FPGA-RPI: A Novel FPGA Architecture With RRAM-Based Programmable Interconnects

Jason Cong, *Fellow, IEEE*, and Bingjun Xiao, *Student Member, IEEE*

***Abstract*— In this paper we introduce a novel field programmable gate array (FPGA) architecture with resistive random access memory (RRAM)-based programmable interconnects (FPGA-RPI). Programmable interconnects are the dominant part of FPGA. We use RRAMs to build programmable interconnects, and optimize their structures by exploiting opportunities that emerge in RRAM-based circuits. FPGA-RPI can be fabricated by the existing CMOS-compatible RRAM process. Using an advanced placement and routing tool named VPR-RPI which was developed to deal with the novel architecture, a customized CAD flow is provided for FPGA-RPI. Results show that the programmable interconnects of FPGA-RR have a 96% smaller footprint, 55% higher performance, and 79% lower power consumptions compared to other FPGA counterparts.**

***Index Terms*— Emerging device, field programmable gate array (FPGA), programmable interconnects, reconfigurable logic, resistive random access memory (RRAM).**

I. Introduction

FIELD programmable gate arrays (FPGAs) can be customized to user applications and provide $>10\times$ improvement in power efficiency over CPUs [1]. The programmable interconnects inside FPGAs allow user applications with an arbitrary Boolean network to be mapped to FGPAs, but pay high overhead for such flexibility. The programmable interconnects account for 50%–90% of the total FPGA area [2]–[4], 70%–80% of the total delay [2]–[6], and 60%–85% of the total power consumption [2]–[4], [6], [7].

Emerging technologies, especially emerging nonvolatile memory (NVM) technologies, lead to new opportunities for design improvement. Popular emerging NVMs include spin-transfer torque RAM (STTRAM), phase-change RAM (PRAM), nanoelectromechanical (NEM) relay, and resistive RAM (RRAM). All these technologies have demonstrated CMOS-compatible fabrication and can be integrated in metal layers over CMOS via a back-end-of-line (BEOL) process [8]–[11]; this leads to opportunities for high-density circuit designs. These NVMs also have the desirable property of nonvolatility, which means that they can be turned off during standby to save power. Various semiconductor companies (i.e., Micronic, Panasonic, and Spansio) are manufacturing NVMs in volume [4]. Many studies focus on the memory applications of NVMs, and there have also been a number of works on the introduction of NVMs to FPGA architectures over the past few years [3], [4], [12]–[23].

This paper presents a novel FPGA architecture with RRAM-based programmable interconnects (FPGA-RPI), which are enabled by using the RRAM properties. Our RRAM-based programmable interconnects are composed of three disjoint structures: transistor-less programmable interconnects, a programming grid, and an on-demand buffering architecture. Specific optimizations are performed on each structure. The transistor-less programmable interconnects are built by RRAMs and metal wires alone, and are placed over CMOS transistors. We have designed an RRAM-friendly layout. It meets the tight space constraints coming from the stacked structure and, at the same time, fits into the footprint of the CMOS transistors below. The programming transistors in the programming grid are heavily shared among RRAMs via the transistor-less programmable interconnects. The on-demand buffering architecture provides opportunities to allocate buffers in interconnects during the implementation phase. It allows utilization of the application information for a better allocation of buffers. Note that the feasibility of the disjoint structures, as well as the feasibility of all their improvements mentioned above, is based on the use of RRAMs as programmable switches (Section II-B.2). Simulation results show that our RRAM-based programmable interconnects achieve significant reduction in area, delay, and power.

This paper expands on preliminary work [24] with a deeper examination of NVM candidates, the addition of the programming scheme, a discussion of the benefits of using our on-demand buffering architecture, and additional evaluations of each architectural change.

II. Background

A. Conventional FPGA

Fig. 1 shows a conventional FPGA architecture. It is a typical island-based type, which is made up of an array of tiles. Each tile consists of one logic block (LB), also called configurable logic block (CLB), two connection blocks (CBs), and one switch block (SB). Each LB contains a cluster of basic logic elements (BLEs), typically look-up tables

Manuscript received April 18, 2012; revised October 14, 2012 and January 30, 2013; accepted March 30, 2013. Date of publication May 21, 2013; date of current version March 18, 2014. This work was supported by the Center for Domain-Specific Computing (CDSC) which is funded by the NSF Expedition in Computing Award CCF-0926127.

J. Cong is with the Department of Electrical Engineering and Computer Science, University of California, Los Angeles, CA 90095 USA, and also with California Nano-System Institute, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu).

B. Xiao is with the Department of Electrical Engineering and Computer Science, University of California, Los Angeles, CA 90095 USA (e-mail: xiao@cs.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2259512

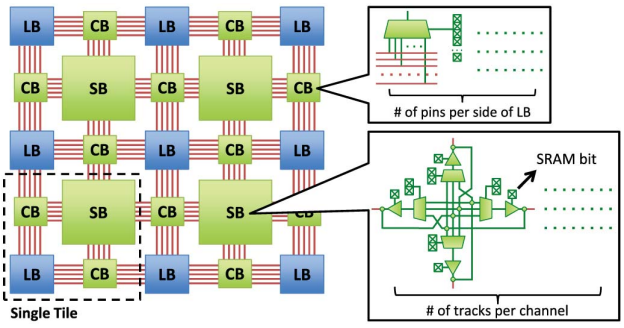


Fig. 1. Typical FPGA architecture. LB: logic block (or CLB). CB: connection block. SB: switch block.

(LUTs), to provide customizable logic functions. Each LB also contains local routing multiplexers (MUXs) to provide connections among BLEs. LBs are connected to the routing channels through CBs, and the segmented routing channels are connected with each other through SBs. Fig. 1 also depicts two typical circuit designs of CBs and SBs based on MUXs and buffers described in [25]. The selector pins of each MUX are connected to a group of storage cells that determine the connectivity of the MUX. The storage cells can be SRAMs, anti-fuses, or flash cells [26]. SRAM-based FPGAs are currently popular due to their standard CMOS manufacturing processes [26]. In this paper we focus our analysis on SRAM-based FPGA architectures. Note that the circuits presented in Fig. 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, and pay a high logic complexity for flexibility.

The FPGA programmable interconnects usually contain three types of components: SRAM-based storage of configuration bits, MUX-based routing switches, and buffers. All three components are nontrivial parts in FPGAs, as shown in Fig. 2.[1] Though CMOS-based programmable interconnects in FPGAs have been optimized, there are fundamental limitations in all three components of programmable interconnects (shown in Fig. 2).

1) Most FPGAs use SRAMs to store programming bits [26]. Each SRAM cell uses more than six transistors to store only one bit. Modern FPGAs enable serial bitstream programming by storing configuration bits in latch nodes embedded in a shift register structure, with an area overhead no less than the SRAM-based storage. In addition, these storage media are volatile—this causes excessive power consumption during standby.
2) MUX-based routing switches in FPGAs are implemented in tree structures with serial pass transistors for less SRAM-based storage of select bits. Each pass transistor has to be wide enough to provide sufficient drive, and this leads to a large footprint.
3) In most cases, routing buffers in FPGAs exceed the buffering demand of a given application. The quantity and positions of routing buffers placed in each track of programmable interconnects are optimized to meet the worst-case demand of the track.

[1]This breakup is based on a recent commercial FPGA (details in Section V-A) and shows a large percentage of LB area. The increasing overhead of programmable interconnects pushes FPGA vendors to larger and more complex LBs. In evaluations [2]–[4], the proportion of programmable interconnects is >75%, which provides sufficient improvement space for these interconnects. We show in Section V-D that after the overhead of programmable interconnects is significantly reduced by our technologies, smaller LBs will be preferred.

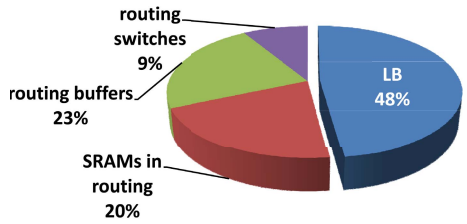


Fig. 2. Area breakdown of different components in an FPGA based on an architectural model [27] that mimics the Xilinx Virtex7 FPGAs [28]. All three components (SRAM bits, routing switches, and routing buffers) take up significant area in programmable interconnects.

B. FPGAs With Emerging NVMs

With the recent development of emerging NVM technologies, a number of novel FPGA architectures based on those technologies have been proposed.

1) Replace SRAMs With NVMs: In [12]–[18], the SRAM-based configuration bits are moved to STTRAMs, PRAMs, NEM relays, or RRAMs, as shown in Fig. 3(a). FPGA area is reduced since NVMs have a five to 25 times higher density than SRAMs [15] and/or can be placed over CMOS transistors [13], [14], [17]. The nonvolatility of NVMs also saves the excessive leakage power during standby and the long configuration loading time at boot-up [15]. The main disadvantage of NVMs is the poor performance of write operations in terms of latency, energy, and endurance, although their read operations are competitive with SRAMs [15]. These drawbacks in write operations are masked when they are used to store FPGA configuration bits; this is because there are write accesses to these bits only during FPGA programming [3]. The number of programming cycles is expected to be small (e.g., < 500 [29]) for typical FPGA users. Note that in [12]–[18], NVMs are applied to not only the SRAMs in programmable interconnects but also the SRAMs in LBs.

2) Use NVMs as Programmable Switches: In [3], [4], and [19], 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. 3(b). 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 the ON state and the OFF state. The programmed resistance value can be kept either under operating voltages or without supply voltage due to nonvolatility. This kind of NVM use saves the area of SRAMs and also the pass transistors that build routing switches. The use of RRAM in this paper belongs to this category.

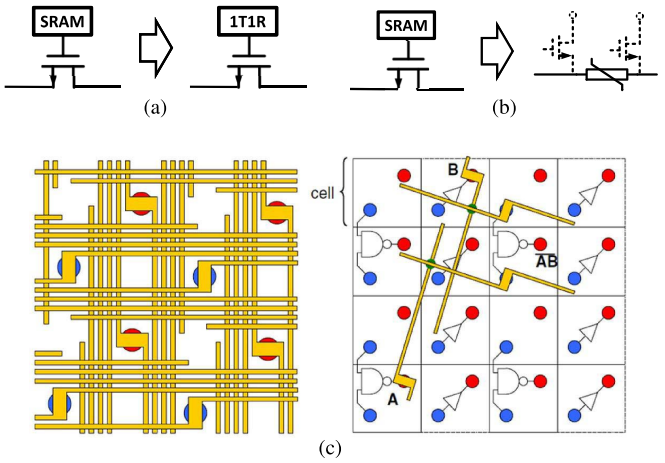


Fig. 3. FPGA with emerging nonvolatile memories. (a) Replace SRAMs with emerging NVMs [12]–[18]. (b) Use NVMs as programmable switches [3], [4], [19]–[21]. (c) Integration of CMOS, NVMs and nanowire in field-programmable nanowire interconnect [22], [23].

There are still many challenges to using NVMs as programmable switches in FPGAs. One of the key challenges is how to solve the tight space constraints in a layout where CMOS, NVMs, and metal wires are stacked together. Another key problem is the programmability of NVMs integrated in interconnects.[2] NVMs, such as PRAMs and RRAMs, have only two terminals. When they are used as routing switches, the two terminals are shared between the programming and signal paths. Their programming circuits need careful design; otherwise there will be interference between the two modes of operation. Furthermore, all of the above focuses on the replacement of routing switches with NVMs. However, as [3] concludes, routing buffers become a bottleneck after this replacement and thus need further improvement.

3) Integrate NVMs With Nanowire Crossbars: An FPNI [22], [23] is an attempt to further introduce nanowire crossbars to FPGAs. As shown in Fig. 3(c), the programmable interconnects are 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 that the feature size of nanowire crossbars be much smaller than that of CMOS logics to achieve higher RRAM density. Another problem is that, in the path between two cells, at least two long nanowires have to be driven, even if the two cells are adjacent. Because of the large capacitance of these long nanowires and fine granularity of logic cells, the FPGA performance decreases by 30%, as reported in [22]; the dynamic power increases by 17.5%, as reported in [6].

III. FPGA-RPI Architecture

In this paper we focus on the FPGA programmable interconnects, which are the dominant components in FPGA. As discussed in Section II-B.1, much work already exists that introduces emerging technologies to LBs [12]–[17]. Our work can be combined with these studies for further improvement. As analyzed in Section II-B.2, using NVMs as programmable switches is clearly a good idea. Although there are still problems with this method, we will show that they can be solved by the methodologies proposed in this paper.

A. Choice of NVMs

First, we need to decide which type of NVM to use. Different NVMs have their own advantages/shortcomings when used as programmable switches. For example, STTRAM usually has an ON/OFF ratio below 10. This ratio is sufficient for a memory application but far from enough for a routing switch. The NEM relay has the highest ON/OFF ratio since it shows almost zero off-current [30]. However, it has a complex structure with a large fabrication size of $92.5F^2$ (F is the feature size) and three metal layers [3]. The area of NEM relays can easily exceed that of the CMOS transistors below them, especially in cases where the area of an FPGA tile is much reduced by emerging technologies. In contrast, one RRAM device can be fabricated within an F^2 region and one metal layer [31]. The cell area of a PRAM can also be as small as that of an RRAM. However, the programming of PRAMs relies on temperature and is hard to control. In contrast, RRAMs can be set/reset by applying high positive/negative voltages [31], [32]. In addition, RRAMs provide more freedom in tuning their device properties than PRAMs. We can choose some fabrication technologies to manufacture RRAMs with a very small write latency ($<$5 ns [33]) or a very high endurance ($>10^{12}$ [34]) for the memory application. We can also choose some other fabrication technologies to manufacture RRAMs with a very high on/off ratio ($\sim 10^6$ [32]) for the application of routing switches. In contrast, the on/off ratio of PRAMs fabricated by different technologies usually remains around 10^3 [35]. There will be problems with both leakage and signal integrity if PRAMs are used as routing switches. Therefore we choose the RRAMs described in [32] to act as the routing switches in this paper.

B. Overall Architecture

In FPGA-RPI, the programmable interconnects are composed of three disjoint structures:

1) transistor-less programmable interconnects;
2) a programming grid;
3) an on-demand buffering architecture.

This composition takes routing buffers from programmable interconnects and puts them in a separate architecture. It enables an RRAM-friendly layout, sharing of more programming transistors, and on-demand buffer allocation (Sections III-C, III-D, and III-E, respectively). The transistor-less programmable interconnects correspond to SRAM-based configuration bits and MUX-based routing switches in conventional FPGAs. In FPGA-RPI, they are built by RRAMs and metal wires alone and are placed over CMOS transistors, as shown in Fig. 4. The programming grid and the on-demand buffering architecture will be optimized to

[2]NEM relays are exempt from this issue. An NEM relay can have four terminals—two for the reconfigurable connection and two for programming. Programming circuits of this kind of NVM in the application of routing switches can remain the same as those used in the memory application [3]. The main problem with NEM relays is their large cell sizes (Section III-A).

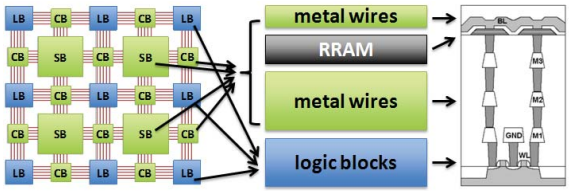


Fig. 4. In FPGA-RPI, SB and connection blocks in transistor-less programmable interconnects are placed over LBs in the same die according to existing RRAM fabrication structures [31]–[33], [36].

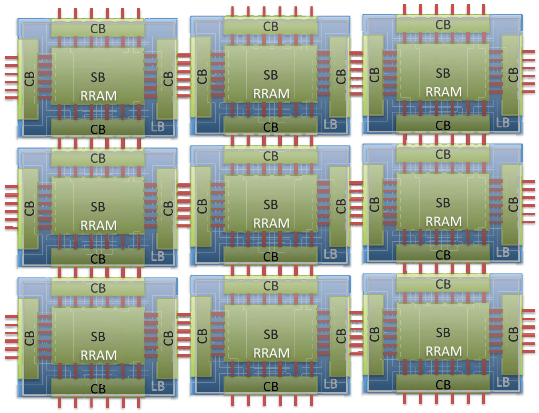


Fig. 5. Overview of FPGA-RPI where the FPGA area is mainly contributed by logic blocks instead of programmable interconnects.

consume much fewer CMOS transistors than LBs. FPGA-RPI then becomes a highly compact array, as shown in Fig. 5. The area of FPGA-RPI is almost solely determined by LBs, which take only 10%–50% of the conventional FPGA area [2]–[4].

C. RRAM-Friendly Layout Design

In the structure of the transistor-less programmable interconnects, RRAMs and metal wires are stacked over CMOS transistors. The layout will be very different from that of conventional FPGAs and will be applied with very tight space constraints. In this section we provide an RRAM-friendly layout design which solves these constraints and at the same time fits into the footprint of the CMOS transistors below. First we give the circuit schematic of the transistor-less programmable interconnects in Fig. 6. We use a universal type switch block [37] for the RRAM-friendly layout design. We are aware that the universal type needs 1.26% more routing tracks than the Wilton type [38]. However, the area of programmable interconnects in our FPGA-RPI will be significantly reduced. Our FPGA-RPI can afford the increase in routing channel width. The layout design for Fig. 6 is shown in Fig. 7 (the number of tracks in Fig. 7 is increased to six to show scalability). It addresses the following design issues.

1) To ease the FPGA-RPI 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 shown in the far right part of Fig. 4.
2) The placement of all metal wires is designed to avoid any blockage caused by a metal via. This blockage issue occurs only in SB where there are overlaps of metal wires from at least three layers. Fig. 8 shows how we address this issue. In the abstract structure of FPGA-RPI SB, as shown in Fig. 8(a), 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 3-D view in Fig. 8(b), the via blockages caused by vertical connections to the 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.
3) RRAMs can easily fit into the layout design in Fig. 7 without extra area overhead. As stated in Section III-A, the size of an RRAM cell can be close to or even smaller than the width of a metal wire [31].
4) Each metal layer in M5 to M9 consists of a set of parallel metal wires without any turns. This reduces fabrication complexity and avoids the high resistance of turning points.
5) The metal wires in the metal layers M1 to M4 within the LB boundary in Fig. 7 are left for the routing within logic LBs. They are sufficient for practical FPGA LBs. We verify this by producing a compact layout of the configurable LBs used in the Xilinx Virtex6 and Virtex7 families according to their datasheets [28], [39].
6) Though our design uses multiple metal layers, a signal rarely goes through many metal layers. While a signal

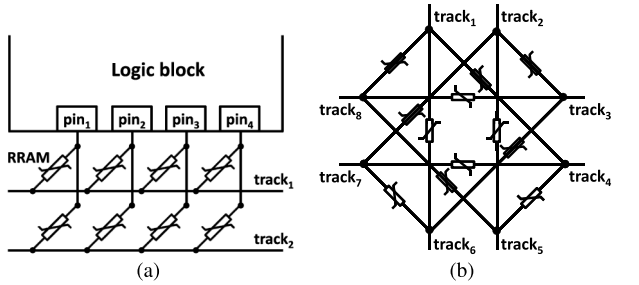


Fig. 6. Circuit schematics of transistor-less programmable interconnects. (a) RRAM-based connection block. (b) RRAM-based switch block.

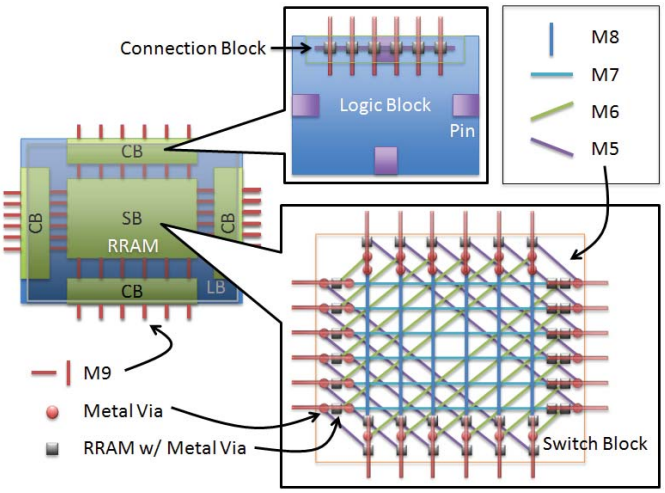


Fig. 7. RRAM-friendly layout design of the programmable interconnects in FPGA-RPI using the RRAM fabrication structure shown in Fig. 4. Here, each metal via (marked as a red point) at the intersection between wires of M9 and M5–M8 refers to a vertical connection between them through metal via(s).

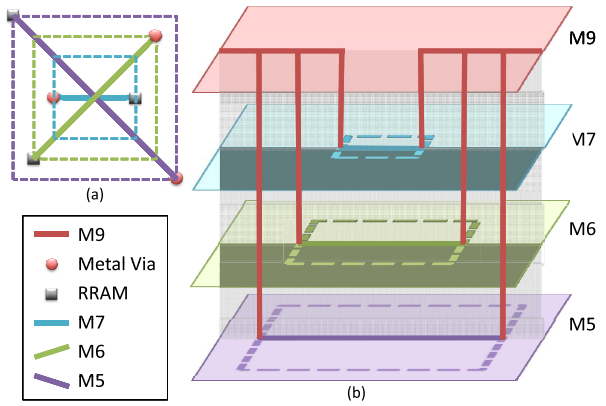


Fig. 8. (a) Abstract structure of FPGA-RPI SB (M9 and M8 are omitted for clarity). (b) 3-D view to show how via blockages are avoided by metal wires in our SB design (RRAM layer is omitted for clarity). For demonstration purposes, vertical connections are bounded in a plane to construct the case most challenging for via blockage avoidance.

is transmitted from one LB to another, the straight paths in SB are used most frequently. To save extra latency on metal via, we place these paths in M8 and M7, which are close to the RRAM layer. Only when a signal reaches a LB does it need to access metal layers at lower levels, i.e., M1–M4, from the RRAM layer.

7) The metal wires in the top metal layer M9 in the middle of the switch block in Fig. 7 are left for the power and clock grids.
8) The CMOS and metal wire resources in the gaps between LB boundaries in Fig. 5 are left for the programming grid and the on-demand buffering architecture.
9) The RRAMs in the middle of the switch block in Fig. 7 can be used to implement BLEs and local interconnects in LBs for further improvement (Section VI).

D. Programming Schematic

1) Basic Programming Schematic: To program the RRAMs integrated in programmable interconnects, there must be a programming transistor at each of the two terminals of an RRAM, as shown in Fig. 9. The RRAM can be switched between the ON and OFF states by turning on both the programming transistors and applying the programming voltages [paired as $(V_p, 0)$ or $(0, V_p)$], as shown in Fig. 9. The values of programming voltages may be the same as those in [4]. Gaillardon *et al.* [19] propose a different schematic where some PRAMs are not connected directly to any programming transistors.[3] The paths for programming currents going to these NVMs need to contain some other NVMs that have direct accesses to programming transistors. When we want to program the NVMs to the OFF state in the path, some NVMs will be turned off before the other NVMs and will block the programming current for the other NVMs. Therefore, we only consider the design of a programming grid that can guarantee that each of the two terminals of any RRAM

[3]Though [19] is based on PRAMs, the programming schematics of RRAMs and PRAMs could be instructive to each other since both these NVMs are two-terminal devices (Section III-A).

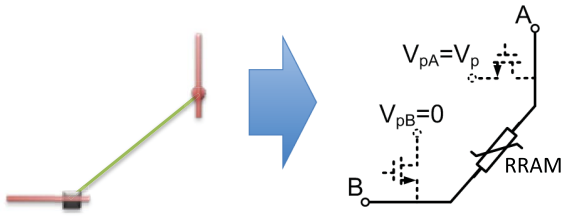


Fig. 9. Programming circuits for RRAMs in interconnects. The two programming transistors will apply programming voltages to program the RRAM between them. V_p is the threshold voltage that switches the RRAM state.

connects to a programming transistor. A basic programming schematic is to allocate two programming transistors for each RRAM.

2) Programming Transistor Sharing: The problem with the basic programming grid is the large area of two programming transistors compared to a single RRAM cell which undermines the area savings brought by the high density of RRAMs. We built a programming grid to allow RRAMs to share programming transistors. In the connection block in Fig. 6(a), there are six nodes (four LB pins and two routing tracks). We allocate only six programming transistors, one per node. Then each terminal of the eight RRAMs connects to a programming transistor. In this case, each programming transistor is shared by all the RRAMs in the same row/column. This kind of sharing in connection blocks has already been proposed in [4]. However, Tanachutiwat *et al.* [4] did not apply the sharing to SB, in which the area is four times larger than connection blocks. The reason is that, in their work, the sharing paths are blocked by the transistors in SB, as shown in Fig. 10. This problem does not exist in our transistor-less programmable interconnects, where buffers are taken away and placed in a separate structure. As shown in Fig. 11, a programming transistor in our architecture can be shared by RRAMs, not only in one switch block but also across adjacent SB and connection blocks. In our transistor-less programmable interconnects, any RRAM is between either a track in a routing channel or a pin of a LB, and it never touches an intermediate node in the interconnects. We allocate programming transistors only at these nodes, and then there will be programming transistors at the two terminals of all RRAMs. Table I shows a comparison of programming transistor counts in programmable interconnects based on CMOS, the schematic in [4], and this paper. We can achieve a 12× programming transistor reduction in SB compared to [4]. Note that this improvement also stems from the 50% reduction of RRAMs in our transistor-less SB. In [4], two unidirectional buffers are used for the connection between two tracks, and each buffer needs one RRAM to control.[4] In our transistor-less SB, only one RRAM is needed to connect two tracks since there are no unidirectional buffers.

3) Programming Transistor Selection: An 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

[4]The switch block design in [4] is being replaced by directional switches [40] and is not used in this paper either.

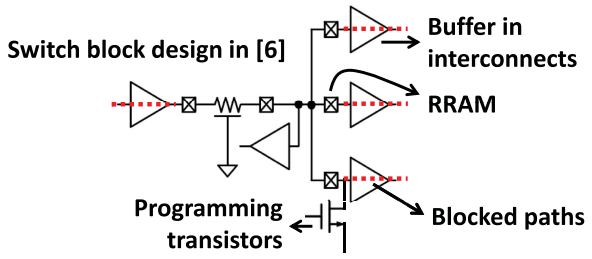


Fig. 10. Sharing paths blocked by transistors (marked as dotted lines), as in [4].

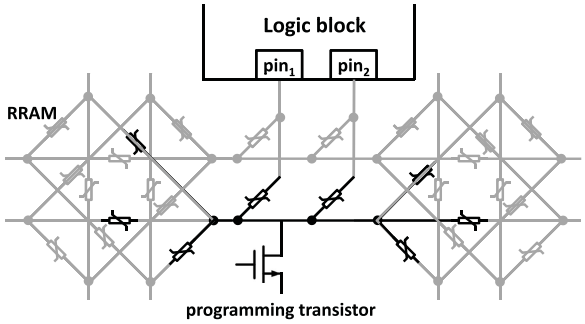


Fig. 11. One programming transistor shared by RRAMs among SB and connection blocks, as in this paper.

TABLE I

COMPARISON OF PROGRAMMING TRANSISTOR COUNTS IN PROGRAMMABLE INTERCONNECTS BASED ON CMOS, THE SCHEMATIC IN [4], AND THIS PAPER. N REPRESENTS THE ROUTING CHANNEL WIDTH. M REPRESENTS THE NUMBER OF LB PINS ACCESSIBLE BY ONE ROUTING CHANNEL

	CMOS	Work in [4]	This Paper
$N \times M$ MUX in CB	$M(N+13\sqrt{N})$	$N+M$	$N+M$
$N \times N$ SB	$64N$	$24N$	$2N$
100×14 MUX in CB	3220	114	114
100×100 SB	6400	2400	200
One tile	12840	2628	228

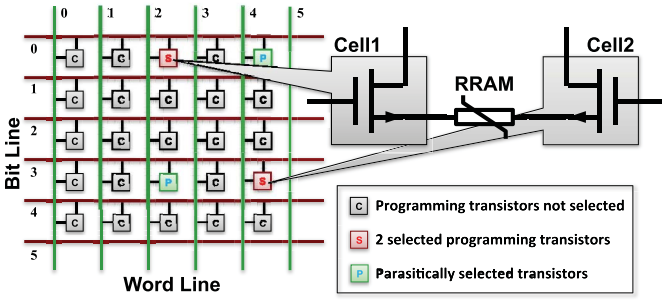


Fig. 12. Row/column addressing for the programming transistor array of RRAMs in interconnects. If two programming transistors in one array are selected simultaneously, two other transistors will be selected parasitically.

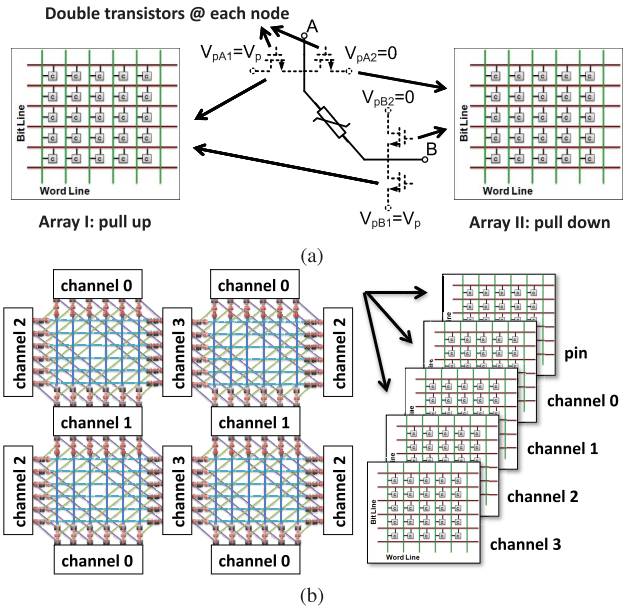


Fig. 13. Two solutions to solve parasitic selection. (a) Double programming transistors at each programming node. (b) Partition programming transistors into five banks, where the two programming transistors at the terminals of any arbitrary RRAM lie in two different banks.

(or RRAM cells), the row–column structure is often used to select a programming transistor, similar to Fig. 12. Contemporary FPGAs can contain 10^8 configuration bits [41]. The selecting circuit will require only $\sim 2 \times 10^4$ drivers (one per row/column) for the entire FPGA. It will not significantly increase the area of the programming grid which is dominated by the $O(10^8)$ programming transistors for all RRAMs (or for SRAMs in conventional FPGAs). Note that in the memory application of RRAMs, an RRAM is dedicated to only one programming transistor. To program an RRAM that is used as a routing switch in programmable interconnects, we need to select two programming transistors, as discussed in Section III-D.1. This is equivalent to a dual-port random access memory. This kind of access may cause problems in the structure designed for single-port access, as shown in Fig. 12.

To realize the dual-port random access without malfunction, one choice is to follow a simple solution in the memory application, where the programming transistors are doubled in all cells, as shown in Fig. 13(a).

However, we could have a better choice that utilizes the structure of our programming grid to completely eliminate the extra increase in programming transistor counts. In the memory application, pseudo dual-port accesses are supported in a single-port memory if there are multiple memory banks, and the memory controller can guarantee that the two simultaneous requests always access two different banks. For FPGA-RPI, we can also partition the programming transistors into different banks. The partitioning must guarantee that the two programming transistors of any arbitrary RRAM belong to different banks. We create a feasible partition of five banks based on the structure of our programming grid, as shown in in Fig. 13(b). Recall that in Section III-D.2, we allocate programming transistors at all pins of LBs and all tracks in routing channels. The programming transistors in y-directional routing channels are assigned to two banks named channel 0 and channel 1 by a parity of channel indices; so are the programming transistors in x-directional routing channels. The programming transistors at the LB pins are assigned to a separate bank.

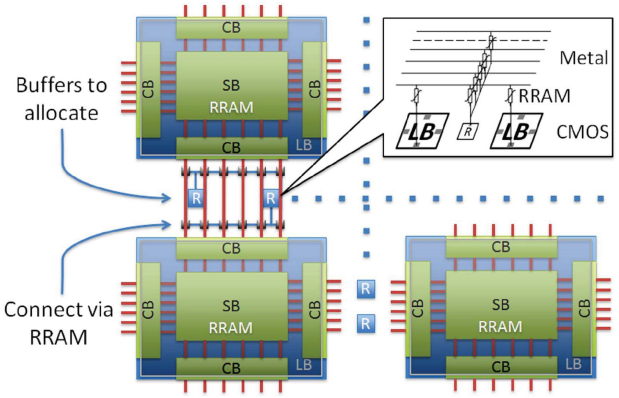


Fig. 14. Circuit structure and layout of the on-demand buffering architecture in our FPGA-RPI. R = repeater.

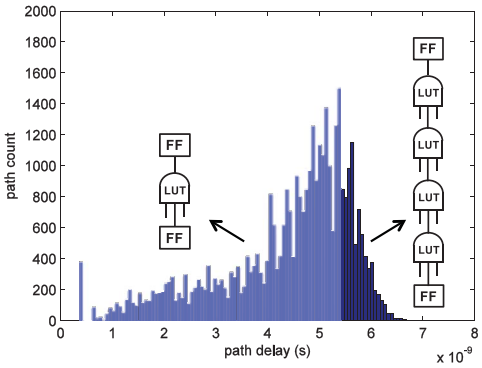


Fig. 15. Delay distribution of all paths in a typical application for mapping onto FPGA. Most paths can be relaxed for less use of buffers.

E. On-Demand Buffering Architecture

In our FPGA-RPI, buffers are taken away from programmable interconnects and put in the on-demand buffering architecture. This section will discuss the structure, the prerequisite, and the benefits of this architecture.

1) Circuit Schematic and Layout: Fig. 14 shows the circuit structure and layout of the on-demand buffering architecture in our FPGA-RPI. 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 the routing paths of the user application that is implemented on FPGA-RPI. This mechanism brings benefits of both area and performance; these will be analyzed in the following sections.

To simply the interconnects between shared buffers and routing tracks, we attach each buffer to a routing track via one single RRAM. Therefore we choose the regenerative feedback repeater described in [42] as our buffer cell. This cell design has one signal terminal and can provide drive in both signal directions, as opposed to the conventional unidirectional buffers. As a result, it saves at least half the buffers by serving the function of two complementary unidirectional buffers with only one repeater.

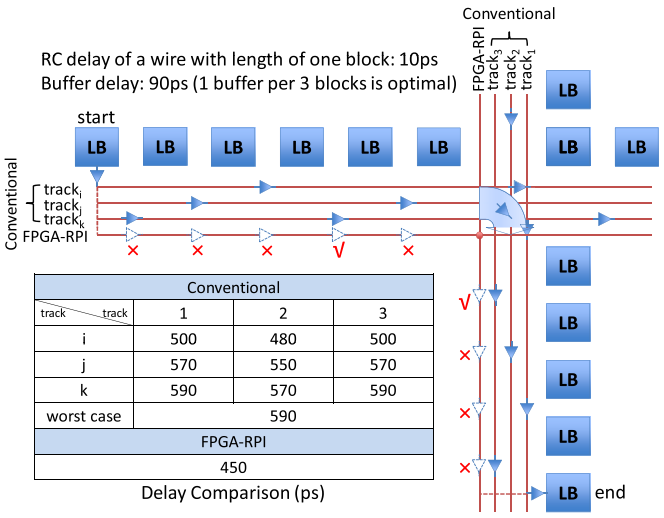
2) Overhead and Prerequisite: The area overheads of the on-demand buffering architecture are composed of two parts. First, the buffers themselves consume the CMOS area. Second, we also need to allocate programming transistors to program the RRAM-based connections between the buffers and the transistor-less programmable interconnects. The connections are the same as RRAM-based multiplexers used in connection blocks in Fig. 6(a), and their overhead is small, as discussed in Section III-D.2. Recall the comparison in Table I and assume every channel has 100 tracks and 10 buffers. The connection between the tracks and buffers in one channel is implemented by a 100×10 multiplexer which costs 110 programming transistors. Among them, 100 programming transistors are placed at the 100 routing tracks and can be shared with SB (similar to the sharing between SB and connect blocks discussed in Section III-D.2). Therefore, the exact area overhead of the programming part of the on-demand buffering is only 10 transistors per routing channel.

The on-demand buffering architecture can also be implemented in conventional CMOS technology, but the area overhead of the multiplexers between buffers and the rest of the interconnects is much higher. According to the cost of the $N \times M$ MUX in Table I, the area overhead is 2300 extra transistors. This large overhead undermines the benefits of the on-demand buffering architecture and prevents this technology from being applied to conventional FPGAs. In summary, the on-demand buffering architecture requires efficient programmable connections between buffers and routing tracks enabled by emerging technologies like RRAMs.

3) Savings of Unnecessary Buffers: In conventional FPGAs, the locations of buffers in programmable interconnects are predetermined during FPGA design. For example, as shown in Fig. 1, each switch from one track segment to another in a 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. But this may not be always necessary, especially in short routing paths. Buffers are needed only in the limited number of routing paths that are long enough to exhibit a quadratic increase in RC delay.

Buffers with fixed locations also result in unnecessary buffers in noncritical paths. Fig. 15 shows a typical delay distribution of all paths in a typical MCNC benchmark "tseng" mapped onto a conventional FPGA at 32-nm technology node. The paths with less criticality (light color in Fig. 15) usually go from an FF through a few LBs to another FF, and can be relaxed for less use of buffers. In conjunction with the paragraph above, we conclude that a routing path needs buffers only if it is a long and critical path.

4) Performance Benefit: The fixed locations of buffers in conventional FPGAs also lead to a deviation from optimal timing results. A case study in Fig. 16 shows the timing benefit offered by the on-demand buffering architecture in FPGA-RPI when compared to the fixed buffer pattern in a 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 $0.5 R_{\text{wire}} C_{\text{wire}} = 10$ ps, and that the buffers in interconnects are considered ideal buffers with infinite drive, no input/output capacitance, and a fixed intrinsic delay of 90 ps. The optimal



Conventional			
track \ track	1	2	3
i	500	480	500
j	570	550	570
k	590	570	590
worst case	590		
FPGA-RPI			
450			

Fig. 16. Performance comparison between the fixed buffer pattern in conventional FPGAs and the on-demand buffering architecture in FPGA-RPI. To simplify, only three routings tracks per channel are shown for the conventional FPGA, and one track is shown for our FPGA-RPI. Note that the buffers in FPGA-RPI are also shared by other tracks in the same channel (not shown here, but discussed in Section III-E.1). ✓ = buffer allocated. ✗ = not allocated.

length of a wire between two adjacent buffers would be

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

In conventional FPGAs, we suppose that the patterns of prefabricated buffers in interconnects already follow this optimal result to distribute buffers evenly with a distance of three blocks (Fig. 16).[5] Since the starting point of each net is unknown before fabrication (the offset can be 0, 1, or 2), the patterns are staggered among different tracks in the same channel, as shown in Fig. 16. When a net is driven by routing congestion to a specific track, it is forced to use all the buffers in the track. The table in Fig. 16 lists all the possible delays of a net from the logic block start to the LB end according to the settings in the conventional FPGA discussed above. The buffers at the LB pins are not counted in the delay calculation. As shown in Fig. 16, the worst case delay in a conventional FPGA is 590 ps. On the contrary, the on-demand buffering architecture in our FPGA-RPI will not suffer from the deviation from the optimal buffer placement in interconnects as does the conventional FPGA. Buffers are allocated exactly one per three blocks, resulting in a total delay of only 450 ps in Fig. 16.

5) Glitch Hazard and Elimination: The regenerative feedback repeater proposed in [42] is chosen as the buffer cell in our buffering architecture. It is vulnerable to glitches that are shorter than half of the repeater's delay t_d. The authors of [42] concluded that the use of the repeater was limited to FPGA architectures that use self-timing or clocking to eliminate glitches. But the experiments in [42] were based on 1.2 μm, which is more than 10 generations away from the current technology node. We revisit this glitch issue at the 32-nm technology node and find that this hazard is much alleviated. As shown in Fig. 17, when there is a glitch from combinational logics within an LB, the glitch will go through the *RC* network formed by routing wires before reaching a regenerative feedback repeater. The *RC* network will act as a low-pass filter and put a lower bound on the width of a signal that can pass the network. This lower bound can be estimated by the *RC* delay of the network. According to ITRS [43], the *RC* delay of the metal wire with a constant length increases by 50% with every generation advancement. Meanwhile, the delay of gate decreases as scaling down continuous. These two trends significantly reduce the probability that glitches exposed to a repeater are shorter than $0.5t_d$. In addition, our on-demand buffering architecture guarantees that the wire segments before any repeater are long enough to filter out all glitches that are shorter than $0.5t_d$. The timing optimization in Section III-E.4 is achieved when buffers in routing paths are allocated with an *RC* delay equal to t_d per segment. Fig. 18 shows the simulation results of the scenario in Fig. 17. We set the width of the glitch coming from the LB to be $0.4t_d$. The buffer cells in our architecture can still work correctly. Note that in conventional FPGAs without our on-demand buffering architecture, $< 0.5t_d$ glitches may still be exposed to regenerative feedback repeaters due to fixed repeater locations in interconnects. For example, the leftmost repeater in track$_k$ in Fig. 16 may have glitch hazard since it immediately follows its predecessor without protection from wires.

[5]In fact, it is not always the case in conventional FPGAs. The distribution of buffers is not usually designed for optimal delay for the sake of area reduction and power consumption savings.

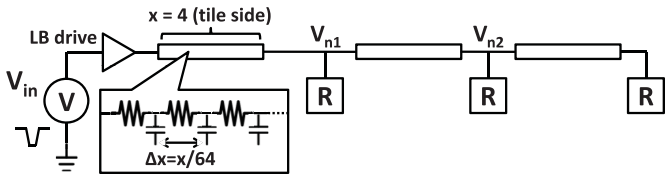


Fig. 17. Glitch elimination. Glitches $< 0.5t_d$ are filtered out by the *RC* network of wire segments before regenerative feedback repeaters. Our on-demand buffering architecture always guarantees sufficient *RC* delay of wires for this filtering.

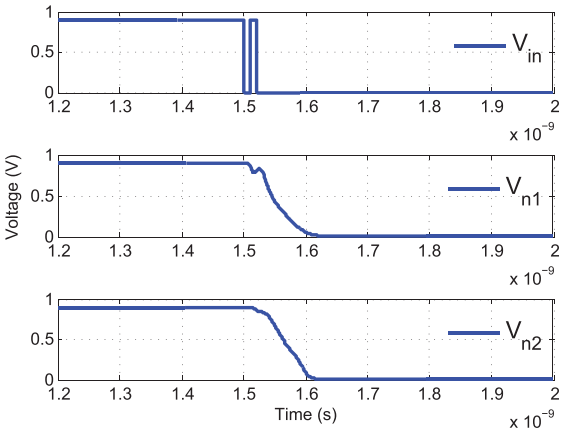


Fig. 18. SPICE simulation results of the scenario in Fig. 17. Simulation settings can be found in Section V-A. Wire segments are modeled as a distributed *RC* network.

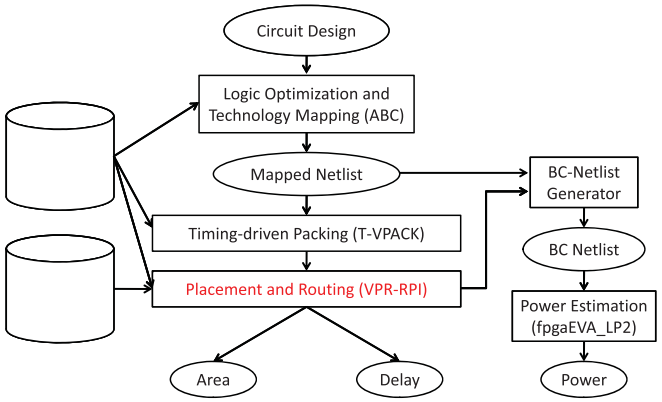


Fig. 19. Customized CAD flow for FPGA-RPI. Enhanced P&R tools are developed for FPGA-RPI.

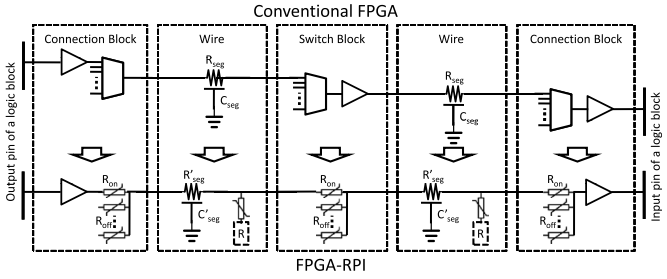


Fig. 20. Extracted equivalent circuit model of the FPGA-RPI routing structure and comparison with a conventional FPGA.

IV. Design Tool Support

A. CAD Flow

To evaluate performance and energy consumption of FPGA-RPI 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 [27]. We adopt it and modify it into a flow for our FPGA-RPI, as shown in Fig. 19. The input of the flow is the user application to be implemented on FPGA-RPI. The output includes the placement and routing (P&R) result used for programming the application on FPGA-RPI, as well as an estimation of area, delay, and power consumption. The main changes of FPGA-RPI compared to the conventional FPGA are in the programmable interconnects, and we develop a new P&R tool VPR-RPI to adapt to the changes. We will introduce the features of this tool in Sections IV-B, IV-C, and IV-D.

B. Equivalent Circuit Model for Interconnect

To perform the timing and power analysis for FPGA-RPI, we first develop an equivalent circuit model for the FPGA-RPI routing structure. Fig. 20 shows the equivalent circuit of a representative routing path from the output pin of an LB to the input pin of another LB in FPGA-RPI and makes a comparison with that of a conventional FPGA. In the circuit model, the MUXs in connection blocks and SB are replaced by the RRAMs, of which one is the ON state and the others are at the OFF state. The wire segments are still modeled as distributed *RC* lines, but with buffers if allocated. The *RC* values of wire segments are updated to reflect the tile area reduction in FPGA-RPI.

C. On-Demand Buffer Allocation Algorithm

1) Role in the Flow: To fully utilize the benefits of the on-demand buffering architecture in FPGA-RPI, we develop an algorithm to choose the suitable positions for buffers that will be allocated in programmable interconnects. This will be performed after FPGA routing. Given all the nets routed in programmable interconnects, the goal is to find a buffer option that minimizes the critical delay. The constraint is the total number of prefabricated buffers in each channel. Since the on-demand buffering architecture in FPGA-RPI allows a buffer allocation that is similar to that in ASIC, we considered implanting into FPGA-RPI several state-of-art methods of ASIC buffer allocation [44]–[46]. After comparison of their optimization scenarios with ours, we decided to create a new algorithm that borrows the concepts of buffer placement for minimal delay in ASICs [44] and the negotiation-based iterative approach for FPGA routing [47].

2) Optimization Without Resource Constraints: We first focus on the delay minimization without constraints of buffer resource. As analyzed in Section III-E.4, more buffers do not necessarily lead to better timing. We extend a method of buffer placement in ASIC [44] to FPGA-RPI. Buffers are placed in suitable positions in each routed net in FPGA-RPI that connects an output pin and multiple input pins of LBs together through programmable interconnects. Each net is equivalent to a routing tree with one source and multiple sinks. 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 capacitance). During the combination of buffer options from two subtrees in the search, if both delay and $C_{\text{downstream}}$ of a buffer option are larger than those of another option, the former one will be pruned. The complexity of the algorithm is $O(B^2)$, where B is the total number of legal buffer positions in a routing tree.

3) Consideration of Resource Constraints: If the number of buffer allocated in Section IV-C.2 exceeds that of available buffers in some regions, we need to consider the constraint of buffer resource during delay minimization. The method that we use is inspired by the negotiation-based routing iteration procedure in [47]. 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 for pruning buffer options (the other metric is still $C_{\text{downstream}}$). The total cost of a buffer option y for a subtree i of a routing tree is expressed as

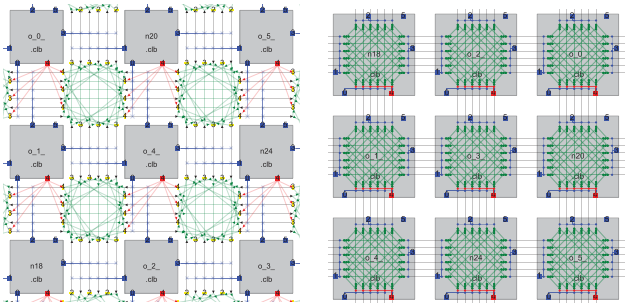
$$\begin{aligned}\text{Cost}(y) =& \text{Crit}(i) \cdot \text{delay}(y) \\ &+ [1 - \text{Crit}(i)] \cdot \text{DNF} \cdot h(y) \cdot p(y)\end{aligned} \tag{1}$$

where $\text{Crit}(i)$ is the largest criticality among all the paths through which subtree i goes. Criticality of a timing-critical path is close to 1, and criticality of a noncritical path is close

Algorithm 1: A brief outline of the buffer allocation algorithm under the constraint of buffer resources.

Input : Routing trees $RT_1, RT_2, RT_3, ..., RT_n$
Output: A buffer option $Y = \{y_1, y_2, ..., y_n\}$ that defines buffer allocation in each RT_i

```
while ∃ overused buffers do
    foreach RT_i do
        rip-up y_i and update buffer congestion p(y);
        y_i = AllocBuf (RT_i) in [47] with Cost(y) in Eq. 1 ;
        update buffer congestion p(y);
    end
    update historical buffer congestion h(y);
end
```



(a) (b)

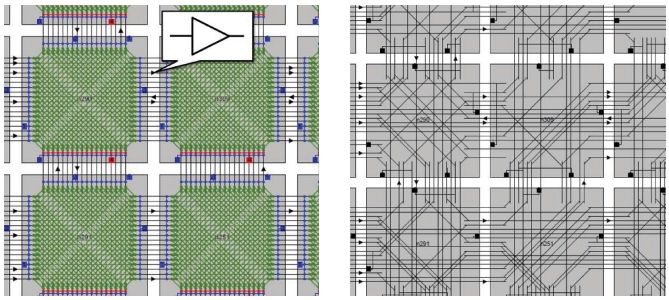
Fig. 21. Overview of the two tools VPR and VPR-RPI. (a) VPR for conventional FPGA. (b) Our VPR-RPI for FPGA-RPI.

to zero. DNF is the delay normalization factor. $h(y)$ and $p(y)$ are the historical and present overuse of buffer resources, respectively, in buffer option y. $p(y)$ will grow as iteration continues, and the buffers in uncritical paths will be pushed away from congested regions or even be removed. A brief outline of our methodology is provided in Algorithm 1.

D. VPR-RPI: the P&R Tool for FPGA-RPI

To deal with the novel routing structure of FPGA-RPI in the P&R step of CAD flow, we developed an advanced tool named VPR-RPI (VPR for RRAM-based programmable interconnects) on the basis of the state-of-art FPGA P&R tool in academia, VPR [27]. The main contributions of this tool are as follows.

1) VPR-RPI can deal with the routing graph of FPGA-RPI as shown in Fig. 21(b). In this figure, the gray blocks are I/O blocks and LBs of FPGA, and the diagonal wires are the routing paths available in SB. The SB and connection blocks in VPR-RPI are placed over LBs and provide connections for LBs nearby in a different way than that of the conventional FPGA shown in Fig. 21(a).
2) VPR-RPI is integrated with the algorithm to perform the on-demand buffer allocation described in Section IV-C. The tool can display where buffers are needed for allocation in the final routing result, as shown in Fig. 22. The positions for the allocation of buffers are marked with a black delta shape. The interconnect view of the routing result in Fig. 22(b) is quite similar to that of ASIC. We describe the good performance of FPGA-RPI in Section V.



(a) (b)

Fig. 22. View of the buffer allocation result generated by VPR-RPI. (a) Routing resource view. (b) Routing result view.

V. Simulation Results

A. Settings

We tried to mimic the architecture of the commercial FPGA, the Xilinx Virtex7 family, for meaningful evaluations. It uses 28-nm processes [28]. Due to lack of technology libraries, we are unable to choose the exact 28-nm models for our experiments. Instead, we choose the nearby 32-nm technology node. We use the settings of eight six-input look-up tables (6-LUTs) per LB; these are used starting from Virtex5 to the most recent Virtex7 [28]. The routing channel width is obtained with the help of the FPGA Editor in the Xilinx ISE environment. We refer to the intelligent FPGA architecture repository (iFAR) [48] for other less important architectural settings that are not provided by the Virtex7 datasheets [28]. We estimate the timing parameters using the lookup tables in ITRS 2011 [43] and SPICE simulation with PTM device models [49], [50]. The RRAM model is extracted from the measurement results of the RRAMs fabricated by the process in [32]. Its ON resistance is $\sim 10^3$ Ω and its OFF resistance is $\sim 10^9 \Omega$. We also perform a sensitivity analysis on the RRAM parameters. The 20 largest MCNC benchmark circuits [51] are used as the input of the CAD flow for conventional FPGA and FPGA-RPI, and comparisons are made on the output of the CAD flow from the aspects of area, delay, and power.

B. Optimize the On-Demand Buffering Architecture

Compared to a conventional FPGA, FPGA-RPI has a unique on-demand buffering architecture. Before evaluation, we need to explore the new parameters related to this architecture. We found that a uniform distribution of buffers over routing channels best serves the buffer demands of different applications, similar to the uniform distribution of routing channel widths arrived at [27]. 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. Based on the observation that a smaller buffer size will increase the number of buffers demanded by each channel, we explore these two parameters simultaneously. Note that both of the parameters have an impact on both the area and performance of FPGA-RPI. Therefore, we use the area–delay product as the optimization goal. Fig. 23 shows the result. Here, buffer size is normalized to the optimal size of

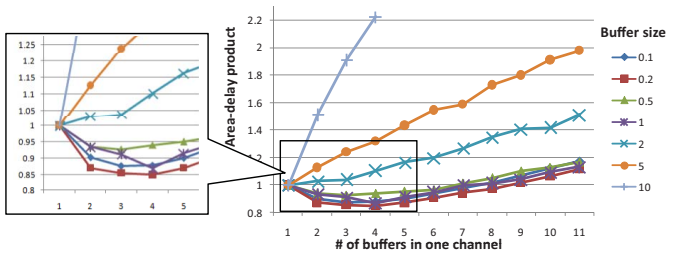


Fig. 23. Exploration of the impact of buffer sizing and richness on the area–delay product. Buffer sizes and area–delay products are normalized.

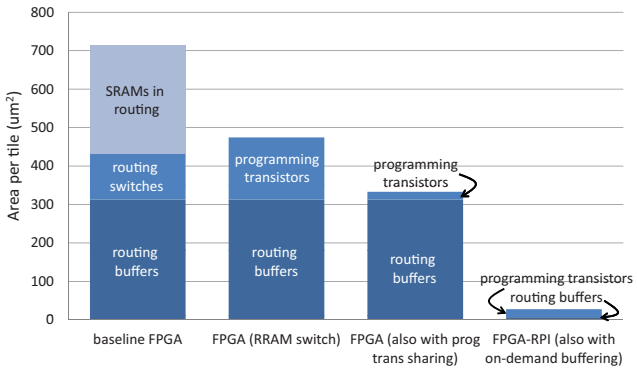


Fig. 24. Area comparison of programmable interconnects in a single tile.

the conventional FPGA in iFAR [48]. The area–delay product is calculated from the geometric mean of the results over the 20 benchmarks and normalized to the value of the case without any buffer allocated. The drop of the area–delay product curve in Fig. 23 is a result of the timing improvement when buffer demand is not fully satisfied. The rise of the curve is a result of the larger area with more buffers. Based on the enlarged portion of Fig. 23, we determine that in our FPGA-RPI the number of buffers per channel is 4, and the buffer size is 0.2 times that used in a conventional FPGA (based on the optimal settings in iFAR).

C. Evaluation of FPGA-RPI

1) Footprint: Fig. 24 provides an area comparison of programmable interconnects in different architecture settings. It shows the area reduction breakdown of the FPGA programmable interconnects achieved by different technologies proposed in this paper. Starting from the baseline FPGA, we first replace the routing switches and their SRAMs in programmable interconnects with RRAM-based switches (Sections II-B.2 and III-B). These RRAMs are placed over CMOS transistors, and their footprint fits into that of the CMOS transistors below; this is guaranteed by our RRAM-friendly layout design in Section III-C. So we can safely skip the area of RRAMs and metal wires in this evaluation. The contribution of RRAMs to the CMOS footprint will be their programming transistors shown in Fig. 24. Then we apply our programming transistor sharing (Section III-D.2). Finally, we implement our on-demand buffering architecture (Section III-E.1). The final area savings are 96% of the programmable interconnects, which corresponds to 50% of the total FPGA area (recall the FPGA area breakdown in Fig. 2).

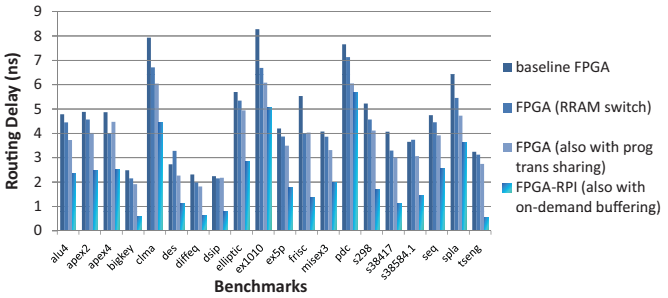


Fig. 25. Comparison of the delays contributed by programmable interconnects in critical paths in FPGAs over benchmarks.

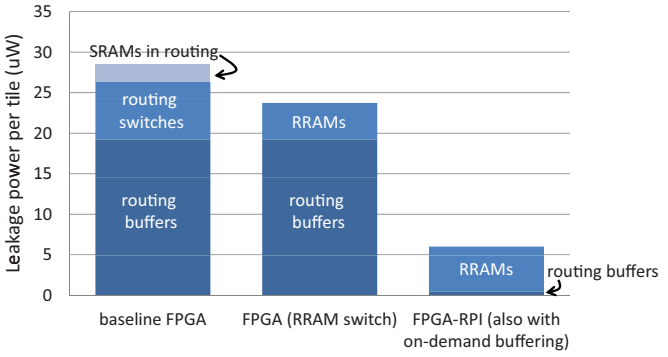


Fig. 26. Comparison of the leakage powers contributed by programmable interconnects in a single tile.

2) Performance: Fig. 25 shows a performance comparison of programmable interconnects in different architecture settings.[6] The speedup of FPGAs with the three technologies applied step by step mainly stems from the shorter signal paths due to the area reduction. This observation corresponds with that of the 3-D FPGAs [2]. In addition, the on-demand buffering architecture also drives the buffer solutions closer to the optimal. The geometric mean of the delay reduction achieved by our FPGA-RPI is 55%.

3) Power Consumption: Fig. 26 shows a leakage power comparison of programmable interconnects in different architecture settings. Note that the programming transistors of RRAMs are not counted in the evaluations of leakage power since the programming grids will be completely turned off during the operation of FPGAs. Our FPGA-RPI reduces the leakage power of programmable interconnects by 79%. Note that RRAM is also nonvolatile. Power gating can be applied to RRAM-based FPGAs to further reduce the leakage power during standby. Fig. 27 shows a dynamic power comparison of programmable interconnects in different architecture settings. All three technologies applied step by step lead to area reduction and thus smaller capacitances of shorter wire interconnects. In addition, the on-demand buffering architecture removes many unnecessary buffers in both short paths and uncritical paths. The geometric mean of the dynamic power savings achieved by our FPGA-RPI is 56%.

[6]The improvement trend is not always consistent for every benchmark due to certain delay noise in VPR [52].

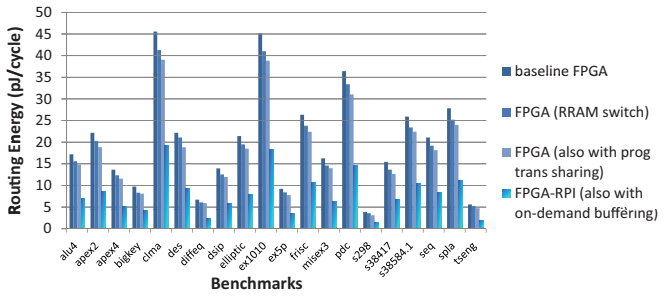


Fig. 27. Comparison of the dynamic powers contributed by programmable interconnects in FPGAs over benchmarks. To separate the impact of the clock period from the dynamic powers, we use pJ/cycle as the metric.

D. Impact on LB Architectural Design

The increasing overhead of programmable interconnects pushes FPGA vendors to larger and more complicated LBs. For example, the Xilinx FPGAs in the Virtex family before Virtex5 put eight 4-input look-up tables (4-LUTs) in one LB [53]. The FPGAs starting from Virtex5 to the most recent Virtex7 switched to eight 6-LUTs in one LB [28]. The proportion of LBs in today's FPGAs is close to 50%, and this large proportion masks the benefits coming from the improved programmable interconnects. In the literature [2]–[4], researchers intentionally chose smaller LBs as their architecture settings to highlight the benefits from programmable interconnects. Here, we provide a more complete study to see which LBs will lead to a smaller footprint, higher speed, and lower power consumption for both logic and routing parts. Fig. 28 shows the overall footprints, delays, and power consumptions in the FPGA architectures with LBs made up of 6-LUTs and 4-LUTs. Before we apply our RRAM-based programmable interconnects, the FPGA architecture with more complicated LBs is generally comparable or even better than that with simpler LBs. After we improve the programmable interconnects with our FPGA-RPI, the FPGA architecture with simpler LBs is better since it receives more benefits from our techniques. This architectural change further reduces the total area, delay, and power consumption by averages of 34%, 7.6%, and 25%, respectively.

E. Sensitivity Analysis

Since the 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. The key parameter of RRAMs is the ON/OFF ratio. We sweep the ON/OFF ratio by two orders of magnitude, and Fig. 29 shows the impact. The analysis is performed on the FPGA-RPI with 4-LUT, and the results are shown in the form of gains compared to the conventional FPGA with 6-LUT. On each curve in Fig. 29, the ON/OFF ratio is fixed and there is a trade-off between the ON resistance R_{ON} and the OFF resistance R_{OFF}. Larger R_{OFF} leads to smaller leakage power and larger energy savings. Larger R_{ON} leads to larger resistance on signal paths and smaller speedup. We suggest trading off energy savings for larger speedup since we can apply power gating to save more energy by using the nonvolatility of RRAMs. When

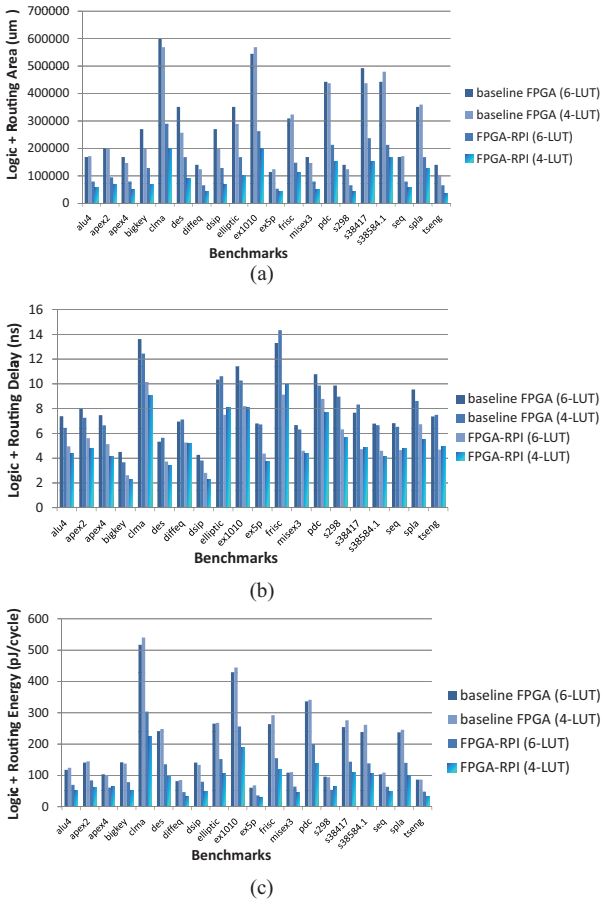


Fig. 28. Comparisons of FPGA architectures with more complicated LBs and simpler LBs. (a) Total footprints. (b) Total delays. (c) Total energy consumptions per clock cycle (including both leakge and dynamic, assumed to operate at highest speed).

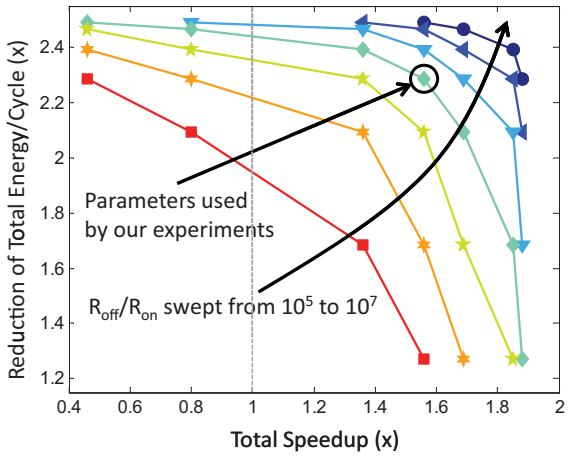


Fig. 29. Sensitivity analysis of the RRAMs ON/OFF ratio for FPGA-RPI. Each line corresponds with a fixed ON/OFF ratio. There is a trade-off between delay and power savings on each line.

we switch from a line of a smaller ON/OFF ratio to one of a larger ratio in Fig. 29, we will achieve improvement in both performance and power savings.

VI. Interconnect Improvement in LBs

There are local interconnects in LBs. We can improve the local interconnects as we do for the global interconnects. For

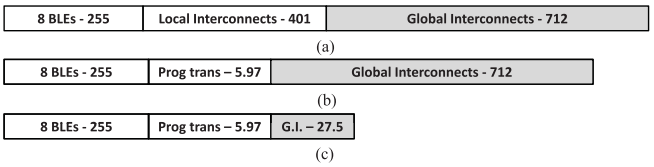


Fig. 30. Area comparison of a single tile in an FPGA. The white parts belong to LBs. Each LB contains eight 6-LUTs. The number of LB inputs is 27, optimized by design experience in [27] and [48]. (a) Baseline FPGA. Total area = 1370 μm^2. (b) FPGA with RRAM-based local interconnects. Total area = 973 μm^2. (c) FPGA-RPI also with our renovated global interconnects. Total area = 289 μm^2.

the sake of simplicity, we consider a basic LB model taken from [27]. Note that there are large MUXes between the input pins of LBs and those of BLEs. They allow each of the BLE inputs to connect to any of the LB inputs or any of the BLE outputs. This feature of being fully connected simplifies the design of CAD tools and is implemented in many commercial FPGAs [27]. The structure of these local interconnects is similar to the $N \times M$ MUXes used in connection blocks in Fig. 6(a). We can also build them by RRAMs and apply our programming transistor sharing as discussed in Sections III-B and III-D.2. Fig. 30 shows the area reduction achieved by RRAM-based local interconnects. The reduction of the LB area due to RRAM-based local interconnects is 60%. It further increases the overall FPGA area reduction enabled by RRAMs. It also leads to extra benefits in performance and power savings due to shorter signal paths, as discussed in Section V-C and [2]. Since there are many different kinds of LB structures in modern FPGAs, we only use Fig. 30 as a case study and expect that the area reduction will vary for different types of LB designs.

VII. Conclusion

This paper presented the FPGA-RPI, a novel FPGA architecture with RRAM-based programmable interconnects. Programmable interconnects are the dominant part of conventional FPGA. We used RRAMs to build programmable interconnects and optimize their structures for the RRAM properties. A customized CAD flow was provided for FPGA-RPI, with an advanced P&R tool named VPR-RPI that was developed for FPGA-RPI to deal with its novel structures. Results showed that the programmable interconnects of FPGA-RPI have a 96% smaller footprint, 55% higher performance, and 79% lower power consumptions compared to other FPGA counterparts.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful suggestions.

References

[1] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *IEEE Design Test Comput.*, vol. 28, no. 2, pp. 6–15, Mar. 2011.

[2] M. Lin, A. E. Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 681–229, Feb. 2007.

[3] C. Chen, S. Mitra, H.-S. P. Wong, R. T. Howe, J. Watt, D. Lewis, J. Provine, K. Akarvardar, S. Chong, N. Patil, and R. Parsa, "Efficient FPGAs using nanoelectromechanical relays," in *Proc. 18th Annu. Int. Symp. Field Program. Gate Arrays*, 2010, pp. 273–282.

[4] S. Tanachutiwat, M. Liu, and W. Wang, "FPGA based on integration of CMOS and RRAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 11, pp. 2023–2032, Nov. 2011.

[5] E. Ahmed and J. Rose, "The effect of LUT and clustersize on deep-submicron FPGA performance and density," *IEEE Trans. VLSI Syst.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.

[6] C. Dong, C. Deming, S. Haruehanroengra, and W. Wang, "3-D nFPGA: A reconfigurable architecture for 3-D CMOS/nanomaterial hybrid digital circuits," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 54, no. 11, pp. 2489–2501, Nov. 2007.

[7] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1712–1724, Nov. 2005.

[8] P. K. 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 Lett.*, vol. 32, no. 1, pp. 57–59, Jan. 2011.

[9] A. L. Lacaita and D. J. Wouters, "Phase-change memories," *Phys. Status Solidi (a)*, vol. 205, no. 10, pp. 2281–2297, Oct. 2008.

[10] W. W. Jang, J. O. Lee, J.-B. Yoon, M.-S. Kim, J.-M. Lee, S.-M. Kim, K.-H. Cho, D.-W. Kim, D. Park, and W.-S. Lee, "Fabrication and characterization of a nanoelectromechanical switch with 15-nm-thick suspension air gap," *Appl. Phys. Lett.*, vol. 92, no. 10, pp. 103110-1–103110-3, Mar. 2008.

[11] R. Huang, L. Zhang, D. Gao, Y. Pan, S. Qin, P. Tang, Y. Cai, and Y. Wang, "Resistive switching of silicon-rich-oxide featuring high compatibility with CMOS technology for 3D stackable and embedded applications," *Appl. Phys. A*, vol. 102, no. 4, pp. 927–931, Mar. 2011.

[12] S. Paul, S. Mukhopadhyay, and S. Bhunia, "Hybrid CMOS-STTRAM non-volatile FPGA: Design challenges and optimization approaches," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2008, pp. 589–592.

[13] N. Bruchon, L. Torres, G. Sassatelli, and G. Cambon, "New non-volatile FPGA concept using magnetic tunneling junction," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerging VLSI Technol. Architectures*, Mar. 2006, pp. 269–276.

[14] P.-E. Gaillardon, M. H. Ben-Jamaa, M. Reyboz, G. B. Beneventi, F. Clermidy, L. Perniola, and I. O'Connor, "Phase-change-memory-based storage elements for configurable logic," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2010, pp. 17–20.

[15] Y. Chen, J. Zhao, and Y. Xie, "3D-nonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2010, pp. 55–60.

[16] 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 Comput. Digital Tech.*, vol. 3, no. 6, pp. 609–624, Nov. 2009.

[17] Y. Y. Liauw, Z. Zhang, W. Kim, A. E. Gamal, and S. S. Wong, "Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2012, pp. 406–408.

[18] O. Turkyilmaz, J. Figueras, and Y. Zorian, "RRAM-based FPGA for ACM normally off, instantly on ACM applications," in *Proc. Int. Symp. Nanosc. Archit.*, 2012, pp. 101–108.

[19] P.-E. Gaillardon, M. H. Ben-Jamaa, G. B. Beneventi, F. Clermidy, and L. Perniola, "Emerging memory technologies for reconfigurable routing in FPGA architecture," in *Proc. Int. Conf. Electron., Circuits Syst.*, Dec. 2010, pp. 62–65.

[20] S. Onkaraiah, P.-E. Gaillardon, M. Reyboz, F. Clermidy, J. Portal, M. Bocquet, and C. Muller, "Using OxRRAM memories for improving communications of reconfigurable FPGA architectures," in *Proc. Int. Symp. Nanosc. Archit.*, Jun. 2011, pp. 65–69.

[21] P. Gaillardon, D. Sacchetto, G. B. Beneventi, M. H. Ben Jamaa, L. Perniola, F. Clermidy, I. O'Connor, and G. De Micheli, "Design and architectural assessment of 3-D resistive memory technologies in FPGAs," *IEEE Trans. Nanotechnol.*, vol. 12, no. 1, pp. 40–50, Jan. 2013.

[22] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, no. 3, p. 035204, Jan. 2007.

[23] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams, "Memristor–CMOS hybrid integrated circuits for reconfigurable logic," *Nano lett.*, vol. 9, no. 10, pp. 3640–3645, Oct. 2009.

[24] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *Proc. Int. Symp. Nanosc. Archit.*, Jun. 2011, pp. 1–8.

[25] G. Lemieux and D. Lewis, "Circuit design of routing switches," in *Proc. Int. Symp. Nanosc. Archit.*, 2002, pp. 19–28.

[26] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

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

[28] Xilinx, (2013). *Virtex-7 FPGA Data Sheets* [Online]. Available: http://www.xilinx.com/support/documentation/7_series.htm

[29] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends Electron. Design Autom.*, vol. 2, no. 2, pp. 135–253, 2007.

[30] R. Nathanael, V. Pott, K. Hei, J. Jaeseok, and T.-J. King Liu, "4-terminal relay technology for complementary logic," in *Proc. IEDM*, Dec. 2009, pp. 1–4.

[31] C.-H. Wang, Y.-H. Tsai, K.-C. Lin, M.-F. Chang, Y.-C. King, C.-J. Lin, S.-S. Sheu, Y.-S. Chen, H.-Y. Lee, F. T. Chen, and M.-J. Tsai, "Three-dimensional $4F^2$ ReRAM cell with CMOS logic compatible process," in *Proc. IEEE IEDM Tech. Dig.*, Dec. 2010, pp. 664–667.

[32] W. Guan, S. Long, Q. Liu, M. Liu, and W. Wang, "Nonpolar nonvolatile resistive switching in Cu doped ZrO_2," *IEEE Electron Device Lett.*, vol. 29, no. 5, pp. 434–437, May 2008.

[33] S.-S. Sheu, P.-C. Chiang, W.-P. Lin, H.-Y. Lee, P.-S. Chen, Y.-S. Chen, T.-Y. Wu, F. T. Chen, K.-L. Su, M.-J. Kao, K.-H. Cheng, and M.-J. Tsai, "A 5ns fast write multi-level non-volatile 1 K bits RRAM memory with advance write scheme," in *Proc. VLSI Circuits, Symp.*, Jun. 2009, pp. 82–83.

[34] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chung, I.-K. Yoo, and K. Kim, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_5-x/TaO_2-x bilayer structures," *Nature Mater.*, vol. 10, no. 8, pp. 625–30, Jan. 2011.

[35] H.-S. P. Wong, S. Raoux, K. SangBum, L. Jiale, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[36] 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 *Proc. IEDM*, Dec. 2007, pp. 767–770.

[37] Y.-W. Chang, D. F. Wong, and C. K. Wong, "Universal switch modules for FPGA design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 1, no. 1, pp. 80–101, Jan. 1996.

[38] Q. Shen, Y. Chen, W. Lingli, T. Jiarong, and J. Tan "FPGA routing architecture optimization," in *Proc. 8th Int. Conf. Solid-State Integr. Circuit Technol.*, 2006, pp. 1934–1936.

[39] Xilinx, (2012). *Virtex-6 FPGA Configurable Logic Block User Guide* [Online]. Available: http://www.xilinx.com/support/documentation/virtex-6.htm

[40] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2004, pp. 41–48.

[41] Xilinx, (2012). *Virtex-6 Family Overview* [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[42] I. Dobbelaere, M. Horowitz, and A. El Gamal, "Regenerative feedback repeaters for programmable interconnections," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1246–1253, Nov. 1995.

[43] D. Burger, (2011). *International Technology Roadmap for Semiconductors* [Online]. Available: http://www.itrs.net/Links/2011ITRS/Home2011.htm

[44] L. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[45] J. Cong and D. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. Int. Conf. Comput.-Aided Design*, 1999, pp. 358–363.

[46] C. Alpert, H. Jiang, S. S. Sapatnekar, and P. G. Villarrubia, "A practical methodology for early buffer and wire resource allocation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 5, pp. 573–583, May 2003.

[47] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. Int. Symp. FPGAs*, 1995, pp. 111–117.

[48] I. Kuon and J. Rose, (2008). *iFAR–Intelligent FPGA Architecture Repository* [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures/

[49] W. Zhao and Y. Cao, (2012). *Predictive technology model (PTM) website* [Online]. Available: http://ptm.asu.edu/

[50] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *Proc. Int. Symp. Qual. Electron. Design*, 2006, pp. 585–590.

[51] S. Yang, "Logic synthesis and optimization benchmarks, version 3.0," Dept. MCNC, Carolina Univ., Research Triangle Park, NC, Tech. Rep. NC2709, 1991.

[52] R. Y. Rubin and A. M. DeHon, "Timing-driven pathfinder pathology and remediation: Quantifying and reducing delay noise in VPR-pathfinder," in *Proc. Int. Symp. FPGAs*, 2011, pp. 173–176.

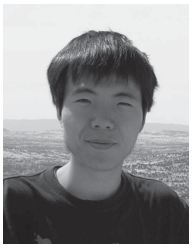
[53] Xilinx, (2009). *Virtex-4 FPGA data sheets* [Online]. Available: http://www.xilinx.com/support/documentation/virtex-4.htm

[54] J. Cong and B. Xiao, "Defect tolerance in nanodevice-based programmable interconnects: Utilization beyond avoidance," in *Proc. Design Autom. Conf.*, 2013, pp. 277–278.

Jason Cong (S'88–M'90–SM'96–F'00) received the B.S. degree in computer science from Peking University (PKU), Beijing, China, in 1985, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign (UIUC), Urbana, IL, USA, in 1987 and 1990, respectively.

He is currently a Chancellor's Professor with the Computer Science Department, University of California, Los Angeles (UCLA), the Director of the Center for Domain-Specific Computing (CDSC), and the Co-Director of the VLSI CAD Laboratory. He served as the Department Chair from 2005 to 2008, and has graduated 30 Ph.D. students. He is also a Distinguished Visiting Professor with Peking University and the Co-Director of the UCLA/PKU Joint Research Institute in Science and Engineering. His current research interests include synthesis of very large-scale integration circuits and systems, programmable systems, novel computer architectures, nanosystems, and highly scalable algorithms. He has authored over 350 publications in his areas of expertise.

Dr. Cong was a recipient of eight Best Paper Awards from IEEE T-CAD in 1995, ACM TODAES in 2005, ISPD in 2005, HPCA in 2008, SASP in 2009, FCCM in 2011, ACM TODAES in 2012, FPGA Symposium in 2013, and of the 2011 ACM/IEEE A. Richard Newton Technical Impact Award in Electric Design Automation. He was a recipient of the 2010 IEEE Circuits and System Society Technical Achievement Award for seminal contributions to electronic design automation, especially in FPGA synthesis, very large-scale integration interconnect optimization, and physical design automation. He was named an ACM Fellow in 2008.



Bingjun Xiao (S'11) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2010 and the M.S. degrees in electrical engineering from the University of California, Los Angeles, CA, USA, in 2012, where he is currently pursuing the Ph.D. degree.

His current research interests include emerging computing architectures and interconnect optimization.

Dr. Xiao was the recipient of the Best Graduate Award from Peking University in 2010 and the Outstanding M.S. Award from UCLA in 2012.