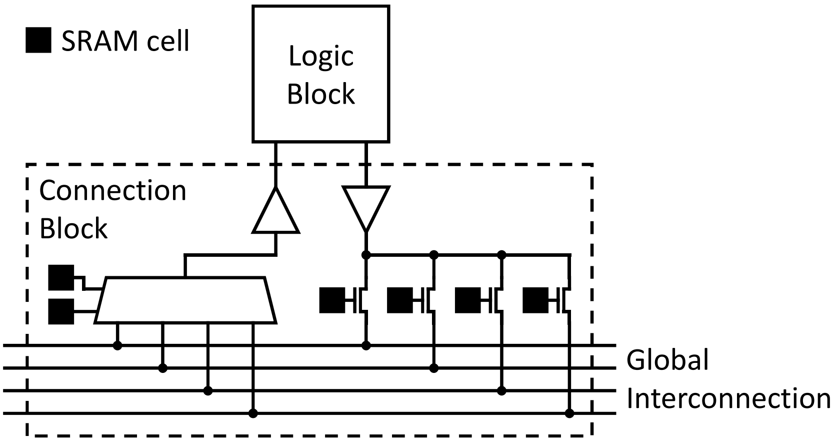


Figure 1.3: SRAM-based connection block structure.

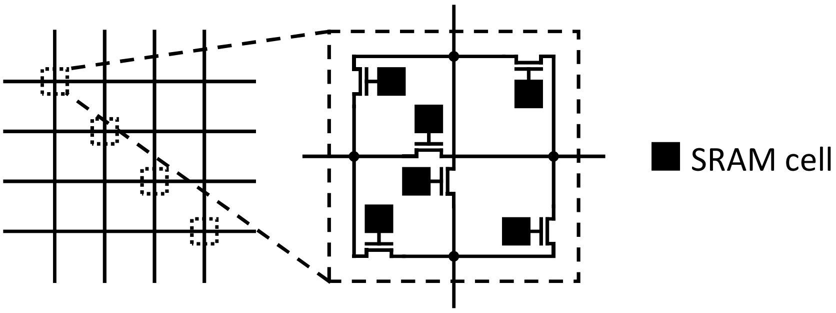


Figure 1.4: SRAM-based switch block structure.

1.3 Emerging FPGAs with Non-volatile Memories

This section firstly compares conventional computing hardware in terms of programmability and energy efficiency. Then, Section 1.3.1 introduces non-volatile memory (NVM)-based FPGAs that are developed for improving the energy efficiency of conventional FPGAs. There are traditional NVM-based FPGAs and emerging NVM-based FPGAs. Section 1.3.2 discusses the advantages and disadvantages of traditional NVM-based FPGAs. Emerging NVM technologies are actively developed in recent years, and they are becoming the most promising candidates to replace SRAM memories in conventional FPGAs. Section 1.3.3 explains three emerging NVM technologies, their structure and operating principle, followed by the reviews of related works of emerging NVM-based FPGAs in Section 1.3.4.

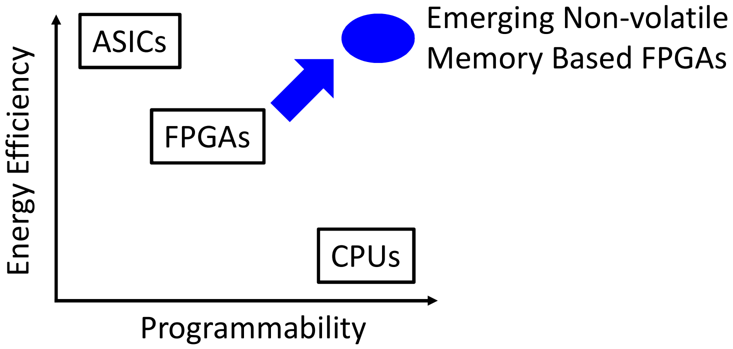


Figure 1.5: Trade-off between programmability and energy efficiency of conventional computing devices, and goal of next-generation FPGAs with non-volatile memories.

1.3.1 Programmability and Energy Efficiency Trade-off in Conventional Hardware

Conventional computing hardware has a trade-off relation between programmability and energy efficiency as shown in Figure 1.5. The programmability of CPUs is the highest since we can easily program them with high-level programming languages such as Python, C++, and so on. However, CPUs suffer from Von Neumann bottleneck due to their slow access speed between the processor and memory, and hence their energy efficiency becomes quite low [20, 21]. On the other hand, ASICs are designed to achieve the highest performance for a specific function, and therefore the energy efficiency is very high. Instead of this, ASICs cannot be programmed after manufacturing except the programs of embedded processors and hardly cope with design changes or bug fixes. Conventional FPGAs stand between CPUs and ASICs. FPGAs can be programmed like software by HDL, which is slightly difficult to use compared to high-level software programming languages. They often offer much better performance than CPUs since arbitrary circuit functionality can be implemented as dedicated hardware with parallelization.

In recent years, many researchers have studied how to boost up the programmability and performance of FPGAs. In terms of programmability, we have to describe cycle-by-cycle circuit behavior at a low level of abstraction with HDL, and this requires advanced hardware expertise. To improve the programmability, high-level synthesis techniques are well developed [22–27]. These techniques automatically generate an HDL description from a given algorithm written by high-level language, e.g., C and Java. The synthesis process can be optimized taking into account the performance, power, and area. Thanks to high-level synthesis, designers without hardware expertise can easily utilize FPGAs, and

Table 1.2: Comparison of SRAM-based FPGA and traditional NVM-based FPGAs.

	SRAM-Based	Flash-Based	Antifuse-Based
Non-volatility	No	Yes	Yes
Switch Area	Large	Medium	Small
On-Resistance	High	High	Low
Capacitance	Large	Large	Small
Reconfigurability	Yes	Yes	No
Manufacturing Process	Simple	Complex	Simple

we can also shorten the development time.

For improving the energy efficiency of FPGAs, on the other hand, various FPGAs that exploit other programmable switches instead of SRAM-based one are studied. As described in the previous section, SRAM-based programmable switches are performance bottleneck in conventional FPGAs. Requirements of desirable programmable switches are as follows.

- Small circuit area.
- Low on-resistance and high off-resistance.
- Small parasitic capacitance.
- Simple manufacturing process and high yield rate.
- Non-volatility.

According to the above requirements, many FPGA architectures using non-volatile memories (NVMs) are widely developed. There are traditional NVM-based FPGAs and emerging NVM-based FPGAs. The following subsections explain the characteristics of each FPGA.

1.3.2 Traditional Non-volatile FPGAs

There are two types of traditional non-volatile FPGAs, namely, flash memory-based FPGA [28–32] and antifuse FPGA [33–37]. Table 1.2 summarizes the characteristics of these traditional NVM-based FPGAs with the SRAM-based one.

Flash-based FPGAs replace SRAM cells by flash memories. The flash memory is a kind of floating-gate metal-oxide-semiconductor field-effect transistors (MOSFETs), and we program it by injecting an electrical charge to the floating-gate. Compared to SRAM-based switches, the area per switch can be reduced, but flash-based switches still have high on-resistance and large parasitic capacitance. Besides, combining flash and complementary MOS (CMOS) technologies requires a complicated and costly manufacturing process.

When we program antifuse FPGAs, we apply a high voltage to the antifuse switch. Initially, the antifuse switch is insulated. The applied high voltage makes a small hole in the insulation layer, and the switch turns on. Antifuse switch has advantages of a small area, low on-resistance, and small capacitance. However, once a hole is formed in the insulation layer, we cannot fill up the hole. Therefore, an antifuse FPGA allows only one-time programming, and there is no reconfigurability after programming once.

1.3.3 Emerging Non-volatile Memories

The previous subsection explains that traditional NVM-based FPGAs have many disadvantages despite their non-volatility. Recently, novel FPGA architectures using emerging non-volatile memories are widely proposed for boosting energy efficiency compared to conventional FPGAs [38, 39]. Firstly, this subsection summarizes the structure and operation of emerging NVM technologies, and then reviews emerging NVM-based FPGAs in the next subsection.

Emerging NVMs are two-terminal devices, and we change their resistance by imposing an electrical stimulus, i.e., voltage or current pulse. A high resistance state (HRS) is regarded as off-state and a low resistance state (LRS) corresponds to on-state. The switching between off-state and on-state is repeatable. NVMs have a non-volatility, namely, each on/off-state is maintained after power off. Emerging NVMs can be categorized into three types according to switching physics: spin-transfer-torque magnetic random access memory (STT-MRAM), phase-change random access memory (PCRAM), and resistive random access memory (RRAM) [40–42].

Table 1.3 compares the device characteristics of SRAM and emerging NVM technologies [41]. Thanks to very high cell density and low leakage power of emerging NVMs, replacing SRAM cells or whole SRAM-based switches including switch gate with these NVMs can improve the area efficiency and energy efficiency of FPGA. Especially, PCRAM and RRAM have a large on-off ratio, which represents the ratio of off-resistance to on-resistance. The large on-off ratio is beneficial to FPGA performance since low resistance in on-state enables high-speed signal transmission and high resistance in off-state cuts off the leakage current. PCRAM and RRAM also have good process compatibility with mainstream CMOS technology. Therefore, these NVM technologies can be easily combined with general CMOS logic circuits at a lower cost. On the other hand, cycling endurance of emerging NVMs, which means the maximum number of on-off switching, is lower than the SRAM cell. Thanks to the above features of emerging NVMs, they are becoming the most promising candidate

Table 1.3: Device Characteristic Comparison of SRAM and Emerging NVM Technologies.

	SRAM	STT-MRAM	PCRAM	RRAM
Non-volatility	No	Yes	Yes	Yes
Cell Area	> 100 F^2	6–50 F^2	4–30 F^2	4–12 F^2
Leakage Power	High	Low	Low	Low
On-Off Ratio	N/A	< 2X	10^2–10^3X	10^1–10^6X
CMOS Compatibility	Yes	Poor	Yes	Yes
Cycling Endurance	> 10^{16}	> 10^{15}	10^6–10^9	10^4–10^{12}

F: minimum feature size.

to replace SRAM-based programmable switches in conventional FPGAs. The following paragraphs explain the structure and operating mechanism of each emerging NVM.

The STT-MRAM cell is based on a magnetic tunnel junction (MTJ) structure [43–49]. MTJ structure consists of two ferromagnetic layers separated by a thin tunneling insulator layer, and one ferromagnetic layer is called a free layer and the other is a pinned layer as shown in Figure 1.6. We can change the magnetization direction in the free layer by injecting a large amount of electrons with the same spin direction, while the magnetization direction in the pinned layer is fixed. When both free and pinned layers have the same magnetization direction, which is called a parallel state, the resistance of STT-MRAM is low and a large tunneling current flows in the cell. On the other hand, in an antiparallel state where magnetization directions of free and pinned layers are opposite, the resistance becomes high and the current becomes small. This resistance change phenomenon that depends on the magnetization direction is known as tunneling magnetoresistance (TMR) effect. The on-off ratio of STT-MRAM is typically small, specifically less than 2X. The fabrication of STT-MRAM becomes complicated and has relatively poor compatibility to CMOS process technology since ten or more layers of ferromagnetic materials are stacked in MTJ structure.

The PCRAM cell is composed of chalcogenide materials that have two phases of the amorphous phase and crystalline phase as illustrated in Figure 1.7 [50–55]. The amorphous phase corresponds to off-state and has high resistance, whereas the crystalline phase which has low resistance corresponds to on-state. When we turn on a PCRAM cell, we heat the cell at a relatively low temperature and then cool down slowly. In the turning off operation, on the other hand, we use a high temperature for heating and then rapidly cool the cell. The on-off resistance ratio of PCRAM is much larger in the range from 100X to 1,000X. Due to the slow cooling down process in the turning on operation, switching speed of PCRAM cell

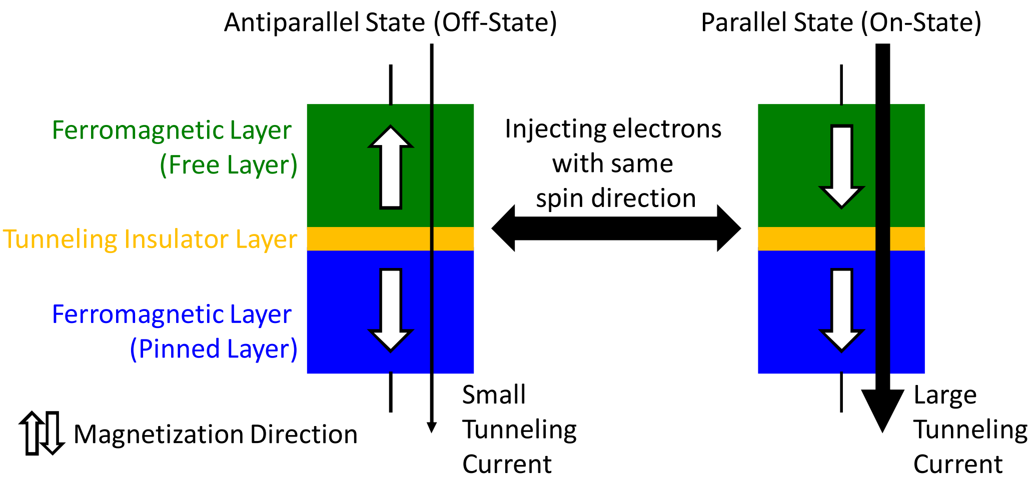

Antiparallel State (Off-State)
Parallel State (On-State)
Ferromagnetic Layer (Free Layer)
Tunneling Insulator Layer
Ferromagnetic Layer (Pinned Layer)
Magnetization Direction
Injecting electrons with same spin direction
Small Tunneling Current
Large Tunneling Current

Figure 1.6: Structure and operation of STT-MRAM cell.

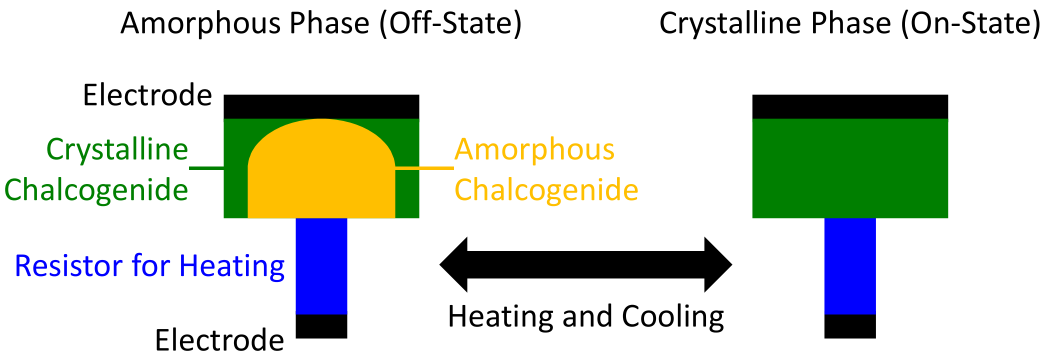


Figure 1.7: Structure and operation of PCRAM cell.

is limited, specifically slower than 50 ns which is ten times longer than RRAM cell. PCRAM has a long endurance of 10^6–10^9 cycles. In general, PCRAM has good process compatibility with CMOS technology.

The RRAM cell consists of metal oxide or solid electrolyte sandwiched between two electrodes, where metal oxide-based RRAM and solid electrolyte-based RRAM are called oxide random access memory (OxRAM) [56–60] and conductive bridge random access memory (CBRAM) [61–65], respectively. In an RRAM cell, low resistance conductive filaments are formed (on-state) and ruptured (off-state) between two electrodes by applying a voltage as depicted in Figure 1.8. Here, this figure shows an OxRAM cell and its filaments consist of oxygen vacancies, while filaments of CBRAM consist of metal atoms. OxRAM has a small on-off resistance ratio of 10–100X and better cycling endurance up to 10^{12} cycles. On the other hand, the on-off ratio of CBRAM is very large in the

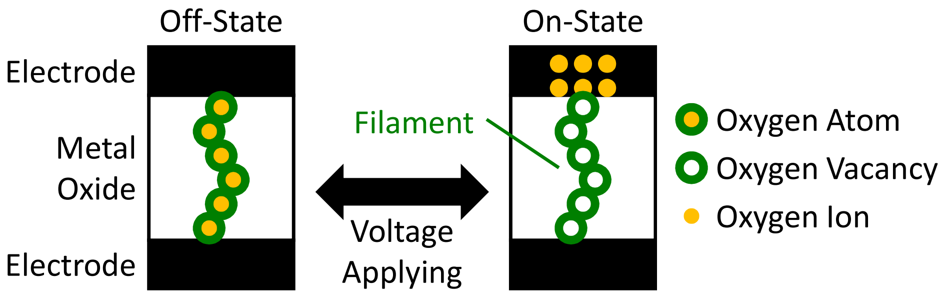


Figure 1.8: Structure and operation of RRAM cell.

range from 10^3X to 10^6X and endurance is limited to 10^4 cycles. RRAMs have excellent process compatibility with CMOS technology.

1.3.4 Emerging NVM-Based FPGAs

In recent years, FPGA architectures that utilize emerging NVMs are widely studied. There are three categories of emerging NVM-based FPGAs. The first category is FPGAs based on non-volatile SRAM (NV-SRAM) cells and non-volatile logic (NV-logic) circuits [66, 67]. The second category replaces SRAM cells with NVM cells [68, 69], while the third category exploits NVMs instead of whole SRAM-based switches including not only SRAM cells but also transistor switches whose gates are connected to the SRAM cells [39, 70]. The following paragraphs review FPGAs in each category and their contributions.

The first category adds NVMs to SRAMs and flip-flops for building the NV-SRAMs and NV-logic cells. The NVMs record the cell states before power off and restore after power on. The goal of this category is to completely cut off leakage power at standby and be immediately ready for computing on demand. Xue et al. proposed a low-power variation-tolerant non-volatile LUT, and demonstrated 38% reduction in power and 22% reduction in delay [66]. Huang et al. presented a low active leakage and high reliability PCRAM-based NV-SRAM [67]. The proposed NV-SRAM based LUT achieves 174 times reduction in active leakage power compared to the state-of-the-art. However, NV-SRAM and NV-logic based FPGAs remain having SRAM-based switches, which are the performance bottleneck in conventional FPGAs, and consequently the improvement on processing speed and area efficiency is limited.

In the second category, SRAM cells that control switch gates are replaced by NVM cells. Typically, NVM-based memory cell consists of two NVM cells in series where one end is connected to power rail and the other is connected to the ground rail. By letting only one NVM cell on-state, i.e., the other is off-state, a value of 0 or 1 can be read from the signal line between two NVM cells. Thanks to the elimination of SRAM cells, FPGAs in the second category reduce the area and energy consumption. Ju et al. proposed an STT-MRAM memory block based FPGA that reduced the critical path delay by 8.55% and the power by 54.34% [68]. Yuan et al. proposed an overall FPGA architecture with RRAM-based memory cells, and stated that they reduced the energy consumption by 39.5% compared with the state-of-the-art techniques [69]. However, these FPGAs still use transistor switches with high resistance in signal paths, which increases the signal delay.

The third category replaces both a transistor switch and its corresponding SRAM cell with an NVM cell since NVMs naturally have both switching and memory functionalities. Thanks to low resistance NVMs being placed in signal paths instead of high resistance transistor switch, the critical path delays can be reduced. Gaillardon et al. proposed NVM-based SB and LUT structures, and demonstrated an area and delay reduction of up to 28% and 34% compared to conventional FPGA [39]. They also claimed that PCRAM leads to the lowest leakage power, whereas RRAM gives the best area and delay improvements. Khaleghi et al. presented an RRAM-based FPGA architecture including SB, LUT, and also programming circuitry [70]. Their architecture reduces the area and delay by 59.4% and 20.1% compared to SRAM-based FPGA, and also improves the area and power by 49.7% and 33.8% compared to recent RRAM-based architecture. Miyamura et al. fabricated an FPGA based on atom switches [71], where the atom switch is one of the via-switch component and is detailed in the next section. They also demonstrated that their FPGA performed 60% active power saving and three times faster operation compared with a conventional FPGA. However, these NVMs need a few transistors per each cell for programming and those transistors prevent the further improvement of area efficiency.

1.4 Via-Switch FPGA and Issues

This section introduces via-switch FPGA that is mainly focused on in this dissertation. The via-switch FPGA is a novel non-volatile FPGA under active development for the practical application. First, Section 1.4.1 describes via-switch structure and characteristics. Then, Section 1.4.2 explains a fundamental structure and switch programming steps. Finally, Section 1.4.3 discusses challenges to be solved for the practical application of the via-switch FPGA.

1.4.1 Via-Switch

To overcome the drawbacks of conventional SRAM-based FPGAs, various FPGAs that exploit emerging NVMs as programmable switches instead of

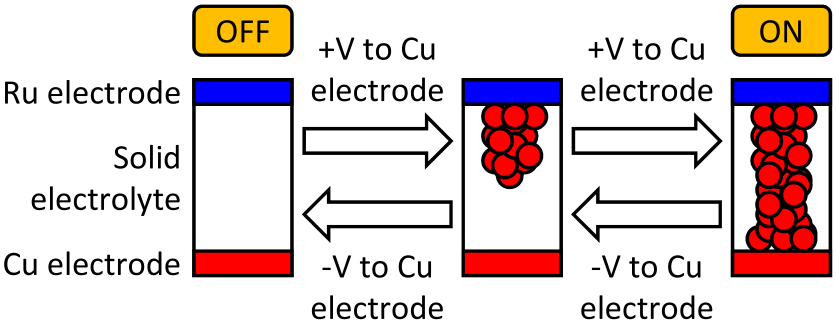


Figure 1.9: Structure and operation of atom switch.

SRAM-based ones are widely studied as explained in the previous section. In these emerging NVM-based FPGAs, however, one or two access transistors per a programmable switch are required for switch programming. The access transistor is relatively large despite the small footprint of an NVM-based switch, and hence it prevents further area reduction. To eliminate access transistors, non-volatile via-switch is actively developed [72–74]. The via-switch is a non-volatile, rewritable, and compact switch that is developed to implement a crossbar switch by Banno et al. [72]. This switch consists of atom switches, which are a kind of CBRAMs in RRAMs family, and varistors in place of access transistors. This subsection explains the device structure, functionality, and characteristics of the via-switch.

The atom switch consists of a solid electrolyte sandwiched between copper (Cu) and ruthenium (Ru) electrodes as shown in Figure 1.9. By applying a positive voltage to the Cu electrode, a Cu bridge is formed in the solid electrolyte, and the switch turns on. On the other hand, when a negative voltage is applied, Cu atoms in the bridge are reverted to the Cu electrode, and then the switch turns off. The switching between on-state and off-state is repeatable, and each state is non-volatile [75–79]. In an atom switch, both electrodes are connected by Cu metal ions, and therefore the on-resistance becomes low and capacitance becomes small. The on-resistance depends on the programming current through the atom switch when we form the Cu bridge, and it can be down to 200 Ω that is one order of magnitude lower compared with a MOS transistor switch. Off-resistance of an atom switch is 200 MΩ. Capacitance per atom switch is 0.14 fF [72, 80]. The atom switch can be reprogrammed about 1,000 times [71]. Furthermore, the atom switch can be integrated in the back-end-of-line (BEoL) layers with a small area, where BEoL layers correspond to wiring layers, while the front-end-of-line (FEoL) layer corresponds to transistor layer [81–84]. Since atom switches do not use transistors, we can fully utilize the FEoL layer to implement the logic, memory, or arithmetic computing unit.

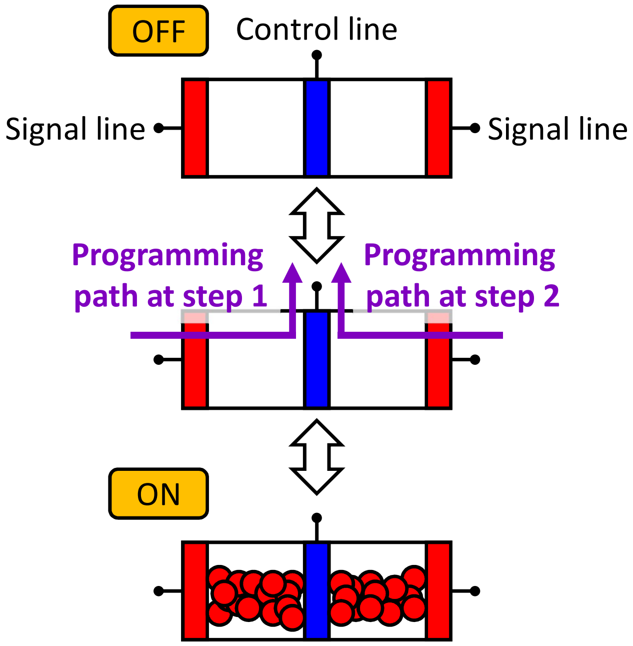


Figure 1.10: CAS structure.

On the other hand, when we use a single atom switch as a programmable switch, a device reliability issue arises. The turning on/off operation applies a relatively high voltage (programming voltage) to the atom switch, whereas we use lower voltage than programming voltage during normal operation. However, even when a lower voltage is applied to the atom switch, a small amount of Cu ions move, and hence it may lead to the transition of on/off states. To prevent the above reliability issue, the complementary atom switch (CAS) is devised, where it consists of two atom switches connected in series with opposite direction as shown in Figure 1.10 [85–89]. By regarding two atom switches as a programmable switch, the voltage applied to each atom switch is divided and it alleviates Cu ions transfer. Besides, when a positive voltage is given to either atom switch, a negative voltage is applied to the other atom switch thanks to the anti-series structure of a CAS, and this improves the device reliability. In the programming of CAS, a pair of the signal line and control line supply a programming voltage to each atom switch, and two atom switches are programmed sequentially. During normal operation, on the other hand, only signal lines are used for routing [71].

To accurately provide the programming voltage only to the target atom switch in a switch array, the varistor, which is a kind of bidirectional diode, is introduced into the via-switch [72–74]. Figure 1.11 shows the structure of via-switch, where the varistor is connected to the control terminal of CAS. When a voltage higher than a threshold value (programming voltage) is applied between the signal and

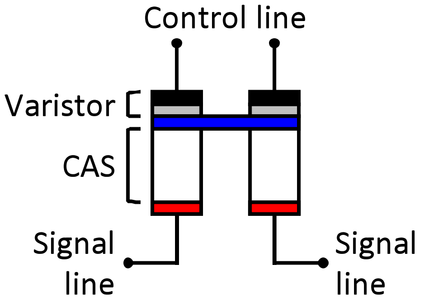


Figure 1.11: Via-switch structure.

control lines, the varistor supplies programming current to an atom switch. On the other hand, the varistor isolates the control lines from the signal lines during normal operation, and we can regard a via-switch as a two-terminal device with two signal lines. Both CAS and varistor can be implemented in the BEoL layer, and therefore the via-switch is a fully transistor free programmable switch. The minimum footprint per via-switch is 18 F^2 [72, 80].

1.4.2 Via-Switch FPGA

Figure 1.12 illustrates the concept of via-switch FPGA. By using non-volatile via-switches with a small footprint, resistance and capacitance instead of SRAM-based switches, the area and energy efficiency can be expected to improve. Furthermore, the BEoL layer integration of via-switches without transistors enables us to exploit the whole FEoL layer for computing logic. This promotes to implement rich computing units, e.g., arithmetic and logic units (ALUs), multiply-accumulators, accelerators for AI applications, etc. The following paragraphs explain the via-switch FPGA structure and switch programming steps.

The structure of via-switch FPGA is an array of configurable logic blocks (CLBs), and each CLB is composed of a logic block and a crossbar where a via-switch is placed at each intersection of signal lines as shown in Figure 1.13 [80]. The via-switch in the crossbar is responsible for connection and disconnection between the horizontal and vertical signal lines by changing the switch on/off states. Besides, the top half of the crossbar serves as input and output multiplexers to the logic block and corresponds to the connection block in conventional FPGAs. On the other hand, the bottom half of the crossbar, which corresponds to the switch block, routes global interconnections. The logic block organizes combinational and sequential circuits. In via-switch FPGA, we reconfigure each CLB separately. To acquire this, via-switches are also placed at the boundary between adjacent CLBs as depicted in Figure 1.13.

Next, the following describes how to program the via-switch in the crossbar

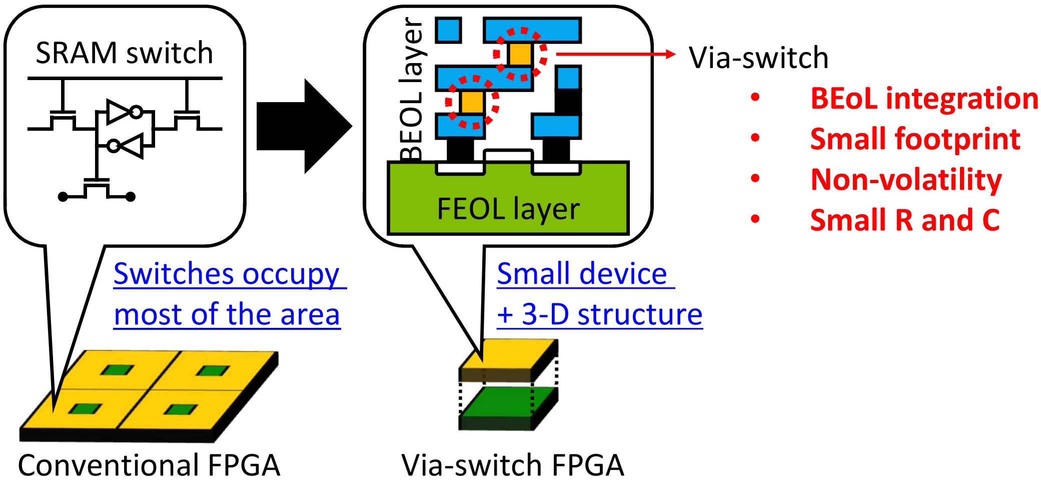


Figure 1.12: Concept of via-switch FPGA.

structure. Figure 1.14 illustrates the via-switch based crossbar structure. Both signal and control lines are aligned horizontally and vertically. This figure exemplifies programming steps in a 2x2 crossbar where an atom switch is turned on at each step. A pair of the perpendicular signal and control lines crossing at the via-switch of interest are used for switch programming. As mentioned in the previous subsection, when a high programming voltage is given between the signal and control lines, the varistor turns on and supplies programming current to the switch. Therefore, two programming drivers are activated at each step, and a positive voltage is given to one of the signal lines, and a ground voltage is given to one of the control lines. Other lines are floated. As shown in the figure, steps 1 and 2 successfully turn on the via-switch at the bottom left.

1.4.3 Issues of Via-Switch FPGA

Thanks to the advantages of via-switches as mentioned, via-switch FPGA is expected to significantly boost the energy efficiency compared to conventional FPGAs. On the other hand, the via-switch FPGA is facing some challenges for the practical application. First of all, it is primarily important to establish the fabrication and integration technologies of via-switches. For utilizing via-switches as programmable switches, they need to have a large on-off resistance ratio. Also, novel nano-sized devices such as via-switches face a challenge in terms of device yield, and hence highly-reliable manufacturing process is demanded. To tackle these issues, Refs. [74, 79, 90–92] have been actively developing the integration technology of via-switches and improving device characteristics. Besides, developing computer-aided design (CAD) tools for the via-switch FPGA

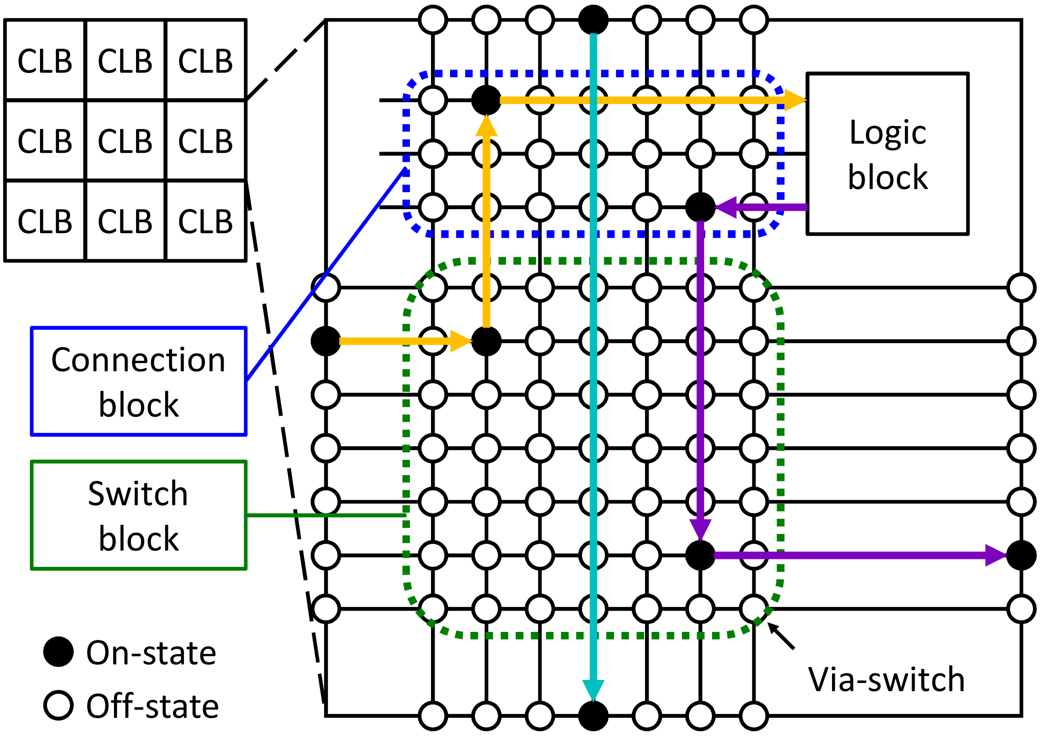


Figure 1.13: Structure of via-switch FPGA.

is also challenging. Based on the features and constraints of the via-switch FPGA architecture, it is essential to build appropriate CAD tools that are responsible for the logic synthesis, technology mapping, place and route. For this challenge, the mapping algorithm dedicated for the via-switch FPGA is devised [93, 94]. However, some critical challenges still remain for the practical application of the via-switch FPGA. This subsection explains these challenges along each phase of via-switch FPGA development, i.e., the chip design phase, manufacturing phase, and reprogramming phase at the user side.

First, as the challenge at the chip design phase, the interconnect structure that fully utilizes the advantages of via-switches is not sufficiently studied. The crossbar structure is considered to be suitable to efficiently accommodate many switches in a small area and to improve the performance in previous studies on FPGAs with atom switches. However, quantitative evaluations on the performance improvement effect in the case that the via-switch FPGA exploits crossbar structure are not enough. Besides, an appropriate connection structure between CLBs has not been considered so far. Furthermore, the repeater insertion technique for improving the interconnect performance, which is a common strategy in conventional circuit design, is not discussed. For example in

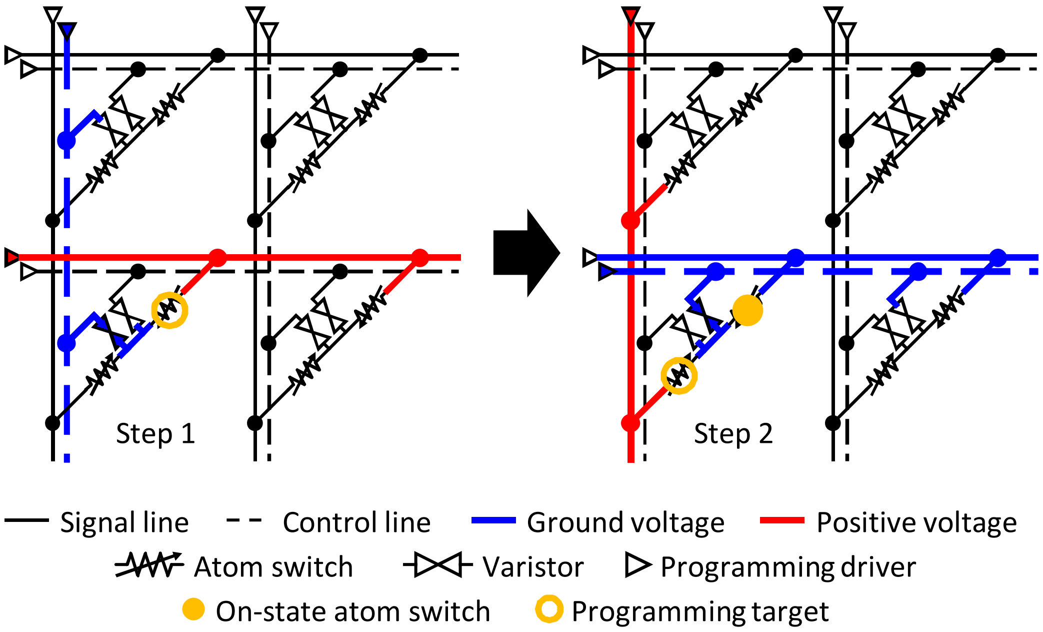


Figure 1.14: Via-switch based crossbar structure and switch programming steps.

conventional FPGAs, signal slope tends to become gentle due to the large circuit area and the large parasitic load of switch circuits. To prevent the signal distortion and improve the performance, conventional FPGAs frequently insert repeaters to the signal paths, where the repeater is a driving circuit to restore the distorted signals and is typically implemented by a buffer or inverter. From the above, the interconnect structure that maximizes the performance improvement effect thanks to via-switches needs to be clarified.

Next, in the manufacturing phase, via-switch FPGA manufacturer needs to inspect the via-switch based crossbar functionality before the shipment for ensuring that FPGA users can implement arbitrary routings. For this purpose, fault testing that investigates whether all the via-switches can be normally turned on and off is essential. However, fault testing methodology for via-switch based crossbar has not been developed. As mentioned, the via-switch has an upper limit on the number of reprogramming. Therefore, the fault testing method with a small number of reprogramming is desirable for maximizing the number of reprogramming at the FPGA user side after the shipment.

After the shipment of the via-switch FPGA, users apply various programs to the FPGA. However, the crossbar programming structure explained in the previous subsection may cause the sneak path problem depending on on-off patterns of via-switches in a crossbar. For example, the programming of the top right via-switch in Figure 1.15 cannot be performed correctly. The atom switch

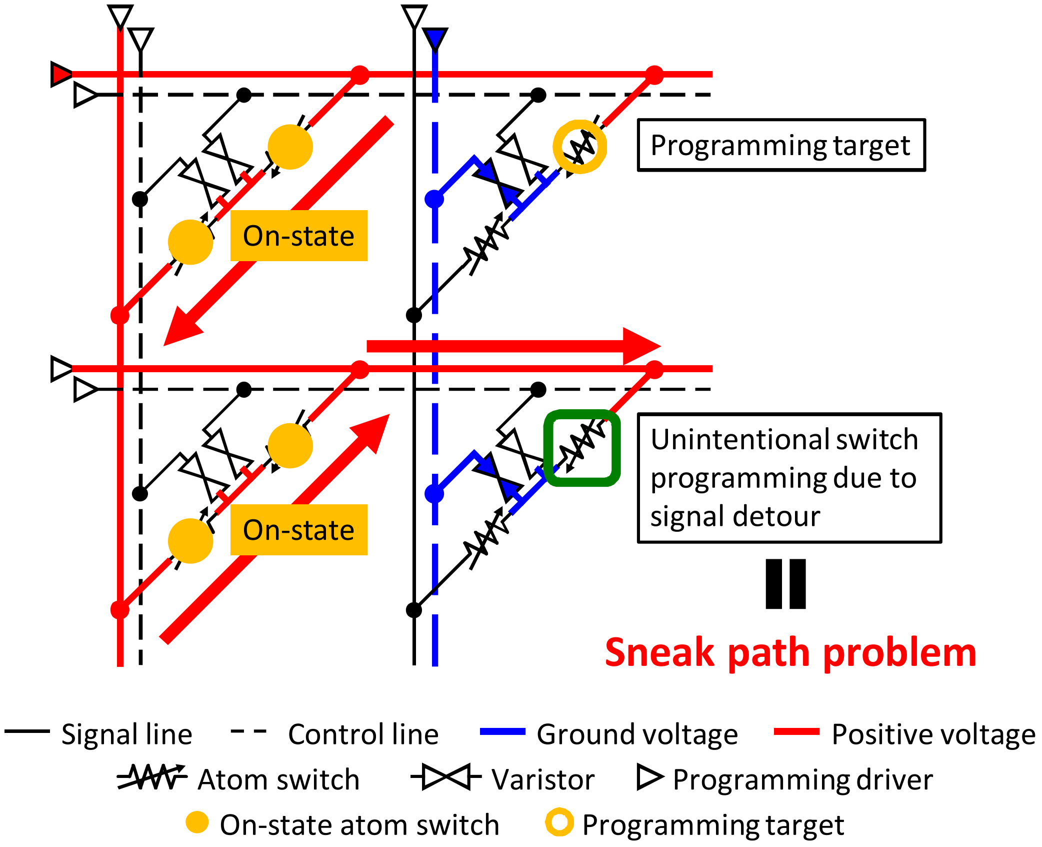


Figure 1.15: Sneak path problem in via-switch crossbar programming.

that composes the bottom right via-switch is under programming unintentionally since the positive voltage is provided through the already on-state via-switches at the bottom left and top left. Such an unintentional switch programming due to signal detouring through on-state via-switches is the sneak path problem. The sneak path problem interferes the reconfiguration of FPGA, and hence it is crucially important to clarify the occurrence conditions and find countermeasures for the sneak path problem.

1.5 Objectives of This Dissertation

The objective of this dissertation is to solve the issues preventing the practical application of the via-switch FPGA and provide an environment where users can utilize the high-performance and defectless via-switch FPGA without the sneak path problem. For this purpose, this dissertation tackles the challenges of via-switch FPGA at the chip design phase, manufacturing phase, and reprogramming phase at the user side as explained in Section 1.4.3. This subsection summarizes the objectives of the following chapters and outlines how to address each issue.

Chapter 2 aims at solving the issues at the design phase. This work investigates interconnect structures of via-switch FPGA focusing on the repeater insertion, and proposes a structure that can utilize the via-switch advantages and can selectively insert repeaters to the signal paths. Then, transistor-level SPICE simulations quantitatively evaluate the interconnect performance of the proposed structure and demonstrate that the significant performance improvement is achieved comparing to conventional SRAM-based FPGA. Besides, this chapter clarifies the appropriate connection structure between CLBs of via-switch FPGA taking the sneak path problem into account. For this purpose, this work also reviews a constraint-based countermeasure for the sneak path problem in the via-switch crossbar and give a formal proof of its effectiveness.

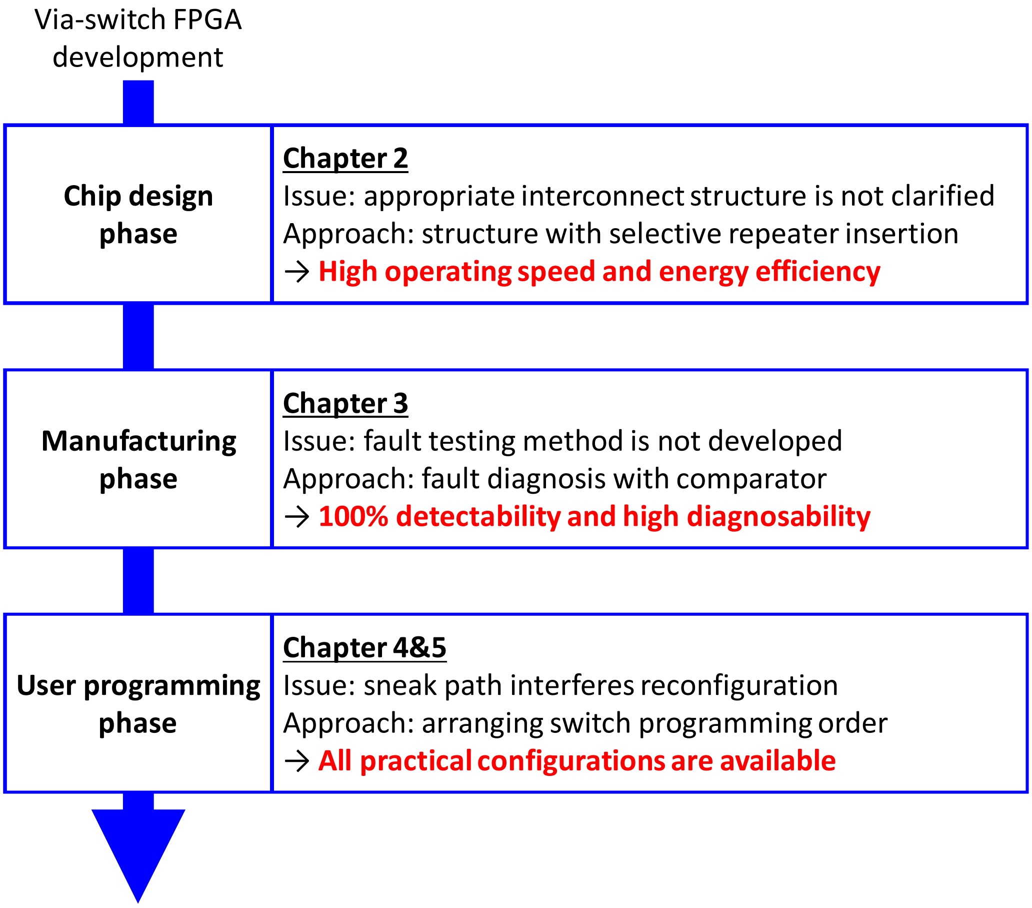
Chapter 3 establishes a fault detection and diagnosis methodology for the manufacturing phase of the via-switch FPGA. This work is the first one to investigate the fault testing of the via-switch crossbar. This chapter utilizes a general differential pair comparator, which can be implemented with a small area at the peripheral a part of via-switch FPGA chip, for distinguishing the via-switch on/off-states in the crossbar. Next, fault modes of a via-switch are clarified by using SPICE simulation that injects stuck-on/off faults to atom switch and varistor. Then, this work proposes a fault detection and diagnosis methodology that identifies faulty via-switches in the via-switch crossbar focusing on the difference of the comparator response in normal and faulty cases. The number of reprogramming in the proposed fault testing method is very small, i.e., each via-switch is reprogrammed only once. The fault detectable ratio and diagnosable ratio are also evaluated.

Chapter 4 addresses the sneak path problem that is a critical issue at the user programming phase. First, the crossbar programming status that causes the sneak path is identified by investigating the occurrence conditions of the sneak path problem. After that, this chapter proposes an initial programming method that avoids the sneak path problem by arranging the programming sequence of via-switches in a crossbar. It should be noted that the crossbar whose via-switches are all off-state is supposed as the initial state in the initial programming. This work develops an algorithm that can effectively find a sneak path free programming order by traversing the connection tree, which represents the connection status of signal lines in a crossbar as a tree structure. A formal proof that a sneak path free programming order necessarily exists for arbitrary on-off patterns without loops in a crossbar is also provided in this chapter. The significant improvement of the routing flexibility in the via-switch crossbar is confirmed by the quantitative simulation-based evaluation.

Chapter 5 extends the proposed method of Chapter 4 for partial reconfiguration in the already programmed crossbar at the initial state. This work proposes a partial reprogramming method for minimizing the number of switch programming steps while avoiding the sneak path problem. The minimization of the number of programmed switches is achieved by arranging the root node of the connection tree. This chapter models the optimal root node selection as a set cover problem with cost minimization, and also proposes a low computational complexity method that obtains the same solution of the set cover problem without solving it. The simulation-based evaluation confirms that the proposed method greatly reduces the number of programming steps and contributes to extending the lifetime of via-switches and speeding up the reconfiguration of the via-switch FPGA.

Finally, Chapter 6 summarizes the dissertation. Future works are also discussed in this chapter.

Primary issues at all phases of via-switch FPGA development that prevent practical application are covered in this dissertation as illustrated in Figure 1.16. The interconnect structure proposed in Chapter 2 boosts both the operating speed and energy efficiency. The fault diagnosis method proposed in Chapter 3 helps the FPGA manufacturer to inspect manufactured products and to prevent the shipment of defective products to FPGA users. The sneak path solution proposed in Chapters 4 and 5 enables to implement all the practical configuration patterns to the via-switch FPGA. As a result, this dissertation realizes that FPGA users can enjoy implementing any applications on high-performance and defectless via-switch FPGA.



Via-switch FPGA development
Chip design phase
Chapter 2
Issue: appropriate interconnect structure is not clarified
Approach: structure with selective repeater insertion
→ High operating speed and energy efficiency
Manufacturing phase
Chapter 3
Issue: fault testing method is not developed
Approach: fault diagnosis with comparator
→ 100% detectability and high diagnosability
User programming phase
Chapter 4&5
Issue: sneak path interferes reconfiguration
Approach: arranging switch programming order
→ All practical configurations are available

Figure 1.16: Contributions and organization of this dissertation.

Chapter 2

Interconnect Structure Design and Evaluation in Via-Switch FPGA

This chapter discusses the interconnect structure that is suitable for via-switch FPGA [80]. By comparing some interconnect structures of via-switch FPGA, this chapter proposes a high performance interconnect structure that can selectively insert repeaters to signal paths. This chapter also clarifies the appropriate programming structure at the connection switch between CLBs taking the sneak path problem into account. Section 2.1 introduces the repeater insertion that is an important technique in circuit design to improve the interconnect performance. Then, Section 2.2 proposes an interconnect structure and provides some structures to be compared. The inter-CLB connection structure is also discussed in this section. Section 2.3 evaluates the interconnect performance of these structures by transistor-level SPICE simulation, and demonstrates that the proposed structure attains high energy efficiency. Lastly, Section 2.4 summarizes this chapter.

2.1 Introduction

This chapter focuses on the repeater insertion which is one of the design considerations in integrated circuits [1]. Both interconnect resistance and capacitance increase in proportion to wiring length L, and hence wiring delay increases in proportion to the square of wiring length L^2. To mitigate this quadratic effect, repeater circuits, which are typically buffers or inverters, are inserted in the middle of interconnection to divide the interconnection and shorten the wiring length per gate circuit. Supposing the wiring is split into N segments with N repeaters, each segment has a delay time of $(L/N)^2$, and hence the total delay through N segments becomes L^2/N. If the number of segments is proportional to the wiring length, the overall delay increases only linearly with L. On the other hand, repeaters also become load factors that have resistance and capacitance, and therefore excessive repeater insertion leads to degrading the operating speed and energy efficiency. Besides, repeater insertion has a disadvantage that signal transmission direction is fixed since repeaters allow only the unidirectional signal transfer.

Conventional FPGAs have large wiring length, and consequent large resistance and capacitance due to the large area of the SRAM-based programmable switches as explained in Section 1.2. Therefore, repeaters are placed at every connection block and switch block, and the signal frequently goes through these repeaters [95]. In via-switch FPGA, on the other hand, the circuit area is expected to be dramatically reduced thanks to the small footprint of a via-switch, and wiring resistance and capacitance become small according to the area reduction. The via-switch resistance and capacitance are also small. Hence, frequent repeater insertion by adopting the conventional strategy may lead to performance degradation. The following sections clarify an interconnect structure that is suitable for via-switch FPGA.

2.2 Interconnect Structures of Via-Switch FPGA

This section introduces the proposed interconnect structure and some structures to be compared.

2.2.1 Proposed Interconnect Structure

Figure 2.1 illustrates the proposed interconnect structure. An important feature of the proposed structure is that repeaters are arranged separately from the global signal lines. In this structure, the connection block is responsible for repeater insertion. This structure enables us to selectively insert repeaters according to the signal transmission distance. For example, in short distance transmission such as transmission to adjacent CLB, the wiring load is small, and hence no repeaters are inserted by using the blue path depicted in Figure 2.1. On the other hand, in the signal transfer over a long distance, the crossbar routes the signal like the green path shown in Figure 2.1 at the middle CLBs and insert repeaters. This selective repeater insertion contributes to optimized signal delay. Besides, the signal lines of the proposed structure are bidirectional since the via-switch is naturally capable of bidirectional signal transfer and repeaters are placed separately from the signal lines. Thanks to the bidirectional transmission, the routing efficiency per signal line improves and consequently the number of necessary signal lines can be reduced. Ref. [80] reports that the number of signal lines with bidirectional interconnects can be reduced by 9–14% compared with unidirectional interconnects in a case study of application mapping.

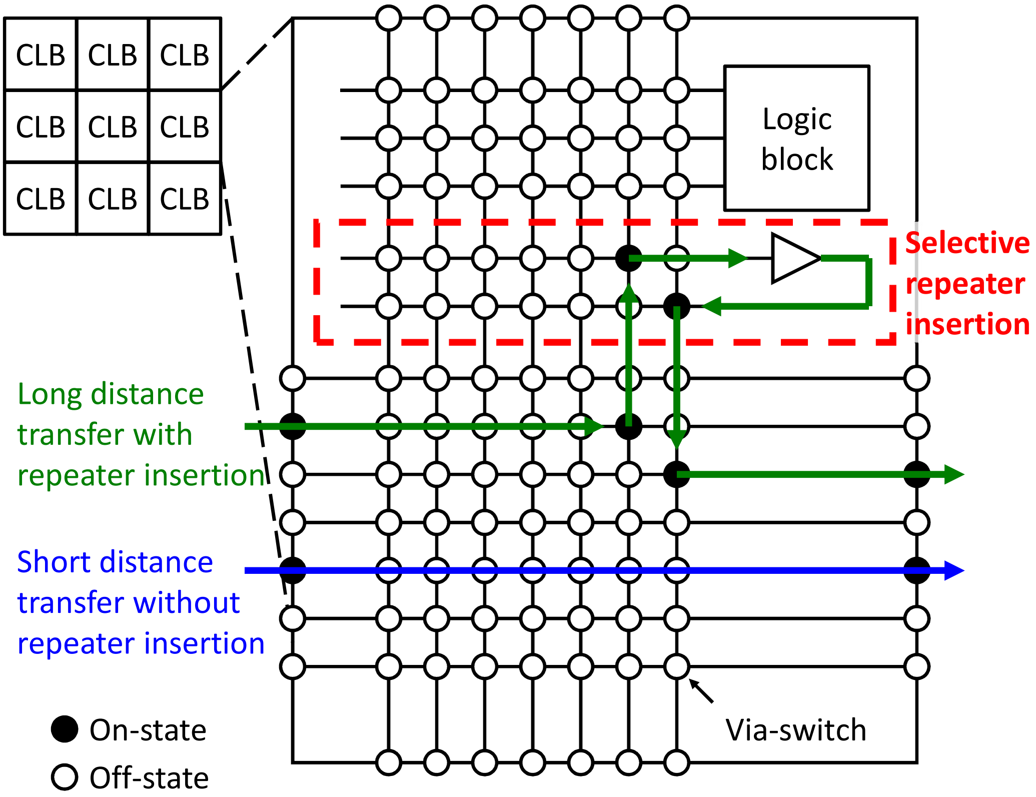

CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
Logic block
Selective repeater insertion
Long distance transfer with repeater insertion
Short distance transfer without repeater insertion
Via-switch
● On-state
○ Off-state

Figure 2.1: Proposed interconnect structure of via-switch FPGA.

2.2.2 Connection Structure Between CLBs

In via-switch FPGA, CLBs are connected to each other by via-switches, but the detailed connection structure is not discussed so far. This subsection clarifies the requirement of connection structure focusing on the sneak path problem in the switch programming as explained in Section 1.4.3. An important design consideration of switch programming is how to arrange control lines of via-switches. For example in the crossbar as mentioned in Section 1.4, each via-switch connects to two control lines that are aligned horizontally and vertically. Ochi et al. intuitively claimed that a programming constraint which prohibits multiple on-state via-switches in the same horizontal line eliminates the sneak path problem. They also mentioned that they could turn on multiple via-switches in the same vertical line for multiple fan-outs by imposing the programming constraint. Therefore, this subsection reveals the appropriate programming structure of between CLBs while assuming that there are multiple on-state via-switches in the same horizontal line but not in the same vertical line. In addition, the next subsection gives a formal proof that the programming constraint surely eliminates the sneak path problem

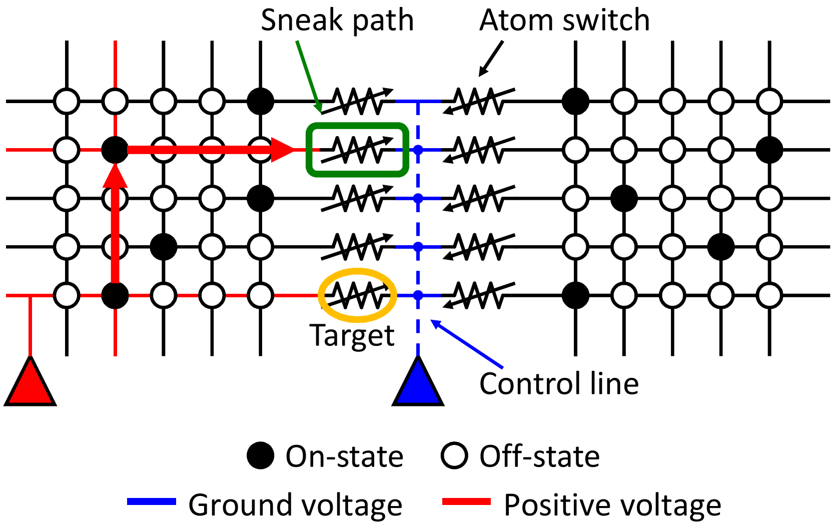


Figure 2.2: Sneak path problem in connection switch between CLBs.

in the crossbar.

First, a smaller number of control lines is better in terms of area efficiency. Therefore, it is preferable that all the connection switches between CLBs share one control line as illustrated in Figure 2.2. However, this structure causes the sneak path problem when multiple via-switches are on-state in the same line of the crossbar, and a non-target connection switch turns on unintentionally in the figure. To prevent the sneak path problem, an individual control line needs to be aligned for each connection switch between horizontal CLBs. On the other hand, connection switches between vertical CLBs can share the same control line since the multiple on-state via-switches in the same horizontal line are not allowed under the programming constraint.

2.2.3 Effectiveness Proof of Programming Constraint Based Countermeasure for Sneak Path Problem

This subsection proves that there is no sneak path problem in the programming of the any-sized via-switch based crossbar structure under a programming constraint that the multiple on-state via-switches are allowed only in one direction. Following the procedure below, a formal proof based on mathematical induction is provided.

1. Prove that the sneak path problem does not arise in the programming of 1×1 crossbar (i.e., single via-switch). This is self-evident.

2. Assume that the sneak path problem does not arise at each programming step for any configuration patterns of $M \times N$ crossbar.

   (a) Prove that the sneak path problem does not arise at each programming step for any configuration patterns in $(M + 1) \times N$ crossbar.

   (b) Prove that the sneak path problem does not arise at each programming step for any configuration patterns in $M \times (N + 1)$ crossbar.

3. Once procedures 1 and 2 are completed, this work can prove there is no sneak path problem in the programming of the any-sized crossbar.

The atom switch at the intersection when a positive voltage is given to the signal line and a ground voltage is given to the control line is turned on. When the number of such intersections is two or more, the sneak path problem arises. Here, the proof focuses on the number of bends of the programming signal given to the signal line. When the number of bends of programming signal given to the signal line is two or more, there are two or more intersections where the corresponding atom switches become on as shown in Figure 1.15. In other words, the programming signal must be bend twice or more by on-state via-switches to cause the sneak path problem. The proof will show that the number of signal bends is at most one in procedure 2. The following assumes that the multiple on-state via-switches are allowed only in the vertical direction without losing generality since the crossbar has a symmetrical structure. Also, turning on and off a via-switch is a symmetrical operation, and by swapping the voltages given to the signal line and control line, the same proof can be easily achieved. Therefore, the following proof only considers the situation in which via-switches are turned on.

In procedure 2-(a), it is necessary to prove that there is no sneak path problem in any possible configuration patterns newly added by the horizontal one-column crossbar extension. Here, the number of on-state via-switches in each row is zero or one because the multiple on-state via-switches in the horizontal direction are not allowed. Therefore, what the proof needs to clarify is that for each row with no on-state via-switches, the switches located at the expanded column can be turned on without the sneak path problem. The upper right of Figure 2.3 illustrates this example. The programming of a via-switch must turn on two atom switches, i.e., the upper atom switch connected to the horizontal signal line and the lower atom switch connected to the vertical signal line as shown in Figure 1.14. When the upper atom switch is under programming, a positive voltage is provided to the horizontal signal line (e.g., step 1 in Figure 1.14). Here, all other switches on the same horizontal line are off-state due to the programming constraint, and hence the signal given to the signal line never bends. On the other hand, the programming

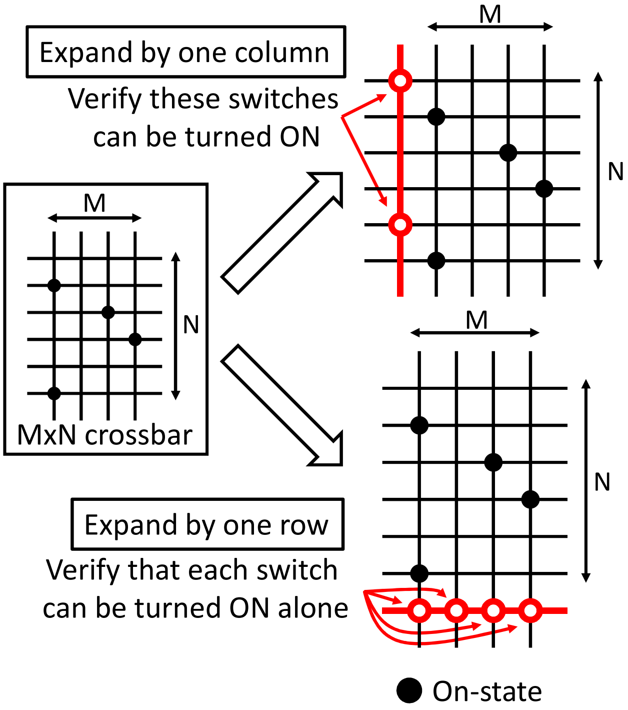

Expand by one column
Verify these switches can be turned ON
M
N
M
N
MxN crossbar
M
N
Expand by one row
Verify that each switch can be turned ON alone
● On-state

Figure 2.3: Crossbar expansion supposed in induction-based proof.

of the lower atom switch uses the vertical signal line (e.g., step 2 in Figure 1.14). In this case, the programming signal can be bent depending on whether there are on-state via-switches on the same vertical line. However, the signal never bends further because the multiple on-state via-switches in the horizontal direction are not allowed. From the above, the proof can conclude that the total number of signal bends is at most one and hence no sneak path problem arises in procedure 2-(a).

Next, let us move to the procedure 2-(b). The crossbar is expanded by one row in the vertical direction. Here, only one via-switch on the expanded row can be turned on because the multiple on-state via-switches in the horizontal direction are not allowed. Therefore, what the proof should verify is that any one of the switches on the expanded row can be turned on, which is illustrated in the lower right of Figure 2.3. When the upper atom switch is under programming, the programming signal given to the horizontal signal line never bends similar to the proof of procedure 2-(a). When the lower atom switch is under programming, the programming signal bends at most once. From the above, the total number of signal bends is one or less, and hence no sneak path problem arises in procedure 2-(b).

In summary, the above proof has completed procedures 1 and 2, and

Table 2.1: Comparison of supposed interconnect structures.

	Proposed	BFR	UFR	SRAM-based
Crossbar Switch	Via-Switch	Via-Switch	Via-Switch	Transmission Gate
Repeater Position	Separated from Signal Lines	Between CLBs	Between CLBs	Between CLBs
Repeater Insertion	Selective	Fixed	Fixed	Fixed
Repeater Circuit	Buffer	Cross-Coupled Tri-State Buffers	Buffer	Cross-Coupled Tri-State Buffers
Signal Direction	Bidirectional	Bidirectional	Unidirectional	Bidirectional

BFR: bidirectional fixed repeater structure.
UFR: unidirectional fixed repeater structure.

consequently clarifies that there is no sneak path problem in the programming of any-sized crossbar under the programming constraint that multiple on-state via-switches are allowed only in one direction.

2.2.4 Interconnect Structures for Performance Comparison

This subsection explains other interconnect structures supposed in this chapter for the comparison purpose. Table 2.1 summarizes these structures with the proposed one. To evaluate the effectiveness of the selective repeater insertion, this chapter supposes two structures for the performance comparison, namely bidirectional fixed repeater (BFR) structure and unidirectional fixed repeater (UFR) structure. In both the BFR structure and UFR structure, a repeater is placed on each signal line at the boundary between CLBs. Therefore, the repeater insertion frequently occurs every time the signal passes through each CLB in BFR and UFR structures. The difference between BFR and UFR structures is the repeater circuit. BFR structure uses cross-coupled tri-state buffers, and hence signal lines are bidirectional, while UFR structure allows only unidirectional signal transmission due to buffers located at the boundary between CLBs. In addition to the above structures, this work supposes an SRAM-based structure to compare the proposed structure with the conventional FPGA. In SRAM-based structure, a transmission gate is placed at each crossbar intersection and cross-coupled tri-state buffers are arranged at the boundary between CLBs.

2.3 Interconnect Performance Evaluation

This section evaluates the interconnect delay and the energy for signal transmission. Section 2.3.1 evaluates the performance improvement thanks to the selective repeater insertion and bidirectional signal transmission, which are the features of the proposed architecture. Section 2.3.2 compares the performance of the proposed and conventional architectures.

2.3.1 Performance Improvement Thanks to Selective Repeater Insertion and Bidirectional Signal Transmission

First, this subsection evaluates the dependence of the delay and energy on the crossbar size aiming to show that the smaller crossbar thanks to bidirectional signal can contribute to higher performance. In general, the bidirectional signaling can utilize programmable interconnections more effectively than the unidirectional signaling, and therefore the bidirectional crossbar size can be reduced compared to the unidirectional one. Ochi et al. reported that the crossbar sizes of 86x153 and 96x163 were required for bidirectional interconnection and for unidirectional interconnection, respectively, when a certain application of the same function is mapped [80]. The following evaluation constructs circuit models of 86x153 crossbar and 96x163 crossbar using the equivalent circuit model of the via-switch shown in Figure 2.4. The circuit models include wire resistance and capacitance of the signal lines. Then, by connecting the crossbar circuit models with inter-crossbar via-switches, the transistor-level netlists of the tiled crossbars are generated. The netlist also includes the LUTs and repeaters where 65 nm thin-buried-oxide fully-depleted-silicon-on-insulator (thin-BOX FD-SOI) transistor [96] that is suitable for the low-voltage operation is assumed. Then, HSPICE performs the circuit simulation and evaluates the propagation delay from the LUT output of the source CLB to the LUT input of the destination CLB by changing the number of CLBs between the source CLB and the destination CLB as illustrated in Figure 2.5.

Figure 2.6 shows the interconnect delay and the energy per signal transition. In this evaluation, no repeaters are inserted. The result shows that the interconnect delay depends on the crossbar size and it decreases by 11% when the crossbar size is reduced from 96x163 to 86x153. The energy per signal transition also depends on the crossbar size and it decreases by 10%. Here, these reduction ratios are evaluated at the distance of seven CLBs since the average distance in the mapping results is roughly seven CLBs [80]. The crossbar size reduction thanks to bidirectional interconnection is effective for interconnect delay and energy reduction.

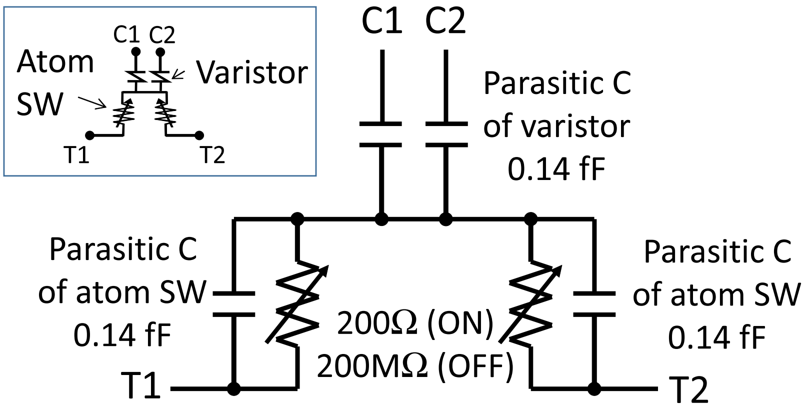

C1 C2
Atom SW
Varistor
T1
T2
C1
C2
Parasitic C of varistor 0.14 fF
Parasitic C of atom SW 0.14 fF
Parasitic C of atom SW 0.14 fF
200Ω (ON)
200MΩ (OFF)
T1
T2

Figure 2.4: An equivalent circuit model of via-switch in normal operation.

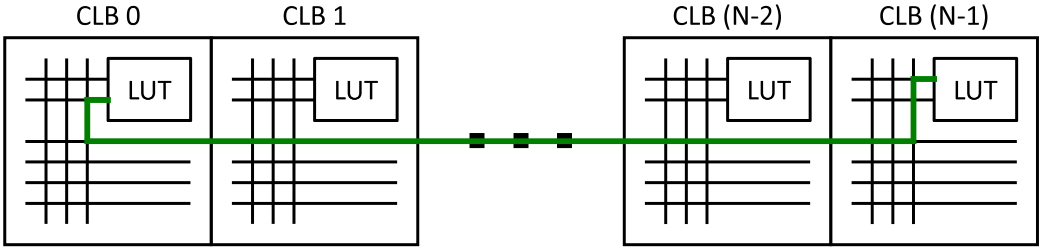


Figure 2.5: Signal transmission path in performance evaluation.

Then, Figure 2.7 compares the proposed structure with BFR and UFR structures in terms of the interconnect delay and energy. Three structures have comparable delay time in short distance transfer up to seven CLBs. Here, the delay in the proposed structure increases in proportion to the square of the distance and becomes larger in long distance transmission. The later evaluation demonstrates that the selective repeater insertion successfully eliminates this quadratic effect and optimizes the delay even in the long distance transmission. Focusing on the energy, on the other hand, the proposed structure has the lowest energy compared to BFR and UFR structures. Figure 2.8 compares the energy-delay product, which is a good metric of energy efficiency. When the distance is seven CLBs, the reduction ratio of the proposed structure from BFR and UFR structures are 67% and 46%, respectively. This result indicates that the proposed structure is suitable for the high energy efficient signal transfer.

Next, this paragraph evaluates the impact of selective repeater insertion. Figure 2.9 shows the relations of interconnect delay and energy to the distance between the source LB and the destination LB. By inserting repeaters, the delay

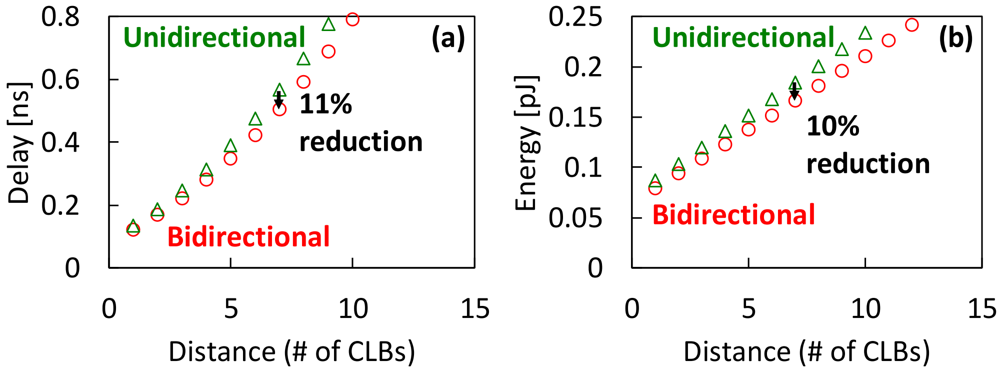


Figure 2.6: (a) interconnect delay and (b) energy per signal transmission in the proposed interconnect structure. Two crossbar sizes of 86x153 and 96x163 are evaluated. Repeaters are not inserted. Supply voltage is 1.0 V.

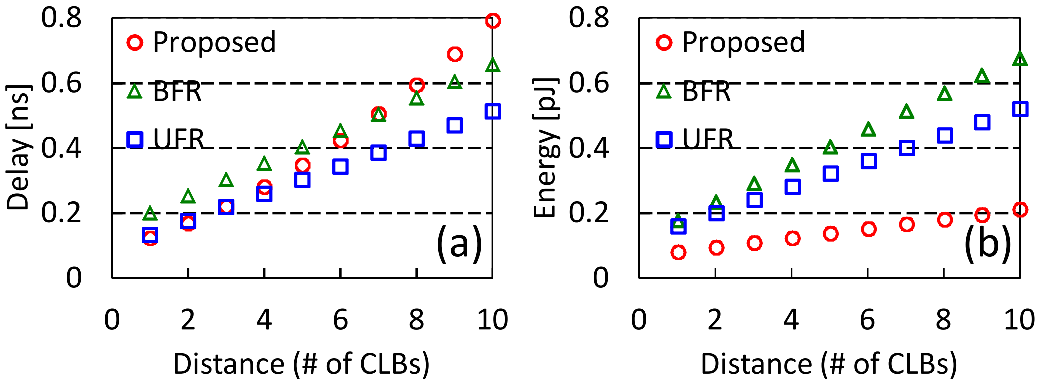


Figure 2.7: (a) interconnect delay and (b) energy per signal transmission in the proposed, BFR, and UFR structures. The proposed and BFR structures are 86x153 crossbar and UFR structure is 96x163 crossbar. Supply voltage is 1.0 V.

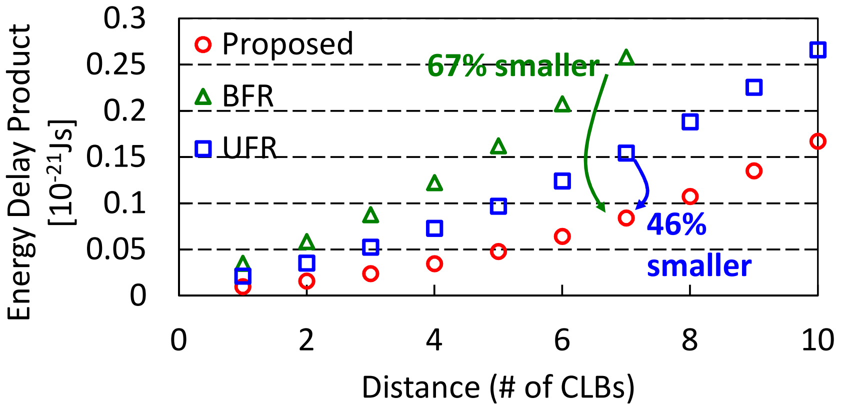


Figure 2.8: Energy-delay product per signal transmission in the proposed, BFR, and UFR structures. The proposed and BFR structures are 86x153 crossbar and UFR structure is 96x163 crossbar. Supply voltage is 1.0 V.

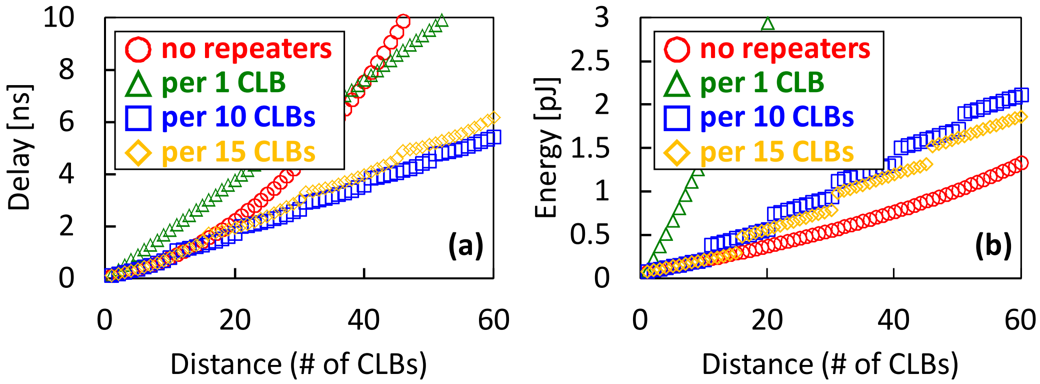


Figure 2.9: (a) interconnect delay and (b) energy per signal transmission in the proposed interconnect structure. Crossbar size is 86x153. Supply voltage is 1.0 V.

becomes proportional to the distance as is expected. The result also indicates that the frequent repeater insertion, e.g., per 1 CLB in Figure 2.9, leads to significant increase in both delay and energy. Therefore, it should be avoided to insert repeaters frequently in via-switch based FPGA. In the case of short-distance transmission, no repeater insertion is needed for minimizing the delay and energy. When the distance is more than 14 CLBs, the repeater insertion per 10 CLBs achieved the minimum delay. On the other hand, the insertion per 15 CLBs reduces the energy with a small delay increase. From these results, the proposed interconnect structure can insert repeaters according to the timing constraint. The routing for selective repeater insertion is accomplished by CAD tools.

In the mapping result reported in reference [80], the most frequent distance from the source to the destination was six CLBs and the ratio of the interconnections whose distance is longer than 14 CLBs was only 2.3%. For 97.7% of interconnections, no repeaters are needed for delay minimization. For larger designs, longer interconnection will appear, but its frequency is expected to be still not high since such a tendency is observed with Rent's rule [97]. Therefore, the number of repeaters is expected to be small. In addition, by introducing long wires, which can be easily accommodated, the proposed structure further reduces the number of repeater insertion. Thus, the flexible repeater insertion well fits the proposed interconnect structure.

2.3.2 Performance Comparison between Proposed and Conventional Architectures

This subsection compares the interconnect delay and energy between the proposed and conventional SRAM-based architectures. The transistor-level netlist of the SRAM-based crossbar is implemented with complementary pass gates and SRAM cells, and the crossbars are connected by back-to-back tristate buffers with SRAM cells for enabling bidirectional signaling. The netlist also includes wire resistance and capacitance. Referring an industrial 65 nm cell library, the crossbar size is estimated by the number of transistors. Here, the 86x153 SRAM-based crossbar size is 223.6 μm × 275.4 μm, and it is 26 times larger compared to the via-switch crossbar which is 516 F × 459 F = 51.6 μm × 45.9 μm. This means the proposed architecture can achieve 26X higher crossbar density. Figure 2.10 shows the performance comparison between the proposed and SRAM-based structures by changing the signal transmission distance. The proposed architecture attains significant delay and energy reduction. The reduction ratios of delay and energy from the SRAM-based structure are 67% and 88%, respectively.

So far, the supply voltage is fixed at 1.0 V. This paragraph sweeps the supply voltage in the range from 0.5 V to 1.0 V to evaluate the performance at the low-voltage operation. Figure 2.11 shows the evaluation result. Here, the signal transmission with seven CLBs distance is assumed, where please remind that seven CLBs is the average distance in the mapping result [80]. The total energy, which is depicted in Figure 2.11-(b), is the sum of all the interconnect energies in the mapping result assuming that a single pulse propagates through each interconnect. Figure 2.11-(a) shows that the interconnect delay reduction ratio from the SRAM-based FPGA becomes higher as the supply voltage becomes lower. At 0.5 V, the ratio of interconnect delay reduction reaches 90% because the ON-resistance of the via-switch is independent of the supply voltage while that of conventional transistor switch depends on the supply voltage. The interconnect delay increasing ratio from 1.0 V to 0.5 V of the proposed architecture is only

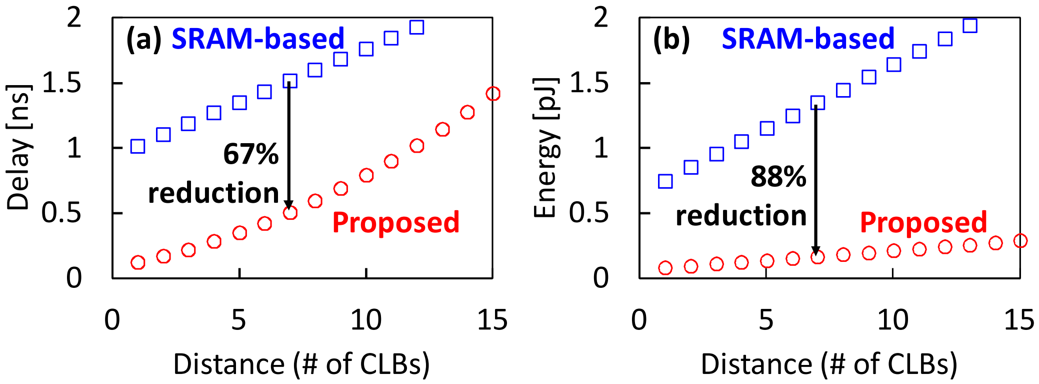


Figure 2.10: Comparison between the proposed and SRAM-based architectures. Repeaters are not inserted. Crossbar size is 86x153. Supply voltage is 1.0 V.

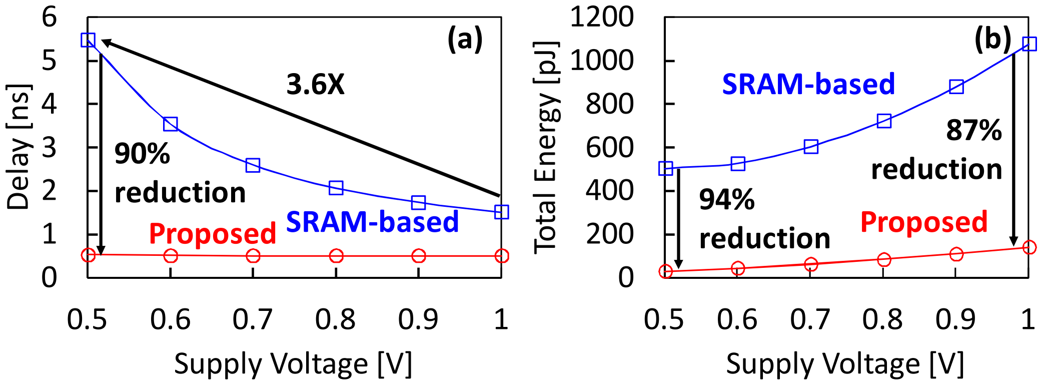


Figure 2.11: (a) Interconnect delay and (b) total energy in the proposed and SRAM-based architectures when supply voltage is varied. Repeaters are not inserted. Crossbar size is 86x153.

1.1X, whereas that of the SRAM-based architecture is 3.6X. The evaluation also indicates that the energy reduction ratio becomes higher with voltage decrease. Focusing on 0.5 V operation, the energy reduction is 94%. These evaluation results show that the proposed architecture can achieve high performance even at low supply voltage.

The next evaluation investigates the impact of on-resistance of via-switch by increasing it from 400 Ω to 1200 Ω. The evaluation results are shown in Figure 2.12. The signal transmission distance is seven CLBs. The delay increases in proportion to the on-resistance. However, even when the on-resistance rises to 1200 Ω, the delay reduction from the SRAM-based architecture is still achievable, especially a large reduction of 76% is attained at 0.5 V operation. On the other hand, the total energy has almost no changes even when the on-resistance rises,

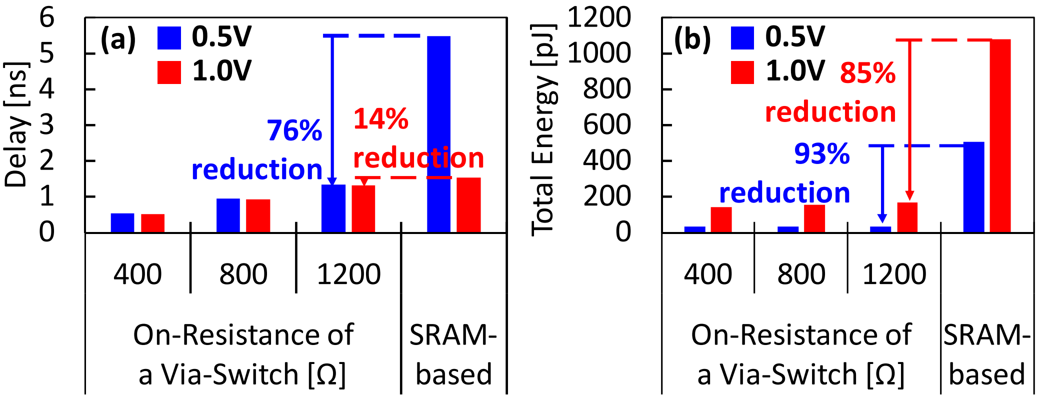


Figure 2.12: (a) Interconnect delay and (b) total energy in the proposed and SRAM-based architectures when the ON-resistance of via-switch is varied. Repeaters are not inserted. Crossbar size is 86x153.

and it is reduced nearly by one order of magnitude from the SRAM-based one at both 1.0 V and 0.5 V operations.

All the above evaluations suppose dense crossbars where via-switches are placed at all the intersections. On the other hand, sparser or depopulated switch block which has fewer intersection switches are often used in conventional FPGAs. The sparsened switch box may degrade the routing flexibility but can reduce the area [98]. In the via-switch FPGA, the sparse crossbar decreases the number of via-switches connected to each signal wire and consequently reduces the load capacitance of each signal wire, whereas the crossbar area is unchanged. In the case of conventional FPGA, the crossbar area is also reduced since the number of transistors is reduced.

This paragraph evaluates the impact of crossbar sparseness on interconnect performance. The evaluation sweeps the ratio of switch removal in the range from 0%, i.e., fully dense crossbar, to 70%. Figure 2.13 shows the crossbar area. Even when the removal ratio reaches 70%, the via-switch crossbar is still 8X smaller than the SRAM-based one. Figure 2.14 shows the interconnection delay and energy, where the crossbar size is 86x153, the supply voltage is 1.0 V, and the signal transmission distance is seven CLBs. As shown in Figure 2.14, both delay and energy decrease according to via-switch removal. When the removal ratio is 70% and on-resistance of via-switch is 400 Ω, the reduction ratios of delay and energy from the full matrix crossbar are 45% and 46%, respectively. The result also shows that the delay and energy of via-switch FPGA remain much smaller than those of SRAM-based FPGA even with the switch removal, and the reduction ratios of delay and energy are 69% and 87%, respectively.

The above results indicate that the proposed architecture achieves significant

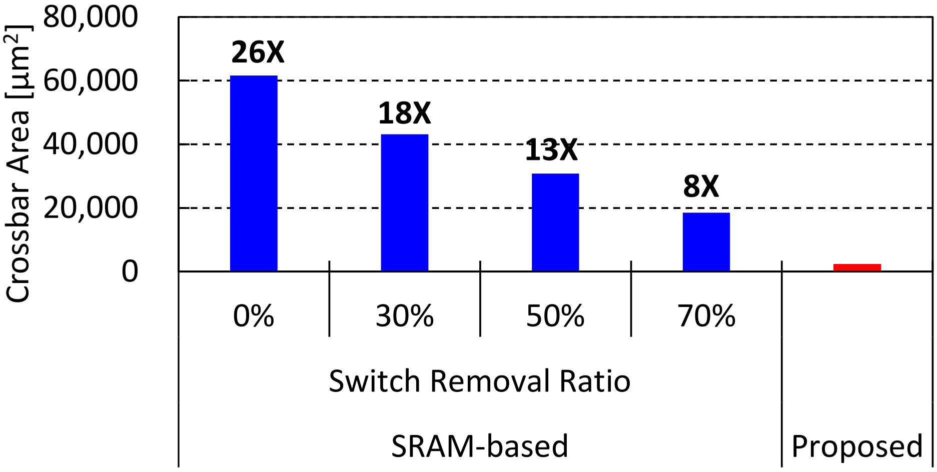


Figure 2.13: Crossbar area variation in terms of switch removal.

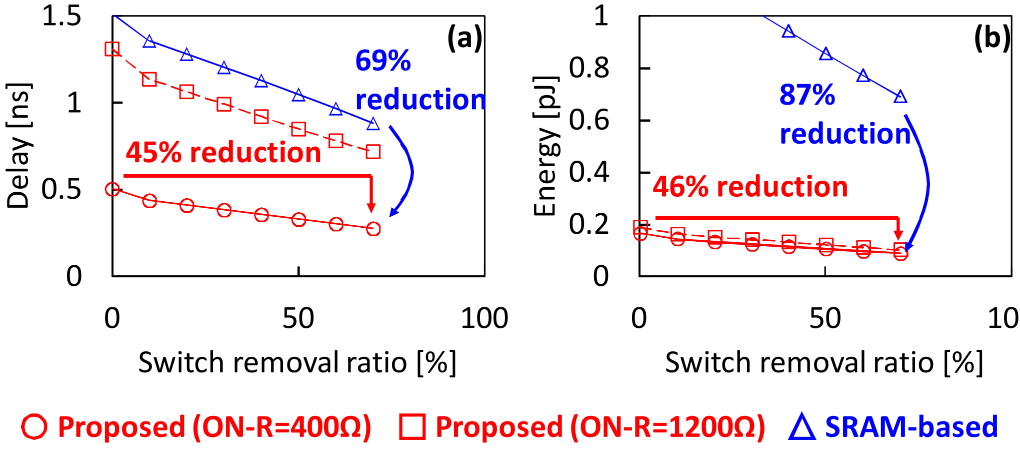


Figure 2.14: (a) Interconnect delay and (b) energy in the proposed and SRAM-based architectures when the thinning ratio of intersection switches is varied. Repeaters are not inserted. Crossbar size is 86x153.

performance improvement compared with the conventional architectures. In particular, it is more significant at the low-voltage operation, and the interconnect delay and energy are reduced by one order of magnitude or more at 0.5 V operation. This improvement can contribute to filling the gap between FPGA and ASIC.

2.4 Conclusion

This chapter has proposed an energy-efficient interconnect structure of via-switch FPGA. The proposed structure can selectively insert repeaters for optimizing the delay according to the signal transmission distance, and adopts bidirectional signaling. These features of the proposed structure contribute to improving the interconnect performance. This chapter also discussed the programming structure of the crossbar focusing on a constraint-based countermeasure for the sneak path problem. This work gave a formal proof that the constraint-based countermeasure works fine, and also identified the requirement of programming structure at inter-CLB connection switches to prevent the sneak path problem. Evaluation results based on transistor-level SPICE simulations show that the proposed interconnect structure can achieve up to 26X higher crossbar integration density and reduce interconnect delay and energy by 90% and 94% at 0.5 V operation compared to conventional SRAM-based crossbars.

Chapter 3

Fault Diagnosis of Via-Switch Crossbar

This chapter proposes a fault diagnosis methodology that identifies the fault modes of via-switches in the crossbar [99]. This work is the first one to investigate the fault testing and diagnosis of the via-switch based crossbar. First, Section 3.1 confirms that a general comparator can distinguish on/off-states of via-switches in the crossbar, and clarifies fault modes of a via-switch by transistor-level SPICE simulation. Then, Section 3.2 proposes a fault diagnosis methodology for via-switches in the crossbar. Section 3.3 discusses the relation between a fault rate of via-switch and a percentage of faulty via-switches in a practical-sized crossbar with simulations of the fault injection, and also confirms that the proposed method is suitable for the practical use. Finally, a summary of this chapter is given in Section 3.4.

3.1 Fault Mode Analysis of Via-Switch

To verify the via-switch crossbar functionality after manufacturing, fault testing that checks whether via-switches can be securely turned on and off is indispensable. Aiming at developing a fault testing method, this section first analyzes fault modes of a via-switch. Section 3.1.1 confirms that a general comparator can distinguish on/off-states of via-switches in the crossbar. Section 3.1.2 then investigates and identifies fault modes of a via-switch by transistor-level SPICE simulation. Based on the discussion in this section, the next section will propose a fault diagnosis method.

3.1.1 Discriminating Via-Switch On/Off-States with Comparator

To discriminate via-switch on/off-states in the crossbar, this work adds a differential pair comparator that connects to every programming driver through a transistor switch as shown in Figure 3.1. This figure illustrates the connection between the comparator, programming drivers, an array that contains four 2x2 crossbars. Here, all the crossbars can share the programming driver by using NMOS pass transistors while Figure 1.14 depicts a driver for each wire. The comparator can also be shared by all the crossbars. Therefore, the proposed fault testing method is feasible by adding only one comparator, and the peripheral circuit for testing is negligibly small for a practically large CLB array.

The read operation applies a voltage to the target atom switch in the same manner as programming operation and turns on only the transistor switch that connects the comparator with the driver that is outputting a ground voltage, which is illustrated in Figure 3.1. In this read operation, the comparator observes the voltage drop in the target atom switch and compares it with a given reference voltage. Here, the applied voltage in the read operation is lower than the programming voltage, and therefore this operation never changes the on/off-states of the target switch. By giving an appropriate reference voltage that makes the comparator output different depending on on/off-states of the target switch, the comparator can read the switch states. The reference voltage is provided as an analog voltage from a large scale integration (LSI) tester outside the chip. In this read method, two programming drivers apply the voltage to a pair of an atom switch and a varistor, and therefore this work calls this operation as atom switch-varistor read (ASV-read) operation.

Table 3.1 summarizes the comparator output simulated by transistor-level SPICE simulation in both cases that the target atom switch is on-state and off-state varying the reference voltage. In this chapter, the crossbar size is set to 90x127 for practical use. Table 3.1 shows that the comparator output changes from 0 to 1 according to an increase in the reference voltage, and the reference voltage at the boundary between 0 and 1 differs depending on the on/off-states of the target atom switch. The boundary reference voltage for the on-state atom switch is higher than that for off-state atom switch, and the voltage difference is about 50 mV as shown in the row of ASV-read in Table 3.2. General LSI testers can provide the analog voltage with millivolt accuracy, and therefore the comparator can discriminate the on/off-states by exploiting the boundary difference between on and off states. For example in Table 3.1, the atom switch state can be distinguished by choosing 0.56 V as the reference voltage.

The boundary reference voltage depends on the on-resistance of the via-switch. Figure 3.2 shows the boundary reference voltage obtained by SPICE

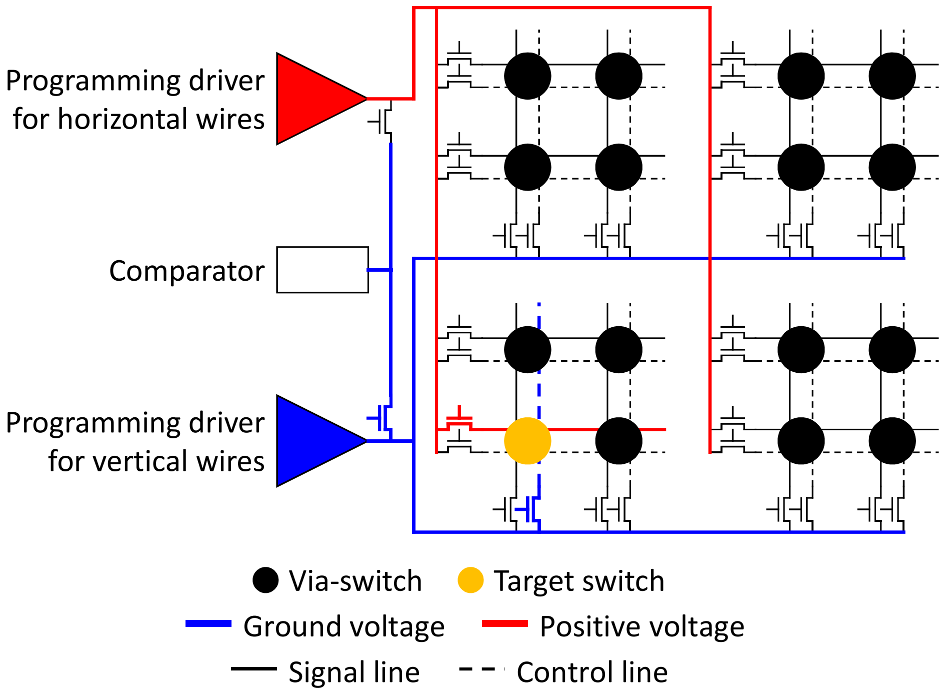


Figure 3.1: Connection between comparator, programming drivers, and crossbar array.

simulations varying the on-resistance. On the other hand, even when the on-resistance varies from 1 kΩ to 10 kΩ, the voltage difference between on and off states is still tens of millivolts. Therefore, the comparator can distinguish the on/off states. Also, the location of the target via-switch in the CLB array affects the boundary reference voltage because the interconnect resistance varies depending on the via-switch location. However, the interconnect resistance is about one order of magnitude lower than the via-switch resistance, and hence the impact of the via-switch location on the boundary reference voltage is relatively small. SPICE simulations confirmed that the comparator successfully discriminates the on/off states of via-switches at different locations.

In addition to the above ASV-read, this work introduces two read methods, namely complementary atom switch read (CAS-read) operation and two varistors read (TVR-read) operation. CAS-read applies a read voltage to the CAS by activating a pair of drivers that drive the perpendicular two signal lines crossing at the target intersection. One the other hand, TVR-read uses perpendicular two control lines to apply a read voltage to two varistors connected in series in a via-switch. Figure 3.3 illustrates each path to apply a read voltage in ASV-read, CAS-read, and TVR-read. Here, this work does not use the signal path using parallel signal and control lines at the target intersection, where the signal also

Table 3.1: Comparator output when reference voltage is varied in read operation of atom switch.

Reference	Comparator output	
voltage [V]	Atom switch is on	Atom switch is off
0.50	0	0
0.52	0	0
0.54	0	1
0.56	0	1
0.58	0	1
0.60	1	1

Table 3.2: Boundary reference voltage in ASV-, CAS-, and TVR-read.

Read type	Target switch state	
	On-state	Off-state
ASV-read	0.58 V	0.53 V
CAS-read	0.70 V	0.53 V
TVR-read	0.58 V	0.58 V

passes through an atom switch and a varistor, due to the sneak path problem. In both CAS-read and TVR-read, one driver gives a positive voltage and the other applies a ground voltage. Both the operations turn on only the transistor switch that connects the comparator with the driver outputting a ground voltage such that the voltage of interest is delivered to the comparator. The applying voltage in CAS-read is lower than the programming voltage. On the other hand, in TVR-read, the drivers apply a voltage of the same level as the programming voltage to check the varistors state correctly.

The SPICE simulation has confirmed that the comparator response in CAS-read is similar to ASV-read except for the absolute value of the reference voltage. The comparator output changes from 0 to 1 as the reference voltage elevates, and the boundary reference voltage for the on-state CAS is higher than that for off-state CAS, where on-state CAS and off-state CAS mean both of atom switches in the CAS are on-state and off-state, respectively. The row of CAS-read in Table 3.2 shows the values of the boundary reference voltage. The SPICE simulation also confirms that the comparator response for a CAS containing one on-state atom switch and one off-state atom switch is the same as the response to off-state CAS. Meanwhile, the boundary in TVR-read does not change regardless of on/off-states of atom switches as shown in Table 3.2 since there is no atom switch in the signal path in TVR-read.

ASV-read uses both an atom switch and a varistor, whereas CAS-read uses

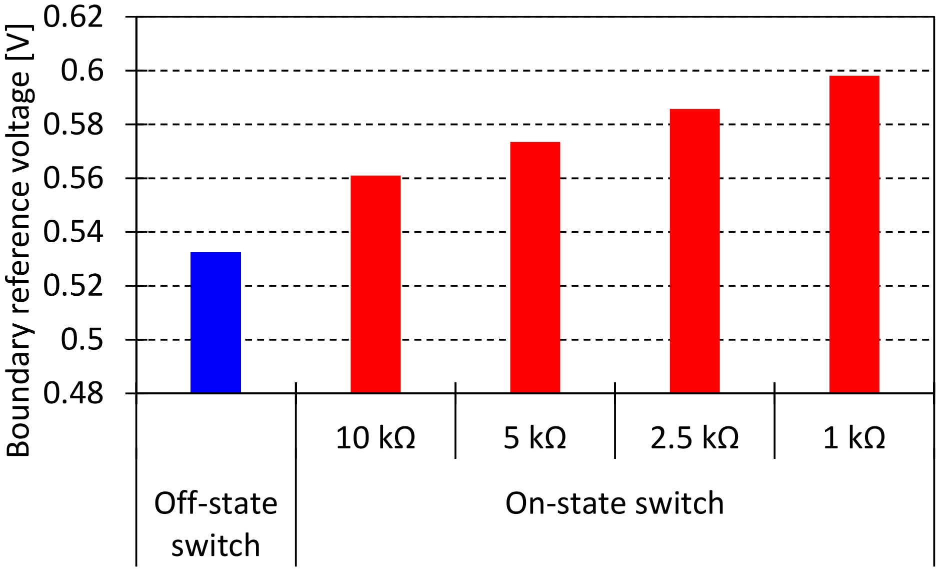


Figure 3.2: Boundary reference voltage of comparator when via-switch on-resistance is varied.

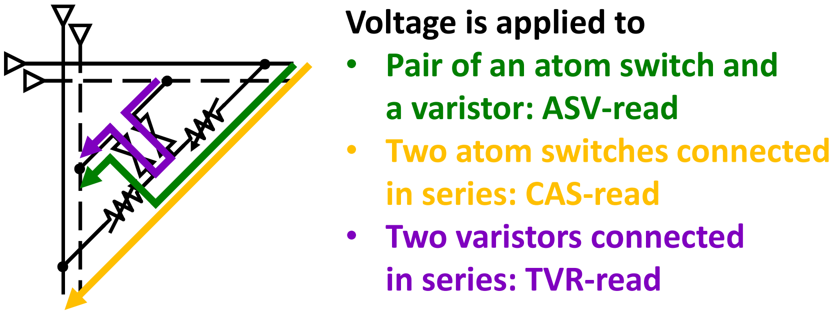

Voltage is applied to
• Pair of an atom switch and a varistor: ASV-read
• Two atom switches connected in series: CAS-read
• Two varistors connected in series: TVR-read

Figure 3.3: Path to apply read voltage in ASV-, CAS-, and TVR-read.

only atom switches and TVR-read uses only varistors. By combining the comparator responses in these three read operations, this work can improve the fault diagnosis capability. The details will be explained in Section 3.2.

3.1.2 Via-Switch Fault Modes

This subsection discusses how the comparator response varies when a via-switch includes faulty atom switch or varistor. This work injects stuck-on/off faults to atom switch and varistor, and evaluates the comparator response by SPICE simulation. Here, stuck-on/off faults mean that the two terminals of atom switch or varistor are shorted/opened.

Table 3.3: Boundary reference voltage in ASV- and CAS-read with faulty varistor.

Read type	Target switch state	Varistor fault type	
		Stuck-on	Stuck-off
ASV-read	On-state	0.77 V	0.53 V
	Off-state	0.53 V	0.53 V
CAS-read	On-state	0.70 V	0.70 V
	Off-state	0.53 V	0.53 V

First, this paragraph studies the case where the atom switch is stuck-on/off. When an atom switch is stuck-on, the boundary reference voltage is unchanged from the non-faulty on-state case even after the drivers apply a programming voltage to turn off the atom switch. Then, this observation indicates that there is a fault. The same discussion holds for the stuck-off case. The CAS can be in a state where one atom switch is on-state and the other is off-state in addition to the states that both atom switches are on-state or off-state. As mentioned in the previous subsection, when at least one atom switch is stuck-off in a CAS, the boundary reference voltage is identical to the boundary of CAS-read for off-state CAS. Stuck-on/off faults of atom switches do not affect the boundary in TVR-read as explained in the previous subsection.

Next, the following discusses the case where the varistor is stuck-on/off. Table 3.3 summarizes the boundary reference voltage in ASV-read and CAS-read with faulty varistor. Focusing on ASV-read with stuck-on varistor in Table 3.3, the boundary for the on-state switch changes from that of the normal case, specifically from 0.58 V in Table 3.2 to 0.77 V in Table 3.3. Therefore, this observation can know there is a fault. On the other hand, when reading the off-state atom switch, the boundary is the same for normal and stuck-on cases. In ASV-read with stuck-off varistor, the boundary is fixed to 0.53 V, which is the boundary in the normal case with off-state switch, regardless of on/off-states of the target switch.

The row of CAS-read in Table 3.3 indicates that the boundary reference voltage for faulty varistor does not change from the normal case. The CAS-read operation applies a read voltage only to the CAS, and hence stuck-on/off faults of the varistor do not affect the comparator response.

Table 3.4 shows the boundary reference voltage of TVR-read in normal and faulty cases. When either varistor in a via-switch is stuck-off, the boundary voltage drops from the normal boundary. On the other hand, when both varistors are not stuck-off and either varistor is stuck-on, the boundary voltage rises compared to the normal case.

This work utilizes these differences in the boundary reference voltage between

Table 3.4: Boundary reference voltage in TVR-read with normal and faulty varistors.

Varistors state	Boundary reference voltage
No fault	0.58 V
If either varistor is stuck-off	0.53 V
Else if either varistor is stuck-on	0.72 V

normal and faulty cases for the fault diagnosis method proposed in the next section. It should be noted that the comparator response for a via-switch with multiple faulty components is a combination of the above fault modes.

3.2 Proposed Fault Diagnosis Method

This section proposes a fault diagnosis method for the via-switch crossbar exploiting the comparator response difference between normal and faulty via-switches explained in the previous section. First, Section 3.2.1 clarifies prerequisites in the proposed method. Then, Section 3.2.2 proposes a fault diagnosis method.

3.2.1 Prerequisites

The proposed method assumes the following prerequisites.

- Even when a varistor is stuck-on, the drivers can program the corresponding atom switch normally. This can be achieved by a current-limiting circuit that restricts the programming current appropriately.
- When a varistor is stuck-off, the drivers cannot program the corresponding atom switch since the programming current cannot be provided to the target atom switch.
- Initial state of non-faulty atom switch is off-state, which is a feature of via-switch.
- There is no fault in the comparator, programming drivers, and interconnect wires.

3.2.2 Fault Diagnosis

This subsection proposes a fault diagnosis method that identifies faulty components in a via-switch on the crossbar. The proposed method utilizes the difference of the boundary reference voltage of the comparator in read operation discussed in Section 3.1.

The proposed method enumerates all the combinations of stuck-on/off faults of two atom switches and two varistors in a via-switch. Then, the proposed method makes a look-up table beforehand that summarizes the boundary reference voltage of ASV-read, CAS-read, and TVR-read after turning on/off the target switch for each fault combination. When actually diagnosing faults in a via-switch, the proposed method investigates the boundary of three read operations after programming the target switch, performs a pattern matching with the look-up table prepared above, and identifies the faults.

Table 3.5 enumerates all the patterns of comparator response when the number of faulty components in a via-switch is up to two, which correspond to the left half of the table. A via-switch has four components and each component can be stuck-on/off. Then, supposing the number of faulty components is n, the number of combinations of fault components is given by ${}_4\mathrm{C}_n \times 2^n$. Therefore, the number of combinations in case of up to n faulty components can be calculated by $\sum_{k=0}^{n} {}_4\mathrm{C}_k \times 2^k$. When n is 2, there are 33 combinations listed in Table 3.5.

The proposed method performs ASV-read operations for both cases after turning on and off the target atom switch, where this operation corresponds to "US"/"LS" and "UR"/"LR" in Table 3.5, respectively. For example, "US" and "LR" stand for "Upper atom switch is Set" and "Lower atom switch is Reset", respectively. For a CAS, there are four combinations to turn on (S) and off (R) both upper and lower atom switches, and hence the proposed method evaluates the boundary reference voltage in all four cases, which are "SS", "SR", "RS", and "RR" in Table 3.5. The proposed method also uses TVR-read operation in the proposed method. For each via-switch, the above read operations can be attained by reprogramming the via-switch only once. The following steps exemplify a procedure of read operations.

1. Initial state of upper and lower atom switches is (upper: off-state, lower: off-state).

2. Turn on the upper atom switch [switch state is (on, off)].

3. Perform the upper ASV-read operation ("US").

4. Perform the CAS-read operation ("SR").

5. Turn on the lower atom switch [switch state is (on, on)].

6. Perform the lower ASV-read operation ("LS").

7. Perform the CAS-read operation ("SS").

Table 3.5: Comparator response difference and diagnosability in case of up to two faulty components in a via-switch.

ID	Fault states of via-switch components												Read operation results									Diag.	
	Upper VR			Lower AS			Lower VR			Upper AS			U-ASV		L-ASV		CAS				TVR	1F	2F
	NF	SN	SF	NF	SN	SF	NF	SN	SF	NF	SN	SF	US	UR	LS	LR	SS	SR	RS	RR			
1	✓			✓			✓			✓			N	N	N	N	N	N	N	N	N	Yes	Yes
2	✓			✓			✓				✓		M	H	N	N	M	M	H	M	N	Yes	Yes
3	✓			✓			✓					✓	L	M	N	N	L	M	M	M	N	Yes	Yes
4	✓			✓				✓		✓			R	M	N	N	N	N	N	N	R	Yes	Yes
5	✓			✓				✓			✓		R	R	N	N	M	M	H	M	R	—	Yes
6	✓			✓				✓				✓	L	M	N	N	L	M	M	M	R	—	Yes
7	✓			✓					✓	✓			L	M	N	N	L	M	M	M	D	Yes	No^1
8	✓			✓					✓		✓		L	M	N	N	M	M	H	M	D	—	Yes
9	✓			✓					✓			✓	L	M	N	N	L	M	M	M	D	—	No^1
10	✓				✓		✓			✓			N	N	M	H	M	H	M	M	N	Yes	Yes
11	✓				✓		✓				✓		M	H	M	H	M	H	H	H	N	—	Yes
12	✓				✓		✓					✓	L	M	M	H	L	M	M	M	N	—	Yes
13	✓				✓			✓		✓			R	M	M	H	M	H	M	M	R	—	Yes
14	✓				✓				✓	✓			L	M	M	H	L	M	M	M	D	—	Yes
15	✓					✓	✓			✓			N	N	L	M	L	M	M	M	N	Yes	Yes
16	✓					✓	✓				✓		M	H	L	M	L	M	M	M	N	—	Yes
17	✓					✓	✓					✓	L	M	L	M	L	M	M	M	N	—	Yes
18	✓					✓		✓		✓			R	M	L	M	L	M	M	M	R	—	Yes
19	✓					✓			✓	✓			L	M	L	M	L	M	M	M	D	—	No^2
20		✓		✓			✓			✓			N	N	R	M	N	N	N	N	R	Yes	Yes
21		✓		✓			✓				✓		M	H	R	M	M	M	H	M	R	—	Yes
22		✓		✓			✓					✓	L	M	R	M	L	M	M	M	R	—	Yes
23		✓		✓				✓		✓			R	M	R	M	N	N	N	N	R	—	Yes
24		✓		✓					✓	✓			L	M	R	M	L	M	M	M	D	—	Yes
25		✓			✓		✓			✓			N	N	R	R	M	H	M	M	R	—	Yes
26		✓				✓	✓			✓			N	N	L	M	L	M	M	M	R	—	Yes
27			✓	✓			✓			✓			N	N	L	M	L	M	M	M	D	Yes	No^3
28			✓	✓			✓				✓		M	H	L	M	L	M	M	M	D	—	Yes
29			✓	✓			✓					✓	L	M	L	M	L	M	M	M	D	—	No^2
30			✓	✓				✓		✓			R	M	L	M	L	M	M	M	D	—	Yes
31			✓	✓					✓	✓			L	M	L	M	L	M	M	M	D	—	No^2
32			✓		✓		✓			✓			N	N	L	M	M	H	M	M	D	—	Yes
33			✓			✓	✓			✓			N	N	L	M	L	M	M	M	D	—	No^3

VR: varistor, AS: atom switch, NF: no fault, SN/SF: stuck-on/off
U-ASV/L-ASV: ASV-read of upper/lower atom switch
US/UR/LS/LR: read after turning on/off/on/off upper/upper/lower/lower atom switch
SS/SR/RS/RR: read after turning on/on/off/off upper atom switch and turning on/off/on/off lower atom switch
N: normal response, M: fault is masked, H/L: boundary is the same as on-state/off-state switch, R/D: boundary rises/drops
Diag.: diagnosability, 1F/2F: up to one/two faulty components in a via-switch
Rows that have the same superscript number of "No" in diagnosability column share the same comparator response.

8. Turn off the upper atom switch [switch state is (off, on)].
9. Perform the upper ASV-read operation ("UR").
10. Perform the CAS-read operation ("RS").
11. Turn off the lower atom switch [switch state is (off, off)].
12. Perform the lower ASV-read operation ("LR").
13. Perform the CAS-read operation ("RR").
14. Perform the TVR-read operation.

There are six characters that represent the state of the boundary reference voltage in Table 3.5. “N” means that the component has no fault and the boundary is normal. When there are faulty components but the boundary is the same as normal, it expressed as “M”, e.g., when the comparator reads a stuck-on switch after turning on the switch. When the boundary is expected to be that of the off-state switch but is the same as the on-state switch, this work categorizes this case as “H”. For example, “H” arises when the comparator reads a stuck-on switch after turning off the switch. “L” is the opposite situation to “H”. “R” and “D” correspond to the cases that the boundary rises and drops from the normal, respectively. After obtaining the pattern of these six characters with ASV-read of upper and lower atom switches, CAS-read, and TVR-read, the proposed method diagnose faulty components in a via-switch.

The following paragraphs discuss fault detectability and diagnosability. Here, the fault detection only evaluates whether the via-switch has faulty components, while the fault diagnosis identifies faulty components in the via-switches and their fault types. First, this paragraph evaluates fault detectability. The ASV-read of an upper atom switch uses the upper atom switch and the lower varistor. Here, ID #1-9 in Table 3.5 cover all combinations of non-faulty and stuck-on/off upper atom switch and lower varistor. In this case, the response of the ASV-read, which corresponds to the column of “U-ASV”, becomes (“N”, “N”) only when both upper atom switch and lower varistor have no fault. The response of the remaining eight cases is different from (“N”, “N”). By utilizing this difference, the proposed method can detect whether a pair of upper atom switch and lower varistor are faulty. The same discussion holds in the ASV-read of lower atom switch with the upper varistor. Therefore, the proposed method can achieve 100% fault detection of a via-switch by using ASV-read for both upper and lower atom switches.

On the other hand, in terms of fault diagnosability, the above observation cannot identify the faulty components in a via-switch uniquely only with the ASV-read. The column of “U-ASV” in Table 3.5 indicates that ID #3 and 6-9 have the same response of (“L”, “M”), and hence ASV-read cannot distinguish these patterns. This is mainly because the boundary reference voltage of ASV-read for stuck-off varistor is fixed to that in the normal case explained in Section 3.1. For improving fault diagnosability, the proposed method combines the responses of ASV-read, CAS-read, and TVR-read. The column of “Diag.” shows that fault diagnosability using these three read methods in cases that there are up to one and up to two fault components in a via-switch. When the comparator response is unique in the table, the corresponding fault is diagnosable. The table demonstrates that the proposed method can identify the fault component perfectly when there is up to one fault component in a via-switch. When there are up to two faulty components, the diagnosability ratio is $26/33 \times 100 = 79\%$. On the other hand, when only AVS-read is used, this ratio decreases to 33%. CAS-read and TVR-read

Table 3.6: Diagonosable faults ratio.

Maximum number of faults	Number of fault patterns	Number of diagnosable patterns	Diagnosis ratio [%]
1	9	9	100
2	33	26	79
3	65	34	52
4	81	34	42

help elevate the fault diagnosability by 46%.

Table 3.6 summarizes the fault diagnosis ratio when the maximum number of faulty components in a via-switch is varied from 1 to 4. When the maximum number of fault is 1, the proposed method can discriminate the faulty component and fault type no matter which component is stuck-on/off. The table also indicates that the diagnosis ratio diminishes as the maximum number of faulty components increases. This is because the number of fault patterns that have the same response of read operations increases. Fortunately, the probability that there are 3 or 4 faulty components in a via-switch is low. The next section discusses a relation between the number of faulty components and the fault rate of via-switch components in a practically-sized crossbar, and confirms that the proposed method is effective for practical use.

3.3 Discussion

This section investigates the relation between a fault rate of via-switch components, a percentage of faulty via-switches, and the number of faulty components in a via-switch in a practical-sized crossbar. Then, this section confirms that identifying the faulty component in via-switches where there is one faulty component in the via-switch is the most important in the crossbar for practical use, and, from this point of view, the proposed method is suitable.

Figure 3.4 shows a percentage of faulty via-switches in a 100x100 crossbar when a fault rate of via-switch components is varied from 0.01 to 0.25. This evaluation randomly injects faults assuming that four components in a via-switch have the same fault rate, and plots the average value of 10,000 trials. This figure also categorizes faulty via-switches according to the number of faulty components. The result demonstrates that via-switches with a single faulty component are dominant, especially when the fault rate is low. When the fault rate of each component is 0.1, the percentage of faulty via-switches reaches more than 30%. Therefore, in the crossbar that has a practical percentage of faulty via-switches, it is important to identify the faulty component in via-switches that

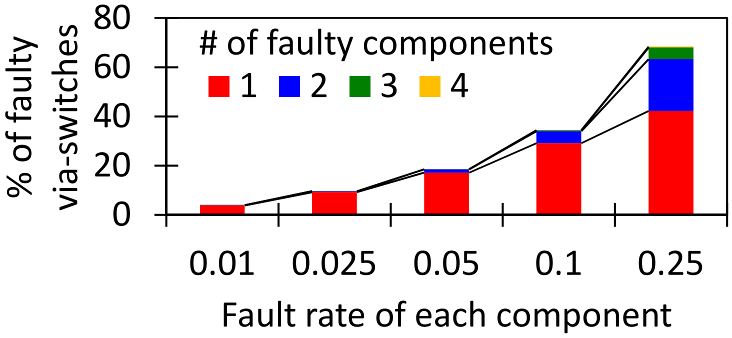


Figure 3.4: Percentage of faulty via-switches in 100x100 crossbar when fault rate of each component in a via-switch varies.

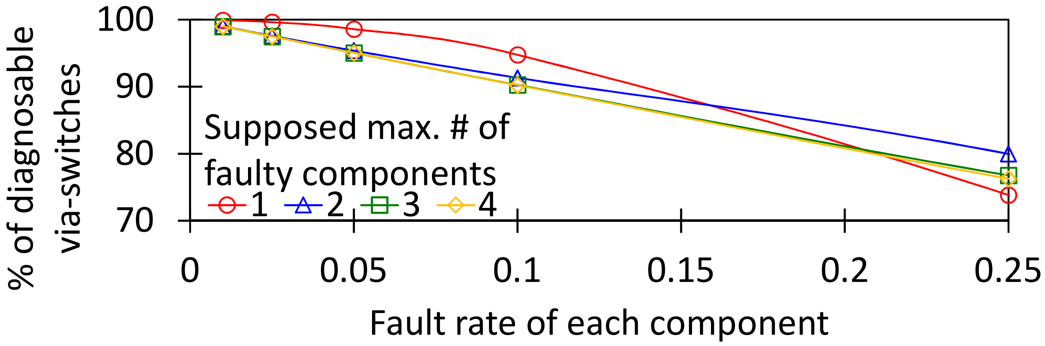


Figure 3.5: Percentage of diagnosable via-switches in 100x100 crossbar when the fault rate of each component in a via-switch and the supposed maximum number of faulty components are varied.

have only one faulty component.

Next, this paragraph evaluates how the percentage of diagnosable via-switches in a 100x100 crossbar changes when the supposed maximum number of faulty components in a via-switch varies. Figure 3.5 shows the evaluation result. In case of lower fault rates, the highest diagnosability can be achieved by supposing that there is up to one fault in a via-switch. For example, the percentage of diagnosable via-switches is 99% when the fault rate is 0.05. The result also indicates that it is better to suppose multiple faults in a via-switch and diagnose faulty components when the fault rate becomes high.

The high diagnosability of the proposed method is useful for yield analysis. In novel devices such as via-switches, it is important to investigate the cause of faults in detail for improving the yield. The proposed method can identify which component is faulty and also discriminate the fault type with high diagnosability. Therefore, the proposed method helps the manufacturer to increase the yield rate. Besides, the proposed method also contributes to the efficient use of programmable resources. This is because, even when a crossbar has faulty via-switches, it is possible to utilize the same crossbar normally by identifying faulty switches with the proposed method and avoiding the faulty part with sophisticated signal routings.

3.4 Conclusion

This chapter has confirmed that a general comparator can discriminate on/off-states of via-switches in the crossbar-based FPGA and clarified fault modes of a via-switch by SPICE simulation. Then, this chapter has proposed a fault diagnosis method that exploits three read modes and identifies faulty via-switch components according to the comparator response difference between normal and faulty cases. The proposed method achieves 100% fault detection. As for the diagnosability, the successful ratios of the fault diagnosis are 100% and 79% in cases that the number of faulty components in a via-switch is up to one and up to two, respectively. The number of reprogramming in the proposed fault testing method is very small, i.e., each via-switch is reprogrammed only once.

Chapter 4

Sneak Path Free Initial Programming in Via-Switch FPGA

This chapter proposes a sneak path free initial programming method [100], where the initial programming means the programming that is performed for the crossbars whose via-switches are all off-state. Partial reconfiguration for the crossbars where some via-switches are on-state will be discussed in the next chapter. Section 4.1 firstly reviews conventional countermeasures for the sneak path problem and states the advantages of the proposed method. After that, Section 4.2 investigates the occurrence conditions of the sneak path problem and identifies the crossbar programming status that causes the sneak path. Based on the discussion in Section 4.2, Section 4.3 proposes a sneak path avoidance method that provides a sneak path free programming sequence of via-switches in a crossbar. Then, Section 4.4 generalizes the proposed method and gives a proof that a sneak path free programming order necessarily exists for arbitrary on-off patterns in a crossbar as long as no loops exist. Section 4.5 gives a pseudo code and execution examples of the proposed method. Section 4.6 clarifies the advantages of the proposed method in terms of routing flexibility by simulation-based evaluation. Lastly, Section 4.7 summarizes this chapter.

4.1 Introduction

In recent decades, countermeasures for the sneak path problem in RRAM-based crossbars are widely studied [101–108]. These countermeasures can be categorized into three groups, namely device-level structure modifications [102–104], voltage bias schemes [105, 106], and multistage reading schemes [107, 108]. Refs. [102] and [103] propose one transistor-one resistor (1T1R) and one diode-one resistor (1D1R) structures for each crossbar intersection, respectively.

These structures mitigate the sneak path current since the transistor or diode act as a gating device, but the integration density of crossbars decreases due to the added gating devices. Ref. [104] adopts a complementary resistive switch (CRS), which is composed of two anti-serial memristors, as an intersection switch. This structure ensures that either memristor in a CRS is always off-state (high resistance) whenever the CRS retains logical 0 or 1, and hence it alleviates the sneak path problem. However, write and read operations for the CRS become complicated. The bias schemes have to apply VDD/3 or VDD/2 to unused wires for minimizing the sneak path current [105, 106]. These schemes need to drive all the wires, and therefore large switching power is consumed. Also, they require complicated controls and additional hardware overheads for write/read operations. Refs. [107, 108] propose a reading scheme that reads the target switch multiple times while changing conditions to eliminate the sneak path effect. The drawbacks of the multistage reading schemes are a large amount of reading time and large reading circuits.

The above summarizes the sneak path countermeasures reported in the literature, but all the countermeasures focus on the sneak path problem in the crossbar used as a memory. On the other hand, via-switch FPGA utilizes the crossbar as programmable interconnections and this work solves the sneak path problem in the write operation. To reduce the signal propagation delay in crossbars for FPGA, on-resistance of a via-switch is set to 400 Ω [80] whereas the on-resistance for memory purpose is in a range of a few kilo-ohms to hundreds kilo-ohms [109–112]. This smaller on-resistance in FPGA purpose makes the sneak path problem more severe since the countermeasures that insert high-resistance gating devices to signal lines increase the delay and diminish the integration density. Besides, voltage bias countermeasures are undesirable due to the large power consumption and hardware overheads. Therefore, the sneak path countermeasure that does not degrade the FPGA performance and does not need hardware modifications is required.

As a sneak path countermeasure for FPGA purpose, Ochi et al. have revealed that the sneak path problem can be avoided by imposing a programming constraint [80]. This constraint allows multiple on-state via-switches on the same signal line only in one direction. In other words, this constraint prohibits the configurations in which multiple on-state via-switches exist in both the same horizontal line and the same vertical line such as Figure 1.15. However, their countermeasure involves a clear disadvantage. The programming constraint prohibits some configurations of via-switch FPGA, and hence imposing the constraint leads to a decrease in the number of available configurations. Consequently, routing flexibility is limited. For example, practical applications often use routing patterns illustrated in Figure 4.1. The left pattern changes vertical routing tracks, and the right one realizes multiple fan-outs in the vertical direction. However,

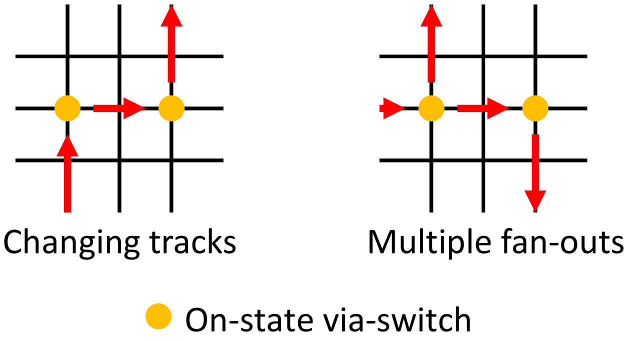


Figure 4.1: Routing patterns that are prohibited in conventional countermeasure. These patterns are often used in practical applications.

these patterns cannot be programmed in a crossbar when the constraint prohibits multiple on-state switches in the same horizontal line [80]. To achieve the same function, it is required to consume more interconnect resources due to detour routing.

This chapter, on the other hand, proposes a programming constraint-free countermeasure for the sneak path problem. The proposed method can program arbitrary practical configuration patterns including the patterns depicted in Figure 4.1. The advantages of the proposed method are as follows. The proposed method completely eliminates the programming status that causes the sneak path problem by arranging the programming order, and accepts all the practical configuration patterns. The computational complexity of the proposed algorithm is low, and there is no additional hardware overhead. Furthermore, the elimination of programming constraints can simplify the algorithm and data structure in the routing CAD software.

It should be noted that the proposed method can turn on multiple via-switches in the same line both vertically and horizontally. As discussed in Section 2.2.2, when there are multiple on-state via-switches in the same line of the crossbar, it is necessary to align an individual control line for each inter-CLB connection switch. Therefore, the via-switch FPGA that adopts the proposed programming method requires the individual control lines for both vertical and horizontal inter-CLB connection switches.

4.2 Occurrence Conditions of Sneak Path Problem

This section clarifies the programming status of a crossbar that leads to the sneak path problem for developing a more efficient countermeasure. As explained in Section 1.4.3, the atom switch at the intersection is turned on when a positive

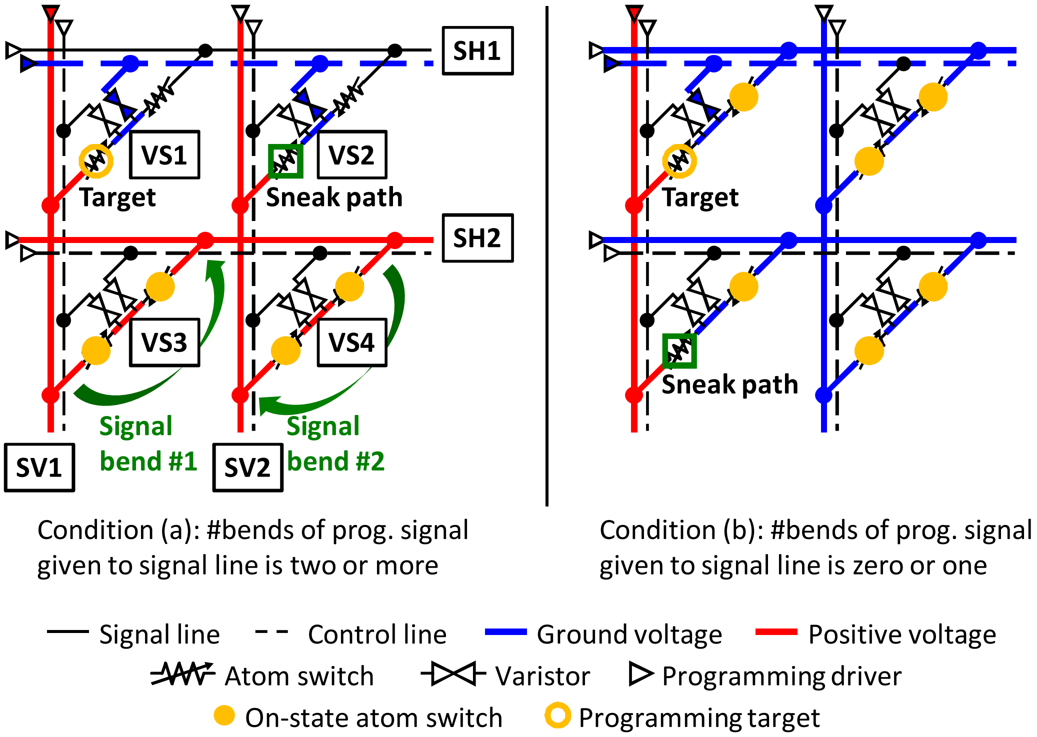


Figure 4.2: Two occurrence conditions of sneak path problem.

voltage is provided to the signal line and a ground voltage is provided to the control line. When the number of such intersections is two or more, the sneak path problem occurs. Focusing on the number of bends of the programming signal given to the signal line, the circuit status that causes the sneak path problem can be classified into two situations, namely conditions (a) and (b) as shown in Figure 4.2. In condition (a), the programming signal provided to the signal line bends twice or more, whereas the number of signal bends is one or zero in condition (b). The following discusses each condition in detail. It should be noted that this section only considers the programming operations to turn on the atom switch in the following because programming operations to turn on and off an atom switch are symmetrical operations and the same discussion can be done by swapping the voltage given to the signal line and control line.

In condition (a) of Figure 4.2, two vertical signal lines SV1 and SV2 are connected by multiple on-state via-switches VS3 and VS4 in the same line SH2. When a programming signal is given to one of the signal lines SV1 and SV2, the same signal is provided to the other signal line, which means we cannot distinguish SV1 and SV2 anymore in the programming. Therefore, when we try to turn on the lower atom switch of via-switch VS1 in the line SV1, the atom switch in the same position of signal line SV2, i.e., lower atom switch of via-switch VS2 is programmed simultaneously. In summary, the sneak path problem arises when programming an atom switch in already indistinguishable vertical lines or indistinguishable horizontal signal lines.

From the opposite point of view, we could avoid the sneak path problem if we would program such an atom switch before multiple vertical/horizontal signal lines become indistinguishable. For example, in condition (a) in Figure 4.2, we need to turn on the lower atom switch of via-switch VS1 before programming VS3 and VS4. It should be noted that, for programming a via-switch, we must turn on two atom switches, which are the lower atom switch connected to the vertical signal line and the upper atom switch connected to the horizontal signal line. The vertical signal line is used when programming the lower atom switch (e.g., step 2 in Figure 1.14), and hence we need to pay attention to multiple on-state via-switches in the same horizontal signal line that connect multiple vertical signal lines. On the other hand, we should care about multiple on-state via-switches in the same vertical signal line when programming the upper atom switch.

Let us move to condition (b) in Figure 4.2, where the number of bends of programming signal given to the signal line is one or zero. In this case, the programming signal that is provided to a control line is detoured and the sneak path problem arises. On the other hand, such a condition is satisfied only when a loop is intentionally programmed in a crossbar. For example, in Figure 4.2-(b), all the four via-switches are intended to be turned on, otherwise the top two atom switches of VS1 and VS3 never be on. This condition arises only when programming the last two atom switches that compose a loop, and the ground voltage applied to the control line is propagated to the non-target switch because of the loop structure.

From the above discussion, the sneak path problem cannot be avoided in the configurations that include a loop. Fortunately, such configurations with a loop are not used in practical applications since the looped signal routing increases the wire capacitance and degrades delay and power compared to non-loop routing. Therefore, there is no need to take care of condition (b). Consequently, the only thing to consider for sneak path avoidance is only condition (a).

4.3 Proposed Sneak Path Free Initial Programming

Based on the discussion in the previous section, this section proposes a sneak path avoidance method that provides a sneak path free initial programming sequence of via-switches in a crossbar. Here, the initial programming means the programming that is performed for the crossbars whose via-switches are all off-state. The proposed method gives a sneak path free programming order, where the target configurations are non-looped configurations for signal routing. Section 4.3.1 explains the overview of the proposed sneak path avoidance method followed by its details with examples in Section 4.3.2. Some key properties necessary for proving that a sneak path free programming order necessarily exists are in *italic*. With those properties, the next section generalizes the proposed method and gives a proof that a sneak path free programming order necessarily exists for arbitrary non-looped configurations.

4.3.1 Overview of Proposed Method

Table 4.1 shows the proposed method consisting of two steps: STEP 1 turning on all the upper atom switches of interest and STEP 2 turning on all the lower atom switches of interest. Each step is explained in the following.

In STEP 1, all the upper atom switches of the via-switches to be turned on in a given target configuration are programmed. The via-switch connects the vertical and horizontal signal lines only when both the upper and lower atom switches composing a via-switch are on-state. Therefore, in STEP 1, any signal lines are not connected to each other and the programming signal never detours. Hence, no sneak path problem arises in this step. Then, *the arbitrary programming order works fine in STEP 1.*

STEP 2 turns on all the lower atom switches to be programmed. In STEP 2, a vertical line and a horizontal line are connected by a via-switch each time a lower atom switch is turned on since the corresponding upper atom switch is already turned on in STEP 1. Therefore, it is necessary to determine the programming order of the lower switches paying attention to the occurrence condition of the sneak path problem, which is discussed in the previous section. Please remind that the driver provides a programming signal to a vertical signal line when programming a lower atom switch, and hence this step considers only multiple on-state via-switches in the same horizontal signal line since they make multiple vertical lines indistinguishable. Multiple on-state via-switches in the same vertical signal line do not matter. STEP 2 categorizes those multiple on-state switches in the same horizontal line as connector switches (CSs) and other switches as non-connector switches (NCSs). STEP 2a and 2b turn on NCSs and CSs, respectively.

Table 4.1 demonstrates that all the target switches are programmed by the proposed STEPs. The fifth column of Table 4.1 shows lemma numbers that correspond to each STEP. Proving those lemmas in Section 4.4, this work ensures that no sneak path problems arise in the proposed method.

It should be noted that the swapped sequence of STEP 2 followed by STEP 1, i.e., programming all the upper atom switches after turning on all the lower atom switches can also avoid the sneak path problem since the crossbar has a

Table 4.1: Summary of proposed initial programming method.

Prog. STEP	Upper atom switches	Lower atom switches		Relevant lemma
		NCSs	CSs	
Start				
STEP 1	Turned on			Lem. 1
STEP 2a	Programmed	Turned on		Lem. 2
STEP 2b	Programmed	Programmed	Turned on	Lem. 3-5
End	Programmed	Programmed	Programmed	

NCSs: non-connector switches, CSs: connector switches

symmetrical structure. In this case, it needs to determine the programming order of the upper switches.

4.3.2 Programming Order Determination with Connection Tree

This subsection details the proposed method explaining how to derive a sneak path free programming order of via-switches in a crossbar. The following uses a configuration of a 5x5 crossbar shown in Figure 4.3 as an example.

As mentioned in Section 4.3.1, the only thing to do is to determine the programming order in STEP 2 since STEP 1 does not cause sneak path problems with an arbitrary programming order. STEP 2 has to pay attention to multiple on-state via-switches in the same horizontal signal line that connect multiple vertical signal lines and make them indistinguishable. Therefore, this work introduces two categories of the via-switch, namely connector switches and non-connector switches, which are defined in the previous subsection. For example, via-switches B, C, D, E, F, G, and H are connector switches and via-switches A and I are non-connector switches in Figure 4.3. A pair of connector switches connects two vertical signal lines, e.g., via-switches E and F connect vertical signal lines SV1 and SV2 in Figure 4.3. The non-connector switch, on the other hand, does not connect any vertical signal lines.

Please remind that the sneak path problem occurs when turning on the lower atom switch included in the already connected vertical signal lines as explained in Section 4.2. Therefore in STEP 2a, *all the non-connector switches should be programmed before the connector switches* so that this step can avoid the sneak path problem in programming the non-connector switches since the connector switches which may connect vertical lines are still off-state. In this case, *the programming order of the non-connector switches is arbitrary.*

Next, STEP 2b determines the programming order of connector switches. In

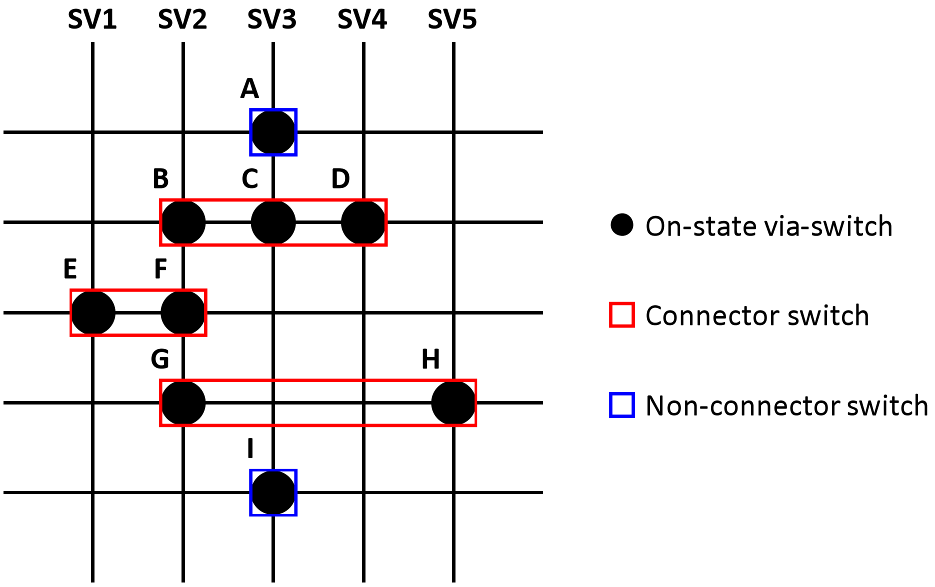


Figure 4.3: Example of non-looped configuration and definition of connector/non-connector switches.

this step, the programming order is not arbitrary since the sneak path problem may arise depending on the programming order. For example in Figure 4.3, when the drivers are turning on the connector switches B or G after programming the connector switches of E and F, the sneak path problem occurs since switches E and F have connected vertical lines SV1 and SV2.

To determine the programming order, this work constructs a connection tree that represents the connection status of vertical signal lines in a crossbar. Figure 4.4 exemplifies a connection tree for the connector switches in Figure 4.3, where each node corresponds to a vertical signal line. The root node can be arbitrarily selected. Vertical line SV3 is selected as the root node in Figure 4.4. Here, when another node is chosen as the root node instead of SV3, the structure of the connection tree changes. On the other hand, regardless of the tree structure, the proposed method can avoid the sneak path, and the number of switches to be turned on does not change since all the connector switches must be turned on. When two vertical signal lines are supposed to be connected in the configuration of interest, an edge is given between the two nodes corresponding to these two vertical signal lines. Two black dots located at both ends of an edge represent connector switches, and when both the two connector switches are turned on, this work supposes the edge is activated and two vertical lines are connected. From the definition, *any non-looped configurations can be necessarily expressed by a tree structure*, which means loops are not included in the graph.

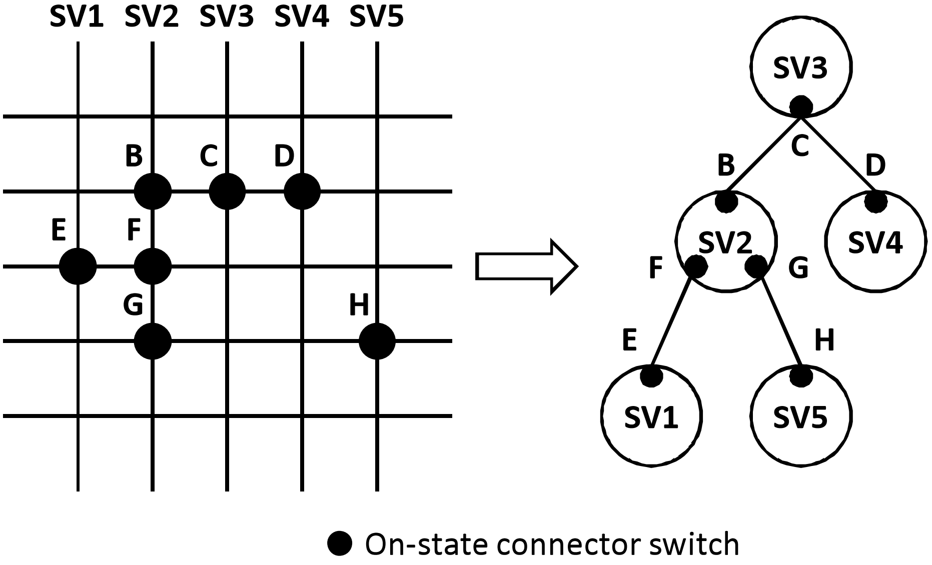
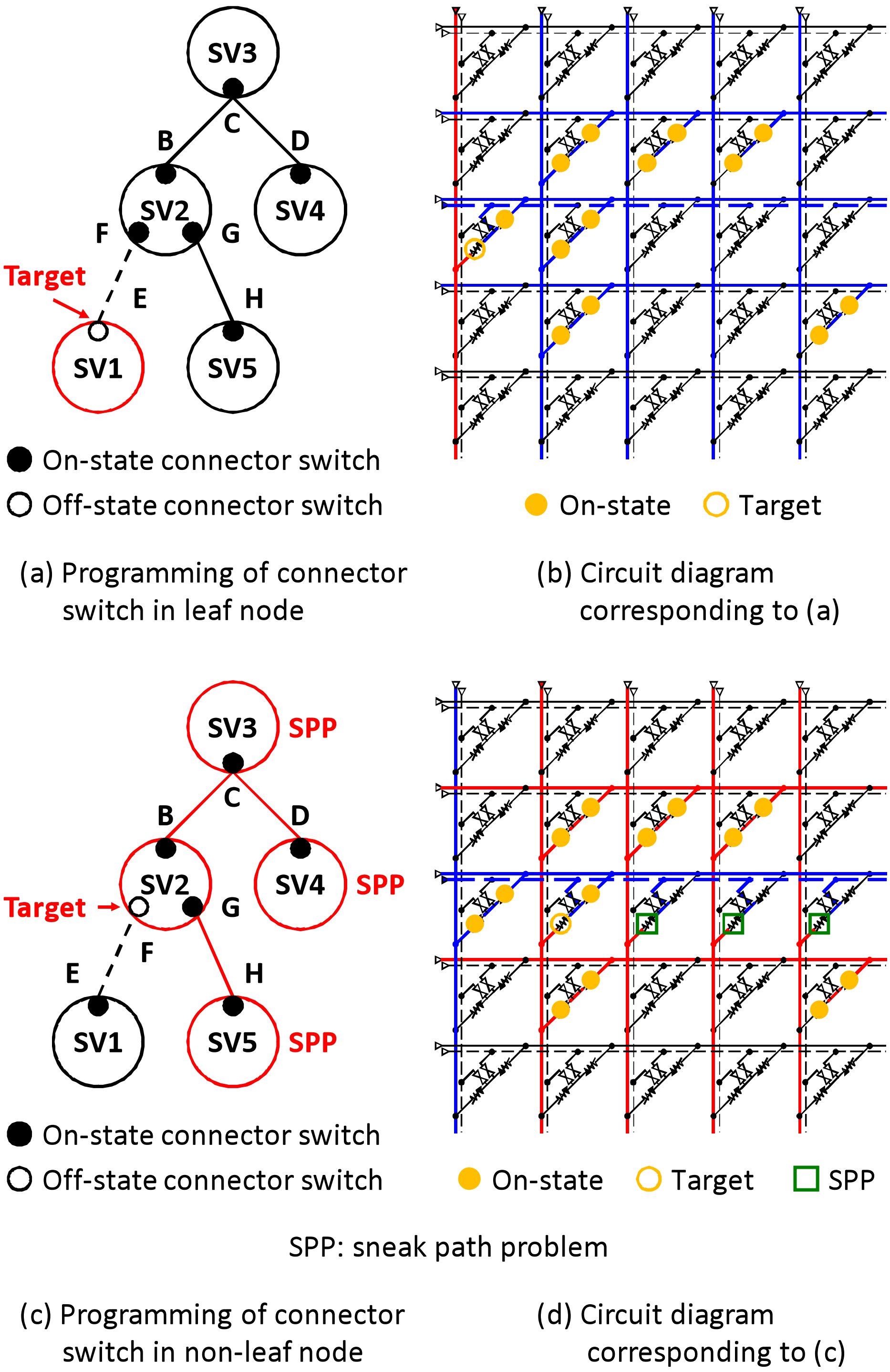


Figure 4.4: Example of connection tree for connector switches in Figure 4.3.

The connection tree tells us which connector switch can be programmed such that *the connector switch that can be turned on at the end of programming is indicated as the leaf node of the connection tree.* Let us explain what happens when programming the connector switch in the leaf node and non-leaf node of the connection tree at the last programming step, where the last programming step means that only one switch remains off and the others are already turned on in the target configuration.

In Figure 4.5-(a), the connector switch in the leaf node of SV1 is under programming, and the other connector switches are already on-state. In this case, node SV1 and node SV2 are not connected yet, and hence the programming signal never propagates to any other vertical signal lines. Consequently, the target connector switch can be turned on without the sneak path problem. Figure 4.5-(b) can also confirm that there is no sneak path problem when programming the connector switch in the leaf node, where this figure is the circuit diagram corresponding to Figure 4.5-(a). Next, let us turn on the connector switch in non-leaf node SV2 in Figure 4.5-(c) at the last programming step. In this case, the programming signal reaches the other vertical signal lines through the connector switches that are already on-state, and consequently the sneak path problem arises. The circuit diagram of Figure 4.5-(d) also indicates that atom switches placed at the same vertical position as the target on the connected indistinguishable vertical signal lines are unintentionally programmed.

Then, this work proposes to *recursively search for a connector switch that can be turned on at the final programming step* for obtaining a sneak path free



(a) Programming of connector switch in leaf node

(b) Circuit diagram corresponding to (a)

(c) Programming of connector switch in non-leaf node

(d) Circuit diagram corresponding to (c)

Figure 4.5: Programming of connector switch in leaf/non-leaf node at the last programming step.

programming order of connector switches. Figure 4.6 illustrates the recursive process. Here, there are two types of connector switches in each node, namely the connector switch connecting with the parent node (e.g., switch B in node SV2) and the connector switch connecting with the child node (e.g., switches F and G in node SV2). *In each node, all the switches connecting with the child node must be turned on before the switch connecting with the parent node.* Otherwise, the sneak path problem arises when programming the switch connecting with the child node since the programming signal is propagated to the parent node through the on-state switch to the parent node as pointed out in Figure 4.5-(c).

Let us roll back the recursive programming step one by one with Figure 4.6. As discussed, we can program only the connector switch in the leaf node at the final programming step. Then, switch H in SV5 is selected as the last switch to be programmed and node SV5, and the edge between SV2 and SV5 are deleted. This modified graph is again analyzed to find the next last switch to be programmed. In this case, switch E is selected. After SV5 and SV1 are removed from the graph, SV2 has no child nodes, and hence switch B in leaf node SV2 connecting with parent node SV3 can be programmed. In this way, one recursive process chooses one leaf node, identifies the switch in the leaf node connecting with the parent node as the last switch to be programmed in the current graph, and remove the leaf node and its edge to the parent node. Eventually, *all the edges are removed from the connection tree, and the recursive process finishes.* At this time, all the vertical lines are not connected to any other vertical lines anymore, and hence we can distinguish all the vertical lines. It should be noted that there remain some on-state connector switches, for example, switches C, F, and G in Figure 4.6. *These switches can be programmed in an arbitrary order as long as they are programmed before the switches selected in the recursive processes.* One of the finally obtained programming orders is C, F, G, B, D, E, and H.

Depending on the target configuration, multiple connection trees may be constructed for a non-looped configuration. Each tree has no connection to other trees, and hence the programming signal never propagates to other trees when programming the switch in the tree of interest. Consequently, the proposed method can handle each connection tree independently.

Thus far, this section discussed the programming order for turning-on operations. Programming operations to turn on and off an atom switch are symmetric except that applied voltages to the signal line and control line are reversed. Therefore, the proposed method can turn off all the switches without the sneak path problem in the reverse order.

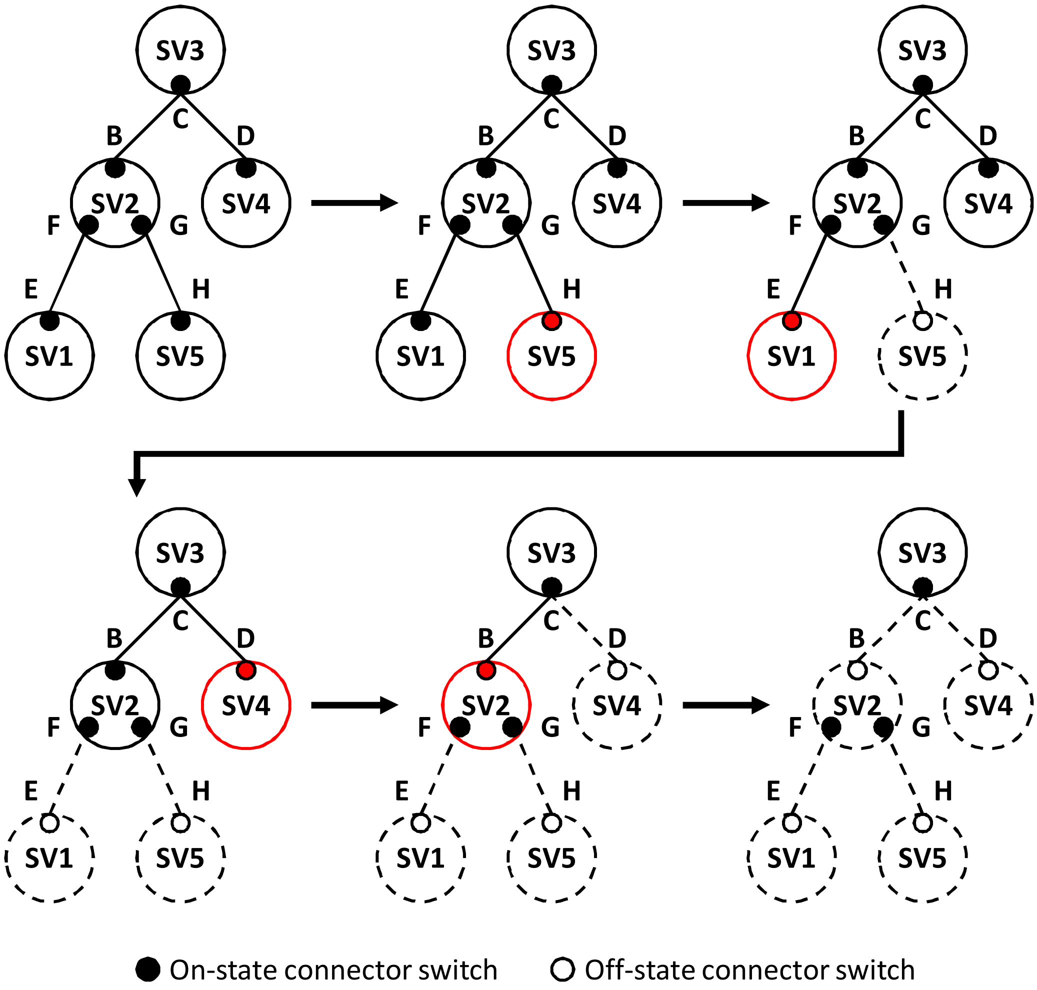


Figure 4.6: Recursively searching switch which can be programmed lastly for each shrinking graph.

4.4 Proof of Existence of Sneak Path Free Programming Order

This section formally proves that a sneak path free programming order always exists for arbitrary non-looped configurations. As indicated in Table 4.1, the proof consists of five lemmas, and this section proves them in the following, where Lemma 1, Lemma 2, and Lemma 3-5 demonstrate that there is no sneak path problem in the programming of upper atom switches, non-connector switches, and connector switches, respectively.

***Lemma* 1.** *All the upper atom switches can be programmed in an arbitrary order* *without the sneak path problem.*

Proof. Only when both the upper and lower atom switches composing a via-switch are on-state, the via-switch connects the vertical and horizontal signal lines. Hence, any signal lines are not connected to each other in this programming step because all the lower atom switches are still off-state. Therefore, the programming signal never detours and no sneak path problem occurs. From the same reason, the programming order of this step is arbitrary. □

***Lemma* 2.** *There is no sneak path problem in the programming of all the non-connector switches, and arbitrary programming order works fine in this step.*

Proof. In programming lower atom switches, the sneak path problem occurs when the drivers turn on an atom switch in already indistinguishable vertical signal lines as mentioned in Section 4.2. The indistinguishable signal lines originate from on-state connector switches that connect multiple vertical lines. By programming all the non-connector switches before connector switches, we can distinguish all the vertical lines in programming non-connector switches since all the connector switches are still off-state in this step. Therefore, no sneak path problem arises and arbitrary programming order is acceptable in this programming step. □

***Lemma* 3.** *Given a non-looped configuration, it can be expressed by a connection tree or multiple connection trees.*

Proof. When we treat each vertical signal line as a node and draw an edge between corresponding nodes if connector switches exist, these nodes and edges compose a graph or multiple graphs that represent the connection status of vertical lines. Each graph does not contain closed loops unless the target configuration has loops. When a node is selected as the root node, the graph is expressed by a tree structure, which is called the connection tree defined in Section 4.3.2. □

***Lemma* 4.** *At the last programming step for a connection tree, only a switch in a leaf node connecting to its parent node can be programmed without the sneak path problem.*

Proof. When programming a switch in a leaf node at the final programming step, the target switch is still off-state and the connection between the leaf node and its parent node is not established yet. Hence, the programming signal given to the leaf node never propagates to any other nodes, and consequently there is no sneak path problem in this programming. On the other hand, a non-leaf node has at least two connections, i.e., connections to its parent node and child node. Therefore, the target non-leaf node has at least one connection to the other node at the last programming step since all the switches except the target switch are already on-state. Consequently, programming a switch in a non-leaf node at the end always causes the sneak path problem by propagating the programming signal to other nodes through the connection to the parent or child node. □

***Lemma* 5.** *Recursively searching a switch that can be programmed at the last programming step always finds the sneak path free programming order.*

Proof. By recursively searching a switch that can be turned on at the final programming step in the current graph and removing the leaf node and its edge to the parent node from the graph, all the nodes except the root node must be eventually eliminated and the recursive search necessarily finishes since the tree must have at least one leaf node when the number of nodes is two or larger. The remaining switches can be programmed in an arbitrary order before the switches chosen in the recursive search since each node has no connection to other nodes at this moment, i.e., all the vertical signal lines are distinguishable. □

4.5 Pseudo Code and Execution Example

Algorithm 1 summarizes the overall determination procedure of a programming order. This algorithm determines a sneak path free programming order of non-connector and connector switches, and stores it to queue O_{prog}. Line 1 defines a set $\mathbb{S}$ of switches to be turned on. Lines 2-4 search non-connector switches by checking the number of on-state switches in each horizontal signal line. Specifically, line 2 creates a set $\mathbb{H}_j$ of on-state switches in j-th horizontal signal line, line 3 enumerates non-connector switches in j-th horizontal line where the number of elements of $\mathbb{H}_j$ is one, and line 4 enqueues all the non-connector switches to O_{prog}. Then, line 5 removes all the non-connector switches from the set $\mathbb{S}$, and subsequent lines 6-8 determine the programming order of connector switches. Line 6 selects i-th vertical signal line, in which the connector switch to be turned on exists, as the root node of a connection tree, and the programming order of connector switches in this connection tree is determined by the function SEARCH in line 7.

SEARCH is a recursive function and traverses a connection tree from the root node to leaf nodes. Please remind that all the switches connected to child nodes need to be programmed before programming any switch connected to its parent node. SEARCH classifies the switches connected to child nodes of the parent node i (line 10) and the switches connected to the parent node i (line 13). The former is enqueued to O_{prog} in line 14 since switches connected to the child have to turn on before the switches connected to the parent. On the other hand, the latter is enqueued in O_{tmp} in line 15 until all the switches connected to the child node are enqueued to O_{prog} by the recursive function SEARCH. This function is recursively

Algorithm 1 Programming order determination of non-connector and connector switches.

```
 1: $\mathbb{S} = \{S_{i,j} \mid S_{i,j}$ is on-state, $0 \le i < W,\ 0 \le j < H\}$
 2: $\mathbb{H}_j = \{S_{i,j} \mid S_{i,j} \in \mathbb{S}\}$
 3: $\mathbb{S}_{\text{non-con}} = \{S_{i,j} \mid |\mathbb{H}_j| = 1,\ S_{i,j} \in \mathbb{S}\}$
 4: Enqueue($\mathbb{S}_{\text{non-con}}$) to $O_{\text{prog}}$
 5: $\mathbb{S} = \mathbb{S} - \mathbb{S}_{\text{non-con}}$
 6: for $i \in \{i \mid \exists S_{i,j} \in \mathbb{S}\}$ do
 7:     Search($i$, $\mathbb{S}$, $O_{\text{prog}}$, $O_{\text{tmp}}$)
 8: Enqueue($O_{\text{tmp}}$) to $O_{\text{prog}}$

 9: function Search($i$, $\mathbb{S}$, $O_{\text{prog}}$, $O_{\text{tmp}}$)
10:     $\mathbb{S}_{\text{child}} = \{S_{i,j} \mid S_{i,j} \in \mathbb{S}\}$
11:     if $\mathbb{S}_{\text{child}} = \emptyset$ then
12:         return
13:     $\mathbb{S}_{\text{parent}} = \{S_{k,j} \mid k \ne i,\ \exists S_{i,j} \in \mathbb{S}_{\text{child}}\}$
14:     Enqueue($\mathbb{S}_{\text{child}}$) to $O_{\text{prog}}$
15:     Enqueue($\mathbb{S}_{\text{parent}}$) to $O_{\text{tmp}}$
16:     $\mathbb{S} = \mathbb{S} - (\mathbb{S}_{\text{child}} \cup \mathbb{S}_{\text{parent}})$
17:     for $k \in \{k \mid \exists S_{k,j} \in \mathbb{S}_{\text{parent}}\}$ do
18:         Search($k$, $\mathbb{S}$, $O_{\text{prog}}$, $O_{\text{tmp}}$)

    $W$: crossbar width, $H$: crossbar height
```

executed for all the child nodes of the node of interest (lines 17-18), and returns when the node of interest is a leaf node of the connection tree, i.e., no child nodes exist (lines 11-12). In the case that there exist multiple connection trees, $\mathbb{S}$ is not empty after line 7 completion. In this case, "for statement" in line 6 re-executes Search with the updated $\mathbb{S}$ until $\mathbb{S}$ becomes empty. Finally, all the switches connected to the parent node in all nodes are enqueued to O_{prog} in line 8.

Let us explain an example when the proposed algorithm is applied to the configurations as shown in Figure 4.3. Lines 2-4 find non-connector switches A and I, and enqueue them to O_{prog}. Line 7 determines a programming order of the remaining connector switches B-H. Assuming the vertical line SV3 is selected as a root node i, at the first execution of the function Search, the set $\mathbb{S}_{\text{child}}$ contains the switch C connected to the child node as shown in Figure 4.4, and the set $\mathbb{S}_{\text{parent}}$ contains switches B and D connected to the parent node (root node). Line 14 enqueues the switch C to O_{prog} and line 15 stores switches B and D to O_{tmp}. After that, function Search is executed again for vertical lines SV2 and SV4 where switches B and D exist. When Search is executed for line SV2, set $\mathbb{S}_{\text{child}}$ contains switches F and G, and set $\mathbb{S}_{\text{parent}}$ contains switches E and H. On the other hand, when Search is executed for line SV4, set $\mathbb{S}_{\text{child}}$ is empty and Search returns. Eventually, the proposed method successfully obtains a sneak path free programming order and it is A, I, C, F, G, B, D, E, and H.

4.6 Evaluation Results

This subsection discusses an increase in the number of available configurations thanks to the proposed sneak path-aware programming method. The conventional countermeasure for the sneak path problem, which is explained in Section 4.1, imposes a programming constraint that prohibits a class of configurations of the via-switch FPGA. Therefore, the conventional countermeasure reduces the number of usable configurations and consequently diminishes the routing flexibility. On the other hand, the proposed method can give a sneak path free programming order for any non-looped configurations. As discussed in Section 4.2, the sneak path problem is unavoidable in looped configurations, but those configurations are practically meaningless for signal routing.

Figure 4.7 compares the number of programmable configurations with the conventional countermeasure and the proposed method in 2x2, 3x3, 4x4, and 5x5 crossbars, where such small crossbars are used to enumerate all the non-looped configurations. In this evaluation, the conventional countermeasure prohibits configurations in which multiple on-state via-switches exist in the same horizontal line [80]. We can see that the proposed method increases the number of usable configurations compared to the conventional countermeasure. Even in the small 5x5 crossbar, the number of available configurations increases by over two orders of magnitude. This evaluation also confirms that the number of usable configurations in the proposed method is equal to the number of all the non-looped configurations. As proved in Section 4.4, the proposed method can find a sneak path free programming order for arbitrary non-looped configurations in an arbitrarily-sized crossbar. Another observation is that the increasing ratio of the number of usable configurations becomes larger as the crossbar size increments, which suggests a significant increase in the number of configurations in practically-sized crossbars.

Motivated by this, this work assesses the number of available configurations in a practically-sized crossbar. Here, the total number of configurations exponentially increases as the crossbar size n becomes larger, like 2^n, and the comprehensive simulation of larger crossbars is infeasible. Instead, this evaluation generated random configurations for a large crossbar in Monte Carlo manner and compared the number of programmable configurations with conventional and

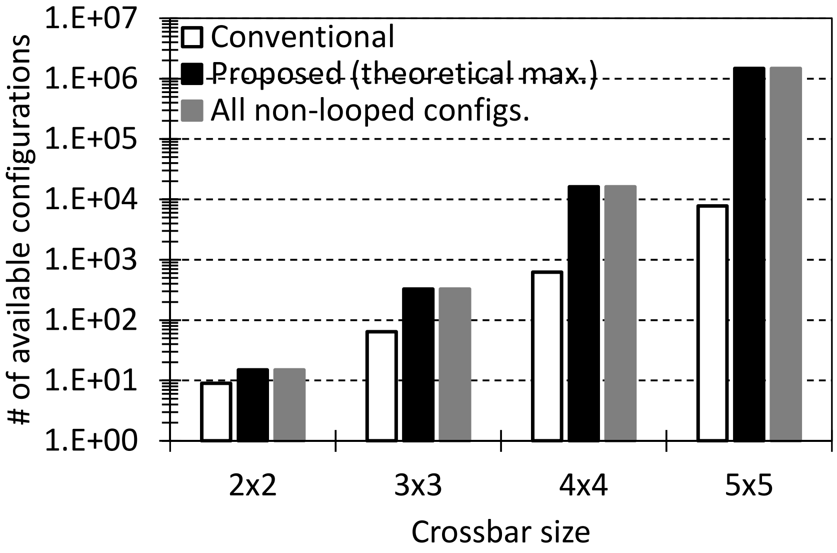


Figure 4.7: Number of available configurations with conventional countermeasure and proposed method in small crossbars.

Table 4.2: Number of usable configurations among 10,000 random configurations in a practically-sized 100x100 crossbar.

% of on-state	# of samples	# of available configs.	
		Conventional	Proposed
0.1	10,000	6,347	10,000
0.2	10,000	1,324	10,000
0.3	10,000	91	10,000
0.4	10,000	1	10,000
0.5	10,000	0	10,000

proposed methods. The locations of on-state via-switches were determined by uniformly distributed random numbers. In this evaluation, the crossbar size was set to 100x100 and the number of trials was 10,000. This evaluation also varied the percentage of on-state via-switches from 0.1% to 0.5%. Table 4.2 shows the evaluation results. We can see that the proposed method achieves a significant increase in the number of usable configurations. When 0.5% of via-switches are on-state, the number of programmable configurations increases by over four orders of magnitude.

4.7 Conclusion

This chapter identified the programming status of the via-switch crossbars based FPGA that cause the sneak path problem, and clarified that the sneak path problem could not be avoided in looped configurations. On the other hand, this chapter has proved that a via-switch programming order which can avoid the sneak path problem always exists for all the non-looped configurations, and this chapter proposed a sneak path avoidance method that gave sneak path free programming order of via-switches in a crossbar. The devised algorithm finds the programming order by representing the connection status of signal wires in a crossbar as a tree structure. Simulation-based evaluation results confirmed that the proposed method increases the number of available configurations by over four orders of magnitude in a practically-sized crossbar, and improves the routing flexibility of via-switch FPGA. The proposed method successfully solves the sneak path problem in any practical configurations of via-switch FPGA.

Chapter 5

Minimization of Programming Steps in Partial Reconfiguration of Via-Switch FPGA

This chapter proposes a partial reconfiguration method that minimizes the number of switch programming steps while avoiding sneak path problem [113]. First, Section 5.1 explains the proposed method that partially turns off and on via-switches in an already programmed crossbar. Then, Section 5.2 provides a proof that there is no sneak path problem in the proposed partial reconfiguration method. After that, Section 5.3 proposes a minimization method of programming steps, followed by a pseudo code in Section 5.4. The minimization effect is quantitatively evaluated by simulations in Section 5.5. Section 5.6 discusses an application example of the proposed method for the chip testing. Finally, Section 5.7 summarizes this chapter.

5.1 Proposed Partial Reprogramming Method

This chapter discusses partial reprogramming for changing crossbar configurations. Needless to say, we can change the crossbar configuration by turning off all the on-state switches in the previous configuration and then writing the next configuration to the crossbar with the proposed method explained in Chapter 4. However, this approach involves unnecessary reprogramming when both the previous and next configurations partially share the same on-state via-switches. The unnecessary reprogramming of via-switches directly leads to an increase in the reconfiguration time, and also shortens the lifetime of via-switches since the maximum number of reprogramming is limited. For solving this problem, this chapter proposes a partial reconfiguration method that minimizes the number

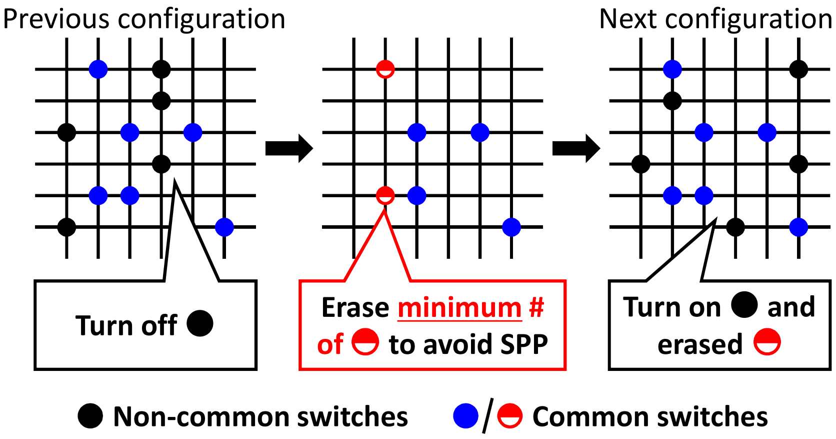


Figure 5.1: Concept of partial reprogramming.

of programmed switches while avoiding the sneak path problem. This method contributes to extending the lifetime of via-switch FPGA and speeding up the reconfiguration.

Table 5.1 summarizes the proposed partial programming steps and indicates switches to be turned on/off in each step and their abbreviations in parentheses. Figure 5.1 illustrates the basic strategy that programs non-common switches and minimizes the number of common switches to be programmed for avoiding the sneak path problem, where the proposed method erases only either upper or lower atom switch in such common switches depicted with red semicircle symbols in Figure 5.1. The following subsections explain how to partially turn off and on via-switches in an already programmed crossbar.

5.1.1 Partial Erasing

STEP 1 of partial erasing turns off both upper and lower atom switches of non-common parts, which are S_{PU} and S_{PL}, respectively, in Table 5.1. Similar to the case of turning on an atom switch, the sneak path problem arises if we turn off an atom switch in the already indistinguishable lines. Figure 5.2 illustrates such a case. This figure indicates that the harmful SPP problem is not occurring since the detoured routing provides the turning-off voltage to the off-state switch. Fortunately, in non-looped configurations, this work can ensure that switches under unintentional programming by the sneak path problem are always off-state. If the switch under unintentional programming is on-state, a loop is composed by these switches beforehand. This work uses only non-looped configurations, and therefore we can turn off any switch regardless of the programming order. Note that the programming order obtained in the previous section can erase all the

Table 5.1: Summary of proposed partial reconfiguration method.

Prog. STEP		Non-common in prev. config.		Common in both config.				Non-common in next config.			Relevant lemma
		Upper ASs	Lower ASs	Upper ASs	Lower ASs			Upper ASs	Lower ASs		
		(S_{PU})	(S_{PL})	(S_{CU})	Horizontal CSs (S_{CH})	Vertical CSs (S_{CV})	Others (S_{CO})	(S_{NU})	Horizontal CSs (S_{NC})	Horizontal NCSs (S_{NN})	
Start		Proged.	Proged.	Proged.	Proged.	Proged.	Proged.				
Partial erase (STEP 1)		Turn off	Turn off	Proged.	Proged.	Proged.	Proged.				Lem. 6
Partial write	STEP 2a			Proged.	Proged.	Turn off	Proged.				Lem. 7
	STEP 2b			Proged.	Proged.		Proged.	Turn on			Lem. 7
	STEP 3a			Proged.	Proged.		Proged.	Proged.			Lem. 8
	STEP 3b			Proged.	Turn off		Proged.	Proged.			Lem. 8
	STEP 3c			Proged.		Turn on	Proged.	Proged.		Turn on	Lem. 8
	STEP 3d			Proged.	Turn on	Proged.	Proged.	Proged.	Turn on	Proged.	Lem. 8
End				Proged.	Proged.	Proged.	Proged.	Proged.	Proged.	Proged.	

ASs: atom switches, CSs: connector switches, NCSs: non-connector switches, Proged.: programmed

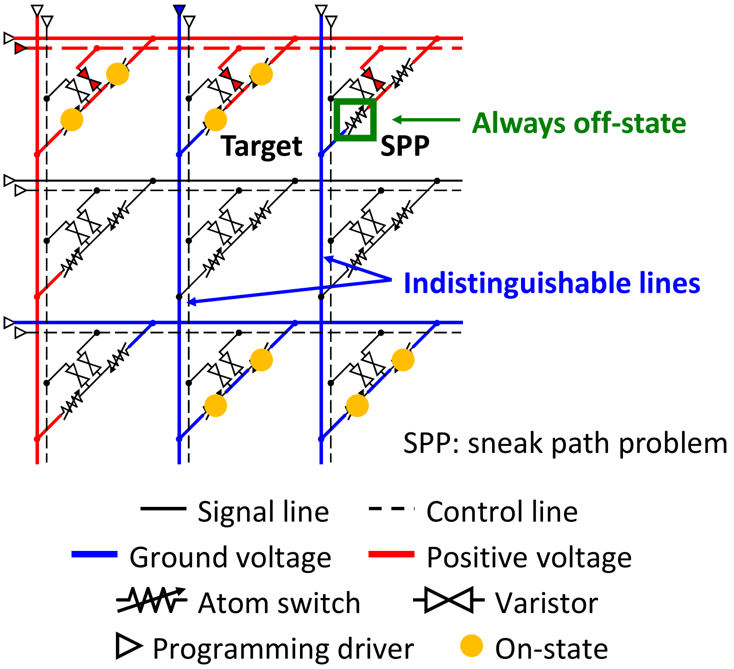


Figure 5.2: Sneak path problem in erasing process.

switches without the harmful or non-harmful sneak path problems.

5.1.2 Partial Writing

The partial writing process consists of two steps, where STEP 2 and 3 turn on all the upper (S_{NU}) and lower (S_{NC} and S_{NN}) atom switches, respectively, of the via-switches to be newly turned on in the target configuration as shown in Table 5.1. For avoiding the sneak path problem at each STEP, the proposed method turns off a part of common and already on-state switches S_{CH} and S_{CV} before programming S_{NU}, S_{NC}, and S_{NN} so that the programming signal does not detour to any non-target switch. The following explains the details of each step.

STEP 2 is to turn on S_{NU} while avoiding the sneak path problem. As discussed in Section 4.2, in programming upper atom switches, the sneak path problem arises when we turn on an atom switch in already indistinguishable horizontal lines. Therefore, it is necessary to care about S_{CV}, which represents the connector switches in the same vertical signal line. When S_{CV} exists in the same horizontal line of the target switch S_{NU}, the proposed method erases the lower atom switches S_{CV} in the same line of S_{NU} at STEP 2a. Such erasing is always possible as explained in Section 5.1.1. The erased S_{CV} will be reprogrammed later at STEP 3. After STEP 2a, STEP 2b can turn on S_{NU} without the sneak path problem since the programming signal is never propagated to any other horizontal lines. Thus, STEP 2 ensures that all the upper atom switches of via-switches to be turned on are on-state at the beginning of STEP 3.

STEP 3 turns on S_{NC} and S_{NN}, which are lower atom switches to be newly turned on in the target configurations. In addition, S_{CV} erased at STEP 2a is also turned on. This step utilizes the connection tree that is constructed based on the final configuration of partial reprogramming. The basic idea to avoid the sneak path problem in STEP 3 is as follows. The first step turns off a part of connector switches and disconnects the node containing a target switch from the connection tree so that the programming signal is never propagated to other nodes. Then, the next step turns on the target switch in the isolated node, followed by turning on the erased connector switches to restore the connection between the isolated node and the connection tree.

STEP 3 is decomposed into the following four steps, which are listed in Table 5.1. Let us explain each step with an example shown in Figure 5.3. As explained in Section 4.3.2, the root node can be arbitrarily selected. Using this property, STEP 3a changes the root node of the connection tree for minimizing the number of switches to be programmed in STEPs 3b-3d, where the detail will be discussed in Section 5.3. In Figure 5.3, node A is selected as the root node. Next, STEP 3b is to turn off S_{CH}, which represents horizontal connector switches connecting to the parent node in the target and its descendant nodes. All the connections below the target node are disconnected from the connection tree by this step. The sneak path problem does not arise in STEP 3b since only erasing is performed. In Figure 5.3, STEP 3b turns off the connector switches in nodes B, C, and E connecting to their parent nodes. Then, STEP 3c turns on switches S_{NC} and S_{CV}. There is no connection between the target node and the others thanks to STEP 3b, and therefore no sneak path problem arises in this step. Finally, STEP 3d reconnects the target and its descendant nodes to the connection tree. By turning on the connector switches S_{CH} and S_{NC} in order from the shallow level to the deep level of the connection tree, each phase always turns on a connector switch in a leaf node, and hence there is no sneak path problem. As proved in Section 4.4, a connector switch in a leaf node of the connection tree can be turned on without the sneak path problem. In Figure 5.3, STEP 3d turns on connector switches in order of nodes B, C, and E.

It should be noted that the swapped sequence of STEP 3 followed by STEP 2, i.e., programming upper atom switches after turning on lower atom switches is also acceptable since the crossbar has a symmetrical structure. In this case, it is necessary to determine the programming order of the upper switches. Depending on the target configuration, the number of switches to be programmed is different for upper switch programming first or lower switch programming first. Therefore, this work evaluates both the cases for reducing the number of programmed

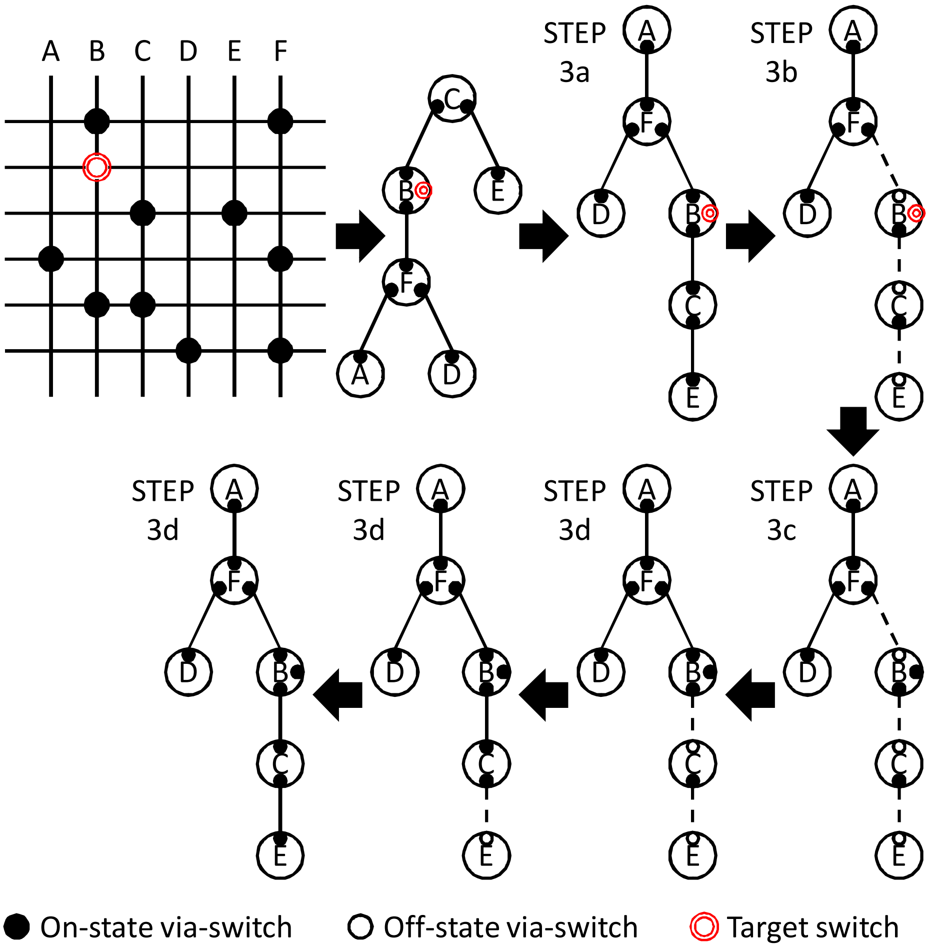


Figure 5.3: Proposed partial writing method.

switches.

5.2 Proof of Sneak Path Avoidance in Partial Reconfiguration

This section summarizes a proof of sneak path problem avoidance with three lemmas that correspond to STEP 1, 2, and 3 in the last column of Table 5.1.

***Lemma* 6.** *Lemma for the partial erasing process (STEP 1): The sneak path problem in the erasing process always applies a voltage to turn off to off-state switches that are placed in the same position as the target switch in the indistinguishable lines.*

Proof. When we turn off an atom switch in the indistinguishable lines, the sneak path problem arises in switches at the same position as the target switch in the indistinguishable lines. Assuming that these switches are on-state, these on-state switches and the target on-state switch connect already indistinguishable lines, i.e., the configuration has loops. This causes a contradiction since this work programs only non-looped configurations. Therefore, these switches that are affected by the sneak path problem in the erasing process are always off-state. Applying a voltage to turn off an already off-state switch does not matter, and hence we can accept the sneak path problem in the erasing process. □

***Lemma* 7.** *Lemma for STEP 2 of the partial writing process: The sneak path problem in the turning-on process of upper atom switches can be avoided by pre-erasing lower atom switches of on-state vertical connector switch in the same horizontal line of the target switch.*

Proof. As discussed in Section 4.2, indistinguishable horizontal lines lead to the sneak path problem in the turning-on process of upper atom switches. The cause of indistinguishable horizontal lines is vertical connector switches that are multiple on-state via-switches in the same vertical line. If these on-state via-switches exist in the same horizontal line of the target switch, the proposed method turns off the lower atom switches of these via-switches before the target switch programming. In this case, we can distinguish the target horizontal line and the programming signal never detours to other horizontal lines. □

***Lemma* 8.** *Lemma for STEP 3 of the partial writing process: The sneak path problem does not arise in each STEP 3a-3d.*

Proof. STEPs 3a and 3b do not include turning on operations, and therefore the sneak path problem never occurs thanks to Lemma 6. STEP 3c turns on atom switches in nodes that are isolated from the connection tree, and hence the programming signal is never propagated to other vertical lines and there is no sneak path problem. STEP 3d turns on connector switches to restore the connections of isolated nodes. When turning on connector switches in order from the shallow to the deep of the connection tree, each turning on operation becomes a leaf node programming. As proved in Lemma 4, the leaf node switch programming does not cause the sneak path problem, and hence there is no sneak path problem in STEP 3d. □

5.3 Proposed Minimization Method of Programming Steps

This section discusses the minimization of the programming steps in the partial reconfiguration. First, Section 5.3.1 models this minimization problem as a set cover problem. Then, Section 5.3.2 proposes a low computational complexity method that obtains the same solution of the set cover problem without solving it.

5.3.1 Minimizing Number of Switches Programed in Partial Reconfiguration

This subsection proposes a minimization method of the number of switches programmed in partial reconfiguration. The total number of programmed switches can be calculated by summing the number of switches programmed in partial erasing (STEP 1) and writing (STEP 2 and 3) process, which are expressed in equations (5.1)-(5.4).

$$\text{(Total \#SW}_{\text{prog}}{}^{[*]}\text{ in partial reprogramming)} = \text{(\#SW}_{\text{prog}}\text{ in STEPs 1, 2, and 3),}$$
$$^{[*]}\text{\#SW}_{\text{prog}}\text{: \# of programmed switches.} \quad (5.1)$$

$$\text{(\#SW}_{\text{prog}}\text{ in STEP 1)} = \text{(\# of S}_{\text{PU}}\text{ and S}_{\text{PL}}\text{).} \quad (5.2)$$

$$\text{(\#SW}_{\text{prog}}\text{ in STEP 2)} = \text{(\# of S}_{\text{NU}}\text{)} + \text{(\# of S}_{\text{CV}}\text{ in same horizontal line of S}_{\text{NU}}\text{).} \quad (5.3)$$

$$\text{(\#SW}_{\text{prog}}\text{ in STEP 3)} = \text{(\# of S}_{\text{CH}}\text{ to be pre-erased)} \times 2$$
$$+ \text{(\# of S}_{\text{CV}}\text{ in same horizontal line of S}_{\text{NU}}\text{)} + \text{(\# of S}_{\text{NC}}\text{ and S}_{\text{NN}}\text{).} \quad (5.4)$$

STEP 1 programs twice as many switches as the number of non-common via-switches in the previous configuration since the via-switch is composed of two atom switches and the proposed method have to turn off both switches $\mathrm{S_{PU}}$ and $\mathrm{S_{PL}}$. STEP 2 turns on all the upper atom switches $\mathrm{S_{NU}}$ of the non-common via-switches in the next configuration at STEP 2b. In addition, STEP 2a turns off the lower atom switch of on-state via-switches in the same horizontal line of $\mathrm{S_{NU}}$ if this on-state via-switch is a vertical connector switch $\mathrm{S_{CV}}$. Therefore, the number of switches programmed in STEP 2 is given by equation (5.3). In STEP 3, the number of programmed switches can be calculated by equation (5.4). The pre-erased connector switches $\mathrm{S_{CH}}$ are programmed twice, i.e., turning off (pre-erasing) before the target programming at STEP 3b and turning on after the target programming at STEP 3d. The target switches $\mathrm{S_{NC}}$, $\mathrm{S_{NN}}$, and $\mathrm{S_{CV}}$, where $\mathrm{S_{CV}}$, which is turned off at STEP 2a, is programmed back at STEP 3c.

The number of S_{CH} to be pre-erased in equation (5.4) varies depending on the chosen root node of the connection tree since it changes the tree structure and the number of descendant nodes of the target node. On the other hand, equations (5.2), (5.3), and remaining terms of equation (5.4) are fixed for a given configuration. Hence, this work minimizes the number of pre-erased connector switches S_{CH}.

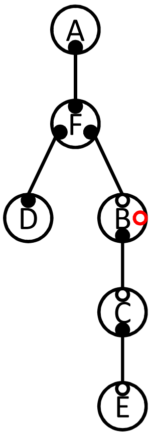
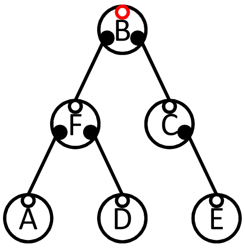
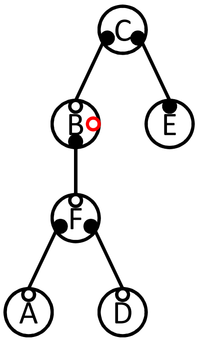
Supposing that there is only one target switch, the number of connector switches S_{CH} to be turned off before the target programming can be given by

$$\begin{aligned}&(\text{\# of } S_{CH} \text{ to be pre-erased for one target switch})\\&\quad= (\text{\# of } S_{CH} \text{ to parent in target and its descendant nodes})\\&\qquad- (\text{\# of } S_{CH} \text{ already turned off among first term}). \quad (5.5)\end{aligned}$$

The pre-erasing phase at STEP 3b turns off the connector switches connecting to the parent node in the target and its descendant nodes for disconnecting these nodes from the connection tree. There are already off-state connector switches since these switches are turned off in STEP 2a, and therefore the number of pre-erased S_{CH} is obtained by equation (5.5). Figure 5.4 exemplifies that the structure of the connection tree and the number of pre-erased S_{CH} change depending on the root node selection. By choosing the root node that minimizes the number of pre-erased S_{CH}, the total number of programmed switches in the entire partial reconfiguration can be minimized.

On the other hand, when there are multiple target switches in a connection tree, the total number of pre-erased connector switches is not necessarily the sum of equation (5.5) for each target switch. This is because some target switches could be programmed during the programming process of another target switch, where this work regards the former switches are dominated by the latter switch. For example in the connection tree of Figure 5.5, target switches a and f can be programmed in the programming process of target switch b when the root node is node E. Here, switches a and f are dominated by switch b. In this case, STEP 3b disconnects target node B and its descendant nodes A, C, D, F from the connection tree. Then, STEP 3c can turn on not only target switches b but also switches a and f since nodes A, B, and F are isolated, and the programming signal never propagates to any other nodes.

Thanks to this dominance property, this work can reduce the number of programmed switches compared to the case that STEP 3 is applied separately to each target switch. In the example of Figure 5.5, the minimum number of programmed switches is achieved by selecting node E or H as the root node and dividing target switches into three groups, i.e., "switches a, b, and f", "switch g", and "switch i", where switches a and f are dominated by switch b. This work separately applies STEP 3 to each switch group, and all the switches in the same group are programmed in the same process of STEP 3. The following paragraphs

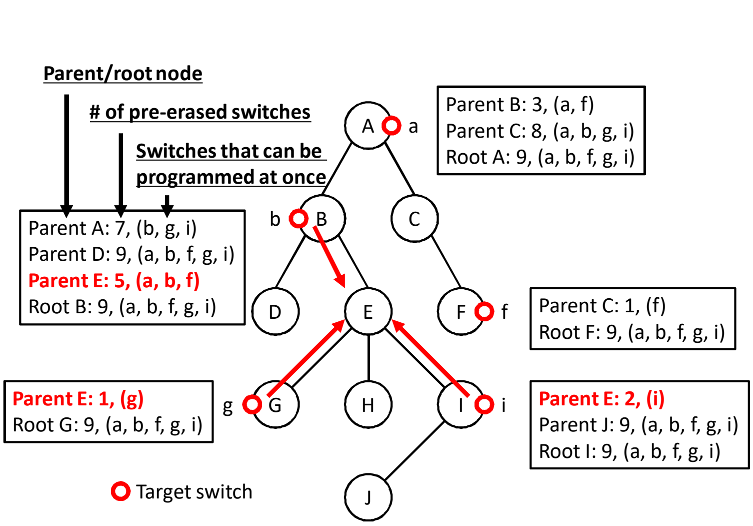
Root node	Node A	Node B	Node C
Structure of connection tree ● On-state ○ Off-state ○ Target			
Number of S_{CH} to be pre-erased (Number of ○)	**3**	5	4

Choose the smallest one for minimizing the number of switch programming

Figure 5.4: Changes in the structure of connection tree and the number of pre-erased S_{CH} depending on the root node selection.

explain how to derive the optimal root node and the target switch groups.

This work models this optimization problem as a set cover problem with cost minimization. For each target switch, the proposed method calculates the number of pre-erased connector switches and enumerates other target switches that are dominated by the target switch of interest while changing the root node. The number of pre-erased connector switches, which is the cost of this problem, can be given by equation (5.5). The proposed method can enumerate dominated target switches that can be programmed in the same process by checking whether other target switches are included in the target node of interest and its descendant nodes. Here, the dominance relation is determined once the parent node of the target switch is fixed. Therefore, the above enumeration for a target switch is repeated for the number of edges of the target node, not for the number of nodes in the connector tree. In the other case, the target node is the root node, and the dominance relation is also fixed. The above procedure gives some pairs of the number of pre-erased connector switches and the group of target switches that can be turned on in the same process of STEP 3. From the combination of these pair


Parent/root node
# of pre-erased switches
Switches that can be programmed at once
Parent A: 7, (b, g, i)
Parent D: 9, (a, b, f, g, i)
Parent E: 5, (a, b, f)
Root B: 9, (a, b, f, g, i)
Parent B: 3, (a, f)
Parent C: 8, (a, b, g, i)
Root A: 9, (a, b, f, g, i)
Parent C: 1, (f)
Root F: 9, (a, b, f, g, i)
Parent E: 1, (g)
Root G: 9, (a, b, f, g, i)
Parent E: 2, (i)
Parent J: 9, (a, b, f, g, i)
Root I: 9, (a, b, f, g, i)
A
a
B
b
C
D
E
F
f
G
g
H
I
i
J
Target switch

a	b	f	g	i	Cost
X		X			3
X	X		X	X	8
X	X	X	X	X	9
	X		X	X	7
X	X	X	X	X	9
X	X	X			5
X	X	X	X	X	9
		X			1
X	X	X	X	X	9
			X		1
X	X	X	X	X	9
				X	2
X	X	X	X	X	9
X	X	X	X	X	9

Figure 5.5: Minimization method of the number of switch programming.

information, the proposed method finds a set that covers all the target switches and minimizes the total number of pre-erased connector switches.

Figure 5.5 exemplifies the set cover problem defined above. For target switch a, the proposed method counts the number of pre-erased connector switches and the covered target switches in three cases; i.e., when node B is parent, when node C is parent, and when node A is root. In case when node B is parent, it is necessary to disconnect nodes A, C, and F in STEP 3b, and hence the number of pre-erased switches is three and the dominated switches are a and f. In the same way, the proposed method counts the number of pre-erased switches and dominated switches for each target switch. Then, this work constructs the table in the right side of Figure 5.5, where each row represents a group of a dominant node and dominated nodes and the corresponding cost of the number of pre-erased switches. From this table, the proposed method finds a set of three red rows that covers all the target switches and minimizes the cost to 8. In this example, we can minimize the number of programmed switches so that node E or H is selected as the root node and the target switches are divided into three groups “switches a, b, and f”, “switch g”, and “switch i”.

5.3.2 Root Node Selection with Lower Computational Complexity

The previous subsection demonstrates a minimization method of the number of programmed switches in partial reconfiguration by solving the set cover problem. However, the set cover problem is well known to be NP-hard. Thus, this subsection proposes an efficient root selection method to minimize the number of programmed switches without solving the set cover problem.

The proposed method exploits a property of the connection tree that at least one solution set of rows in the set cover problem can share the same root node. With this property, the proposed method only needs to count the number of programmed switches sequentially supposing each node of the connection tree is the root node, and chooses the one with the minimum number of programmed switches. This approach reduces the computational complexity compared to solving the set cover problem since the number of nodes of the connection tree is the number of vertical signal lines at most. The proposed root selection method can be implemented as a polynomial-time algorithm and the details will be explained in Section 5.4.

The following paragraphs prove that the above property is always satisfied. For this proof, this work introduces representative switches, which are defined as the target switches that are not dominated by other target switches in a solution of the set cover problem. One representative switch is included as one row in the table like Figure 5.5, and this entry is obtained by assuming the parent node or the root node. Therefore, the direction to the root node is specified. For example in Figure 5.5, the solution selects three representative switches b, g, and i, and three red arrows indicate the direction to the root node. In this case, node E or H is selected as the root node that satisfies all the red arrows at the same time. If such a root node exists in any case, there is no need to solve the set cover problem. The following calls the node that contains a representative switch as the representative node.

To derive the property that optimal representative switches can share the same root node, this work proves that representative switches that have common dominated switch never compose an optimal solution first. The proof supposes that there are two cases of representative switches that cover all the target switches, and the one is with one or more common dominated switches and the other has no common dominated switch. For example, a pair of 3rd and 7th rows of the table in Figure 5.5 belongs to the former case, which has common dominated switches a and b. On the other hand, a pair of three red (7th, 11th, and 13th) rows corresponds to the latter case without common dominated switches. When we compare the number of pre-erased switches in both cases, the former is always costlier than the latter. This is because the node including the dominated switch is disconnected and reconnected in each programming of a representative switch. The former disconnects/reconnects such a node multiple times, but the latter disconnects/reconnects it only once. Therefore, the optimal solution selects representative switches that have no common dominated switches.

Consequently, the proof can conclude that optimal representative switches are not dominated by other target switches. Hence, the original property can be proved by

$$\begin{aligned}
&\mathrm{SW_{opt\text{-}rep}}^{[**]} \text{ are not dominated by other target switches.}\\
&\Rightarrow \text{Multiple } \mathrm{SW_{opt\text{-}rep}} \text{ are not dominated by each other.}\\
&\Leftrightarrow \text{Multiple } \mathrm{SW_{opt\text{-}rep}} \text{ are not descendants of each other.}\\
&\Rightarrow \text{Multiple } \mathrm{SW_{opt\text{-}rep}} \text{ share a root node.}\\
&{}^{[**]}\mathrm{SW_{opt\text{-}rep}}\text{: optimal representative switches.}
\end{aligned} \tag{5.6}$$

Another proof in an exhaustive manner for equation (5.6) is given in Appendix A.

5.4 Pseudo Code of Partial Reprogramming

This section gives a pseudo code of the proposed partial reprogramming method. Algorithm 2 includes only STEP 3 of the partial writing process since STEPs 1 and 2 are straightforward. Lines 3-8, line 9, lines 10-12, and line 13 correspond to STEP 3a, 3b, 3c, and 3d, respectively.

As explained in Section 5.3.2, lines 3-8 calculate the number of programmed switches by function CalcCost repeatedly changing the root node, and choose the one with the minimum number of programmed switches. Function CalcCost recursively traverses the connection tree with depth-first search, and count the number of programmed switches. Inside this function, lines 15-16 search representative nodes from the closer nodes to the root node for each edge. After that, lines 17-18 enumerate on-state connector switches connecting to the parent in a representative node and its descendant for all representative nodes. The proposed method puts these connector switches in $\mathbb{S}_{\text{erase}}$ and increments the cost, which is the number of programmed switches, by one each time a connector switch is put. STEP 3a performs a depth-first search as many times as the number of nodes in the connection tree, and hence the worst time complexity is $O(|V|(|V| + |E|))$ where $|V|$ and $|E|$ are the number of nodes and edges of the connection tree, respectively. Here, $|E| = |V| - 1 < |V|$ holds in a tree structure, and $|V|$ is N_{vertical} at most, which is the number of vertical signal lines in a crossbar. Therefore the worst time complexity can be written as $O({N_{\text{vertical}}}^2)$.

After the completion of lines 3-8, the switches to be pre-erased are in $\mathbb{S}_{\text{erase}}$, and the proposed method turns off them in line 9 as STEP 3b. Then, STEP 3c turns

Algorithm 2 Minimizing programmed switches.

```
 1: $\mathbb{S}_{\text{target}} = \{\text{S}_{\text{CV}},\ \text{S}_{\text{NC}},\ \text{S}_{\text{NN}}$ to be turned on$\}$
 2: $\mathbb{S}_{\text{ch}} = \{$On-state $\text{S}_{\text{CH}}\}$
 3: $\mathbb{S}_{\text{erase}} = \emptyset,\ \text{cost} = \infty$
 4: for each node $N_i$ do
 5:     $\mathbb{S}_{\text{tmp}} = \emptyset,\ \text{cost}_{\text{tmp}} = 0$
 6:     CalcCost($N_i,\ \emptyset,\ \mathbb{S}_{\text{tmp}},\ \text{cost}_{\text{tmp}},$ False)
 7:     if $\text{cost}_{\text{tmp}} < \text{cost}$ then
 8:         $\mathbb{S}_{\text{erase}} = \mathbb{S}_{\text{tmp}},\ \text{cost} = \text{cost}_{\text{tmp}}$
 9: Turn off all $\mathbb{S}_{\text{erase}}$
10: for $S_i \in \mathbb{S}_{\text{target}}$ do
11:     if $S_i$ is not connector switch to parent then
12:         Turn on $S_i,\ \mathbb{S}_{\text{target}} = \mathbb{S}_{\text{target}} - \{S_i\}$
13: Turn on all $\mathbb{S}_{\text{erase}} \cup \mathbb{S}_{\text{target}}$ in order from shallow to deep

14: function CalcCost($N_i,\ \mathbb{N}_{\text{rep}},\ \mathbb{S}_{\text{erase}},\ \text{cost},\ \text{flag}$)
15:     if flag = False, $N_i$ has $S_j \in \mathbb{S}_{\text{target}}$ then
16:         $\mathbb{N}_{\text{rep}} = \mathbb{N}_{\text{rep}} \cup \{N_i\},\ \text{flag} = \text{True}$
17:     if flag = True, $N_i$ has $S_j \in \mathbb{S}_{\text{ch}}$, $S_j$ is connecting to parent then
18:         $\mathbb{S}_{\text{erase}} = \mathbb{S}_{\text{erase}} \cup \{S_j\},\ \text{cost} = \text{cost} + 1$
19:     for $N_{\text{child}} \in \{N_j \mid N_j$ is child node of $N_i\}$ do
20:         CalcCost($N_{\text{child}},\ \mathbb{N}_{\text{rep}},\ \mathbb{S}_{\text{erase}},\ \text{cost},\ \text{flag}$)
21:     if $N_i \in \mathbb{N}_{\text{rep}}$ then
22:         flag = False
```

on target switches S_{CV} and S_{NN} in lines 10-12. Finally, line 13 turns on connector switches S_{CH} and S_{NC} in order from shallow to deep at STEP 3d.

5.5 Evaluation Results

This subsection experimentally demonstrates how much the number of programmed switches is reduced by the proposed partial reprogramming method. Figures 5.6 and 5.7 compare the number of programmed switches in the conventional method and that in the proposed method. Here, the conventional method turns off all the on-state via-switches in the previous configuration and then writes the next configuration to the crossbar. This evaluation randomly generates non-looped previous and next configurations and derives a sneak path free programming order minimizing the programmed switches by the proposed

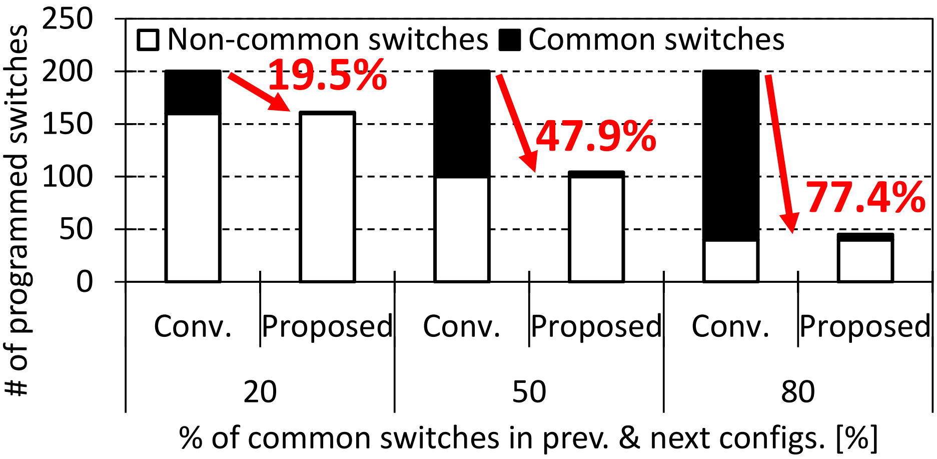


Figure 5.6: Number of programmed switches in reconfiguration with conventional and proposed methods when 0.5% of via-switches are on-state in 100x100 crossbar.

method. This evaluation chose the locations of on-state switches using uniformly distributed random numbers. The crossbar size was set to 100x100, and the percentage of on-state via-switches in a crossbar was 0.5% in Figure 5.6 and 1.5% in Figure 5.7. This evaluation also varied the percentage of common on-state via-switches in the previous and next configurations from 20% to 80%, and performed 10,000 trials for each case. Programmed common and non-common switches are depicted with different colors.

From Figure 5.6, we can see that the number of programmed switches in the conventional method is fixed to 200, which is 100 × 100 via-switches × 0.5% × 2 atom switches × 2 configurations (previous and next), for each case. In the proposed method, the number of programmed non-common switches is the same as that of the conventional method since it is essential to erase and write non-common switches for reconfiguration. On the other hand, the number of programmed common switches is dramatically reduced by the proposed method. As a result, the proposed method reduces the total number of programmed switches by 19.5% to 77.4%. The reduction of 77.4% increases the number of possible reconfiguration executions of the via-switch FPGA by 4.4X. The reduction in the number of programmed switches also reduces reconfiguration time. If the via-switch FPGA shares the programming drivers between the entire CLBs array for reducing the area of peripheral circuits, it is necessary to program via-switches sequentially. In this case, the reconfiguration time is the programming time of each switch multiplied by the number of programmed via-switches. Therefore, 77.4% reduction in the number of programmed switches reduces reconfiguration time by 77.4%. On the other hand, the via-switch FPGA

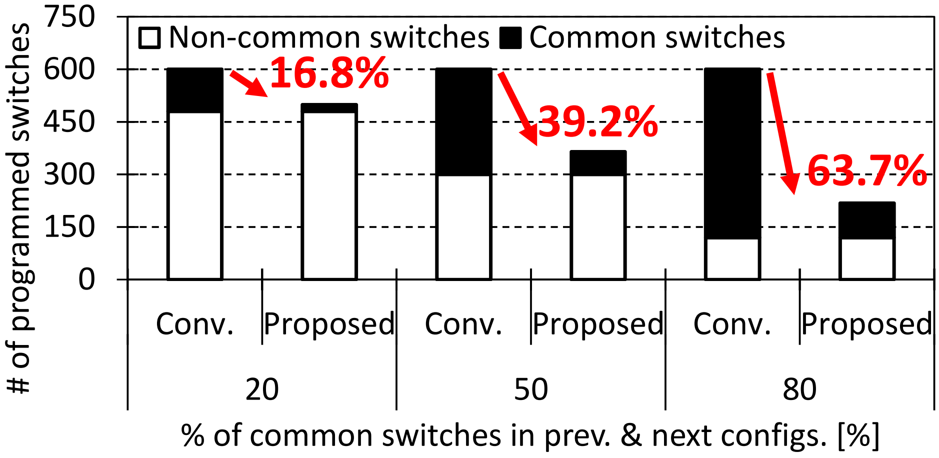


Figure 5.7: Number of programmed switches in reconfiguration with conventional and proposed methods when 1.5% of via-switches are on-state in 100x100 crossbar.

can also adopt a parallel programming scheme by increasing the number of drivers. Even in this case, the reconfiguration time decreases while the reduction ratio may vary depending on the maximum number of programmed switches in independent programming regions.

Comparing Figures 5.6 and 5.7, the reduction ratio on the number of programmed switches decreases slightly as the percentage of on-state via-switches increases from 0.5% to 1.5%. This is because the density of on-state via-switches is relatively high and we have to erase more common switches for avoiding the sneak path problem. However, we can still see that the proposed method considerably reduces the number of programmed switches.

Next, this paragraph discusses the impact of optimal root node selection in STEP 3a of partial writing. This evaluation compares the number of programmed switches in both cases that STEP 3a selects an optimal root node and the worst root node. Figure 5.8 shows a histogram of reduction ratio in the number of programmed switches from the worst case to the optimal case. In this evaluation, the crossbar size is 100x100, the percentage of on-state via-switches in a crossbar of the previous and next configurations are 1% and 1.1%, all the on-state via-switches in the previous configuration are included in the next configuration, and consequently 0.1% via-switches are newly turned on in the next configuration. The number of trials is 10,000. The number of programmed switches can be reduced by 70% at maximum and 29% on average thanks to the optimal root selection. This result indicates that the optimal root selection plays an important role in the proposed method.

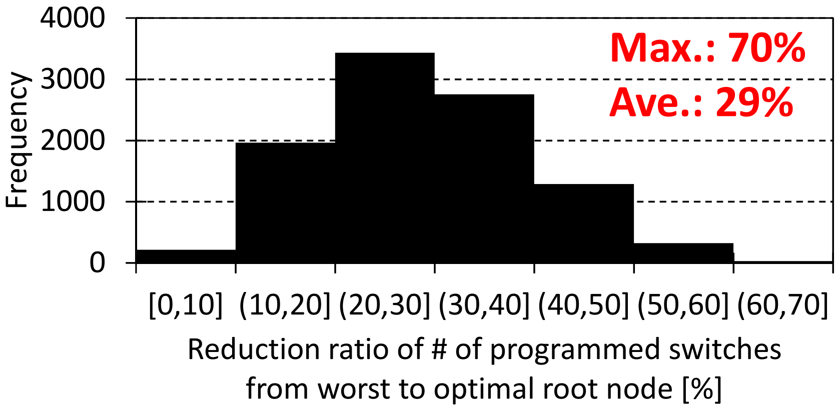


Figure 5.8: Histogram of reduction ratio in number of programmed switches from worst root selection to optimal root selection.

5.6 Discussion

As is proved, the proposed method achieves the sneak path free partial reconfiguration of the via-switch crossbar with minimum programming steps. This section discusses an application example of the proposed method for the chip testing. To verify the via-switch FPGA functionality, the manufacturer has to check whether there are faults in manufactured FPGA chips before the shipment. In the FPGA chip test, built-in self-test (BIST) techniques are often adopted [114–119]. The BIST techniques utilize reconfigurability of FPGAs for testing itself without adding dedicated testing circuits to the chip, i.e., they program some CLBs to test the other CLBs. The following paragraphs explain that the proposed method can accomplish further reduction of programming steps by focusing on the feature of the BIST techniques that test patterns of the BIST can be arranged in arbitrary order.

In the BIST, all the CLBs are configured as groups of block under test (BUT), test pattern generator (TPG), and output response analyzer (ORA). The BIST uses TPGs and ORAs for testing BUTs, specifically test patterns that are generated by TPGs are applied to BUTs, and then the output of BUTs is compared with the expected value in ORAs as illustrated in Figure 5.9. The BIST consists of multiple test sessions so that all CLBs become BUT at least once. For example in Figure 5.10, the BIST is composed of two test sessions. Test session #1 programs CLBs at even rows as BUTs and CLBs at odd rows as TPGs and ORAs, and vice versa in test session #2. Thanks to these two test sessions, all the CLBs in the FPGA are tested. Figure 5.10 also shows that each test session has multiple test phases to check various faults in the BUT, e.g., transistor faults, interconnect faults, and so on. The BIST reprograms CLBs every time a test phase is shifted.

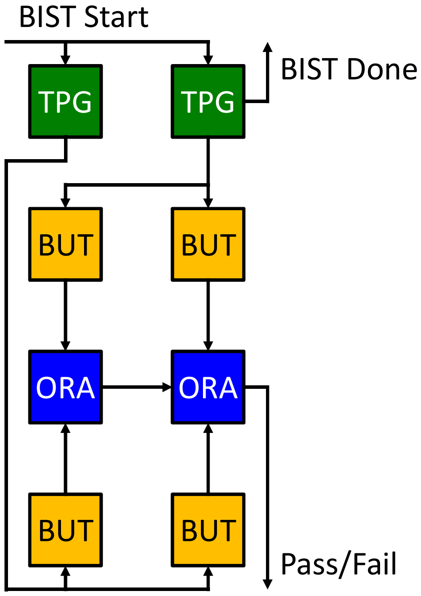


Figure 5.9: Built-in self-test (BIST) approach.

In the above BIST approach, the order of test sessions and test phases is arbitrary. For example, we can perform test session #2 before test session #1, and test phase #1 can be moved after test phase #2 in Figure 5.10. By combining this feature of order independence and the proposed partial reconfiguration method, this section proposes a method that can further reduce the number of programmed switches in the entire BIST sequence. First, this work models the BIST sequence as a graph structure where each node represents one configuration of via-switch FPGA. Here, the beginning and end are a configuration where all via-switches are off-state, and many test configurations exist between them. Each edge weight between arbitrary two nodes, which corresponds to the number of programmed switches required for changing one configuration to another one, can be calculated by equation (5.5). It should be noted that the edge weight from configuration A to configuration B and the edge weight from configuration B to configuration A are different. By calculating the edge weights of all the combinations of two configurations, it gives a complete directed graph as depicted in the top of Figure 5.11. After that, by solving an asymmetric traveling salesman problem with the obtained graph, an optimal BIST sequence with minimum programming steps can be derived as shown in the bottom of Figure 5.11. The proposed optimization method of the BIST sequence minimizes the manufacturing test time. The cost of manufacturing test accounts for a large percentage of a unit price of a chip, and hence the minimization of the test time thanks to the proposed method contributes to reducing the chip cost. The proposed method also contributes to

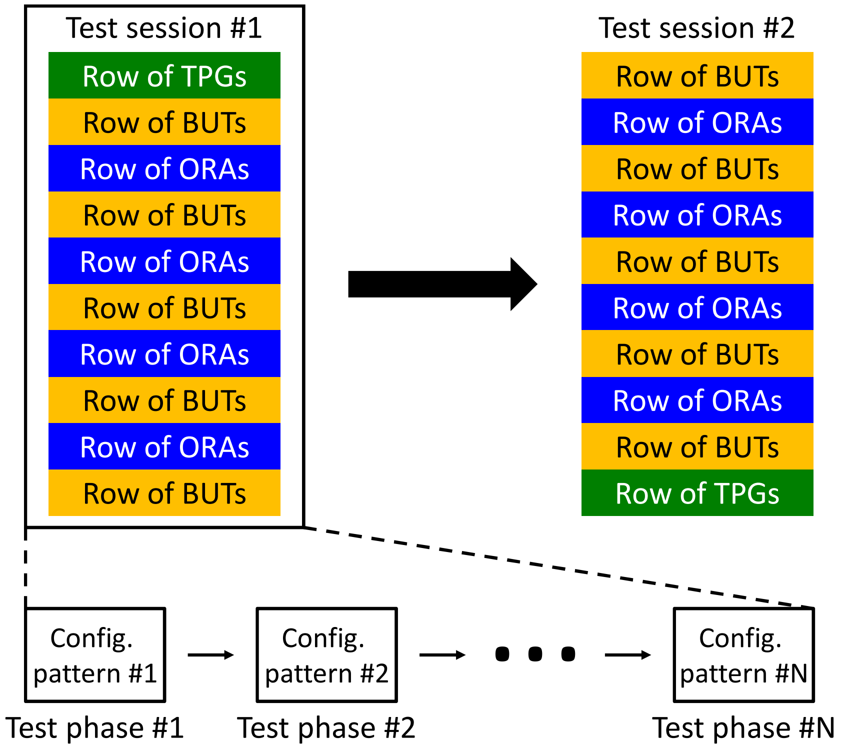


Figure 5.10: Test sessions and test phases in BIST.

maximizing the number of reprogramming in the FPGA lifetime at the FPGA user side after the shipment.

5.7 Conclusion

This chapter proposed a minimization method of the programming steps in the partial reconfiguration of the via-switch crossbar without the sneak path problem. The proposed method minimizes the number of programmed switches by arranging the root node of the connection tree that represents the connection status of signal wires in a crossbar. This chapter models the optimal root node selection as a set cover problem with cost minimization, and also proposes a low computational complexity method that obtains the same solution of the set cover problem without solving it. In a test case, the proposed method reduced the number of programmed switches by 77.4% compared to the conventional approach, which enables 4.4X more reconfigurations of the via-switch FPGA and reduces reconfiguration time by 77.4%. The proposed method contributes to extending the lifetime of via-switches and speeding up the reconfiguration of

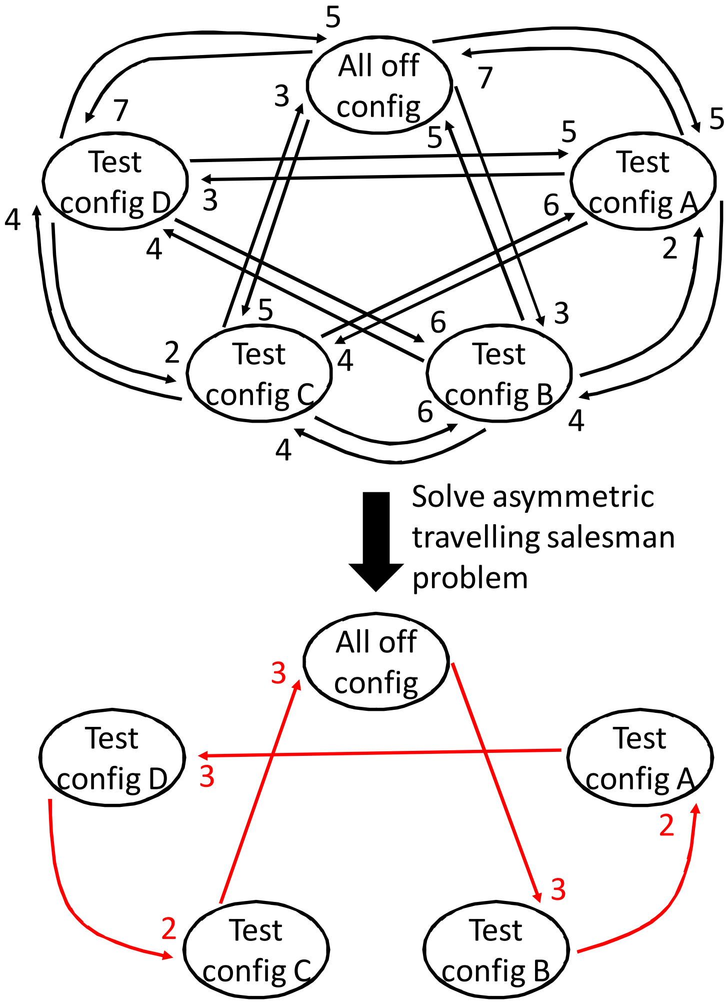


Figure 5.11: Complete directed graph that represents entire BIST sequence and optimal test order derivation by solving asymmetric travelling salesman problem.

the via-switch FPGA for both user programming and manufacturing test.

Future work includes the generalization of the sneak path free reconfiguration method proposed in Chapters 4 and 5 for other crossbar structures with various types of switches beyond the via-switch FPGA. Crossbar structures are widely adopted in many non-volatile FPGAs and memories because of high integration density, and these circuits also suffer from the sneak path problem. When the proposed method in this dissertation is generalized, it may solve the sneak path problem in other FPGAs or memories than via-switch FPGA. For discussing the generality, it is needed to define generalized crossbar structures that can accommodate various types of switches and identify what is the necessary condition for the generalized structures.

Chapter 6

Conclusion

This dissertation discussed primary challenges that the non-volatile via-switch FPGA was facing at the design phase, manufacturing phase, and user programming phase for practical application. This work provided solutions for each challenge to realize the high energy efficiency, defect-free, and sneak path free via-switch FPGA. This chapter summarizes this dissertation, and highlights its contributions, and future works.

For the challenge at the design phase, Chapter 2 focused on the interconnect structure that fully utilizes advantages of via-switches in terms of small footprint, non-volatility, BEoL integration, and small parasitic load. This work proposed an interconnect structure that has a functionality of selective repeater insertion to signal paths and achieves low interconnect delay and high energy efficiency. This chapter also identified the programming structure requirement at the inter-CLB connection switch taking the sneak path problem into account. A substantial performance improvement compared with conventional SRAM-based FPGA thanks to the proposed interconnect structure was demonstrated by transistor-level SPICE-based evaluation. The evaluation result showed that the proposed structure improved the crossbar integration density by 26X and reduced 90% of interconnect delay and 94% of energy at 0.5 V operation. The proposed interconnect structure was actually adopted in the fabricated trial chip of the via-switch FPGA [120].

To ensure arbitrary routings at the manufacturing phase before the shipment, Chapter 3 discussed how to explore faulty via-switches in the crossbar. This chapter demonstrated that on/off-states of via-switches can be discriminated by using a general differential pair comparator. The comparator can be implemented with a small area at the peripheral part of the via-switch FPGA chip. This chapter also clarified fault modes of a via-switch using SPICE simulation that injected stuck-on/off faults to atom switch and varistor. Then, this work proposed a look-up table based fault detection and diagnosis methodology that identifies faulty via-switches in the crossbar, where the look-up table enumerates the comparator response in normal and faulty cases. The proposed fault testing method had 100% fault detectability. In terms of the fault diagnosability, the successful ratios of 100% and 79% were accomplished in cases that the number of faulty components in a via-switch was up to one and up to two, respectively.

At the user programming phase, the sneak path problem that unintentionally changes on/off state of non-target via-switches may arise depending on FPGA configuration patterns. Chapter 4 clarified the crossbar programming status that causes the sneak path by exploring the occurrence conditions of the sneak path problem. Then, this chapter proposed a sneak path free initial programming method based on arranging the programming sequence of via-switches in a crossbar. Here, the initial programming means the programming that is performed for the crossbars whose via-switches are all off-state. This work devised a connection tree based algorithm that effectively obtains a sneak path free programming order, where the connection tree represents the connection status of signal lines in a crossbar as a tree structure. This chapter also formally proved that arbitrary non-looped on-off patterns in a crossbar necessarily had a sneak path free programming order. The quantitative evaluation demonstrated a significant improvement of the routing flexibility in the via-switch crossbar thanks to the proposed method. This work confirmed that the proposed method increased the number of available configurations by over four orders of magnitude in a practically-sized 100x100 crossbar. In any practical configurations of via-switch FPGA, the sneak path problem can be solved by the proposed method.

Chapter 5 extended the proposed method of Chapter 4 for partial reconfiguration in the crossbar where some via-switches are already on-state at the initial state. A partial reconfiguration method that minimizes programming steps without the sneak path problem is proposed in this chapter. The proposed method minimizes the number of programmed switches by arranging the root node of the connection tree, and this chapter modeled the optimal root node selection as a set cover problem with cost minimization. Furthermore, this work also proposed a low computational complexity method that obtains the same solution of the set cover problem without solving it. The simulation-based evaluation demonstrated that the proposed method reduced the number of programmed switches by 77.4% compared to the conventional approach. This 77.4% reduction enables 4.4X more reconfigurations of the via-switch FPGA and shortens reconfiguration time by 77.4%.

This dissertation covers and solves critical challenges at all phases of via-switch FPGA development. The interconnect structure proposed in Chapter 2 improves the operating speed and energy efficiency. The fault diagnosis method proposed in Chapter 3 can identify defective products after manufacturing and prevent the shipment of those products. The sneak path solution proposed in

Chapter 4 can program all the practical configuration patterns to the via-switch FPGA. The partial reconfiguration method proposed in Chapter 5 extends the lifetime of via-switches and improves the reconfiguration speed of the via-switch FPGA for both user programming and manufacturing test. Finally, these contributions enable FPGA users to implement any applications on high-performance and defectless via-switch FPGA.

Future work in the fault diagnosis method of Chapter 3 is to develop a fault testing method for logic circuits and interconnections. For this purpose, the BIST technique, which is briefly introduced in Section 5.6, is one of the suitable solutions. The BIST technique checks faults of manufactured FPGA without adding dedicated testing circuits to the chip by utilizing the FPGA reconfigurability, i.e., some CLBs are configured to test the other CLBs. As discussed in Section 5.6, the proposed partial reconfiguration method can be combined with the BIST technique, and contributes to minimization of the number of programmed switches in the test phase and maximization of the number of reprogramming at user side after the shipment.

Another future work includes the verification of the proposed fault testing and sneak path avoidance methods in the trial via-switch FPGA chip. The trial chip is based on the proposed interconnect structure in Chapter 2 and has been fabricated with 65 nm process node [120]. For the proposed fault diagnosis method in Chapter 3, it is necessary to confirm that the comparator can distinguish the via-switch on/off-states and the proposed fault diagnosis method actually works in the trial chip. Also, for the sneak path free reconfiguration methods in Chapters 4 and 5, it is needed to validate that the proposed methods surely eliminate the sneak path and completely solve this problem.

Appendix A

Another proof for Equation (5.6)

Another proof for equation (5.6) is presented. This chapter separately gives the proof for the two cases; when only one representative switch is given by the solution of the set cover problem, and when multiple representative switches are given.

When there is only one representative switch, the proof is self-evident. The direction to the parent node is specified only by one representative switch, and hence no contradiction occurs. Otherwise, the solution says that the representative node is the root node, and again no contradiction occurs.

Next, the following discusses the proof for the case of multiple representative switches. This work proves the case of two representative switches since this proof is easily extended to the cases that there are three or more representative switches. Figure A.1(a) illustrates the generalized tree structure with two representative switches, and Figure A.1(b) shows all the combinations of two representative switches. No matter how many edges a representative node has, they can be categorized into two groups; (1) an edge directed to another representative node and (2) other edges. All the nodes except representative nodes can be divided into node groups A, B, and C in Figure A.1(a). Node group B contains the nodes between the (1) edges of the two representative nodes. Nodes beyond the edges (2) of representative nodes S and T are categorized into node groups A and C, respectively. In Figure A.1(b), each red arrow indicates the direction to the root node from each representative switch, and the red arrow pointing to the representative node itself indicates that the representative node is the root node. There are three directions for each red arrow (edge (1), edge (2), and the representative node itself), and hence there are 3×3 = 9 patterns of two red arrows. This work divides these 9 patterns into three cases. Case 1 contains patterns that include at least one red arrow pointing to the representative node itself. Patterns that have the red arrow indicating the direction of edge (2) are categorized into case 2. Case 3 contains patterns where both the red arrows indicate the direction

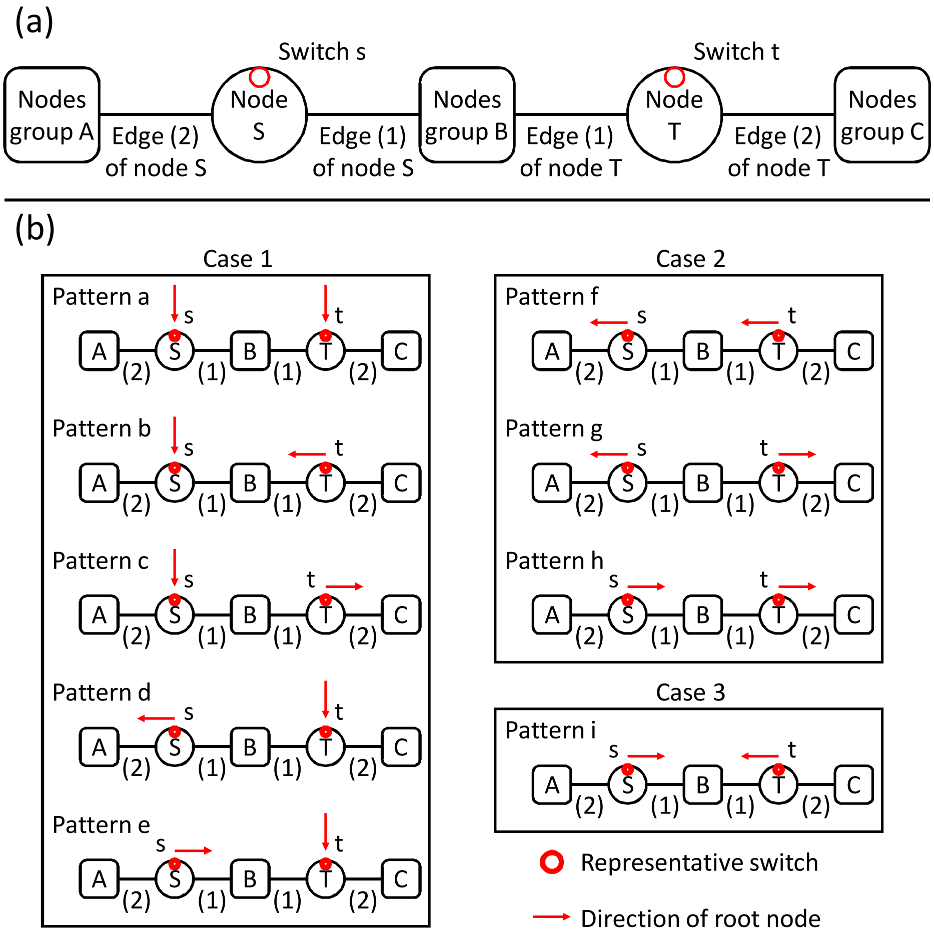


Figure A.1: All combinations of two representative switches.

of edge (1).

From now, this paragraph proves that the possible pattern is only case 3 in Figure A.1(b). In case 1, there is at least one red arrow pointing to the representative node itself, which indicates that choosing this representative node as the root node is optimal. To program the representative switch in the root node, it is needed to disconnect the descendant nodes of the root node, i.e., all the nodes of the connection tree. Therefore, another representative node is also disconnected and can be programmed in this programming step, which means there is a dominance relationship. From the definition, representative switches do not dominate each other, and hence the patterns in case 1 never exist as a solution of the set cover problem. In general, from the above reason, the red arrow pointing to the representative node itself never exists in cases where there are two or more representative nodes. Next, the same discussion is applied to case 2 where at least one red arrow indicates the direction of edge (2). In this case, another representative node is included in the descendant nodes of this representative node. For example in pattern f of Figure A.1(b), representative switch s argues that the root node is included in node group A. At that time, another representative switch t is one of the descendant nodes of node S, and then there is a dominance relationship. Hence, patterns in case 2 do not exist as a solution of the set cover problem. The same discussion is applicable to cases of three or more representative nodes. From the above discussion, the possible pattern of red arrows is only case 3 where all the red arrows indicate the direction to intermediate nodes of all representative nodes. All the red arrows can be satisfied by choosing the root node from intermediate nodes of representative switches, and the proposed method can minimize the number of programmed switches without solving the set cover problem.

Bibliography

[1] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley Publishing Company, 2010.

[2] Intel Corporation, "Microprocessor Quick Reference Guide."

[3] Apple, "iPhone 11 Pro," https://www.apple.com/iphone-11-pro/.

[4] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors (ITRS)."

[5] N. Campregher, P. Y. K. Cheung, G. A. Constantinides, and M. Vasilko, "Yield Enhancements of Design-specific FPGAs," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays (FPGA)*, 2006, pp. 93–100.

[6] K. Flamm, "Measuring Moore ' s Law: Evidence from Price, Cost, and Quality Indexes," National Bureau of Economic Research, Working Paper 24553, April 2018.

[7] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May 2015.

[8] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[9] Microsoft Research, "Project Catapult."

[10] Amazon Web Services, “Amazon EC2 F1 Instances.”

[11] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, “Serving DNNs in Real Time at Datacenter Scale with Project Brainwave,” *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar 2018.

[12] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, “A Configurable Cloud-scale DNN Processor for Real-time AI,” in *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2018, pp. 1–14.

[13] Microsoft Research, “Project Brainwave.”

[14] C. Maxfield, *FPGAs: Instant Access*. Newnes, 2008.

[15] V. George and J. M. Rabaey, *Low-energy FPGAs: Architecture and Design*. Kluwer Academic Publishers, 2001.

[16] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[17] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. now, 2008.

[18] I. Kuon and J. Rose, “Measuring the Gap Between FPGAs and ASICs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[19] M. Lin, A. E. Gamal, Y. C. Lu, and S. Wong, “Performance Benefits of Monolithically Stacked 3-D FPGA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 216–229, Feb 2007.

[20] J. Backus, “Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs,” *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978.

[21] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, and K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," 2008.

[22] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, April 2011.

[23] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A Survey and Evaluation of FPGA High-Level Synthesis Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct 2016.

[24] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. AviÅienis, J. Wawrzynek, and K. Asanovi, "Chisel: Constructing hardware in a Scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[25] F. Winterstein, S. Bayliss, and G. A. Constantinides, "High-level synthesis of dynamic data structures: A case study using Vivado HLS," in *2013 International Conference on Field-Programmable Technology (FPT)*, Dec 2013, pp. 362–365.

[26] J. Choi, S. Brown, and J. Anderson, "From software threads to parallel hardware in high-level synthesis for FPGAs," in *2013 International Conference on Field-Programmable Technology (FPT)*, Dec 2013, pp. 270–277.

[27] P. Coussy and A. Morawiec, Eds., *High-Level Synthesis: from Algorithm to Digital Circuit.* Springer Netherlands, 2008.

[28] K. Zaitsu, K. Tatsumura, M. Matsumoto, M. Oda, and S. Yasuda, "Non-volatile Programmable Switch With Adjacently Integrated Flash Memory and CMOS Logic for Low-Power and High-Speed FPGA," *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4009–4014, Dec 2015.

[29] M. Abusultan and S. P. Khatri, "Exploring static and dynamic flash-based FPGA design topologies," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 416–419.

[30] J. Y. Jia, P. Singaraju, F. Dhaoui, R. Newell, P. Liu, H. Micael, M. Traas, S. Sammie, F. Hawley, J. McCollum, and V. den Abeelen Werner, “Performance and reliability of a 65nm Flash based FPGA,” in *2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology*, Oct 2012, pp. 1–3.

[31] K. Zaitsu, K. Tatsumura, M. Matsumoto, M. Oda, S. Fujita, and S. Yasuda, “Flash-based nonvolatile programmable switch for low-power and high-speed FPGA by adjacent integration of MONOS/logic and novel programming scheme,” in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, June 2014, pp. 1–2.

[32] K. J. Han, N. Chan, S. Kim, B. Leung, V. Hecht, B. Cronquist, D. Shum, A. Tilke, L. Pescini, M. Stiftinger, and R. Kakoschke, “A Novel Flash-based FPGA Technology with Deep Trench Isolation,” in *2007 22nd IEEE Non-Volatile Semiconductor Memory Workshop*, Aug 2007, pp. 32–33.

[33] T. He, F. Zhang, S. Bhunia, and P. X. . Feng, “Silicon Carbide (SiC) Nanoelectromechanical Antifuse for Ultralow-Power One-Time-Programmable (OTP) FPGA Interconnects,” *IEEE Journal of the Electron Devices Society*, vol. 3, no. 4, pp. 323–335, July 2015.

[34] Chiang, Forouhi, Chen, Hawley, McCollum, Hamdy, and Hu, “Antifuse structure comparison for field programmable gate arrays,” in *1992 International Technical Digest on Electron Devices Meeting*, Dec 1992, pp. 611–614.

[35] K. E. Gordon and R. J. Wong, “Conducting filament of the programmed metal electrode amorphous silicon antifuse,” in *Proceedings of IEEE International Electron Devices Meeting*, Dec 1993, pp. 27–30.

[36] Y. Tamura and H. Shinriki, “Most promising metal-to-metal antifuse based 10 nm-thick p-SiN/sub x/ film for high density and high speed FPGA application,” in *Proceedings of 1994 IEEE International Electron Devices Meeting*, Dec 1994, pp. 285–288.

[37] K. L. Chen, D. K. Y. Liu, G. Misium, W. M. Gosney, S. . Wang, J. Camp, and H. Tigelaar, “A sublithographic antifuse structure for field-programmable gate array applications,” *IEEE Electron Device Letters*, vol. 13, no. 1, pp. 53–55, Jan 1992.

[38] Z. Zhang, Y. Y. Liauw, C. Chen, and S. S. Wong, “Monolithic 3-D FPGAs,” *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1197–1210, July 2015.

[39] P. E. Gaillardon, D. Sacchetto, G. B. Beneventi, M. H. B. Jamaa, L. Perniola, F. Clermidy, I. O'Connor, and G. D. Micheli, "Design and Architectural Assessment of 3-D Resistive Memory Technologies in FPGAs," *IEEE Transactions on Nanotechnology*, vol. 12, no. 1, pp. 40–50, Jan 2013.

[40] S. Yu and P. Chen, "Emerging Memory Technologies: Recent Trends and Prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, Spring 2016.

[41] R. Rizk, D. Rizk, A. Kumar, and M. Bayoumi, "Demystifying Emerging Nonvolatile Memory Technologies: Understanding Advantages, Challenges, Trends, and Novel Applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[42] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, July 2008.

[43] J. Zhu, "Magnetoresistive Random Access Memory: The Path to Competitiveness and Scalability," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1786–1798, Nov 2008.

[44] T. Kishi, H. Yoda, T. Kai, T. Nagase, E. Kitagawa, M. Yoshikawa, K. Nishiyama, T. Daibou, M. Nagamine, M. Amano, S. Takahashi, M. Nakayama, N. Shimomura, H. Aikawa, S. Ikegawa, S. Yuasa, K. Yakushiji, H. Kubota, A. Fukushima, M. Oogane, T. Miyazaki, and K. Ando, "Lower-current and fast switching of a perpendicular TMR for high speed and high density spin-transfer-torque MRAM," in *2008 IEEE International Electron Devices Meeting*, Dec 2008, pp. 1–4.

[45] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. A. Wolf, A. W. Ghosh, J. W. Lu, S. J. Poon, M. Stan, W. H. Butler, S. Gupta, C. K. A. Mewes, T. Mewes, and P. B. Visscher, "Advances and Future Prospects of Spin-Transfer Torque Random Access Memory," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 1873–1878, June 2010.

[46] J. Li, P. Ndai, A. Goel, S. Salahuddin, and K. Roy, "Design Paradigm for Robust Spin-Torque Transfer Magnetic RAM (STT MRAM) From Circuit/Architecture Perspective," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 12, pp. 1710–1723, Dec 2010.

[47] X. Fong, R. Venkatesan, A. Raghunathan, and K. Roy, “Non-Volatile Complementary Polarizer Spin-Transfer Torque On-Chip Caches: A Device/Circuit/Systems Perspective,” *IEEE Transactions on Magnetics*, vol. 50, no. 10, pp. 1–11, Oct 2014.

[48] E. Chen, D. Apalkov, A. Driskill-Smith, A. Khvalkovskiy, D. Lottis, K. Moon, V. Nikitin, A. Ong, X. Tang, S. Watts, R. Kawakami, M. Krounbi, S. A. Wolf, S. J. Poon, J. W. Lu, A. W. Ghosh, M. Stan, W. Butler, T. Mewes, S. Gupta, C. K. A. Mewes, P. B. Visscher, and R. A. Lukaszew, “Progress and Prospects of Spin Transfer Torque Random Access Memory,” *IEEE Transactions on Magnetics*, vol. 48, no. 11, pp. 3025–3030, Nov 2012.

[49] D. Apalkov, S. Watts, A. Driskill-Smith, E. Chen, Z. Diao, and V. Nikitin, “Comparison of Scaling of In-Plane and Perpendicular Spin Transfer Switching Technologies by Micromagnetic Simulation,” *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 2240–2243, June 2010.

[50] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, “Phase Change Memory,” *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[51] L. Wu, X. Zhou, Z. Song, M. Zhu, Y. Cheng, F. Rao, S. Song, B. Liu, and S. Feng, “Sb-rich SiSbTe Phase-Change Material for Phase-Change Random Access Memory Applications,” *IEEE Transactions on Electron Devices*, vol. 58, no. 12, pp. 4423–4426, Dec 2011.

[52] C. D. Wright, L. Wang, P. Shah, M. M. Aziz, E. Varesi, R. Bez, M. Moroni, and F. Cazzaniga, “The Design of Rewritable Ultrahigh Density Scanning-Probe Phase-Change Memories,” *IEEE Transactions on Nanotechnology*, vol. 10, no. 4, pp. 900–912, July 2011.

[53] S. W. Fong, C. M. Neumann, and H. . P. Wong, “Phase-Change Memory —Towards a Storage-Class Memory,” *IEEE Transactions on Electron Devices*, vol. 64, no. 11, pp. 4374–4385, Nov 2017.

[54] A. Pirovano, A. L. Lacaita, A. Benvenuti, F. Pellizzer, and R. Bez, “Electronic switching in phase-change memories,” *IEEE Transactions on Electron Devices*, vol. 51, no. 3, pp. 452–459, March 2004.

[55] A. Pirovano, A. L. Lacaita, F. Pellizzer, S. A. Kostylev, A. Benvenuti, and R. Bez, “Low-field amorphous state resistance and threshold voltage drift in chalcogenide materials,” *IEEE Transactions on Electron Devices*, vol. 51, no. 5, pp. 714–719, May 2004.

[56] H. S. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, “MetalOxide RRAM,” *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, June 2012.

[57] B. Traore, P. Blaise, E. Vianello, H. Grampeix, S. Jeannot, L. Perniola, B. De Salvo, and Y. Nishi, “On the Origin of Low-Resistance State Retention Failure in HfO2-Based RRAM and Impact of Doping/Alloying,” *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4029–4036, Dec 2015.

[58] D. Ielmini, “Modeling the Universal Set/Reset Characteristics of Bipolar RRAM by Field- and Temperature-Driven Filament Growth,” *IEEE Transactions on Electron Devices*, vol. 58, no. 12, pp. 4309–4317, Dec 2011.

[59] U. Russo, D. Ielmini, C. Cagli, and A. L. Lacaita, “Filament Conduction and Reset Mechanism in NiO-Based Resistive-Switching Memory (RRAM) Devices,” *IEEE Transactions on Electron Devices*, vol. 56, no. 2, pp. 186–192, Feb 2009.

[60] M. Bocquet, D. Deleruyelle, H. Aziza, C. Muller, J. Portal, T. Cabout, and E. Jalaguier, “Robust Compact Model for Bipolar Oxide-Based Resistive Switching Memories,” *IEEE Transactions on Electron Devices*, vol. 61, no. 3, pp. 674–681, March 2014.

[61] J. Guy, G. Molas, P. Blaise, M. Bernard, A. Roule, G. Le Carval, V. Delaye, A. Toffoli, G. Ghibaudo, F. Clermidy, B. De Salvo, and L. Perniola, “Investigation of Forming, SET, and Data Retention of Conductive-Bridge Random-Access Memory for Stack Optimization,” *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3482–3489, Nov 2015.

[62] S. Yu and H. . P. Wong, “Compact Modeling of Conducting-Bridge Random-Access Memory (CBRAM),” *IEEE Transactions on Electron Devices*, vol. 58, no. 5, pp. 1352–1360, May 2011.

[63] T. Tsai, F. Jiang, C. Ho, C. Lin, and T. Tseng, “Enhanced Properties in Conductive-Bridge Resistive Switching Memory With Oxide-Nitride Bilayer Structure,” *IEEE Electron Device Letters*, vol. 37, no. 10, pp. 1284–1287, Oct 2016.

[64] S. Lv, H. Wang, J. Zhang, J. Liu, L. Sun, and Z. Y. Life, “An Analytical Model for the Forming Process of Conductive-Bridge Resistive-Switching Random-Access Memory,” *IEEE Transactions on Electron Devices*, vol. 61, no. 9, pp. 3166–3171, Sep. 2014.

[65] Z. Dong, H. Zhao, D. DiMarzio, M. Han, L. Zhang, J. Tice, H. Wang, and J. Guo, “Atomically Thin CBRAM Enabled by 2-D Materials: Scaling Behaviors and Performance Limits,” *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4160–4166, Oct 2018.

[66] X. Xue, J. Yang, Y. Lin, R. Huang, Q. Zou, and J. Wu, “Low-Power Variation-Tolerant Nonvolatile Lookup Table Design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 1174–1178, March 2016.

[67] K. Huang, Y. Ha, R. Zhao, A. Kumar, and Y. Lian, “A Low Active Leakage and High Reliability Phase Change Memory (PCM) Based Non-Volatile FPGA Storage Element,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 9, pp. 2605–2613, Sep. 2014.

[68] L. Ju, X. Sui, S. Li, M. Zhao, C. J. Xue, J. Hu, and Z. Jia, “NVM-Based FPGA Block RAM With Adaptive SLC-MLC Conversion,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2661–2672, Nov 2018.

[69] Z. Yuan, Y. Liu, J. Li, J. Hu, C. J. Xue, and H. Yang, “CP-FPGA: Energy-Efficient Nonvolatile FPGA With Offline/Online Checkpointing Optimization,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 7, pp. 2153–2163, July 2017.

[70] B. Khaleghi and H. Asadi, “A Resistive RAM-Based FPGA Architecture Equipped With Efficient Programming Circuitry,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 7, pp. 2196–2209, July 2018.

[71] M. Miyamura, T. Sakamoto, M. Tada, N. Banno, K. Okamoto, N. Iguchi, and H. Hada, “Low-power programmable-logic cell arrays using nonvolatile complementary atom switch,” in *International Symposium on Quality Electronic Design (ISQED)*, March 2014, pp. 330–334.

[72] N. Banno, M. Tada, K. Okamoto, N. Iguchi, T. Sakamoto, M. Miyamura, Y. Tsuji, H. Hada, H. Ochi, H. Onodera, M. Hashimoto, and T. Sugibayashi, “A novel two-varistors (a-Si/SiN/a-Si) selected complementary atom switch (2V-1CAS) for nonvolatile crossbar switch with multiple fan-outs,” in *International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 2.5.1–2.5.4.

[73] N. Banno, M. Tilda, K. Okamoto, N. Iguchi, T. Sakamoto, H. Hada, H. Ochi, H. Onodera, M. Hashimoto, and T. Sugibayashi, “50x20 crossbar

switch block (CSB) with two-varistors (a-Si/SiN/a-Si) selected complementary atom switch for a highly-dense reconfigurable logic," in *International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 16.4.1–16.4.4.

[74] N. Banno, K. Okamoto, N. Iguchi, H. Ochi, H. Onodera, M. Hashimoto, T. Sugibayashi, T. Sakamoto, and M. Tada, "Low-Power Crossbar Switch with Two-Varistors Selected Complementary Atom Switch (2V-1CAS; Via-Switch) for Nonvolatile FPGA," *IEEE Transactions on Electron Devices*, vol. 66, no. 8, pp. 3331–3336, Aug 2019.

[75] K. Okamoto, M. Tada, T. Sakamoto, M. Miyamura, N. Banno, N. Iguchi, and H. Hada, "Conducting mechanism of atom switch with polymer solid-electrolyte," in *International Electron Devices Meeting (IEDM)*, Dec 2011, pp. 12.2.1–12.2.4.

[76] T. Sakamoto, M. Tada, K. Okamoto, and H. Hada, "Electronic Conduction Mechanism in Atom Switch Using Polymer Solid Electrolyte," *IEEE Transactions on Electron Devices*, vol. 59, no. 12, pp. 3574–3577, Dec 2012.

[77] M. Tada, T. Sakamoto, N. Banno, K. Okamoto, N. Iguchi, H. Hada, and M. Miyamura, "Improved ON-State Reliability of Atom Switch Using Alloy Electrodes," *IEEE Transactions on Electron Devices*, vol. 60, no. 10, pp. 3534–3540, Oct 2013.

[78] M. Tada, K. Okamoto, T. Sakamoto, and H. Hada, "ON-state Reliability of Cu Atom Switch Under CurrentTemperature Stress," *IEEE Transactions on Electron Devices*, vol. 62, no. 9, pp. 2992–2997, Sep. 2015.

[79] M. Tada and T. Sakamoto, "Set/Reset Switching Model of Cu Atom Switch Based on Electrolysis," *IEEE Transactions on Electron Devices*, vol. 64, no. 4, pp. 1812–1817, April 2017.

[80] H. Ochi, K. Yamaguchi, T. Fujimoto, J. Hotate, T. Kishimoto, T. Higashi, T. Imagawa, R. Doi, M. Tada, T. Sugibayashi, W. Takahashi, K. Wakabayashi, H. Onodera, Y. Mitsuyama, J. Yu, and M. Hashimoto, "Via-Switch FPGA: Highly Dense Mixed-Grained Reconfigurable Architecture With Overlay Via-Switch Crossbars," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2723–2736, Dec 2018.

[81] N. Banno, M. Tada, T. Sakamoto, K. Okamoto, M. Miyamura, N. Iguchi, T. Nohisa, and H. Hada, "Nonvolatile 32 × 32 crossbar atom switch block

integrated on a 65-nm CMOS platform," in *2012 Symposium on VLSI Technology (VLSIT)*, June 2012, pp. 39–40.

[82] K. Okamoto, M. Tada, N. Banno, T. Sakamoto, M. Miyamura, N. Iguchi, T. Nohisa, and H. Hada, "Bidirectional TaO-diode-selected, complementary atom switch (DCAS) for area-efficient, nonvolatile crossbar switch block," in *2013 Symposium on VLSI Technology*, June 2013, pp. T242–T243.

[83] M. Tada, K. Okamoto, N. Banno, T. Sakamoto, and H. Hada, "Three-Terminal Nonvolatile Resistive-Change Device Integrated in Cu-BEOL," *IEEE Transactions on Electron Devices*, vol. 61, no. 2, pp. 505–510, Feb 2014.

[84] K. Okamoto, M. Tada, N. Banno, N. Iguchi, H. Hada, T. Sakamoto, M. Miyamura, Y. Tsuji, R. Nebashi, A. Morioka, X. Bai, and T. Sugibayashi, "Robust Cu atom switch with over-400 ° C thermally tolerant polymer-solid electrolyte (TT-PSE) for nonvolatile programmable logic," in *2016 IEEE Symposium on VLSI Technology*, June 2016, pp. 1–2.

[85] M. Tada, T. Sakamoto, M. Miyamura, N. Banno, K. Okamoto, N. Iguchi, T. Nohisa, and H. Hada, "Highly reliable, complementary atom switch (CAS) with low programming voltage embedded in Cu BEOL for Nonvolatile Programmable Logic," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 30.2.1–30.2.4.

[86] M. Tada, T. Sakamoto, M. Miyamura, N. Banno, K. Okamoto, N. Iguchi, and H. Hada, "Improved Off-State Reliability of Nonvolatile Resistive Switch With Low Programming Voltage," *IEEE Transactions on Electron Devices*, vol. 59, no. 9, pp. 2357–2362, Sep. 2012.

[87] N. Banno, M. Tada, T. Sakamoto, K. Okamoto, M. Miyamura, N. Iguchi, and H. Hada, "Improved Switching Voltage Variation of Cu Atom Switch for Nonvolatile Programmable Logic," *IEEE Transactions on Electron Devices*, vol. 61, no. 11, pp. 3827–3832, Nov 2014.

[88] N. Banno, M. Tada, T. Sakamoto, M. Miyamura, K. Okamoto, N. Iguchi, T. Nohisa, and H. Hada, "A fast and low-voltage Cu complementary-atom-switch 1Mb array with high-temperature retention," in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, June 2014, pp. 1–2.

[89] N. Banno, M. Tada, T. Sakamoto, M. Miyamura, K. Okamoto, N. Iguchi, and H. Hada, "Cu Atom Switch With Steep Time-to-ON-State Versus

Switching Voltage Using Cu Ionization Control," *IEEE Transactions on Electron Devices*, vol. 62, no. 9, pp. 2966–2971, Sep. 2015.

[90] R. Nebashi, N. Banno, M. Miyamura, Y. Tsuji, A. Morioka, X. Bai, K. Okamoto, N. Iguchi, H. Numata, H. Hada, T. Sugibayashi, T. Sakamoto, and M. Tada, "High-Density and Fault-Tolerant Cu Atom Switch Technology Toward 28nm-node Nonvolatile Programmable Logic," in *2018 IEEE Symposium on VLSI Technology*, June 2018, pp. 127–128.

[91] T. Sakamoto, Y. Tsuji, X. Bai, M. Miyamura, A. Morioka, R. Nebashi, N. Banno, K. Okamoto, N. Iguchi, H. Hada, T. Sugibayashi, and M. Tada, "Atom Switch with Improved Cycle Endurance using Field Enhancement for Nonvolatile SoC," in *2018 IEEE International Memory Workshop (IMW)*, May 2018, pp. 1–4.

[92] X. Bai, T. Sakamoto, M. Tada, M. Miyamura, Y. Tsuji, A. Morioka, R. Nebashi, N. Banno, K. Okamoto, N. Iguchi, H. Hada, and T. Sugibayashi, "A low-power Cu atom switch programmable logic fabricated in a 40nm-node CMOS technology," in *2017 Symposium on VLSI Technology*, June 2017, pp. T28–T29.

[93] T. Higashi and H. Ochi, "Area-Efficient LUT-Like Programmable Logic Using Atom Switch and Its Delay-Optimal Mapping Algorithm," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100.A, no. 7, pp. 1418–1426, 2017.

[94] T. Kishimoto, W. Takahashi, K. Wakabayashi, and H. Ochi, "Range Limiter Using Connection Bounding Box for SA-Based Placement of Mixed-Grained Reconfigurable Architecture," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99.A, no. 12, pp. 2328–2334, 2016.

[95] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (FPT)*, Dec 2004, pp. 41–48.

[96] Y. Yamamoto, H. Makiyama, H. Shinohara, T. Iwamatsu, H. Oda, S. Kamohara, N. Sugii, Y. Yamaguchi, T. Mizutani, and T. Hiramoto, "Ultralow-voltage operation of Silicon-on-Thin-BOX (SOTB) 2Mbit SRAM down to 0.37 V utilizing adaptive back bias," in *2013 Symposium on VLSI Circuits*, June 2013, pp. T212–T213.

[97] D. Stroobandt, *A Priori Wire Length Estimates for Digital Design*. Springer US, 2001.

[98] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Kluwer Academic Publishers, 2004.

[99] R. Doi, X. Bai, T. Sakamoto, and M. Hashimoto, "Fault Diagnosis of Via-Switch Crossbar in Non-volatile FPGA," in *Design, Automation, and Test in Europe Conference and Exhibition (DATE)*, (accepted).

[100] R. Doi, J. Yu, and M. Hashimoto, "Sneak Path Free Reconfiguration of Via-switch Crossbars Based FPGA," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

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

[102] M. Zangeneh and A. Joshi, "Design and Optimization of Nonvolatile Multibit 1T1R Resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1815–1828, Aug 2014.

[103] M. Lee, Y. Park, B. Kang, S. Ahn, C. Lee, K. Kim, W. Xianyu, G. Stefanovich, J. Lee, S. Chung, Y. Kim, C. Lee, J. Park, I. Baek, and I. Yoo, "2-stack 1D-1R Cross-point Structure with Oxide Diodes as Switch Elements for High Density Resistance RAM Applications," in *International Electron Devices Meeting (IEDM)*, Dec 2007, pp. 771–774.

[104] C. Jung, J. Choi, and K. Min, "Two-Step Write Scheme for Reducing Sneak-Path Leakage in Complementary Memristor Array," *IEEE Transactions on Nanotechnology*, vol. 11, no. 3, pp. 611–618, May 2012.

[105] S. Ham, H. Mo, and K. Min, "Low-Power $V_{\mathbf{DD}}/3$ Write Scheme With Inversion Coding Circuit for Complementary Memristor Array," *IEEE Transactions on Nanotechnology*, vol. 12, no. 5, pp. 851–857, Sep. 2013.

[106] Y. Yang, J. Mathew, M. Ottavi, S. Pontarelli, and D. K. Pradhan, "Novel Complementary Resistive Switch Crossbar Memory Write and Read Schemes," *IEEE Transactions on Nanotechnology*, vol. 14, no. 2, pp. 346–357, March 2015.

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

[108] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. H. Fahmy, and K. N. Salama, "Memristor Multiport Readout: A Closed-Form Solution for Sneak Paths," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 274–282, March 2014.

[109] Y. Deng, P. Huang, B. Chen, X. Yang, B. Gao, J. Wang, L. Zeng, G. Du, J. Kang, and X. Liu, "RRAM Crossbar Array With Cell Selection Device: A Device and Circuit Interaction Study," *IEEE Transactions on Electron Devices*, vol. 60, no. 2, pp. 719–726, Feb 2013.

[110] S. Kim, J. Zhou, and W. D. Lu, "Crossbar RRAM Arrays: Selector Device Requirements During Write Operation," *IEEE Transactions on Electron Devices*, vol. 61, no. 8, pp. 2820–2826, Aug 2014.

[111] I. Gupta, A. Serb, R. Berdan, A. Khiat, A. Regoutz, and T. Prodromakis, "A Cell Classifier for RRAM Process Development," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 62, no. 7, pp. 676–680, July 2015.

[112] W. Banerjee, N. Lu, Y. Yang, L. Li, H. Lv, Q. Liu, S. Long, and M. Liu, "Investigation of Retention Behavior of TiOx/Al2O3 Resistive Memory and Its Failure Mechanism Based on MeyerNeldel Rule," *IEEE Transactions on Electron Devices*, vol. 65, no. 3, pp. 957–962, March 2018.

[113] R. Doi, J. Yu, and M. Hashimoto, "Sneak Path Free Reconfiguration with Minimized Programming Steps for Via-switch Crossbar Based FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (accepted).

[114] C. E. Stroud, *A Designer's Guide to Built-In Self-Test*. Kluwer Academic Publishers, 2002.

[115] S. Dutt, V. Verma, and V. Suthar, "Built-in-Self-Test of FPGAs With Provable Diagnosabilities and High Diagnostic Coverage With Application to Online Testing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 2, pp. 309–326, Feb 2008.

[116] C. Hsu and T. Chen, "Built-in Self-Test Design for Fault Detection and Fault Diagnosis in SRAM-Based FPGA," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 7, pp. 2300–2315, July 2009.

[117] M. Abramovici, C. E. Stroud, and J. M. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks," *IEEE Transactions on Very*

Large Scale Integration (VLSI) Systems, vol. 12, no. 12, pp. 1284–1294, Dec 2004.

[118] M. Abramovici and C. E. Stroud, "BIST-based test and diagnosis of FPGA logic blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 159–172, Feb 2001.

[119] I. G. Harris and R. Tessier, "Testing and diagnosis of interconnect faults in cluster-based FPGA architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1337–1343, Nov 2002.

[120] M. Hashimoto, X. Bai, N. Banno, M. Tada, T. Sakamoto, J. Yu, R. Doi, Y. Araki, H. Onodera, T. Imagawa, H. Ochi, K. Wakabayashi, Y. Mitsuyama, and T. Sugibayashi, "Via-switch FPGA: First Implementation in 65-nm CMOS and Architecture Extension for AI Applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, (accepted).