University of California

Los Angeles

# FPGA-RR: A Novel FPGA Architecture with RRAM-Based Reconfigurable Interconnects

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering

by

**Bingjun Xiao**

2012

ABSTRACT OF THE THESIS

# FPGA-RR: A Novel FPGA Architecture with RRAM-Based Reconfigurable Interconnects

by

**Bingjun Xiao**

Master of Science in Electrical Engineering

University of California, Los Angeles, 2012

Professor JINGSHENG JASON CONG, Chair

In this paper we introduce a novel FPGA architecture with RRAM-based reconfiguration (FPGA-RR). This architecture focuses on the redesign of programmable interconnects, the dominant part of FPGA. By renovating the routing structure of FPGA using RRAMs, the architecture achieves significant benefits concerning area, performance and energy consumption. The implementation of FPGA-RR can be realized by the existing CMOS-compatible RRAM fabrication process. A customized CAD flow is provided for FPGA-RR, with an advanced P&R tool named VPR-RR developed for FPGA-RR to deal with its novel routing structure. We use the flow to verify the benefits of area, performance and power of FPGA-RR over the 20 largest MCNC benchmark circuits. Results show that FPGA-RR achieves 6.82x area savings, 3.09x speedup and 4.33x energy savings.

The thesis of Bingjun Xiao is approved.

KANG LUNG WANG

MAU-CHUNG FRANK CHANG

JINGSHENG JASON CONG, Committee Chair

University of California, Los Angeles

2012

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

## ACKNOWLEDGMENTS

Part of this work appears in Proceedings of the 7th IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 2011), with copyright @ 2011 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Here I would like to offer my sincerest gratitude to my advisor, Prof. Jason Cong. Prof. Cong is a Chancellor's Professor at the Computer Science Department and Electrical Engineering Departement of UCLA, Director of Center for Customizable Domain-Specific Computing, and Co-Director of UCLA VLSI CAD Laboratory. His insightful instructions helped me throughout the time of my research and writing of this thesis.

The work in this thesis is partially supported by the Center for Domain-Specific Computing (CDSC) which is funded by the NSF Expedition in Computing Award CCF-0926127, and grants from Altera Corporation and Xilinx Inc.

# CHAPTER 1

# Introduction

The performance/power efficiency of an application implemented in an application specific integrated circuit (ASIC) can be as much as six orders of magnitude higher than its counterpart coded in CPU [SV03]. However the rapid increase of non-recurring engineering cost and design cycle of an ASIC in nanometer technologies makes it impractical to implement most applications in ASIC. As a result, the field programmable gate arrays (FPGAs) have become increasingly more popular. An FPGA can be reconfigured to realize a large range of arbitrary functions according to customer demands. The programmability makes FPGA a quick and reusable hardware implementation platform for specific applications. Compared to ASIC, FPGA has sacrificed its performance to some extent for programmability [KR07], but FPGA can still have orders of magnitude improvement in performance/power efficiency compared to CPU [SV03, CSR11]. The flexibility and performance of FPGA compared to ASIC and CPU makes FPGA an important component in the realm of customizable heterogeneous computing platforms [CSR11].

The study in [KR07] measures the gap between FPGA and ASIC. It shows that the area, delay and power consumption of FPGA are $\sim$21 times, $\sim$4 times and $\sim$12 times as much as those of ASIC, respectively. The programmable routing structure in FPGA is the principal source of FPGA performance inferiority when compared to ASIC [Geo00, DeH96, AR04, LLH05, CWM10]. It is reported that the programmable interconnects in FPGA can account for up to $90\%$ of the total area [Geo00, CWM10], up to $80\%$ of the total delay [DeH96, AR04, CWM10] and up to $85\%$ of the total power consumption [LLH05, CWM10]. If the FPGA routing structure gains some improvement, the gap between FPGAs and ASICs will be significantly reduced.

Emerging technologies, especially emerging non-volatile memory (NVM) technologies, lead to opportunities for circuit improvement. Popular emerging NVMs include spin-transfer torque RAM (STTRAM), phase-change RAM (PRAM), nanoelectromechanical (NEM) relay, and resistive RAM (RRAM). All of these have demonstrated CMOS compatible fabrication and can be integrated in metal layers over CMOS via a back-end-of-line (BEOL) process [KZU11, LW08, JLY08, HZG11], leading to opportunities for high-density circuit design. They also have the desirable property of non-volatility, which means that they can be turned off during stand-by to save power. In light of these new properties of NVMs, a number of novel FPGA architectures based on NVMs have been explored in the past few years [PMB08, BTS06, CZX10, GHB10, CPZ09, CWM10, TLW11].

This thesis presents a novel FPGA architecture with RRAM-based reconfiguration (FPGA-RR). We renovate FPGA programmable interconnects with the integration of RRAMs in order to conquer the limitations of conventional FPGA mentioned above. Our proposed architecture substantially reduces the gap between FPGAs and ASICs in area, delay and power.

This thesis is organized as follows. Chapter 2 reviews the conventional FPGA architecture and recent research work on FPGAs with emerging NVMs. Chapter 3 describes the architecture of FPGA-RR from high-level overview to detailed design. Chapter 4 provides a complete CAD flow and evaluation method for FPGA-RR. Chapter 5 presents detailed optimization and evaluation results of FPGA-RR using the largest 20 MCNC benchmarks. Finally we draw some conclusions in Section 6. A preliminary study of this work was presented at 2011 IEEE/ACM International Symposium on Nanoscale Architectures [CX11].

# CHAPTER 2

# Background

## 2.1 Conventional FPGA

Fig. 2.1 shows a conventional FPGA architecture. It is a typical island-based type, which is made

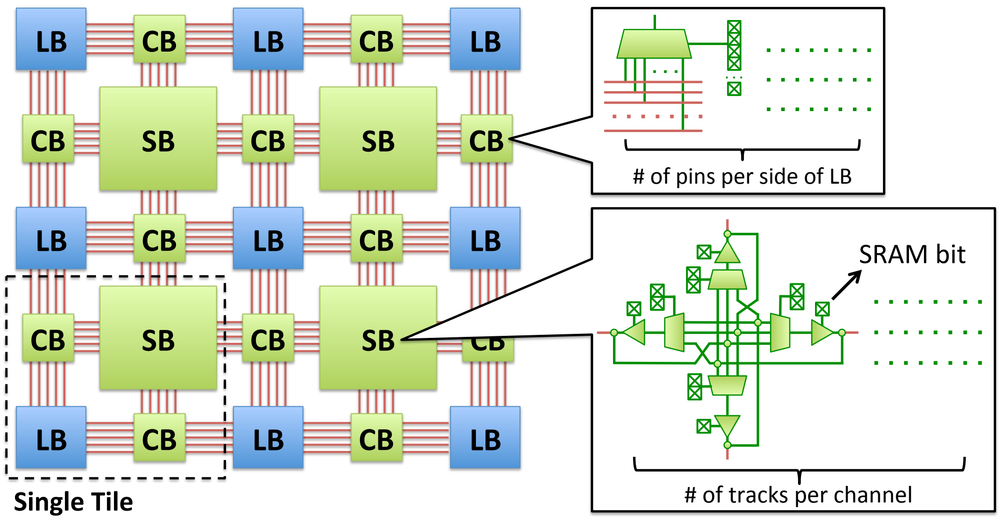


Figure 2.1: Conventional FPGA architecture.

up of an array of tiles. Each tile consists of one logic block (LB), two connection blocks (CB) and one switch block (SB). Each logic block contains a cluster of basic logic elements (BLEs), typically look-up tables (LUTs), to provide customizable logic functions. Each LB also contains local routing multiplexers (MUXs) to provide connection among BLEs. LBs are connected to the routing channels through CBs, and the segmented routing channels are connected with each other through SBs. Fig. 2.1 also depicts two typical circuit designs of CBs and SBs based on MUXs and buffers described in [LL02]. The selector pins of each MUX are connected to a group of 6-

transistor SRAM cells to have their connectivity determined. Note that the circuits presented in Fig. 2.1 have copies of up to the number of pins per side of LBs in a single CB, and up to the number of tracks per channel in a single SB. We see that CBs and SBs make up the interconnects of FPGAs with much larger area and higher complexity compared to the direct interconnects of ASIC.

The FPGA routing structure is usually divided into three components: SRAM-based programming bits, MUX-based routing switches, and buffers. All of these three components are non-trivial parts in FPGA, as shown in Fig. 2.2. Though programmable interconnects in FPGA have been

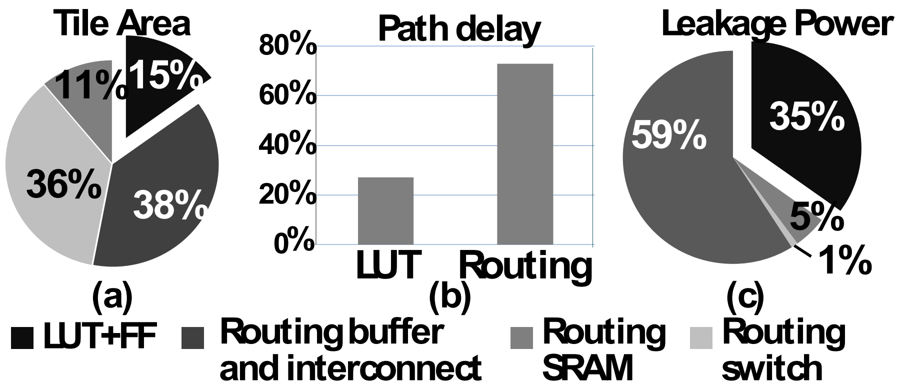


Figure 2.2: Breakdown of different components in FPGA [CWM10]. Area is dominated by programmable interconnects. All three components (SRAM bits, routing switches and routing buffers) take up significant area.

optimized within the design space of CMOS technology for years, there are fundamental weak points in all three components:

1. One SRAM cell contains as many as six transistors, but can store only one-bit data. The low density of SRAM-based storage increases the area overhead of FPGA programmability. SRAM is also a volatile memory — this means that it contributes to excessive power consumption during standby.

2. The MUX-based routing switches also have large area overhead. Fig. 2.3 shows three typical designs of a MUX. For a 16-to-1 MUX, a one-stage MUX needs 16 SRAM cells, but a

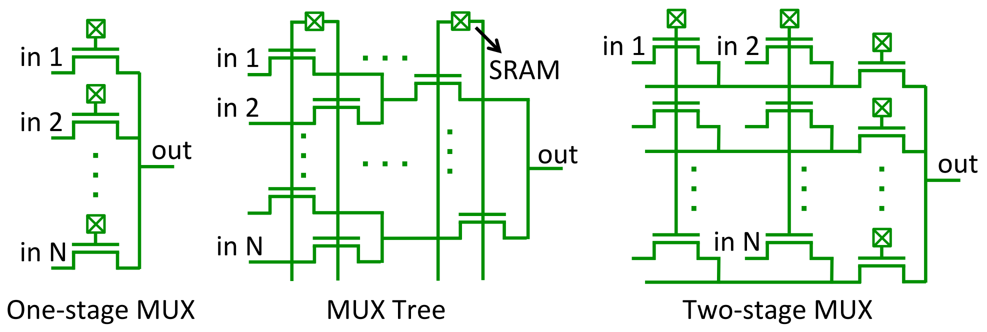


Figure 2.3: Different implementations of a MUX-based routing switch in FPGA, with trade-off between MUX area and number of SRAM bits.

MUX tree needs only four. Though a MUX tree avoids the large overhead of SRAM cells, it contains four serial pass transistors in its path from input to output. Each pass transistor has to be even larger so as to provide sufficient drive. The total number of pass transistors used by a MUX tree is also two times of the one-stage MUX. As a trade-off between SRAM cells and pass transistors, the two-stage MUX shows the minimum area-delay product [CWM10].

3. Routing buffers are extensively fabricated and used in the programmable interconnects of conventional FPGA. Only a small part of them are truly necessary to achieve the optimal timing result. The excessive buffers originate from their fixed locations in the routing network, as shown in Fig. 2.1. The buffer placement cannot be optimized according to a given design implemention on FPGA; it can only be optimized for a general situation. Since we are not sure whether a routing track needs a buffer due to its being in a critical path or a long unbuffered segment, buffers are placed at every terminal of routing track segments for the sake of safety. The overuse of buffers not only increases area costs, but also leads to worse timing results since the intrinsic delay of overused buffers may cancel the benefit brought by their buffering effect.

In summary, these problems contribute to the significant gap between FPGA and ASIC.

## 2.2 FPGAs with Emerging NVMs

With the recent development of emerging non-volatile memory (NVM) technologies, a number of novel FPGA architectures based on those technologies have been proposed in the past few years. NVMs are used to replace some FPGA components to obtain the property of non-volatility and to save area by placing them over CMOS transistors.

### 2.2.1 Replace SRAMs with NVMs

In [PMB08, BTS06, CZX10, CPZ09, TLW11], the SRAM-based programmable bits are moved to STTRAMs, PRAMs, NEM relays and RRAMs respectively, as shown in Fig. 2.4a. Note that in their works, emerging NVMs are used to store '0' or '1' in forms other than voltage levels. So they still need CMOS transistors as sensing circuits to recover voltage signals. In [PMB08, BTS06], the authors give an estimation of the number of transistors, which is two transistors for one bit in a LUT and five transistors for one bit in the routing structure. This cost is still large, especially for the routing structure. That's because each routing switch is connected to the output of the memory cell to store its programming bit, and thus a 4-transistor sense amplifier is needed at every output and cannot be shared among memory cells.

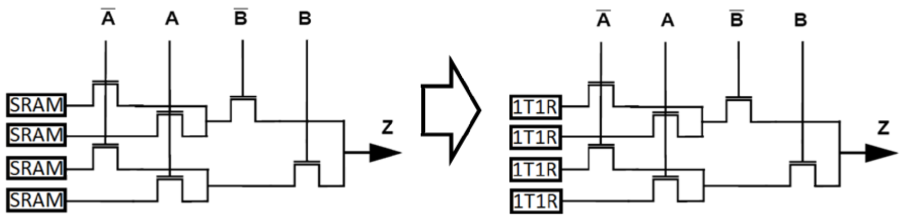
### 2.2.2 Use NVMs as Programmable Switches

In [GHB10, CWM10, TLW11], emerging NVMs, including PRAMs, NEM relays and RRAMs, are used more aggressively as programmable switches in place of SRAM-based pass transistors in conventional FPGA, as shown in Fig. 2.4b. This kind of use is enabled by a common property of these emerging NVMs. That is, the connection between two terminals of these devices can be programmed to turn on or turn off. By applying specific programming voltages, the resistance between the two terminals can be switched between high resistance state (HRS) and low resistance state (LRS). The programmed resistance value can be kept either under operating voltages or without supply voltage due to non-volatility. This kind of NVM use saves the area of not only the sensing circuits but also the pass transistors which build routing switches. Since there are no SRAMs needed by NVM-based routing switches, the one-stage MUX becomes the optimal one among the implementations in Fig. 2.3. The use of RRAM in our work belongs to this category.

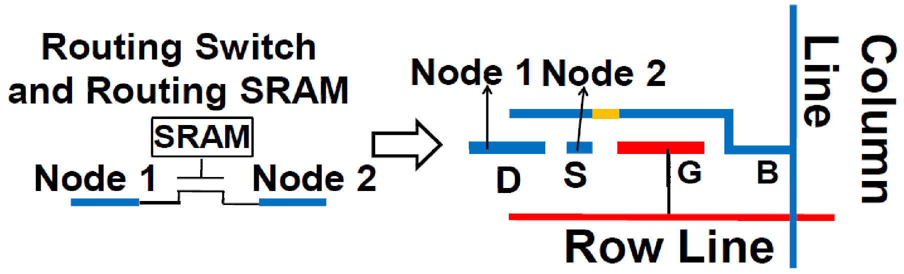
There are still problems to solve for FPGAs with NVMs as routing switches. One of the key problems is the programmability of NVMs integrated in interconnects. Since the programming of some of these NVMs, such as PRAMs, 3T NEM relays and RRAMs, share the same terminals with the signal path when they are used as routing switches, the programming circuits need careful design. Otherwise there will be interference between the two operation modes. A naive programming schematic was proposed in [TLW11]. It needs two programming transistors for each RRAM, one for each terminal. This kind of design undermines the great area benefit brought by RRAMs, which do not occupy CMOS area itself. Besides the programming transistors, there are other programming issues yet to be investigated, such as how to improve the buffering solution of FPGA in light of NVMs. The works referenced above all focus on replacement of routing switches with NVMs and show some benefits. However, as the author of [CWM10] concludes, routing buffers become a bottleneck after this replacement, and further exploration of FPGA architecture is needed on buffers.

### 2.2.3 Integrate NVMs with Nanowire Crossbars

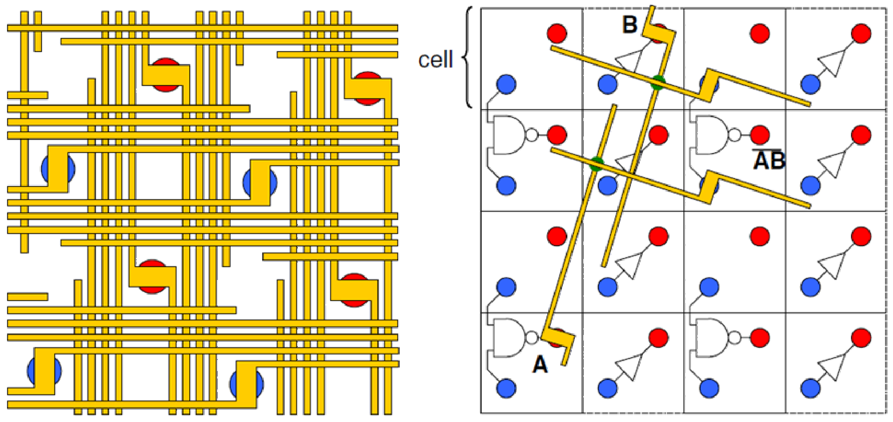
FPNI [SW07,XRC09] is an attempt to further introduce nanowire crossbars to FPGA. As shown in Fig. 2.4c, the routing structure is implemented by nanowire crossbars and RRAM cross-points over CMOS logics. The structure is quite different from the typical island-based FPGA architecture. It shows significant area reduction but requires the feature size of nanowire crossbars to be much smaller than that of CMOS logics to achieve higher RRAM density. In addition, the connection between two cells always has to drive at least two long nanowires in its path, even if the two cells are adjacent. Due to the large capacitance of these long nanowires and fine granularity of logic cells, the FPGA performance decreases by 30% as reported in [SW07], and the dynamic power increases by 17.5% as reported in [DCH07]. The high density of programmable devices in these works may also lead to high leakage power. Furthermore, FPNI fabrication is more complex since it needs integration of three technologies: CMOS, RRAM and nanowire. There are extra technical problems, such as broken nanowires, alignment of nanowire crossbars with CMOS pins, etc., yet to be solved to realize FPNI for practical use. Our work will show that actually we can achieve similar area reduction without high density of programmable devices.



(a) Replace SRAMs with emerging NVMs [TLW11].



(b) Use NVMs as programmable switches [CWM10].



(c) Integration of CMOS, NVMs and nanowire in FPNI [XRC09].

Figure 2.4: FPGA with emerging non-volatile memories.

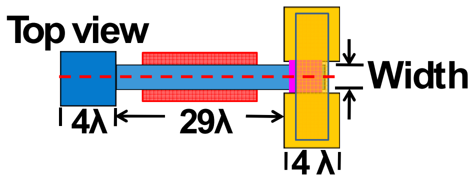
# CHAPTER 3

# The FPGA-RR Architecture

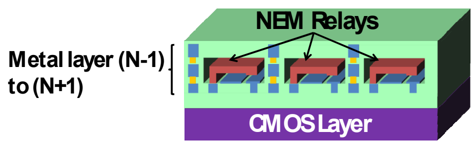
In this paper, we focus on only the renovation of an FPGA routing structure, which is the dominant component in FPGA. There are already many works that propose logic blocks with emerging technologies [PMB08, BTS06, CZX10, CPZ09, TLW11]. Our work can build on these studies and also provide further improvement. As analyzed in Section 2, using NVMs as routing switches is clearly a good idea. Although there are still problems with this method, we will show that they can be solved by renovating FPGA architecture to cater to NVM properties.

## 3.1 Choice of NVMs

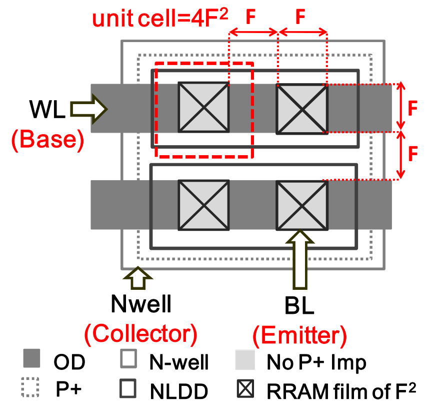
First, we need to decide which type of NVM to use in our work. Different NVMs have their own merits/shortcomings as programmable switches. For example, STTRAM usually has an on/off ratio below 10. This ratio is sufficient for memory application but far from enough for a routing switch. RRAM can have an on/off ratio as high as $10^6$ which is comparable to that of MOSFET [GLL08]. NEM relay has the highest on/off ratio since it shows almost zero off current [LW08]. However, this device has a complex structure, as shown in Fig. 3.1a and Fig. 3.1b. It has a large fabrication size of $92.5F^2$ ($F$ is the feature size) and occupies three metal layers [CWM10]. The area of NEM relays can easily exceed that of CMOS transistors below them, especially in cases where the area of an FPGA tile is much reduced by emerging technologies. In contrast, other NVMs, like STTRAMs, PRAMs and NVMs, have simple junction-like structures and thus have a much smaller size. For example, one RRAM device can be limited within $F^2$ area and one metal layer as shown in Fig. 3.1c and Fig. 3.1d [WTL10]. PRAM also has properties as good as RRAM.



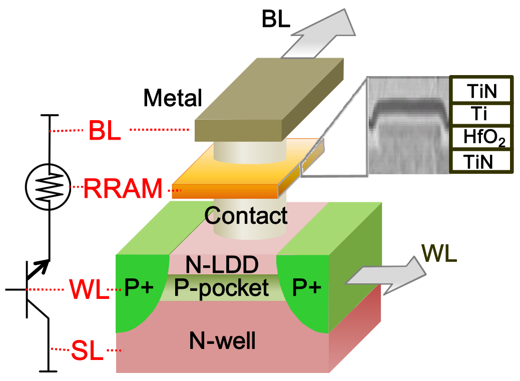
(a) Top view of an NEM relay.



(b) NEM relay with CMOS.



(c) Top view of of a 2x2 RRAM array.



(d) RRAM with CMOS.

Figure 3.1: Different cell sizes of two emerging non-volatile memories — NVM relay [CWM10] and RRAM [WTL10].

However, the programming of PRAM is harder to control than RRAM. PRAM needs to be kept at a very high temperature for a very short time to have a reset operation, and needs to be kept at a moderately high temperature for a long time to have a set operation [LW08]. In contrast, RRAM has a bipolar programming mechanism, i.e., it can be set/reset by applying high positive/negative voltages [GLL08,WTL10]. Therefore we choose RRAM to act as the routing switches in our work.

## 3.2 Overall Architecture

In our FPGA-RR, routing buffers are first separated from the FPGA routing structure. By doing so, only SRAM-based programming bits and MUX-based routing switches are left in the routing structure. They can be built by RRAMs and metal wires alone. Then both connection blocks and switch blocks are free of transistors and can be placed over logic blocks, as shown in Fig. 3.2.

FPGA-RR will become a highly compact array, as shown in Fig. 3.3a. The area of FPGA-RR is

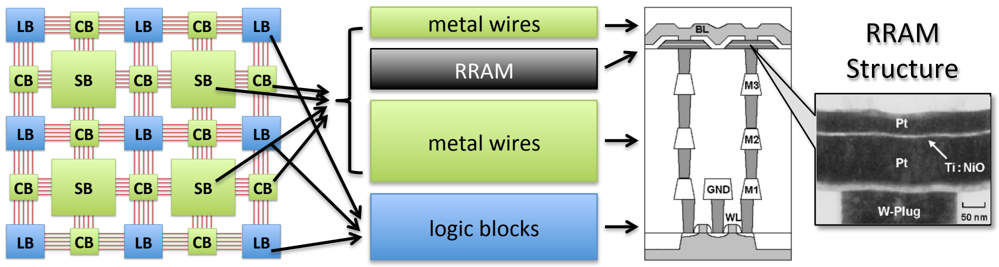


Figure 3.2: In FPGA-RR, switch blocks and connection blocks are placed over logic blocks in the same die according to existing RRAM fabrication structures [TKN07].

almost solely determined by logic blocks, which take only $10\%$ to $20\%$ of the conventional FPGA area [Geo00, CWM10]. There are also programming transistors for RRAMs and routing buffers to be attached to the FPGA-RR routing structure. We will show that, for this additional structure, the area of both programming transistors and routing buffers is much smaller than that of logic blocks.

## 3.3 Layout Design

Since SBs and CBs are now placed over CMOS transistors using RRAMs and metal wires, their layout cannot be borrowed from conventional FPGA experience; it needs to be redesigned. Most of the designers' attention should be put on the consistency of the relative positions of RRAM and metal layers with existing RRAM fabrication technologies. Fig. 3.3b is a detailed design of the connection blocks and switch blocks in FPGA-RR using RRAMs and metal wires only. It addresses some design issues as follows:

1. In this design we use five metal, layers M5 to M9, solely for interconnects. The circuits of logic blocks in FPGA-RR will use the metal layers, i.e., M1 to M4, which are sufficient for practical FPGAs, e.g. Virtex-6 [Xil].

2. To ease FPGA-RR fabrication, the RRAM layer is designed to be located between the M9 and M8 layers. It is close to the top, which is the same as the RRAM fabrication structure

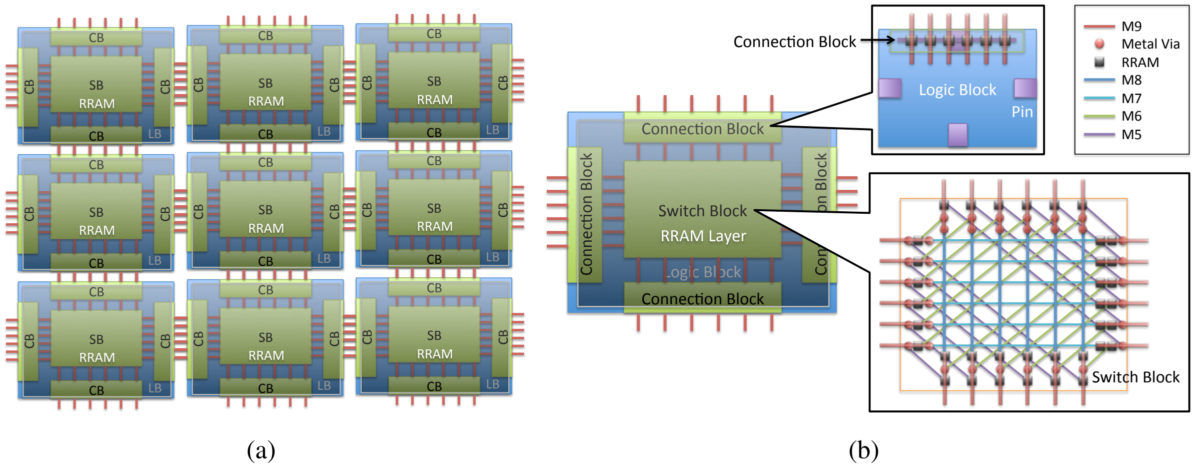


Figure 3.3: The proposed FPGA-RR architecture. (a) Overview of FPGA-RR where the FPGA area is mainly contributed by logic blocks instead of programmable interconnects, and (b) A detailed layout design of the connection blocks and switch blocks in FPGA-RR using the RRAM fabrication structure shown in Fig. 3.2. Here each 'metal via' (marked as a red point) at the intersection between wires of M9 and M5–M8 refers to a vertical connection between them through metal via(s). Each 'RRAM' (marked as a black box) refers to the same vertical connection but also integrated with an RRAM device.

shown in the far right of Fig. 3.2.

3. The placement of all metal wires is designed to avoid any blockage caused by a metal via. This blockage issue occurs where more than two metal layers are used, e.g. in switch blocks. Fig. 3.4 shows how we address this issue. In the abstract structure of FPGA-RR switch blocks, as shown in Fig. 3.4a, the location of any via that connects a metal layer is limited to the perimeter of the rectangular box assigned to that layer. Then, in the 3D view in Fig. 3.4b, the via blockages caused by vertical connections to the most bottom metal layer are located at the most outside box and vice versa. With the application of this principle, metal wires will naturally avoid all via blockages.

4. Though our design uses multiple metal layers, a signal rarely goes through many metal layers. While a signal is transmitted from one logic block to another, the straight paths in

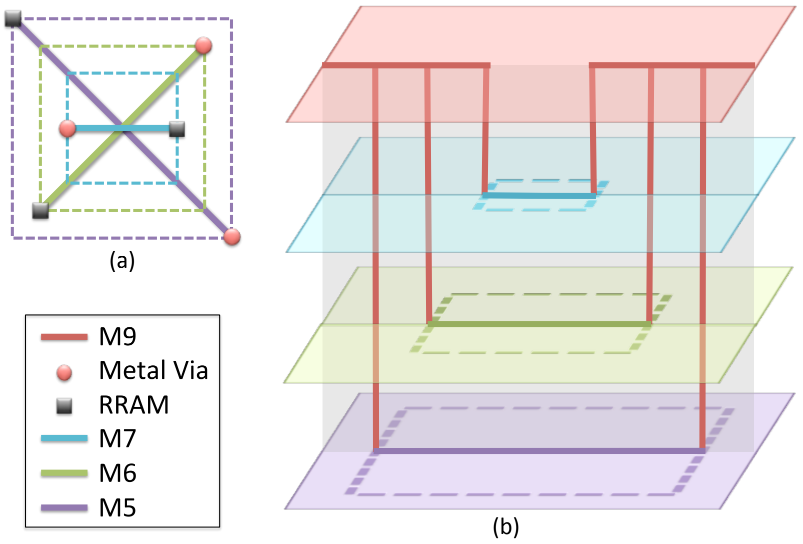


Figure 3.4: An illustration of how via blockages are avoided by metal wires in our layout design of switch blocks. (a) An abstract structure of FPGA-RR switch blocks. M9 and M8 are omitted here for clarity. (b) 3D view to show how via blockages are avoided by metal wires. RRAM layer is omitted here for clarity. For demonstration purposes, Vertical connections are bounded in a plane to construct the case most demanding for via blockage avoidance.

switch blocks are used most frequently. These paths go through M8 and M7, which are close to the RRAM layer to save extra latency on metal via. When a signal reaches a logic block, it then needs to access metal layers at very low levels, i.e. M1–M4, from the RRAM layer.

5. RRAMs can easily fit into the layout design in Fig. 3.3b without extra area overhead. As stated in Section 3.1, the size of an RRAM cell can be close to or even smaller than the width of a metal wire [WTL10].

6. Each metal layer in M5 to M9 consists of a set of parallel metal wires without any turns. This reduces fabrication complexity and avoids high resistance of turning points.

## 3.4 Programming Schematic

### 3.4.1 Programming Transistor Sharing

To program the RRAMs integrated in programmable interconnects, there needs to be two programming transistors at both of the two terminals of an RRAM respectively. Then an RRAM can be correctly selected to program under control of the two programming transistors. To achieve this, the author of [TLW11] proposes to allocate two programming transistors for each RRAM, i.e., a 2T1R structure. Programming transistors are usually larger than RRAMs and lead to extra CMOS area. In our FPGA-RR, we find out that we do not need so many programming transistors. If we denote each track in a routing channel or each pin of a logic block as one programming point, then any arbitrary RRAM is between two adjacent programming points. So, we only need to allocate one programming transistor for each programming point. For example, when we want to program the RRAM between two tracks to connect/disconnect the two tracks, we just turn on the programming transistors on the two tracks and apply the programming voltages to them to program the RRAM, as shown in Fig. 3.5. In the figure, $V_p$ refers to the threshold voltage that switches the RRAM state. The programming voltages may be the same as those in [TLW11]. The programming for the RRAMs between the tracks and the pins of logic blocks follows a similar pattern. The total

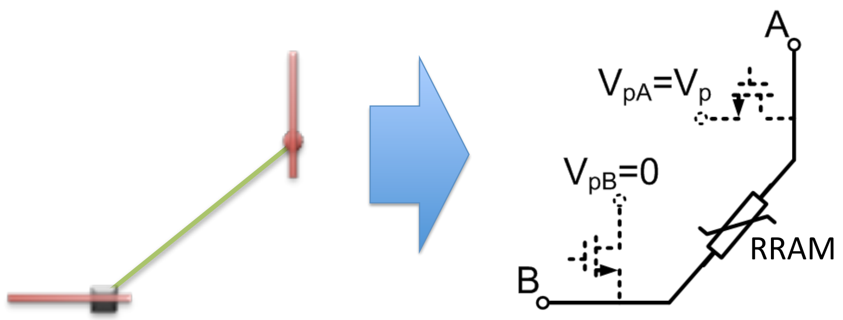


Figure 3.5: Programming circuits for RRAMs in interconnects. The two programming transistors on the two tracks will apply programming voltages to program the RRAM between them.

number of programming transistors per tile is calculated as two times the channel width plus the number of pins of a logic block per channel. It is about 1/6 of 2T1R consumption and also much smaller than the transistor count of a logic block. This area savings comes from the programming transistor sharing (PTS) among RRAMs, as shown in Fig. 3.6. This figure shows the situation in SBs, which contain more RRAMs than CBs. We see that one programming transistor is shared by

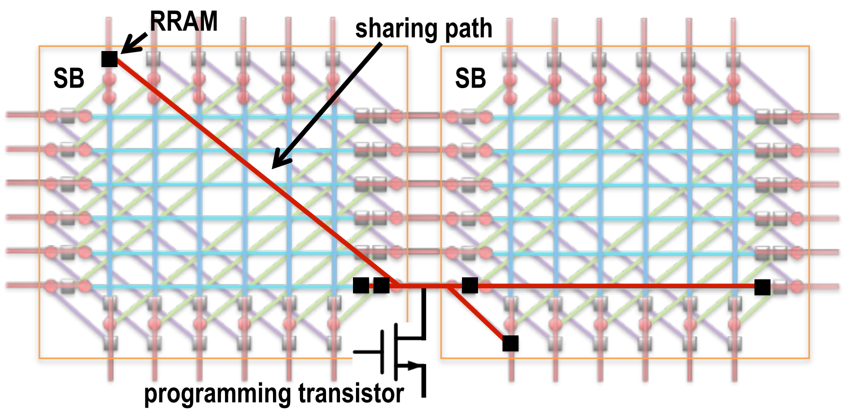


Figure 3.6: One programming transistor can be shared by six RRAMs (marked as black boxes) through sharing paths (marked as red lines) in switch blocks (SBs) of our work.

as many as six RRAMs, which are located not only within one single SB, but can also be across two adjacent SBs in two tiles. The situation in CBs is even better where more RRAMs share one programming point. Table 3.1 shows a comparison of transistor counts among routing switches based on CMOS, RRAM with 2T1R structure, and PTS.

Table 3.1: Comparison of transistor counts among routing switches based on CMOS, RRAM with 2T1R structure, and programming transistor sharing (PTS).

| | CMOS | 2T1R [TLW11] | PTS (this work) |
|---|---|---|---|
| $N$-to-$M$ MUX | $M\left(N+7\sqrt{N}\right)$ | $N+M$ | $N+M$ |
| $N \times N$ SB w/o buf | $64N$ | $24N$ | $2N$ |

The high sharing rate comes from our transistor-free RRAM-based programmable interconnect design. In the related work proposing 2T1R design [TLW11], the possibility of programming transistor sharing is, as shown in Fig. 3.7, blocked by the transistors existing in programmable interconnects. This limits sharing opportunities.

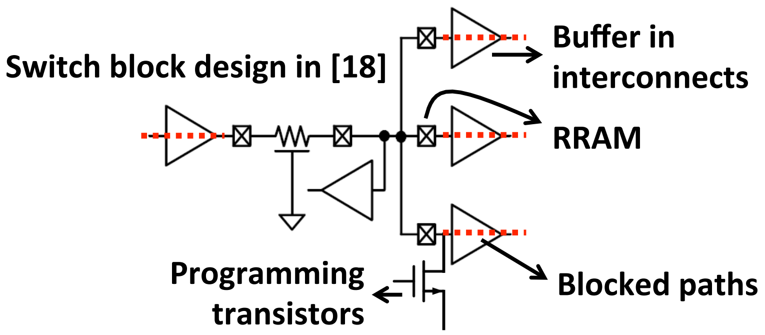


Figure 3.7: Sharing paths are blocked by transistors (marked as dotted lines) in existing work [TLW11].

## 3.4.2 Programming Transistor Selecting

Another important problem that has been ignored in previous work is the selecting circuit of programming transistors. In a memory system made up of 1T1C DRAM cells or 1T1R PCRAM (or RRAM cells), a common way to select a programming transistor is row/column addressing. The motivation is to build the selecting circuit at a complexity of $O\left(\sqrt{n}\right)$ with a lower order than $n$, where $n$ is the number of programming transistors. For large $n$, compared to programming transistors, the cost of the selecting circuit becomes trivial and can be ignored. The structure of row/column addressing is shown in Fig. 3.8. The gate terminal of each programming transistor is

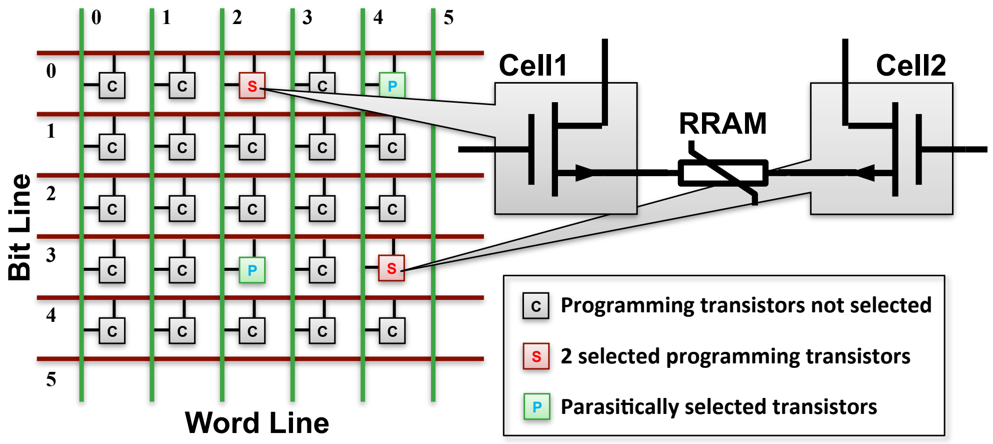
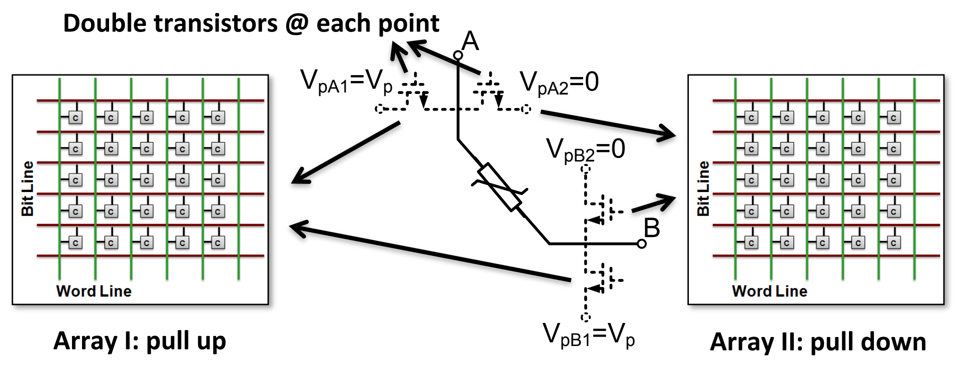
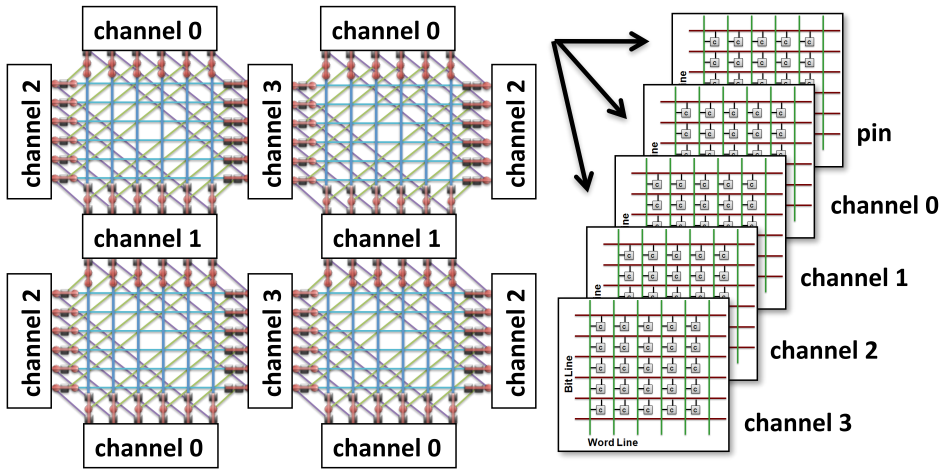

Bit Line
Word Line
Cell1
Cell2
RRAM
Programming transistors not selected
2 selected programming transistors
Parasitically selected transistors

Figure 3.8: Row/column addressing for the programming transistor array of RRAMs in interconnects. If two programming transistors in one array are selected simultaneously, there will be two other transistors selected parasitically.

connected to a word line, and the drain terminal is connected to a bit line. Note that in a memory application of RRAMs, an RRAM is dedicated to only one cell instead of being shared between cells in FPGA-RR, as shown in Fig. 3.8. One problem to programming an RRAM in programmable interconnects is that each time there are two programming transistors being selected. This is equivalent to a dual-port random access. This kind of access may cause troubles in a structure designed for single-port access, as shown in Fig. 3.8. For example, to program the RRAM in Fig. 3.8, the two cells marked as 'S' are selected. Then both the two word lines '2' and '4' and the two bit lines '0' and '3' are enabled simultaneously. It will cause two other programming transistors (marked as 'P') to be parasitically selected as well. To realize dual-port random access without malfunction, one naive solution is to double the programming transistors at each programming point, as shown in Fig. 3.9a. A better solution, without an extra increase in programming cost, is to partition the programming transistors into different blocks. It must guarantee that the two programming transistors of any arbitrary RRAM belong to different blocks. Fig. 3.9b shows a feasible partition which leads to five blocks. The programming transistors at programming points in y-directional channels are assigned into two blocks named 'channel 0' and 'channel 1' by a parity of channel indices. So are the programming points in x-directional channels. The programming points at the pins of logic



(a)



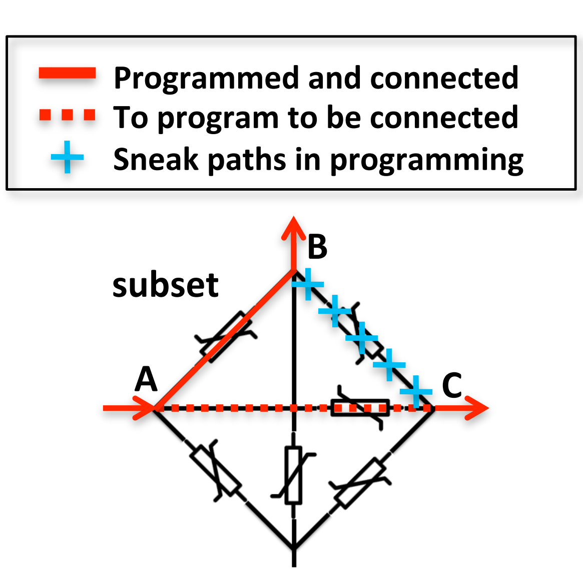
(b)

Figure 3.9: Two solutions to solve parasitic selection — (a) double programming transistors at each programming point, and (b) partition programming transistors into five blocks, where the two programming transistors at the terminals of any arbitrary RRAM lie in two different blocks.

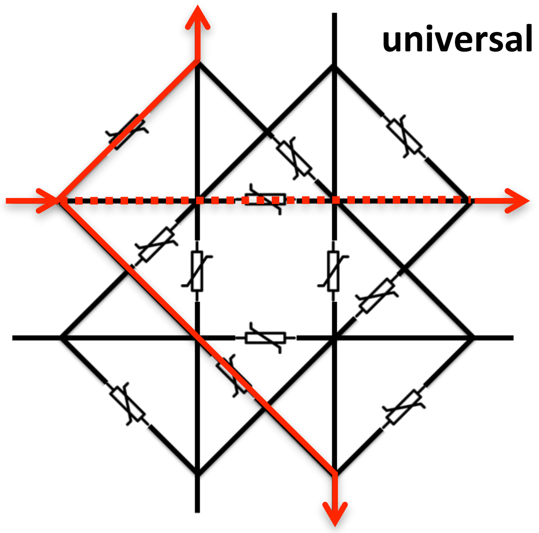
blocks are assigned into a separate block. In this case, no two programming transistors in the same block will be selected simultaneously and no parasitic selection will result.

### 3.4.3 Sneak Paths in Programming

One more problem that we visit in this paper is sneak paths in programming circuits. This problem exists in circuits with two-terminal programmable devices, e.g., anti-fuse, PRAM, RRAM, etc. To illustrate this, we take an RRAM-based subset-type switch block (SB) [WM95] as an example. As shown in Fig. 3.10a, we want to send a signal from point A upwards and rightwards to B and C. It is a very common demand in FPGA routing. We find out that when we want to program the RRAM between A and C after A and B have been programmed to connect, the RRAM between B and C will be programmed as well. One may argue that whether B and C are connected or not



(a) Subset-type switch block [WM95].



(b) Universal-type switch block [CWW96].

Figure 3.10: Subset-type switch blocks always have sneak paths in programming whenever a track with a fanout of two or more is involved. Universal-type switch blocks do not have this problem.

does not matter since A, B and C are always in the same net after programming. But in the next programming process, the connection between B and C can shunt part of the programming current to reset the RRAM between A and B or A and C. The programming current to reset a RRAM is usually large and the minimum size of the programming transistors is designed close to its margin. The unexpected current shunting will likely cause a programming failure and subsequent functional failure of FPGA. The author of [GHB93] found a way to avoid sneak paths by programming devices in a specific order. But in the case of Fig. 3.10a, when we turn to program the RRAM between A and C first to avoid the sneak path between B and C, the path is still there when we program the RRAM between A and B later. To program a subset-type SB to a connection with fanout $\geq 2$ directions, there will always be sneak paths which cannot be eliminated by changing programming order. What we need is a change of architecture to fix this problem. We realize that the reason that SBs of subset type can easily cause this kind of sneak path is that this SB type leads to dense programmable connections between limited tracks. We consider the universal-type SB [CWW96] to be a better choice since it distributes programmable connections among more tracks. As shown in Fig. 3.10b, even to program a connection with fanout of three directions, there will be no sneak path in the universal-type SB. Only in this case can we follow the method in [GHB93] to program in an order that eliminates sneak path. We understand that many commercial FPGA products are using subset-type SBs [WM95] due to easier layout design [SC02]. But we have already given a feasible universal-type RRAM-based SB layout design in Section 3.3. Also, universal SBs can give better routability, as proven in [CWW96].

## 3.5 On-Demand Buffer Insertion

In the previous sections we were discussing our FPGA-RR routing structure without buffers. Separation of buffers from the basic routing structure not only provides opportunities for the programming transistor sharing mentioned in Section 3.4, but also facilitates on-demand buffer insertion which will be described in this section.

### 3.5.1 Buffering Architecture

We propose on-demand buffer insertion as a buffering solution for FPGA-RR. As shown in Fig. 3.11, a limited number of buffers are prefabricated in routing channels. They can be connected to the tracks in channels via RRAMs. Buffers are shared among tracks in the same channel. Only a track with a high demand for a buffer will be programmed to use a buffer. The demand depends on routing paths of the circuit to implement on FPGA-RR. This mechanism brings benefits of both area and performance; these will be analyzed in the following sections.

Since each buffer is attached to routing tracks via one single RRAM, we adopt the regenerative feedback repeater described in [DHE95] as a buffer circuit. This circuit has one signal terminal and

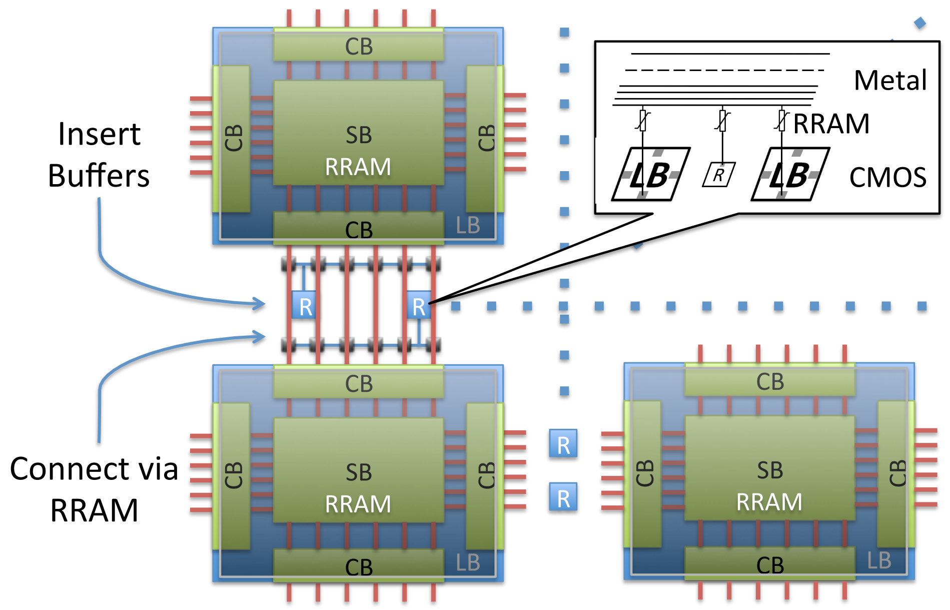


Figure 3.11: On-demand buffer insertion in programmable interconnects of FPGA-RR.

can provide drive in both signal directionals, as opposed to the conventional unidirectional buffers. It saves at least half of the buffers by serving the function of two complementary unidirectional buffers with only one repeater.

Note that the connections between buffers and routing tracks are actually RRAM-based multiplexers. For a channel of $N$ tracks and $M$ buffers, the area overhead of on-demand buffer insertion except buffer area is the $M$ programming transistors at extra programming points of buffer terminals. We reuse the $N$ programming transistors at programming points of tracks which are used to program switch blocks as well. The on-demand buffer insertion can also be implemented in conventional CMOS technology, but the area overhead is much higher. The cost would be an $N$-to-$M$ MUX, i.e., around $MN$ transistors according to Table 3.1.

### 3.5.2 Savings of Unnecessary Buffers

In conventional FPGAs, the locations of buffers in a routing structure are predetermined during FPGA design. For example, as shown in Fig. 2.1, each switch from one track segment to another in a switch block (SB) contains a routing buffer . The motivation for this conservative over-buffering is to avoid a potential large RC delay caused by an unbuffered connection between two long unbuffered track segments. This cautiousness stems from a lack of information about which track a signal comes from. But if we have information of the circuit mapped in FPGA, we do not need so many buffers. For example, the path from A to B in Fig. 3.12 will use a buffer when it goes through the SB. However, a buffer is unnecessary for such a short path. We should only place

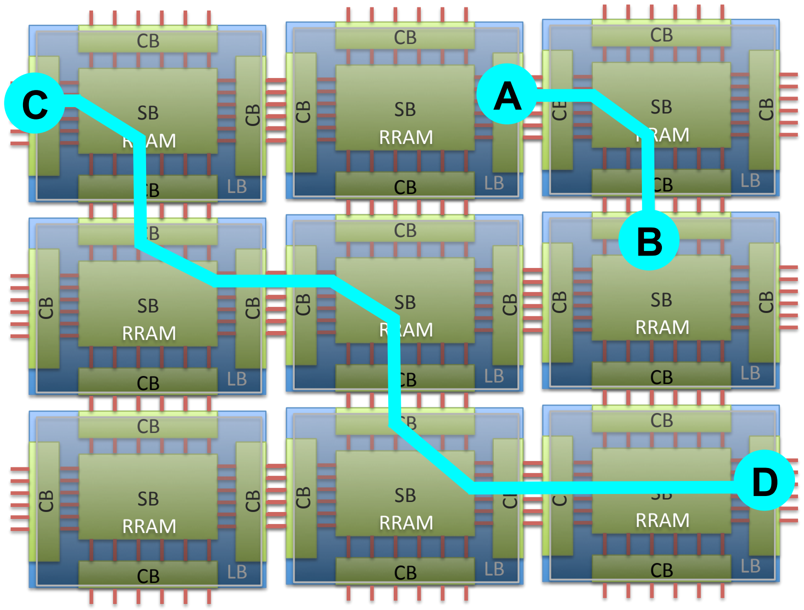


Figure 3.12: Two path examples to show different demands for buffers. Short paths without exhibition of quadratic increase of RC delay do not need buffers.

buffers at certain points in paths that are long enough to exhibit a quadratic increase in RC delay, e.g., from C to D.

Fixed locations of buffers also result in unnecessary buffers in non-critical paths. Fig. 3.13 shows a delay distribution of all paths in a benchmark ”tseng” mapped onto a conventional FPGA. The paths with more criticality (heavy color in Fig. 3.13) usually go from an FF through many logic blocks to another FF. Some of these logic blocks are far away from each other, and takes much effort to go from one to another, like C to D in Fig. 3.12. But for paths which are not so critical (those with light color in Fig. 3.13), even if some portion of them travels, e.g., C to D in

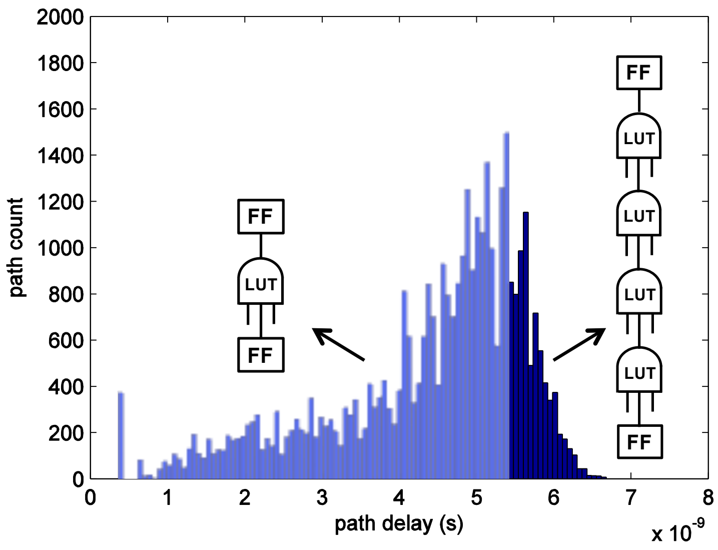
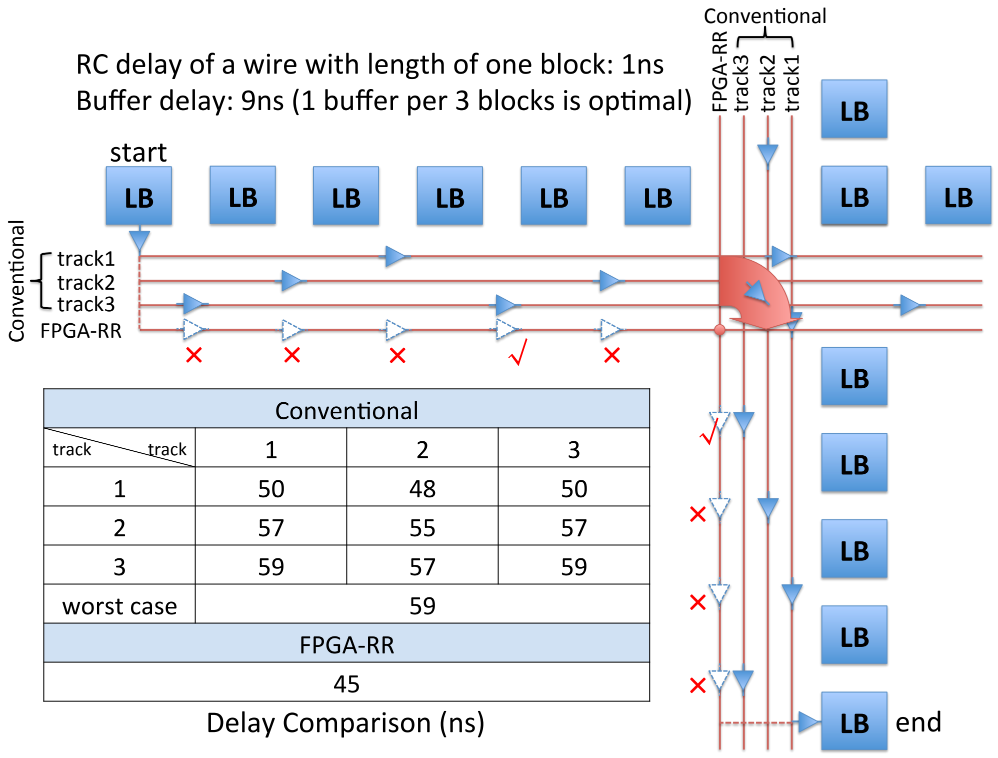


Figure 3.13: Delay distribution of all paths in a design to map onto FPGA. Most paths can be relaxed for less use of buffers.

Fig. 3.12, they may not need buffers since they go through fewer logic blocks and have sufficient timing slack. This results in a savings of routing buffers. But fixed locations of buffers eliminate this opportunity due to an unawareness of the critical paths of circuits that should implemented in FPGA ahead of the design period. Savings of unnecessary buffers can be achieved only during placement and route via on-demand buffer insertion.

### 3.5.3 Performance Benefit

The fixed locations of buffers in conventional FPGA also lead to a deviation from optimal timing results. A case study in Fig. 3.14 shows the timing benefit brought by the on-demand buffer insertion in FPGA-RR when compared to the fixed buffer pattern in conventional FPGA. To simplify the problem, we assume in this case that the RC delay of a wire with the length of one block is $0.5R_{\text{wire}}C_{\text{wire}} = 1$ns, and that the buffers in interconnects are considered ideal buffers with infinite drive, no input/output capacitance, and a fixed intrinsic delay of 9ns. The optimal length of a wire



| Conventional | | | |
|---|---|---|---|
| track \ track | 1 | 2 | 3 |
| 1 | 50 | 48 | 50 |
| 2 | 57 | 55 | 57 |
| 3 | 59 | 57 | 59 |
| worst case | 59 | | |
| FPGA-RR | | | |
| 45 | | | |

Delay Comparison (ns)

Figure 3.14: Performance comparison between the fixed buffer patten in conventional FPGA and the on-demand buffer insertion in FPGA-RR.

between two adjacent buffers would be

$$k = \sqrt{\frac{T_{\text{buffer}}}{0.5 R_{\text{wire}} C_{\text{wire}}}} = 3$$

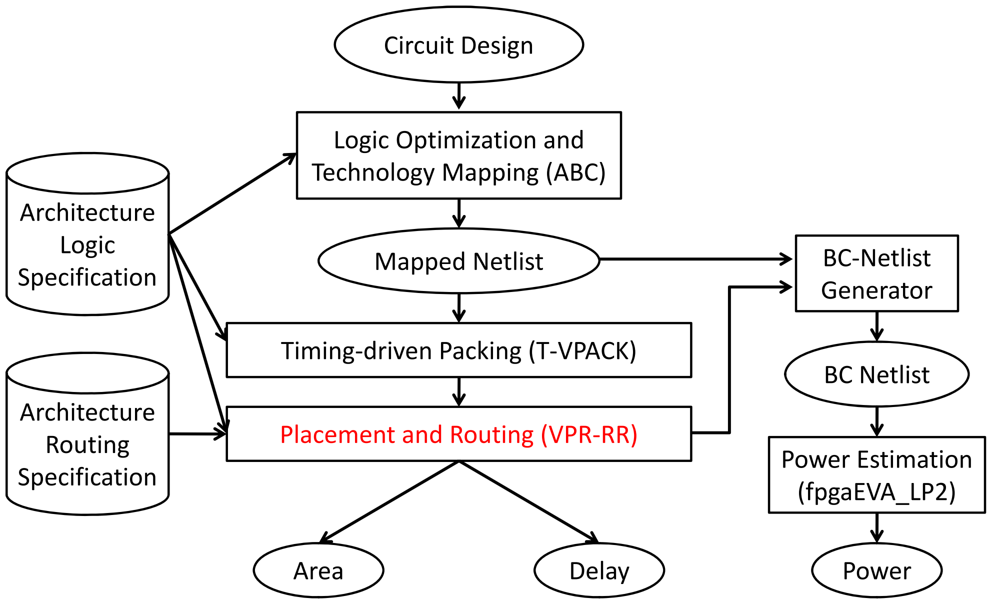
In conventional FPGA, the pattern of pre-fabricated buffers in interconnects will follow this result to distribute buffers evenly with a distance of three blocks (as shown in Fig. 3.14). The problem is that it does not know the starting point of each net (the offset can be 0, 1 or 2). So it just staggers the patterns among different tracks in the same channel, as shown in Fig. 3.14. When the output pin of a logic block happens to be connected to a wire segment that is buffered right after the connection node, just like the block “start” in Fig. 3.14 connected to track3 in the upper channel, the routing result will deviate from the optimal. The problem can be even worse during the switch from one track to another, which is always buffered to avoid potential RC delay. It drives the routing result farther from the optimal. The table in Fig. 3.14 lists all the possible delays from the logic block “start” to the logic block “end” according to the settings in the conventional FPGA discussed above. The first column is the track ID in the upper channel, and the first row is the track ID in the right channel. The buffers at the output pin of the “start” block and the input pin of the “end” block are not counted in the delay calculation. As shown in Fig. 3.14, the worst-case can be $22\%$ slower than the best case in a conventional FPGA. On the contrary, the on-demand buffer insertion in FPGA-RR will not suffer from the deviation from the optimal buffer placement in interconnects as does the conventional FPGA. We see from Fig. 3.14 that in FPGA-RR, buffers are inserted just exactly one per three blocks, resulting in a total delay of only 45ns. It is even better than the best-case in a conventional FPGA.

# CHAPTER 4

# Characterization and Evaluation

## 4.1 CAD Flow

To evaluate performance and energy consumption of FPGA-RR via widely used benchmarks, such as MCNC benchmarks, we need a CAD flow that is able to accept these benchmarks as input. The CAD flow for conventional FPGAs has been extensively studied [BRM99]. We adopt it and modify it into a flow for our FPGA-RR, as shown in Fig. 4.1. The input of the flow is the circuit



Circuit Design
Logic Optimization and Technology Mapping (ABC)
Architecture Logic Specification
Mapped Netlist
BC-Netlist Generator
Timing-driven Packing (T-VPACK)
BC Netlist
Architecture Routing Specification
Placement and Routing (VPR-RR)
Power Estimation (fpgaEVA_LP2)
Area
Delay
Power

Figure 4.1: Customized CAD flow for FPGA-RR. An enhanced P&R tools is developed for FPGA-RR.

design to implement on FPGA-RR. The output includes the placement and routing (P&R) result used for programming the design on FPGA-RR, as well as an estimation of area, delay, and power consumption for the design implementation on FPGA-RR. A circuit design first goes through the design tool ABC [too] to perform logic optimization and technology mapping. A mapped netlist consisting of $K$-LUTs will be obtained. Then the mapped netlist is fed into the design tool T-VPACK [BRM99] to pack LUTs to logic blocks, and then into the design tool VPR-RR that we developed to accomplish P&R. At last, we use the generated basic-cell (BC) netlist with delay and capacitance information to run the power estimation tool *fpgaEVA_LP2* [LLH05]. Since we try to reuse most parts of the existing CAD flow for conventional FPGAs, the development of an FPGA-RR CAD flow is quite easy. Only the P&R tool is specifically developed for FPGA-RR. We will introduce the features of this tool in the rest of this section.

## 4.2 Equivalent Circuit Model for Interconnect

To perform the timing and power analysis for FPGA-RR, we first develop an equivalent circuit model for the FPGA-RR routing structure. Fig. 4.2 shows the equivalent circuit of a representative routing path from the output pin of a logic block to the input pin of another logic block in FPGA-RR and makes a comparison with that of a conventional FPGA. In the circuit model, the MUXs

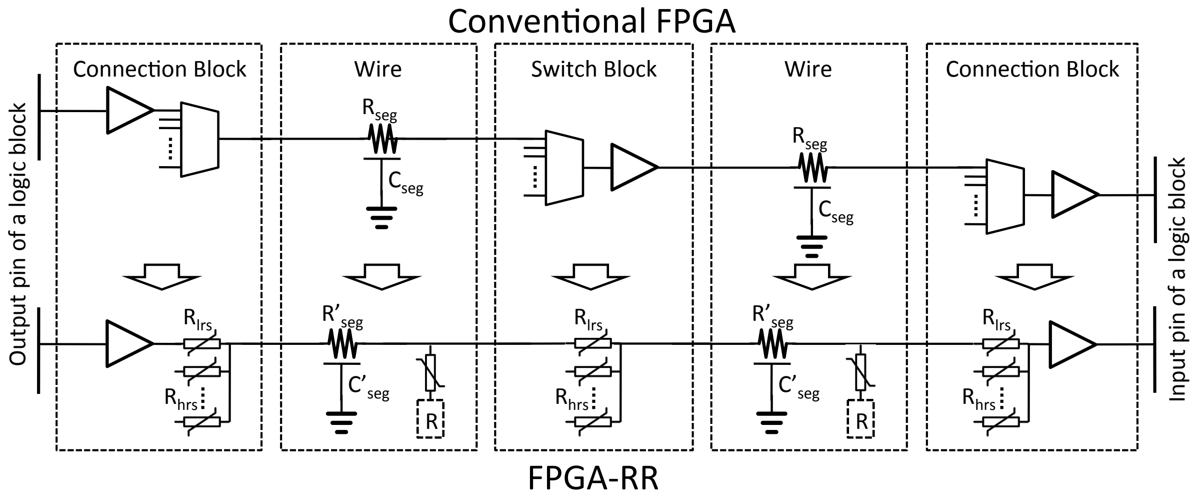


Figure 4.2: Extracted equivalent circuit model of the FPGA-RR routing structure and comparison with conventional FPGA.

in connection blocks and switch blocks are replaced by the RRAMs, of which one is at a low resistance state (LRS) and the others are at a high resistance state (HRS). The wire segments are still modeled as distributed RC lines, but with buffers if inserted. Note that the overall resistance value and capacitance value of wire segments have changed due to the reduction of the tile area in FPGA-RR.

## 4.3 On-Demand Buffer Insertion Algorithm

To fully utilize the benefits of on-demand buffer insertion in FPGA-RR, we develop an algorithm to choose the optimal positions of buffers to insert into programmable interconnects. Given all the nets routed in programmable interconnects, the goal is to find the best buffer option so that the critical delay is minimized. The constraint of the delay minimization is that the total number of inserted buffers should not exceed that of prefabricated buffers in each channel. Since on-demand buffer insertion in FPGA-RR allows a buffer allocation for demanding wires just like ASIC, we have considered implanting into FPGA-RR several state-of-art methods of ASIC buffer insertion [Gin90, CP99, ASV03]. Some methods are not applicable to FPGA-RR due to its difference from ASIC. For example the need to form buffer blocks or to utilize dead areas in ASIC does not exist in FPGA-RR. After these investigations, we decided to go with a negotiation-based iterative approach integrated with buffer placement for minimal delay.

To better present our algorithm, we will first focus on the delay minimization assuming there is a buffer resource. In this case, optimization is applied to every net so that the global minimization of the largest delay among all paths is achieved. Here we extend a method of buffer placement in ASIC [Gin90] to FPGA-RR. A routed net in FPGA-RR connects an output pin and multiple input pins of logic blocks together through programmable interconnects. According to the circuit model in Fig. 4.2, it is equivalent to a routing tree with one source and multiple sinks, as shown in Fig. 4.3. We perform a depth-first search on the routing tree to construct a set of delay/$C_{\text{downstream}}$ pairs that correspond to different buffer options for every subtree ($C_{\text{downstream}}$ refers to downstream

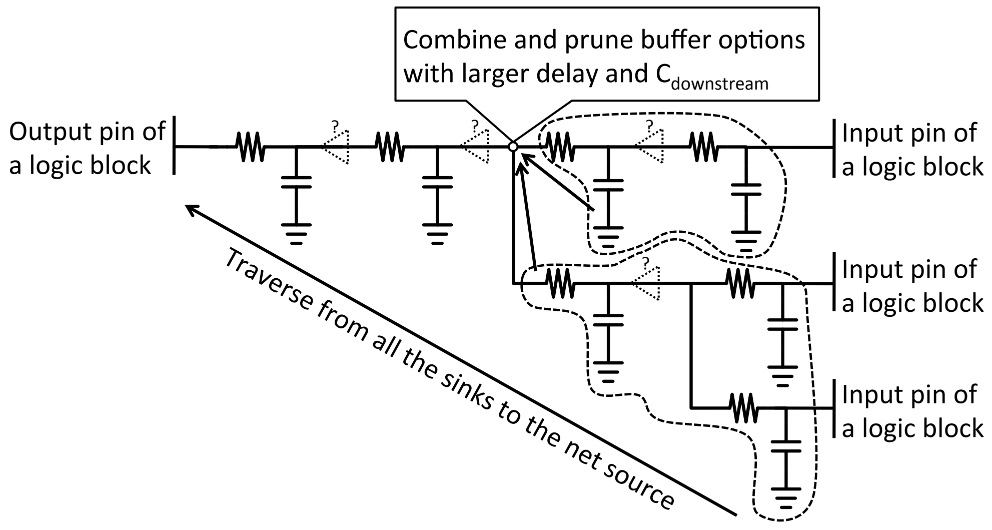

Combine and prune buffer options with larger delay and $C_{\text{downstream}}$
Output pin of a logic block
Input pin of a logic block
Input pin of a logic block
Input pin of a logic block
?
?
?
?
Traverse from all the sinks to the net source

Figure 4.3: The basic idea of the algorithm for the optimal buffer insertion is implemented in our tool.

capacitance). During the search, any pair with both delay/$C_{\text{downstream}}$ larger than another existing pair will be pruned after a combination of buffer options for two subtrees. The complexity of the algorithm is $O(B^2)$ where $B$ is the total number of legal buffer positions in a routing tree.

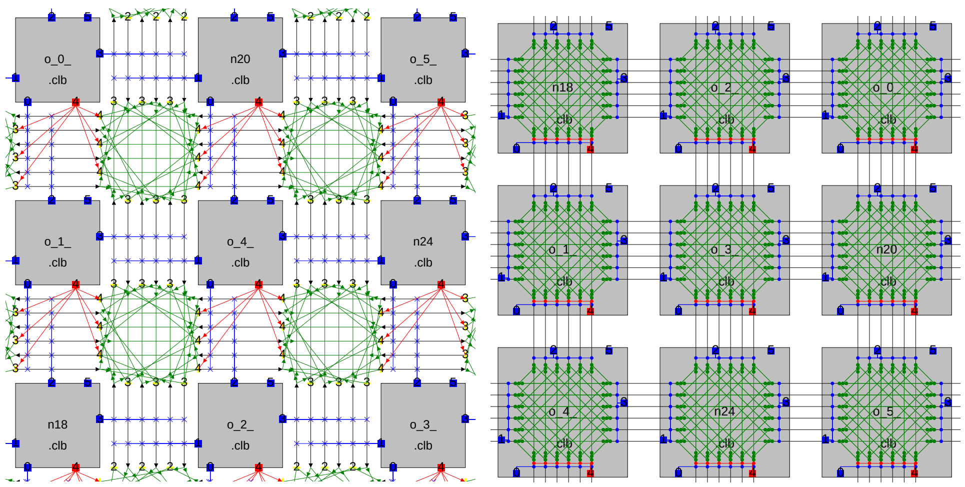
When buffer demands exceed the number of available buffers in some regions, we need to take the constraint of buffer resource into consideration during delay minimization. The method that we use is inspired by the negotiation-based routing iteration procedure in [ME95]. That procedure minimizes the critical delay under the constraint of track resource in every channel, which is similar to our problem. In our case, we add the buffer overuse cost to the delay of a buffer option as one of the metrics to prune buffer options (the other metric is still $C_{\text{downstream}}$). The total cost of a buffer option $n$ for a subtree $i$ of a routing tree shown in Fig. 4.3 is expressed as

$$\text{Cost}(n) = \text{Crit}(n) \cdot \text{delay}(n) + [1 - \text{Crit}(n)] \cdot \text{DNF} \cdot h(n) \cdot p(n)$$

Crit$(i)$ is the largest criticality among all the paths through which subtree $i$ goes. Criticality of a timing-critical path is close to one and cricality of a non-critical path is close to zero. DNF is the delay normalization factor. $h(n)$ and $p(n)$ are the historical and present overuse of buffer resources, respectively, in buffer option $n$. $p(n)$ will grow as iteration continues, and the buffers in uncritical paths will be pushed away from congested regions or even be removed.

## 4.4 VPR-RR: the P&R Tool for FPGA-RR

To deal with the novel routing structure of FPGA-RR in the P&R step of CAD flow, we develop an advanced tool named VPR-RR (VPR for RRAM-based Reconfiguration) on the base of the commonly-used FPGA P&R tool VPR [BRM99]. The main contributions of this tool are as fol-



(a) VPR for conventional FPGA (b) Our VPR-RR for FPGA-RR

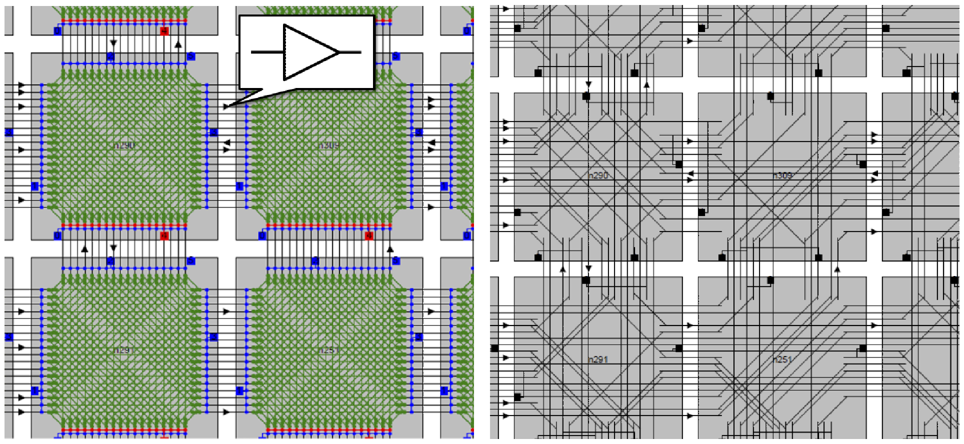
Figure 4.4: Overview of the two tools, VPR and VPR-RR.

lows:

1. VPR-RR can deal with the routing graph of FPGA-RR as shown in Fig. 4.4b. In Fig. 4.4b the gray blocks are I/O blocks and logic blocks of FPGA, and the diagonal wires are the routing paths available in switch blocks. We see that the switch blocks and connection blocks in VPR-RR are placed over logic blocks and provide connections for logic blocks nearby in a different way from that of the conventional FPGA shown in Fig. 4.4a.

2. VPR-RR is integrated with the algorithm to generate the best option for on-demand buffer insertion described in Section 4.3. The tool can display where buffers are needed for insertion in the final routing result, as shown in Fig. 4.5. We see that the positions for buffers to be

inserted are marked with a black delta shape. We also see that the interconnect view of the routing result in Fig. 4.5b is quite similar to that of ASIC. We describe the good performance of FPGA-RR in Section 5.



(a) Routing resource view. (b) Routing result view

Figure 4.5: View of the best option for buffer insertion by VPR-RR.

# CHAPTER 5

# Experimental Results

## 5.1 Settings

We choose 32nm technology node as our experimental platform. The architecture specifications for conventional FPGA are adopted from the 32nm setting in the intelligent FPGA Architecture Repository (iFAR) [KR, KR08a, KR08b]. In this setting, each logic block contains 10 four-input look-up tables (4-LUTs) and 22 input pins [KR]. One problem with iFAR is that all the input/output resistance/capacitance values of routing switches or wire segments in FPGA routing structures are zero. It is claimed in iFAR that all the timing parameters are counted in delay values. This is not an accurate model for evaluating the performance of FPGA. We calculate the timing parameters using the lookup tables in ITRS2010 [ITR10] and HPSICE simulation with PTM device models [ZC, ZC06]. The RRAM model is extracted from the measurement results of the RRAMs fabricated by the process in [GLL08]. The 20 largest MCNC benchmark circuits are used as the input of the CAD flow for conventional FPGA and FPGA-RR respectively, and comparisons are made on the output of the CAD flow from the aspects of area, delay and power. We find that one of the benchmarks becomes unroutable in conventional FPGA after we supplement the architecture settings with detailed timing parameters. So we increase the routing channel width from the original 104 in iFAR [KR] to the minimum achievable number of 120 found by VPR binary search.

## 5.2 Buffer Distribution, Sizing and Amount

Compared to conventional FPGA, FPGA-RR has the unique capability of on-demand buffer insertion. Before its evaluation, we need to explore the architecture settings related to on-demand buffer insertion. The first basic question is whether we should prefabricate more buffers in the routing channels at a specific location in FPGA-RR, e.g. around the center, or just evenly distribute them. This depends on the distribution of buffer demand over the routing channels. We measure the number of buffers inserted in every channel with the constraint of buffer resource removed. Fig. 5.1 shows two buffer distributions of two benchmarks implemented on FPGA-RR. By

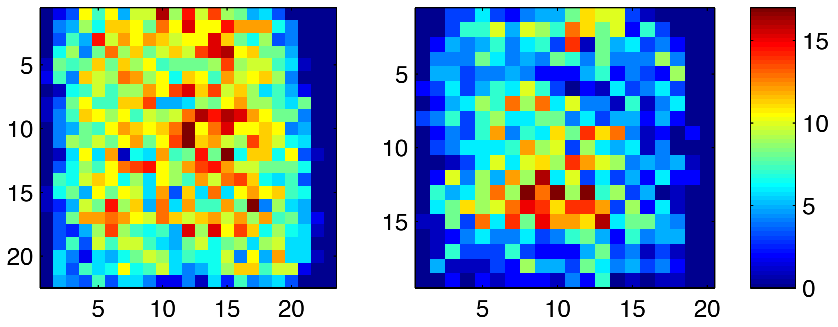


Figure 5.1: Two cases of buffer distribution without buffer limit. Locations with high buffer demand vary among cases.

comparing the buffer distributions of all the benchmarks, including these two, we observe that the hotspot locations of buffer demand differ from one design to another. They could be at the center of FPGA-RR, at boundaries, at some intermediate zones, or just a mixture of multiple patterns. Therefore, a simple uniform distribution of prefabricated buffers tends to work best.

Then we need to decide how many buffers to prefabricate per routing channel. In addition, the size of the buffers needs to be optimized again since the buffering solution has changed as compared to a conventional FPGA. Due to the observation that a smaller buffer size will increase the number of buffers required per channel, we explore these two parameters simultaneously. Note that both of the two parameters have an impact on both the area and performance of FPGA-RR. Therefore we use the area-delay product as the evaluation metric. Fig. 5.2 shows the result. Here

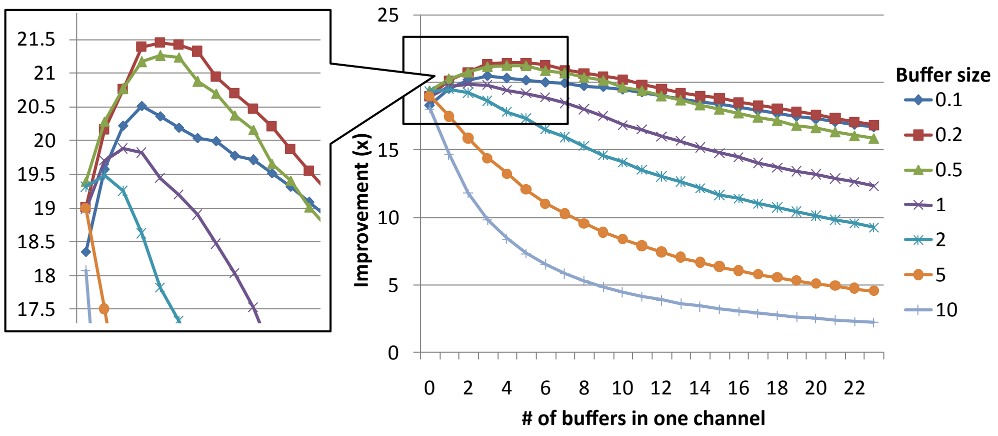


Figure 5.2: Exploration of the impact of buffer sizing and richness on the area-delay product. Buffer size is normalized (the optimal size of conventional FPGA in iFAR [KR] is normalized to 1).

buffer size is normalized, with the optimal size of conventional FPGA in iFAR [KR] equal to 1. The area-delay product is the average improvement of FPGA-RR, compared to conventional FPGA, over the 20 benchmarks. We see that with the increase of the number of buffers per channel, improvement of the area-delay product first increases and then decreases. That's because the FPGA area will always increase, so the area savings will always decrease. But the critical delay will gain a benefit at first and then keep constant when the buffer resource is rich enough. In the enlarged portion of Fig. 5.2, we find the best pair of buffer sizing and richness to achieve the maximum area-delay product. We determine the number of buffers per channel to be three and the buffer size at 0.2 times the optimal size of a conventional FPGA in iFAR.

## 5.3 Evaluation of FPGA-RR

Fig. 5.3 shows the area savings of FPGA-RR at the technology node of 32nm. It shows more than 6.82x saving of total area for FPGA-RR. The SRAM-based programming bits and routing switches are now implemented efficiently by RRAMs on top of CMOS. A large number of routing buffers are also saved due to on-demand buffer insertion. These factors lead to a small area of

|  |  |
|---|---|
| BLEs - 109 | Local / Global Interconnects - 839 |

(a) Baseline FPGA. Total area = $948\mu m^2$.

| RRAM / Metal wires |  |  |
|---|---|---|
| BLEs - 109 | Programming Trans – 22.2 | Buffers – 7.6 |

(b) FPGA-RR. Total area = $139\mu m^2$.

Figure 5.3: Area of a single tile and comparison. BLEs: basic logic elements.

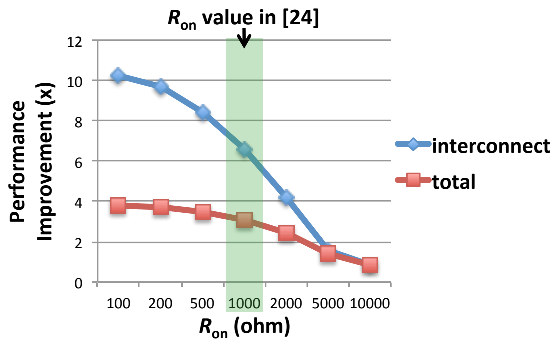
programmable interconnects as compared to that basic logic elements (BLE). Only in this case, where BLEs become the dominant part of FPGA, will technologies for highly compact BLEs [PMB08, BTS06, CZX10, CPZ09, TLW11] be meaningful. Works on BLEs can be combined with this work on programmable interconnects for more area savings.

Table 5.1 shows the performance comparison between baseline FPGA and FPGA-RR. It shows a 3.09x speedup. The speedup mainly stems from shorter communication distance due to area reduction. One-stage multiplexers (MUXs), implemented by RRAMs in routing switches, are also faster than multi-stage MUXs implemented by SRAMs. On-demand buffer insertion also drives buffer solutions closer to the optimal.

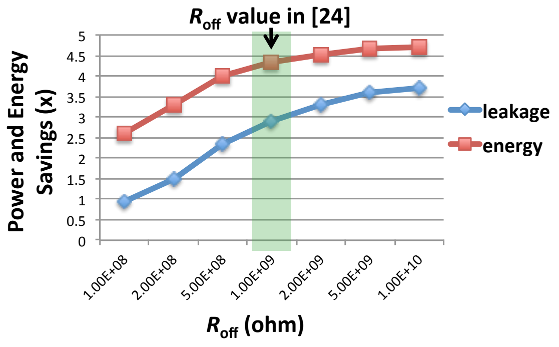
Table 5.2 shows the power and energy consumption of the baseline FPGA and FPGA-RR. An average of 4.33x energy savings per clock cycle is achieved. Due to fewer transistors in programmable interconnects, the leakage power is reduced. Also, since there is smaller capacitance on routing paths, dynamic power is reduced as well. Note that RRAM is also non-volatile. FPGA-RR will enjoy more benefits brought by power gating since no devices other than SRAMs in BLEs need supply power to hold their state. These SRAMs can be further eliminated by BLEs based on emerging non-volatile memories [PMB08, BTS06, CZX10, CPZ09, TLW11] as well. Table 5.3 shows a comparison of improvements achieved by different work on RRAM-based FPGAs. These works perform evaluation at the same technology node (32nm) with the same set of benchmarks. The comparison should be fair. When compared to FPGA-RR in this work, rFPGA [TLW11] shows limited improvement since it only replaces SRAM-based routing switches with RRAMs and does not renovate the routing structure to cater to RRAM property. The area savings of FPNI [SW07, XRC09] is impressive, but the large capacitance value of long nanowires undermines performance and energy consumption.

## 5.4 Sensitivity Analysis

Since RRAM is an emerging technology, its device parameters tend to be hard to control. This motivates us to conduct a sensitivity analysis for RRAM parameters. There are two key parameters: $R_{\text{on}}$ (or so-called $R_{\text{lrs}}$, resistance value at low resistance state) and $R_{\text{off}}$ (or so-called $R_{\text{hrs}}$, resistance value at high resistance state). $R_{\text{on}}$ mainly has an impact on FPGA-RR performance and $R_{\text{off}}$ on leakage power and energy consumption. Fig. 5.4 shows this impact. We sweep the parameters one



(a) Impact of $R_{\text{on}}$ on performance improvement.



(b) Impact of $R_{\text{off}}$ on energy/power savings.

Figure 5.4: Sensitivity analysis of RRAM parameters in FPGA-RR.

order of magnitude upwards or downwards away from the center with original settings. In most of the swept ranges, FPGA-RR maintains the improvement.

Table 5.1: Critical Path Delay and Comparison (unit: ns)

| | Baseline FPGA | | FPGA-RR | |
|---|---|---|---|---|
| | Interconnect | Total | Interconnect | Total |
| alu4 | 5.21 | 6.27 | 0.90 (5.79x) | 1.99 (3.14x) |
| apex2 | 6.09 | 7.28 | 1.04 (5.81x) | 2.15 (3.38x) |
| apex4 | 5.82 | 6.75 | 0.90 (6.46x) | 1.84 (3.66x) |
| bigkey | 2.85 | 3.32 | 0.45 (6.24x) | 0.97 (3.42x) |
| clma | 9.33 | 11.5 | 2.59 (3.59x) | 4.33 (2.67x) |
| des | 4.77 | 5.70 | 0.70 (6.74x) | 1.65 (3.45x) |
| diffeq | 5.33 | 6.90 | 0.52 (10.2x) | 2.56 (2.69x) |
| dsip | 2.89 | 3.21 | 0.37 (7.74x) | 0.88 (3.61x) |
| elliptic | 6.58 | 8.94 | 0.94 (7.00x) | 3.75 (2.38x) |
| ex1010 | 11.4 | 12.5 | 4.50 (2.54x) | 5.60 (2.23x) |
| ex5p | 5.62 | 6.42 | 0.77 (7.29x) | 1.71 (3.74x) |
| frisc | 9.62 | 12.8 | 1.20 (7.97x) | 4.82 (2.65x) |
| misex3 | 4.91 | 5.83 | 0.77 (6.30x) | 1.72 (3.38x) |
| pdc | 8.25 | 9.36 | 2.49 (3.31x) | 3.57 (2.61x) |
| s298 | 6.96 | 8.55 | 0.92 (7.53x) | 2.70 (3.16x) |
| s38417 | 6.22 | 7.82 | 0.66 (9.33x) | 2.31 (3.38x) |
| s38584.1 | 5.29 | 6.55 | 0.72 (7.27x) | 2.17 (3.00x) |
| seq | 5.33 | 6.13 | 0.87 (6.12x) | 1.79 (3.41x) |
| spla | 7.26 | 8.18 | 1.67 (4.32x) | 2.61 (3.13x) |
| tseng | 5.10 | 6.67 | 0.50 (10.0x) | 2.41 (2.76x) |
| average | - | - | - (6.58x) | - (3.09x) |

Table 5.2: Power/Energy Consumption and Comparison (unit: mW or J/cycle).

| | Baseline FPGA | | FPGA-RR | |
|---|---|---|---|---|
| | Leakage | Energy | Leakage | Energy |
| alu4 | 6.38 | 82.7 | 1.87 (3.39x) | 22.7 (3.63x) |
| apex2 | 7.75 | 99.3 | 2.54 (3.05x) | 23.8 (4.15x) |
| apex4 | 6.99 | 71.5 | 2.49 (2.80x) | 15.2 (4.68x) |
| bigkey | 14.4 | 83.2 | 8.61 (1.67x) | 22.7 (3.65x) |
| clma | 25.0 | 402. | 8.10 (3.09x) | 83.1 (4.84x) |
| des | 20.2 | 194. | 12.6 (1.59x) | 56.0 (3.46x) |
| diffeq | 5.53 | 62.3 | 1.75 (3.14x) | 16.6 (3.73x) |
| dsip | 15.0 | 78.9 | 8.93 (1.68x) | 22.4 (3.51x) |
| elliptic | 12.3 | 160. | 4.21 (2.93x) | 39.0 (4.11x) |
| ex1010 | 23.2 | 363. | 8.15 (2.85x) | 74.5 (4.87x) |
| ex5p | 5.87 | 60.6 | 2.09 (2.81x) | 13.6 (4.44x) |
| frisc | 13.4 | 221. | 4.16 (3.22x) | 42.4 (5.21x) |
| misex3 | 6.73 | 75.6 | 2.29 (2.94x) | 20.0 (3.78x) |
| pdc | 19.4 | 236. | 6.89 (2.82x) | 46.4 (5.10x) |
| s298 | 5.55 | 55.8 | 1.70 (3.26x) | 8.55 (6.52x) |
| s38417 | 18.4 | 215. | 5.33 (3.44x) | 47.2 (4.54x) |
| s38584.1 | 20.1 | 217. | 5.71 (3.52x) | 53.6 (4.05x) |
| seq | 7.90 | 91.4 | 2.69 (2.93x) | 23.5 (3.88x) |
| spla | 15.9 | 179. | 5.75 (2.77x) | 35.4 (5.07x) |
| tseng | 4.27 | 55.1 | 1.10 (3.85x) | 15.9 (3.46x) |
| average | - | - | - (2.89x) | - (4.33x) |

Table 5.3: Comparison of improvements in related work and our work.

| | area reduction | performance | energy reduction |
|---|---|---|---|
| rFPGA [TLW11] | 2.10x | 1.67x | 1.19x |
| FPNI [SW07, XRC09] | 7.52x | 0.77x | 0.94x |
| FPGA-RR (this work) | 6.82x | 3.09x | 4.33x |

# CHAPTER 6

# Conclusion

This work presents FPGA-RR, an novel FPGA architecture with RRAM-based reconfiguration. The proposed architecture is based on the existing CMOS-compatible RRAM fabrication process. After reviewing limitations of the conventional FPGA and recent work on FPGA with emerging NVMs, we redesign the dominant part of FPGA, programmable interconnects, to conquer these limitations. We discuss practical issues and improvement opportunities driven by RRAM usage. A complete CAD flow is provided for FPGA-RR, with an advanced P&R tool named VPR-RR developed for FPGA-RR to deal with its novel routing structure. An evaluation of FPGA-RR is done on the 20 largest MCNC benchmark circuits. Results show that FPGA-RR achieves 6.82x area savings, 3.09x speedup and 4.33x energy savings. Note that no die-stacking is needed to achieve this degree of area reduction and speedup. If 3D integration technology is introduced in the future to stack several FPGA-RRs together, at least 2x more improvement in density and speedup can be expected, according to the experimental results on 3D architecture in [LEL07].

# REFERENCES

[AR04] E. Ahmed and J. Rose. "The Effect of LUT and ClusterSize on Deep-Submicron FPGA Performance and Density." *IEEE Transactions on VLSI Systems*, **12**(3):288–298, March 2004.

[ASV03] C.J. Alpert, S.S. Sapatnekar, and P.G. Villarrubia. "A practical methodology for early buffer and wire resource allocation." *TCAD*, **22**(5):573–583, May 2003.

[BRM99] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. MA:Kluwer, Norwell, 1999.

[BTS06] N. Bruchon, L. Torres, G. Sassatelli, and G. Cambon. "New non-volatile FPGA concept using Magnetic Tunneling Junction." In *ISVLSI*, pp. 269–276, 2006.

[CP99] Jason Cong and D.Z. Pan. "Buffer block planning for interconnect-driven floorplanning." In *ICCAD*, pp. 358–363, 1999.

[CPZ09] R.S. Chakraborty, S. Paul, Y. Zhou, and S. Bhunia. "Low-power hybrid complementary metal-oxide-semiconductor-nano-electro-mechanical systems field programmable gate array: circuit level analysis and defect-aware mapping." *IET Computers and Digital Techniques*, **3**(6):609, 2009.

[CSR11] Jason Cong, Vivek Sarkar, Glenn Reinman, and Alex Bui. "Customizable Domain-Specific Computing." *IEEE Design and Test of Computers*, **28**(2):6–15, March 2011.

[CWM10] Chen Chen, H.-S. Philip Wong, Subhasish Mitra, Roozbeh Parsa, Nishant Patil, Soogine Chong, Kerem Akarvardar, J Provine, David Lewis, Jeff Watt, and Roger T. Howe. "Efficient FPGAs using nanoelectromechanical relays." In *International Symposium on FPGAs*, pp. 273–282, 2010.

[CWW96] Yao-Wen Chang, D F Wong, and C. K. Wong. "Universal switch modules for FPGA design." *ACM Transactions on Design Automation of Electronic Systems*, **1**(1):80–101, January 1996.

[CX11] Jason Cong and Bingjun Xiao. "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration." In *Proc. International Symposium on Nanoscale Architectures*, pp. 1–8, June 2011.

[CZX10] Yibo Chen, Jishen Zhao, and Yuan Xie. "3D-nonFAR: Three-Dimensional Non-Volatile FPGA ARchitecture Using Phase Change Memory." In *ISLPED*, p. 55, 2010.

[DCH07] Chen Dong, Deming Chen, Sansiri Haruehanroengra, and Wei Wang. "3-D nFPGA: A Reconfigurable Architecture for 3-D CMOS/Nanomaterial Hybrid Digital Circuits." *IEEE Transactions on Circuits and Systems I: Regular Papers*, **54**(11):2489–2501, November 2007.

[DeH96] Andre DeHon. *Reconfigurable Architectures for General-Purpose Computing*. PhD thesis, MIT, December 1996.

[DHE95] I. Dobbelaere, M. Horowitz, and A. El Gamal. "Regenerative feedback repeaters for programmable interconnections." *IEEE Journal of Solid-State Circuits*, **30**(11):1246–1253, 1995.

[Geo00] Varghese George. *Low Energy Field-Programmable Gate Array*. PhD thesis, UC Berkeley, 2000.

[GHB93] J. Greene, E. Hamdy, and S. Beal. "Antifuse field programmable gate arrays." *Proceedings of the IEEE*, **81**(7):1042–1056, July 1993.

[GHB10] Pierre-Emmanuel Gaillardon, M. Haykel Ben-Jamaa, Giovanni Betti Beneventi, Fabien Clermidy, and Luca Perniola. "Emerging memory technologies for reconfigurable routing in FPGA architecture." In *ICECS*, pp. 62–65, December 2010.

[Gin90] L.P.P.P. van Ginneken. "Buffer placement in distributed RC-tree networks for minimal Elmore delay." In *ISCAS*, pp. 865–868, 1990.

[GLL08] Weihua Guan, Shibing Long, Qi Liu, Ming Liu, and Wei Wang. "Nonpolar Nonvolatile Resistive Switching in Cu Doped $ZrO_2$." *IEEE Electron Device Letters*, **29**(5):434–437, May 2008.

[HZG11] Ru Huang, Lijie Zhang, Dejin Gao, Yue Pan, Shiqiang Qin, Poren Tang, Yimao Cai, and Yangyuan Wang. "Resistive switching of silicon-rich-oxide featuring high compatibility with CMOS technology for 3D stackable and embedded applications." *Applied Physics A*, pp. 927–931, March 2011.

[ITR10] "International Technology Roadmap for Semiconductors.", 2010.

[JLY08] Weon Wi Jang, Jeong Oen Lee, Jun-Bo Yoon, Min-Sang Kim, Ji-Myoung Lee, Sung-Min Kim, Keun-Hwi Cho, Dong-Won Kim, Donggun Park, and Won-Seong Lee. "Fabrication and characterization of a nanoelectromechanical switch with 15-nm-thick suspension air gap." *Applied Physics Letters*, **92**(10):103110, 2008.

[KR] Ian Kuon and Jonathn Rose. "iFAR – intelligent FPGA Architecture Repository.".

[KR07] Ian Kuon and Jonathan Rose. "Measuring the Gap Between FPGAs and ASICs." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **26**(2):203–215, February 2007.

[KR08a] Ian Kuon and Jonathan Rose. "Area and delay trade-offs in the circuit and architecture design of FPGAs." In *International Symposium on FPGAs*, pp. 149–158, 2008.

[KR08b] Ian Kuon and Jonathan Rose. "Automated transistor sizing for FPGA architecture exploration." In *DAC*, p. 792, 2008.

[KZU11] P. Khalili Amiri, Z M Zeng, P Upadhyaya, G Rowlands, H Zhao, I N Krivorotov, J.-P. Wang, H W Jiang, J A Katine, J Langer, K Galatsis, and K L Wang. "Low Write-Energy Magnetic Tunnel Junctions for High-Speed Spin-Transfer-Torque MRAM." *IEEE Electron Device Letters*, **32**(1):57–59, January 2011.

[LEL07] Mingjie Lin, Abbas El Gamal, Yi-Chang Lu, and Simon Wong. "Performance Benefits of Monolithically Stacked 3-D FPGA." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **26**(2):681–229, February 2007.

[LL02] Guy Lemieux and David Lewis. "Circuit design of routing switches." In *Intl. Symp. FPGA*, pp. 19–28, 2002.

[LLH05] Fei Li, Yan Lin, Lei He, Deming Chen, and Jason Cong. "Power modeling and characteristics of field programmable gate arrays." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **24**(11):1712–1724, November 2005.

[LW08] Andrea L. Lacaita and Dirk J. Wouters. "Phase-change memories." *Physica Status Solidi (a)*, **205**(10):2281–2297, October 2008.

[ME95] Larry Mcmurchie and Carl Ebeling. "PathFinder : A Negotiation-Based Performance-Driven Router for FPGAs." In *FPGA*, pp. 111–117, 1995.

[PMB08] Somnath Paul, Saibal Mukhopadhyay, and Swarup Bhunia. "Hybrid CMOS-STTRAM non-volatile FPGA: Design challenges and optimization approaches." In *ICCAD*, pp. 589–592, November 2008.

[SC02] Herman Schmit and Vikas Chandra. "FPGA switch block layout and evaluation." In *Intl. Symp. FPGA*, p. 11, 2002.

[SV03] P. Schaumont and I. Verbauwhede. "Domain-specific codesign for embedded security." *Computer*, **36**(4):68–74, April 2003.

[SW07] Gregory S Snider and R Stanley Williams. "Nano/CMOS architectures using a field-programmable nanowire interconnect." *Nanotechnology*, **18**(3):035204, January 2007.

[TKN07] K. Tsunoda, K. Kinoshita, H. Noshiro, Y. Yamazaki, T. Iizuka, Y. Ito, A. Takahashi, A. Okano, Y. Sato, T. Fukano, M. Aoki, and Y. Sugiyama. "Low Power and High Speed Switching of Ti-doped NiO ReRAM under the Unipolar Voltage Source of less than 3V." In *IEDM Technical Digest*, pp. 767–770, December 2007.

[TLW11] Sansiri Tanachutiwat, Ming Liu, and Wei Wang. "FPGA Based on Integration of CMOS and RRAM." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **19**(11):2023–2032, November 2011.

[too] "Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 70731.".

[WM95] Yu-liang Wu and Malgorzata Marek-Sadowska. "Orthogonal Greedy Coupling - A New Optimization Approach to 2-D FPGA Routing." In *DAC*, pp. 568–573. ACM, December 1995.

[WTL10] Ching-Hua Wang, Yi-Hung Tsai, Kai-Chun Lin, Meng-Fan Chang, Ya-Chin King, Chrong-Jung Lin, Shyh-Shyuan Sheu, Yu-Sheng Chen, Heng-Yuan Lee, Frederick T Chen, and Ming-Jinn Tsai. "Three-Dimensional $4F^2$ ReRAM Cell with CMOS Logic Compatible Process." In *IEDM Technical Digest*, pp. 664–667, 2010.

[Xil] Xilinx. "Virtex-6 FPGA data sheets.".

[XRC09] Qiangfei Xia, Warren Robinett, Michael W Cumbie, Neel Banerjee, Thomas J Cardinali, J Joshua Yang, Wei Wu, Xuema Li, William M Tong, Dmitri B Strukov, Gregory S Snider, Gilberto Medeiros-Ribeiro, and R Stanley Williams. "Memristor-CMOS hybrid integrated circuits for reconfigurable logic." *Nano Letters*, **9**(10):3640–5, October 2009.

[ZC] Wei Zhao and Yu Cao. "Predictive Technology Model (PTM) website.".

[ZC06] Wei Zhao and Yu Cao. "New Generation of Predictive Technology Model for Sub-45nm Design Exploration." In *ISQED*, pp. 585–590, 2006.