R-Accelerator: An RRAM-Based CGRA Accelerator With Logic Contraction

Zhengyu Chen, *Student Member, IEEE*, Hai Zhou, *Senior Member, IEEE*, and Jie Gu, *Senior Member, IEEE*

***Abstract*— In this paper, a novel RRAM-based reconfigurable accelerator (R-accelerator) design is proposed, which makes special use of existing RRAM device for high-efficient reconfigurable application-specific computing. The proposed R-accelerator design consists of RRAM-based arithmetic unit (AU) array, fully integrated into commercial EDA design tools. A significant area saving is achieved compared with conventional digital counterpart due to the proposed logic contraction technique, as well as saving of storage space and routing congestions. For enabling the optimization of the AU array, this paper also proposes a systematical method on the synthesis of the AU array under routing channel constraint for application-specific designs. Two automatic mapping algorithms, including simultaneous mapping and incremental mapping algorithms, are proposed and compared. The experiments using 45-nm CMOS technology on a case study of dynamic time warping example and general benchmark programs show up to 49% area reduction and 28% performance enhancement using the proposed R-accelerator technique compared with conventional application-specified integrated circuit (ASIC) design.**

***Index Terms*— Coarse grain reconfigurable array (CGRA), logic contraction, reconfigurable accelerator (R-accelerator), RRAM.**

I. Introduction

STRONG demand for accelerator enriched microprocessor is recently observed from the consumer electronics industry due to the rapid growth of applications, such as image processing, machine learning, etc. These applications are typically characterized by complex and data-intensive computation, which lead to significant challenges to the existing IC development models. Two approaches are commonly applied for their implementations: 1) hardware in the form of accelerator-based application-specified integrated circuit (ASIC) chips or application-specific instruction processors (ASIPs) [1], [2] and 2) software running on a general-purpose processor. In the case of ASIC designs, although a state-of-the-art performance is achieved, it becomes impractically expensive to implement all potentially used algorithm in an ASIC or ASIP design [5]. On the other hand, the general-purpose processor is flexible enough to support various applications but suffers from insufficient performance for complex algorithms, especially those required for real-time image processing, such as face recognition, and optimal margin classifier, etc. [3], [4]. As a result, a reconfigurable accelerator (R-accelerator)-based hardware solution, which can provide the advantages of both approaches, has become increasingly popular recently [6], [7]. Such architecture has higher performance compared to the general-purpose processor and wider applicability compared to ASIC. Toward reconfigurable processor design, many solutions have been developed so far. The most common example is field-programmable gate array (FPGA), which utilizes SRAM-based look-up tables to provide flexibility in implementing logical solutions. However, the high cost of area and power caused by the bit-level programmability in FPGA poses significant challenges to meet the demand of low-power applications, such as the Internet of Things (IoT) devices.

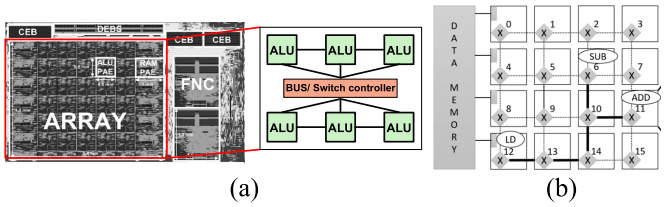


Fig. 1. (a) Layout and core circuit diagram of previous CGRA work [6]. (b) Block diagram of a CGRA with single-cycle multihop interconnect [25].

As an alternative solution, coarse grain reconfigurable array (CGRA) represents the second class of reconfigurable architectures. In CGRA, predefined, and optimized reconfigurable arithmetic units (AUs) are used as building blocks. Complex algorithms are then allocated into AU arrays to realize application-specific instruction sets removing conventional performance limitation from multicycle pipeline operations. CGRA significantly reduces the overhead of fine-grain switches and associated memory components from solutions, such as FPGA while keeping a high level of reconfigurability [6], [8]. An example of such architecture is shown in Fig. 1(a) where totally 30 ALUs are arranged in a 5×6 mesh to realize a complex operation, such as square-root without intermediate storage or pipelines [1], [6]. Fig. 1(b) shows another CGRA design with single-cycle multihop interconnect, which contains 16 nodes; each node can communicate with its four nearest neighbors without using a central crossbar [25]. In spite of the above advantages, CGRA still suffers from

Manuscript received February 1, 2019; revised May 22, 2019; accepted June 11, 2019. Date of publication July 30, 2019; date of current version October 23, 2019. This work was supported by the National Science Foundation (NSF) under Grant CCF-1533656, Grant CNS-1441695, and Grant CNS-1651695. *(Corresponding author: Zhengyu Chen.)*

The authors are with the Electrical and Computer Engineering Department, Northwestern University, Evanston, IL 60208 USA (e-mail: zhengyuchen2015@u.northwestern.edu).

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

Digital Object Identifier 10.1109/TVLSI.2019.2925937

the area, and power overhead for bookkeeping the configuration of the design, as well as the high cost of establishing flexible interconnects between AU units. In this paper, we proposed a novel RRAM-embedded accelerator based on CGRA architecture to address the above issues with a cross-layer solution ranging from device, circuits, to architecture levels.

In recent technology development, the emerging nonvolatile memory (NVM) technologies, i.e., RRAM or memristor, have provided tremendous new opportunities to the development of high-performance microprocessors. The demonstrated use of RRAM can be classified into two large classes: 1) NVM where RRAM is utilized to provide on-chip NVM solution replacing conventional cache or DRAM and 2) a processing unit, where resistive values of such devices are used to represent computing variables, e.g., weights for the matrix computation [11]. In the first class, the nonvolatility of such device provides a significant saving in retention power. In addition, the very simple thin-film topology of RRAM allows monolithic integration of such memory device in vertical stacks on top of the processing units, leading to an extremely high-density storage solution. An example is shown in the commercial crossbar's technology [10]. In the second class, such an implementation dramatically reduced the cost of convolution operation, leading to a tremendous saving of computing power. Furthermore, a mrFPGA technique was also proposed to use RRAM as an interconnect solution to replace existing SRAM-based FPGA, leading to 5.2× improvement in the area [5]. More recently, a processing-in-memory (PIM) solution was proposed, which uses RRAM as a processing unit embedded into memory array for the neural network application. It is shown that such a PIM significantly reduced the costs, leading to 300× saving on energy consumption [12].

A. Prior Works

A similar work, which focuses on the interconnect is in [25]. A novel CGRA architecture, HyCUBE is introduced, which has a reconfigurable interconnect supporting single-cycle communications across distant function units on the chip. The reconfigurable interconnect leads to a new formulation of the application mapping problem that is efficiently handled by their HyCUBE compiler.

In [6], a system on chip implementation of a reconfigurable digital signal processor is proposed. The device is suitable for the execution of a wide range of applications exploiting a balanced mix of heterogeneous reconfigurable fabrics merged together by a flexible and efficient communication infrastructure based on a 64-bit Network on-chip.

In [8] and [9], an edge-centric modulo scheduling (EMS) for coarse-grained reconfigurable architecture is proposed to systematically solve the problem of the placement of operation for the compiler. The distributed nature of CGRAs, including sparse interconnect and distributed register files, presents challenges to a compiler. Overall, EMS improves performance by 25% over traditional modulo scheduling and achieves 85%–98% of the performance compared to a state-of-the-art simulated annealing technique.

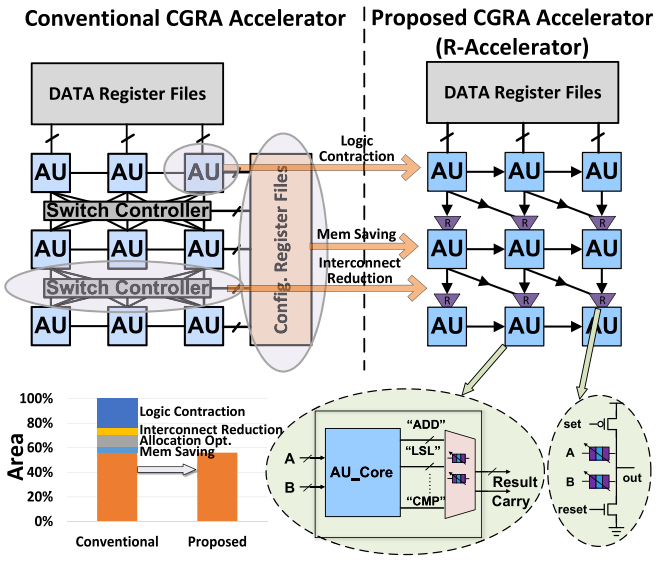


Fig. 2. Conventional CGRA accelerator (left). Proposed RRAM-based CGRA accelerator (R-accelerator) (right).

B. Contributions of This Work

Different from the prior works, this paper proposes a fundamentally new scheme for implementing CGRA using the emerging RRAM/RRAM device, referred to as R-accelerator. Our target design is similar to those from CGRA, where the chip is reconfigured for special applications. The suitable applications for CGRA are pretty wide and mostly special-purpose computing, e.g., image processing, video decoding/encoding, compression, and machine learning operation, etc. Essentially, most of the computing jobs can be mapped to this design as a general-purpose "accelerator" in comparison with the general-purpose microprocessor. Contrary to the popular PIM implementation, we propose a microarchitecture of memory-in-process (MIP), which embeds a nonvolatile RRAM device into the computing logic. Fig. 2 summarizes the proposed R-Accelerator architecture in the following aspects.

1) The conventional AU is replaced by RRAM-based reconfigurable AU with a specially developed logic contraction technique rending significant logic simplification.
2) The switch controller (SC) in conventional design is reduced and simplified by the RRAM-based programmable interconnection.
3) The configuration register file is eliminated as the RRAM works as both nonvolatile storage and programmable interconnect.

As extended from previous works in [31], this work develops comprehensive techniques of RRAM-based accelerator design. Also, a comprehensive modeling of logic contraction and area-driven design is provided. More specifically, the contributions of this paper are highlighted below.

1) To fully utilize the strength of the RRAM-based logic cell, we propose a special logic synthesis technique, referred as logic contraction, which can dramatically reduce the area overhead associated with realizing

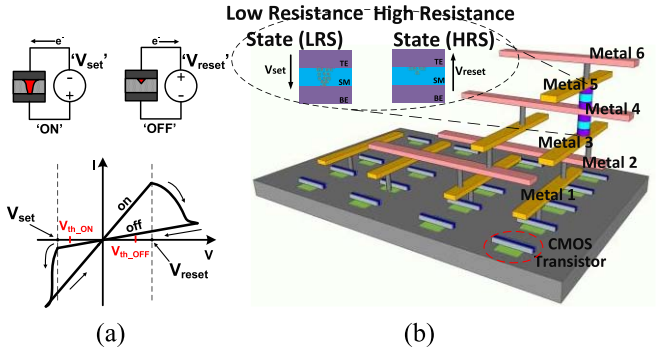


Fig. 3. (a) Typical bipolar I–V linear curve of RRAM. (b) Three-dimensional diagram of the RRAM-based circuit.

the reconfigurability in conventional CMOS design. Additionally, we developed a model of the logic contraction to estimate the area saving.

2) To leverage the help from RRAM device and overcome the challenges of allocating arbitrary algorithm into CGRA AU arrays, we propose special resource allocation solutions based on two heuristic algorithms: simultaneous allocation and incremental allocation algorithm. The proposed resource allocation algorithms lead to orders of magnitude speedup while achieving an almost ideal solution compared with brute-force exhaustive search algorithm.
3) A full layer implementation, including PCell and backend supports, have been developed and fully integrated into the conventional EDA design tools, leading to an automatic design flow of the proposed R-accelerator design. It is demonstrated in our case studies on a dynamic time warping (DTW) example and general benchmark programs.

The organization of this paper is given below. In Section II, a brief introduction of the principle of the RRAM device is described. An overview of the coarse grain reconfigurable architecture is introduced. In Section III, the critical element of RRAM-based AU array, an RRAM-based AU design, is proposed. Detailed design flow and analysis of the proposed logic contraction technique are carried out. In Section IV, the reconfigurable AU array architecture is introduced with detailed interconnect options and modeling. A comprehensive instruction-to-AU array synthesis and allocation algorithm is provided in Section V. Measurement that supports the results of the proposed design flow is presented in Section VI. Section VII concludes the discussion on this paper.

II. Background

A. RRAM Devices

Since the announcement of RRAM device from 2008, a large variety of RRAM device has been developed. A common RRAM uses a metal-insulator-metal structure, where a thin layer of insulation material is sandwiched between two metallic electrodes. The resistance of the RRAM is determined by the content of a conductive filament in the layer between the two electrodes, as illustrated in Fig. 3(a).

TABLE I
Selected Published RRAM/RRAM Characteristics

Material	LRS (ohm)	HRS (ohm)	Write Time(ns)	Set Voltage (V)
NiO [15]	500	10M	10~50	1.5
SiOxNy [18]	100	1M	100	1
TaO [19]	100	100k	2	1.5
HfO2 [20]	1k	100k	100	1.5
HfO2 [21]	5k	10~100M	100	2
This work	10k	1M	10	1.2

The key characteristics, such as resistance tuning range, writing speed, and tuning voltages, vary dramatically based on the potential applications. The two major classes of application are: 1) RRAM memory with binary states and 2) analog RRAM where the resistance states possess continuous tuning and larger range. Normally, higher resistance ratio is observed in the second class due to the requirement of higher resolutions. Here, we briefly survey the selected published RRAM characteristics in Table I.

As this paper explores a novel usage of the resistive memory as a nonvolatile reconfigurable logic, we only utilize the RRAM as a binary state device, i.e., low resistive state (LRS) and high resistive state (HRS) with relaxation on the tuning resolution requirement of the device as compared with previous publication for neuromorphic computing [12]. Although the matching of exact device characteristics to particular published device is beyond the scope of this paper, we devote our effort to incorporate more realistic behavior and electronic properties of the RRAM device by creating: 1) a realistic parameterized cell (PCell) that can be extracted from real device layout and simulated by spice simulator to characterize the performance of the proposed circuits, e.g., the delay of the proposed logic circuits and 2) an integrated library cell that is fully supported by modern EDA tools, e.g., Cadence encounter, to perform synthesis, place and route step of the developed circuit components. All designs in this paper have been fully implemented using commercial EDA tools into backend layout in a 45-nm CMOS.

B. Electrical Characteristics of RRAMs

Fig. 3 shows the RRAM device, which is a two-terminal passive device in which the resistance between its terminals can be configured into an HRS state, or an LRS state. Both HRS and LRS states are controlled by an external voltage applied across its terminals.

Fig. 3(a) shows the representative bipolar I–V linear characteristics of such an RRAM [10], [28]. A voltage V_{set} above the threshold $V_{\text{th\_ON}}$ is applied across its terminals until a conductive filament is formed, to set (or program) the device into a memristive "ON" or LRS state. The conductive filament shorts the device and lowers its resistance to R_{ON}. The reverse operation is applied to "reset" the device into "OFF" or HRS state. A recent development from commercial vendors, such as Crossbar, Inc., Santa Clara, CA, USA, has demonstrated a fully integrated monolithic solution, where a memristive device can be inserted between any metal layers of the existing CMOS chips and possess a large on–off ratio, e.g., 1000 [10], [30].

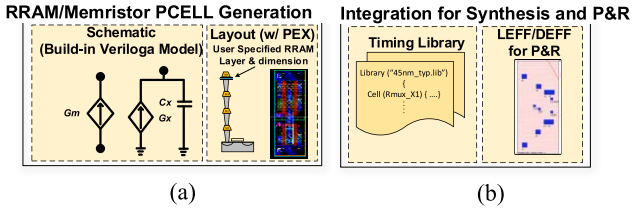


Fig. 4. Model and implementation of RRAM. (a) PCell layout and schematic. (b) Integration into the conventional EDA flow.

Fig. 3(b) shows the 3-D diagram of RRAM-based circuit design where RRAM is inserted between metal 4 and metal 5 on top of CMOS transistor layers.

C. Coarse Grain Reconfigurable Architecture

CGRA consists of an array of computing elements/nodes. Each node executes word-level operations and communicates through an interconnected network. In general, CGRA designs can be described by four characteristics: size, node functionality, interconnected network configuration, and register file communication. The size refers to the number and topology of nodes array and commonly varies from four nodes arranged in a 2×2 grid up to 64 nodes arranged in an 8×8 grid [8]. The functionality of each node can vary from a single-function module (e.g., adder or absolute value), an ALU, or to a full-blown processing element (PE). In addition, the functionality of nodes may be homogeneous or heterogeneous. For example, only a subset of nodes may access data memory [9], [25]. There is a large amount of different interconnect network configurations, such as interconnects between each node and its four (or eight diagonal) nearest neighbors [8], [24], or buses connecting each node to other nodes in the same row or column [6].

III. RRAM-Based Reconfigurable AU Design

A. RRAM Model and PCell Design

Fig. 4(a) shows our developed RRAM PCell, including both schematic representation and layout. The PCell allows variable dimension, as well as placement of RRAM at any user-specific metal layers.

For spice simulation, parasitics due to routing and via connections are first extracted from Calibre. And then, the devices are simulated with the VerilogA model together with extracted parasitics. The VerilogA model was developed based on various reported resources [17]. In our design, we used one of the high-level metals to avoid causing congestion with local routing. The parasitic impact from the connection between local transistors and RRAM devices has been included in our extracted model and library cell component in our digital design environment. Fig. 4(b) shows our integration of the developed RRAM device and RRAM based logic circuits into digital design flow for large-scale integration used in this work. Fig. 4(c) shows the spice simulation waveforms of RRAM-based multiplexer (MUX) circuits as will be discussed in Section III-B.

Practical issues of RRAM/RRAM devices have been reported, including the variability of the cell resistance especially at high resistance state and the endurance of resistance

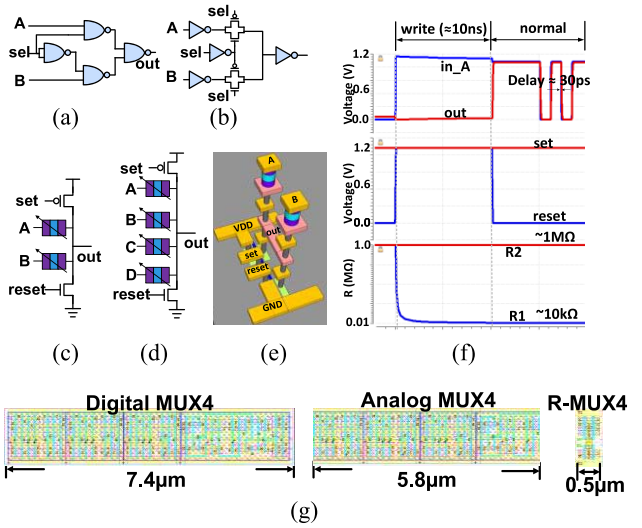


Fig. 5. MUX design. (a) Conv. digital MUX2. (b) Conv. analog MUX2. (c) R-MUX2. (d) R-MUX4. (e) Three-dimensional diagram of R-MUX2. (f) Spice simulation waveforms of RRAM-based MUX. (g) Layout comparison: Conv. digital MUX4 (left), conv. analog MUX4 (middle), and R-MUX4 (right).

value during the writing and reading [16], [17]. Although an exact device level evaluation is out of the scope of this paper, we performed a variability test by varying HRS and LRS by a factor of 300% and evaluated the performance, i.e., cell delay impact. Our evaluation shows the performance variation is within 10% of the nominal value because the logic cell delay is strongly influenced by intrinsic transistors from driving buffers rather than the interconnect variation.

B. RRAM-Based Logic Cell Design

In CGRA, MUX logic cell plays an important role in realizing configurable functionality. Fig. 5(a) and (b) show two conventional implementations of MUX2 cell, which contains 16 and 12 transistors with one select bit provided from a memory cell. As an alternative solution, we propose an RRAM-based MUX (R-MUX) cell with a significant saving on cell footprint. Fig. 5(c) and (e) show our proposed RRAM-based MUX2 (R-MUX2) cells and its 3-D drawing with only two RRAMs and two transistors.

Moreover, since the R-MUX can be fabricated over the logic blocks, integrating more bit into the R-MUX would not significantly increase the area of the device. Fig. 5(d) shows the schema of a 4-to-1 R-MUX with the same footprint as 2-to-1 R-MUX. For comparison, conventional 4-to-1 MUX takes three times the area as 2-to-1 MUX. The footprint of the proposed R-MUX4 cell is reduced by $9\times$ compared with conventional ones. In addition, there is no extra SRAM needed to store the configuration data since the RRAM works as nonvolatile storage. Including memory space saving, an overall saving of $12\times$ to $15\times$ compared to conventional designs can be achieved. At configuration phase of the proposed circuits, by turning ON/OFF the set/reset transistors with particular input combination from the previous stage, we can selectively program the RRAM into LRS and HRS, i.e., only one of RRAM stays in LRS while the rest stay in HRS. In this way, only the signal through the path with the RRAM in LRS

will be passed realizing a MUX logic. For programming the device into an LRS, a voltage V_{set} above the programming threshold is applied across its terminals until a conductive filament is formed; reverse operation is applied to "reset" the device into HRS. Note that to separate programming from the normal operation the configuration phase should be performed at an elevated voltage, e.g., V_{set} of 1.2 V, rather than the normal operation voltage, e.g., 1.0 V. Fig. 5(f) shows the spice simulation of operation waveforms of the R-MUX circuits, exhibiting a write-speed of ∼10 ns, ON–OFF resistive ratio of ∼100 and a logic delay of ∼30 ps with correct functionality. Fig. 5(g) shows the area comparison among conventional MUX, analog MUX, and R-MUX.

C. Logic Contraction Using RRAM-Based Logic Cell for Reconfigurable AU Design

1) Reconfigurable Arithmetic Unit Design: Reconfigurable AU is used as a building element of the CGRA. We implement the RRAM based 8-bit AU which integrates eight basic configurable operations: addition (ADD), subtraction (SUB), logic shift left (LSL), logic shift right (LSR), MUX, comparison (CMP), XOR, and XNOR. In addition, a bypass mode is also introduced to pass input signals directly into the next stages.

The conventional AU uses combinational logics to realize the configuration of different supported functionality. We experiment with the area benefits of using proposed R-MUX by simply replacing some of the conventional MUXs by R-MUXs in the synthesized and place and route step. Note that, only the static MUXs used for configuration can be replaced by the R-MUX since the MUXs used for dynamic logic operations require much faster switching speed than what is offered in our RRAM devices. Despite the significant area reduction shown in Section III-B, our experiments show that only 5% of area saving can be achieved by replacing conventional MUX by R-MUX. This is because, in a highly optimized gate-level netlist, many of the selection logic of the AU are synthesized into more complex logic, e.g., AND-OR-Invert logic gates which cannot be replaced. The left layout in Fig. 6 shows the conventional AU layout with MUXs marked by red color boxes. The area benefit from simply replacing existing MUXs with R-MUXs is insignificant due to the low occurrence of conventional MUXs because of the merging of selection logic with main functional logics from conventional synthesis methodology. Even if we force more usage of MUXs, the area improvement is still small. Hence, without a methodology to clearly distinguish main functional logic from other control/selection logic, we cannot fully utilize the benefits from RRAM based logic cells. This issue will be addressed in Section III-C2.

2) Proposed Logic Contraction Method: In this section, a novel control logic contraction technique is proposed to provide substantial area saving using RRAM based logic cells.

In the conventional design, an 8-bit AU is realized by the digital operation code (control bit) to configure the AU core into different operation modes, e.g., ADD, SUB, CMP, etc. For instance, an 8-bit two-operation conventional AU with a

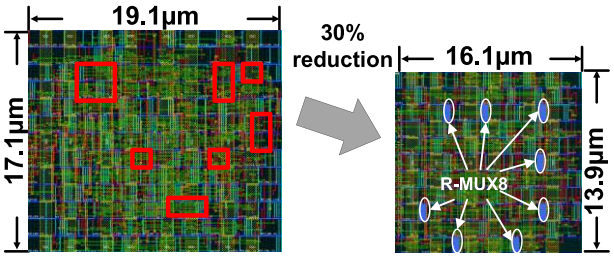


Fig. 6. Layout comparison between the conventional AU (left) and RRAM-based AU (right) with conventional MUXs and R-MUXs highlighted.

1-bit operation code (c_0) can be expressed as follows:

$$O_{\text{conv-AU}} = \overline{c_0}\text{ADD}(A, B) + c_0\text{SUB}(A, B). \tag{1}$$

It is important to highlight the overhead of the configuration. Even with a fully optimized adder/subtractor with technology mapping using complex standard cell, selection logic for realizing (1) requires approximately 16 ANDs, 8 ORs gates, and 8 NORs for this two-operation AU. The block diagram of the conventional AU design is shown in Fig. 8(a) with distributed selection logic circuits marked in gray. Compared to the AU core function, i.e., ADD/SUB, the selection logic circuit consumes a total of 20%–40% area. On the other hand, the selection logic overhead can be dramatically suppressed by utilizing the proposed R-MUX and logic contraction technique for configuration. Its block diagram is shown in Fig. 8(b)

$$O_{\text{RRAM-AU}} = M[\text{ADD}(A, B), \text{SUB}(A, B)]. \tag{2}$$

Equation (2) shows the expression of the RRAM-based AU design. The M function represents the R-MUX logic, which consumes much less area when compared to the conventional digital MUX cells. More importantly, the selection logic operation in (2) is eliminated from the conventional expression as in (1). As a result, the digital logic implementation using proposed R-MUX becomes much simpler. In fact, the more functionality included in the AU design, the more area saving can be achieved due to the small area consumption of R-MUX logic. The output expressions of 8-bit 4-operation conventional and R-MUX-based AUs are shown in the following equations:

$$\begin{aligned} O_{\text{conv-AU}} &= \overline{c_1 c_0}\text{ADD}(A, B) + \overline{c_1}c_0\text{SUB}(A, B) \\ &\quad + c_1\overline{c_0}\text{LSL}(A, Bc_1c_0) + \text{LSR}(A, B) \end{aligned} \tag{3}$$

$$\begin{aligned} O_{\text{RRAM-AU}} &= M[\text{ADD}(A, B), \text{SUB}(A, B), \\ &\quad \text{LSL}(A, B), \text{LSR}(A, B)]. \end{aligned} \tag{4}$$

Unfortunately, standard logic synthesis technique does not support the insertion of the R-MUX logic, which essentially behaves as a reconfigurable "wire" rather than a Boolean logic. To facilitate the logic synthesis of the proposed RRAM-based reconfigurable cells, we developed a special logic contraction flow. As illustrated in Fig. 8(b), at first, we rewrite the register transfer level (RTL) for AU by elaborating outputs from each supporting function and modify the high-level top module with a separation of core functionality and selection logic. Second, we perform R-MUX integration using cells built from RRAM PCell through standard synthesis procedure of the logic design. This method allows us to fully utilize the optimization power

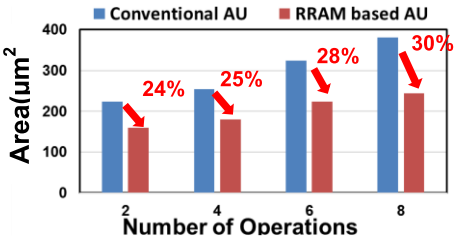


Fig. 7. Area comparison between the conventional and RRAM-based AUs.

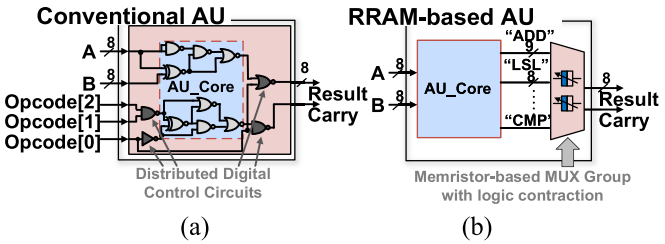


Fig. 8. AU circuit diagram. (a) Conventional AU. (b) RRAM-based AU.

of modern synthesis tool, while still integrating R-MUX automatically into the final netlist and layout. For comparison, we also built a conventional AU, which includes the same amount of functions as the R-MUX-based AU, e.g., ADD, SUB, etc. It is synthesized by the commercial tool and then placed and routed by the same tool. Both designs are generated by Cadence tool in the 45-nm technology. The area comparison between conventional AU and R-MUX-based AU with different operations integrated is shown in Fig. 7. An area-saving from 24% to 30% is achieved. As expected, the more operations integrated into the AU, the more area-saving benefit is achieved. The layouts of our proposed 8-bit 8-operation AU and conventional AU are shown in Fig. 6. Both designs have been synthesized and placed and routed with hierarchically flattened options to achieve optimized area. A maximum area-saving of 30% is achieved by introducing the RRAM-based logic contraction technique. Note that, the additional area-saving of storage bits for configuration is not included in this comparison.

3) Flow and Modeling of Logic Contraction: The proposed logic contraction method is a special method developed for the effective insertion of RRAM-based logic cells. Since different applications have different functionality defined, the benefits reported in Section III-C2 may not be universal to a general design. In this section, we provide an analytical model for area saving from RRAM-based logic contraction.

The general AU or ALU design consists of two main parts: 1) the core function parts that provide the core functionality of AU (e.g., adder, subtractor) and 2) configuration or control logic parts that configure the AU into a particular function mode. In conventional AU design, core function part is mixed with digital control circuits, as shown in Fig. 9(a) and hard to be separated. To facilitate the study and analysis, we add a virtual intermediate step in Fig. 9(b) to create a hybrid AU where the core functional logic is separated from the control/selection logic for reconfigurability.

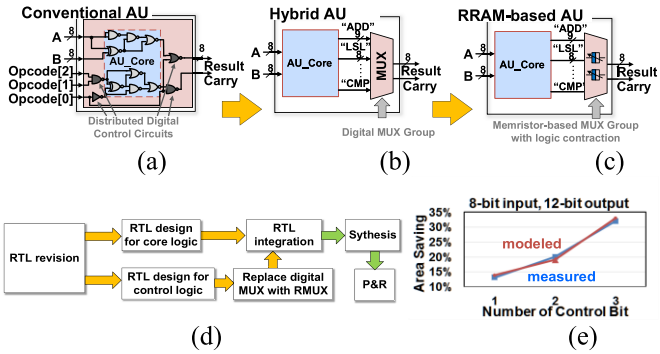


Fig. 9. (a) Conventional AU. (b) Hybrid AU. (c) RRAM-based AU. (d) Flowchart for logic contraction. (e) Comparison between modeled versus measured saving.

Fig. 9 shows the evolution of RRAM-based AU.

1) We first extract the "core function" from the conventional AU. This can be realized by eliminating the control logic in the RTL design step.
2) We generate a separate control logic block, which can be realized by groups of digital MUXs and connect it with the function core to form the "hybrid AU" as in Fig. 9(b). This "hybrid AU" realizes the same functionality as conventional AU with an overhead due to logic separation.
3) Replace the digital MUXs with R-MUX to form the proposed RRAM-based AU, which is shown in Fig. 9(c). In the actual implementation, step (2) in Fig. 9(b) is not necessary. The flowchart of the proposed logic contraction method is shown in Fig. 9(d).

Since the area consumption of R-MUX is very small, the area difference between the RRAM-based AU [shown in Fig. 9(c)] and the "hybrid AU" [shown in Fig. 9(b)] equals to the area of the control logic part (digital MUX group). The area saving $S_{\text{RRAM-hybrid}}$ is shown in the following equation:

$$S_{\text{RRAM-hybrid}} = \frac{A_{\text{hybrid}} - A_{\text{RRAM}}}{A_{\text{hybrid}}} = \frac{A_{\text{ctrl}}}{A_{\text{core}} + A_{\text{ctrl}}} \tag{5}$$

where A_{bybrid} is the area of "hybrid AU," A_{core} is the area of the core function, A_{RRAM} is the area of RRAM-based AU, which is approximately equal to A_{core}, and A_{ctrl} is the area of the control logic, which consists of digital MUXs. Based on experimental results, the area of the control logic group, with a given number of output bit and control bit, can be estimated by the following empirical equation:

$$A_{\text{ctrl}} = (\alpha_0 + \alpha_1 2^{n_{\text{ctrl}}}) \cdot (\beta_0 + \beta_1 a_{n_{\text{out}}}) \tag{6}$$

where the term $\alpha_0 + \alpha_1 2^{n_{\text{ctrl}}}$ in (6) represents that the area of MUX-based control logic exponentially increases when the number of control bit increases. $\beta_0 + \beta_1 a_{n_{\text{out}}}$ represents that area of control logic linearly increase when the number of out bit increases.

Based on the experimental results we found out that the area of hybrid AU is constantly 10% larger than the conventional AU. We then assume $A_{\text{conv}} = \gamma A_{\text{hybrid}}$, where γ is a constant (e.g., $\gamma = 0.9$ in our case). Based on (5), the saving

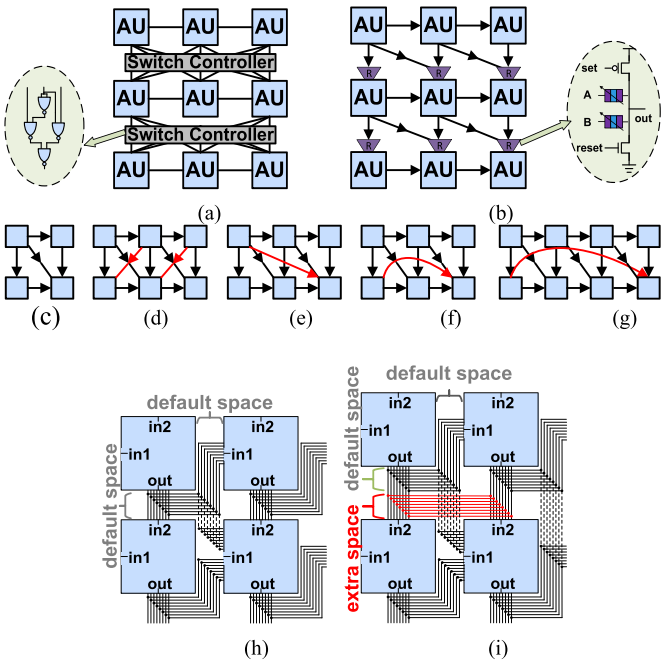


Fig. 10. AU array interconnect. (a) Conventional interconnect. (b) Proposed RRAM-based interconnect. (c) Option 1. (d) Option 2. (e) Option 3. (f) Option 4. (g) Option 5. (h) Detailed routing of (c). (i) Detailed routing of (d) which shows the overhead of extra routing channel.

between conventional AU and RRAM-based AU is shown in the following equation:

$$S_{\text{RRAM-conv}} = \frac{A_{\text{conv}} - A_{\text{RRAM}}}{A_{\text{conv}}} = \frac{\gamma A_{\text{hybrid}} - A_{\text{RRAM}}}{\gamma A_{\text{hybrid}}} = \frac{\gamma A_{\text{ctrl}} + (\gamma - 1)A_{\text{core}}}{\gamma (A_{\text{core}} + A_{\text{ctrl}})}. \quad (7)$$

In summary, for an arbitrary AU design with given functionality, the area saving of RRAM-based AU design can be estimated by following steps: 1) find out the area of core function design by using commercial EDA tools; 2) estimate the area of digital control logic group by (6); 3) calculate the final area saving based on (7). Fig. 9(e) shows the model based on (5)–(7) matches experiment results on a large number of designs.

IV. Reconfigurable AU Array Architecture

A. Interconnect and Reconfigurable AU Arrays

In the CGRA, the reconfigurable AU array should be configured into different logic topology to adapt to the data path of particular instructions. Conventionally, the reconfigurable AU array consists of AUs and SC [8]. The primary function of SC is achieving the interconnect of $m \times n$ configurable AUs setting signals from the top level. The architecture of a conventional interconnecting network with SC is shown in Fig. 10(a).

In order to reduce the complexity and area overhead of interconnecting network, in our work, the conventional SC is simplified into a basic unidirectional interconnect in which only one signal propagation direction is allowed in one signal channel. Besides the basic interconnect, four more optional routing channels depending on the demand of target instruction sets. As a result, we introduce totally five interconnecting options: 1) the basic interconnect, which is shown in Fig. 10(c); 2) adding an extra diagonal interconnect channel, which is shown in Fig. 10(d); 3) adding an extra free interconnect channel between nearby rows, which is shown in Fig. 10(e); 4) adding an extra interconnect channel within the same row but between different columns, which is shown in Fig. 10(f); and 5) continually adding extra routing channels within the same row, which is shown in Fig. 10(g).

Our proposed unidirectional network dramatically decreases the congestion on interconnecting wires compared with a fully connected crossbar. The interconnecting delay and area overhead can be further reduced when introducing proposed R-MUX logic to replace the conventional SC, which consists of a large amount of conventional MUXs. In addition, the memory component used to store the reconfigurable information is no longer needed since the R-MUX also works as NVM unit. Thus, the proposed R-accelerator design can provide a significantly reduced area with detailed results shown in Section VI.

B. Interconnect Options and Modeling

There is a tradeoff between the flexibility of interconnecting and the area consumption. The more interconnection routing options included, the more flexible the AU array can be to implement complex instructions. However, more routing options lead to increase of routing space and performance degradation of the interconnecting network. In this work, the default interconnecting of the AU array is unidirectional. Interestingly, no matter where the input port 1, input port 2 and output port are located, there occurs a free channel for the diagonal interconnects (from top left to bottom right) without introducing extra routing space. Fig. 10(h) shows a detailed routing diagram according to default routing in Fig. 10(c). If more options are offered in the configurable AU array, extra routing space will be needed. Fig. 10(i) shows the detailed routing for routing option in Fig. 10(d) and the extra routing channel is marked in red. Despite the extra routing overhead, more interconnecting options can lead to less number of AUs to be implemented for the target instruction sets. Thus, there exists an optimal interconnecting solution, which leads to the minimum area of AU array for supporting target instruction sets, which will be discussed in Section V.

The cost function (area) of interconnect is given as follows:

$$S_{\text{ALUarray}} = N_{\text{row}} \times (H_{\text{AU}+}H_{\text{intercnct}}) \times N_{\text{column}} \times (L_{\text{AU}+}L_{\text{intercnct}}) \quad (8)$$

where N_{row} is the number of rows, and N_{column} is the number of columns in AU array, H_{AU} is the height and L_{AU} is the length of a single AU, $L_{\text{intercnct}}$ is the extra routing spacing in the horizontal direction, and $H_{\text{intercnct}}$ is the extra routing spacing in the vertical direction of the extra interconnects. The final area of the configurable AU array is determined by both the number of AU (array topology) in the array and the extra routing space introduced by the extra interconnects. In Sections V-B and Section VC, two heuristic allocation algorithms are introduced.

V. Instruction-to-AU Array Synthesis and Allocation Algorithm

An important challenge for reconfigurable AU-based accelerator design is the allocation of instruction set into existing AU arrays. Several resource-constrained allocation algorithms were proposed in [8], [9], [13]. However, the prior work was focused on instruction scheduling based on existing fixed numbers of processing units. Different from prior work, in this paper, we explore optimal design choices where the number and interconnects of AU arrays are not predefined, but a minimum number of target instruction sets are provided. Thus, an optimal design exists to support the required instruction sets, which can be further extended based on reconfigurability provided in the design. Hence, this work is orthogonal to prior scheduling focused study [8], [9], [13], which cannot be used to provide us the optimal design choices. Generally speaking, this kind of graph allocation work is NP-hard [14]. In this work, we proposed two allocation algorithms based on the heuristic algorithm. The target is to find the optimal numbers of AU and optimal interconnect options as given in Fig. 10 to achieve minimum area costs.

A. Instruction Decomposition

One way to represent the instructions is by using a dataflow graph (DFGs). In the DFGs, each vertex represents a basic function operation, such as ADD, SUB, logic shift, etc., and each edge represents the data dependency between the connected operations. In order to allocate a particular instruction into the AU arrays, this instruction needs to be decomposed into several operations. The flow of the function decomposition can be comprehended as follows: 1) decompose the complex instruction into several simple operations, which can be realized by a single AU and 2) generate the DFG of the decomposed instruction. For example, a four-input maximum can be decomposed into two stages two-input maximum [the DFGs of MAX is shown in Fig. 11(e)]

$$\mathrm{MAX}(a,b,c,d) = \mathrm{MAX}(\mathrm{MAX}(a,b),\mathrm{MAX}(c,d));$$
$$\mathrm{MAX}(a,b) = \mathrm{MUX}(a,b,\mathrm{CMP}(a,b)).$$

B. Simultaneous Allocation Algorithm

Suppose there are a total of i_{MAX} number of instructions, and all the operations are in the given topological orders for AU array allocation. The key idea of the simultaneous algorithm is considering all instructions at the same time with all the operations allocated sequentially in the given topological order. Fig. 12(a) shows the allocation growing path of the instruction set consists of seven instructions based on Algorithm 1.

The DFGs of the input instructions are shown in Fig. 11. First, all the operations whose input are from the previous pipeline stage are allocated into the first row of the AU array. Then the remaining operations whose input are not from the previous pipeline stage are allocated simultaneously and try to share as many common AUs as possible. The allocation result of utilized AU for allocating of the first 5 operations is shown as the very left graph of Fig. 12(a).

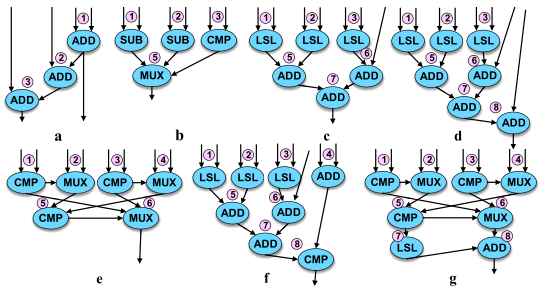


Fig. 11. DFGs of the test instruction set. (a) Long Addition. (b) ABS. (c) MUL. (d) MAC. (e) MAX. (f) Square Root. (g) WTA.

Algorithm 1 Simultaneous Allocation Algorithm

Input: DFGs of Instruction set and data path P of the 5*5 AU array
Output: Minimum Area of configurable AU array
1: Allocate the 1st group operations
2: Initialize the complexity of each instruction
3: **for** j = ($j_{\mathrm{MAX\_1st}}$+1) .. ($j_{\mathrm{MAX\_1st}}+j_{\mathrm{MAX\_2nd}}$), **do**
4: **for** i based on the complexity order, from high to low **do**
5: **repeat**
6: Allocate the operation O_{ij}
7: **until** find the location with minimum cost in that row
8: **end for**
9: Recalculate the complexity of each instruction
10: **end for**
11: Calculate the total rectangular area of the allocated ALUs S
12: **return** S and the floor plan of AU array

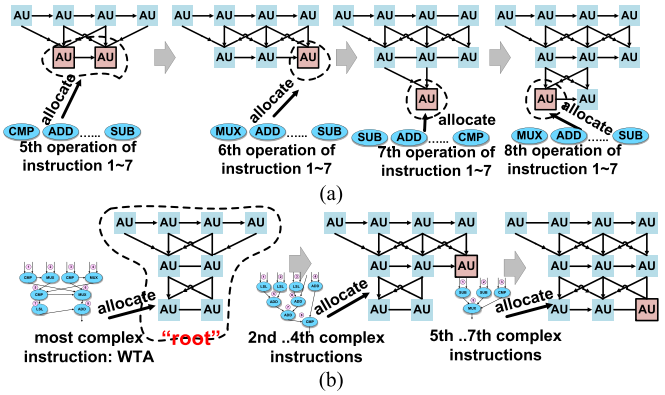


Fig. 12. Allocation algorithm growing paths. (a) Simultaneous algorithm. (b) Incremental algorithm.

After the 1st operation of all instruction sets is allocated, the 2nd, 3rd, …nth operations will be allocated into the AU array continuously. The remaining allocation result of those operations is shown in Fig. 12(a).

C. Incremental Allocation Algorithm

The incremental algorithm (its pseudocode is shown in Algorithm 2) is basically considering the most complex instruction first and then "growing" other instructions based on it. The most complex instruction will be fully optimized in the first place. The remaining instructions will be optimized based on the order of complexity, as given in (9). The previous utilized AU will be recorded and put into high priority when the following instructions are allocated. The complexity of each instruction is calculated based on the sum of the

Algorithm 2 Incremental Allocation Algorithm

Input: DFGs of Instruction set and data path P of the 5*5 AU array
Output: Minimum area of configurable AU array
1: Allocate the 1st group operations
2: Calculate the complexity of each I
3: Initialize the AU usage domain AU_usage
4: **for** i based on the complexity order, from high to low **do**
5: **for** j = (j_{MAX_1st}+1) .. (j_{MAX_1st}+j_{MAX_2nd}) **do**
6: **if** there are available locations in **AU**_$usag$ for operation O_{ij}
7: allocate operation O_{ij} within AU_usage with minimum area cost
8: **else** allocate operation O_{ij} in rest of the AUA and update AU_usage
9: **end for**
10: **end for**
11: **return** S and the floor plan of AU array

weighted interconnects and operations, while the weight of each interconnect is chosen based on the distance between the two operations it connects

$$\text{complexity}_i = \sum O + \sum_{j=1\ldots i\max}^{j\neq i} (C_{ji} I_{ji}) \tag{9}$$

where $\sum O$ represents the number of operations in ith instruction, and I_{ji} represents the number of connects between ith and jth instructions with weight C_{ji}.

Fig. 12(b) shows the allocation growing procedure of the instruction set based on the incremental algorithm. The most complex instruction winner-take-all (WTA) is first allocated. The floor plan of the AU array, which serves as the "root" is shown in the very left graph in Fig. 12(b) after the first allocation. The following instructions just "grow" on this "root" and try to use as many AU as possible within the "root."

The main differences between simultaneous and incremental algorithms are: 1) simultaneous algorithm has a wider vision of the whole allocation tasks, while incremental algorithm just consider the instructions one by one and 2) the search space of incremental algorithm is smaller than the simultaneous algorithm since incremental one considers the AU based on allocation history without taking consideration of the unallocated AUs (AUs which are not in the "root").

D. Evaluation of Allocation Algorithms

In order to test the correctness and efficiency of the simultaneous and incremental allocation algorithms, we run each algorithm on a group of instructions, which consists of seven instructions whose DFGs are shown in Fig. 11.

1) Area and Runtime: In this test, we use seven different instruction groups that contain different numbers $(1, 2, \ldots, 7)$ of instructions from the original instruction set to evaluate the impact from the various number of instruction sets to be supported. The algorithms are implemented in MATLAB and run on a Windows machine with 2.6-GHz i7 Quad-core and 8-GB RAM. For each group, we run the simultaneous and incremental algorithms and compare with a brute-force exhaustive search method where we exhaustively search all possible solutions.

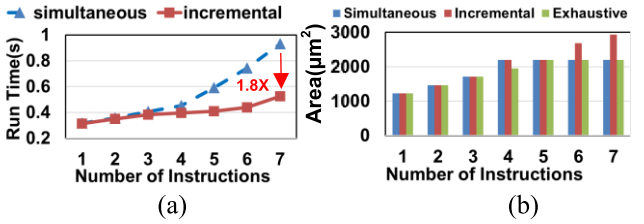


Fig. 13. (a) Runtime comparison. (b) Area comparison during interconnect choice level 1.

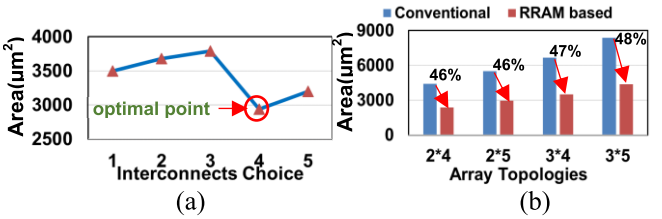


Fig. 14. (a) Optimal interconnects choice. (b) AU array area comparison between the conventional and RRAM-based designs.

Fig. 13(a) shows the runtime results. The incremental allocation consumes less runtime because its searching space is much smaller than the simultaneous allocation. The runtime benefit increases with more instructions included. The incremental allocation would achieve the runtime improvement of 1.8× when there are seven instructions to be allocated. For comparison, the runtime of exhaustive allocation is 32 min which is a thousand times larger than the proposed two algorithms.

Fig. 13(b) shows the area consumption of AU array between the two algorithms with default interconnect. The incremental algorithm introduces more area overhead because it does not consider all instructions simultaneously during allocation, while simultaneous algorithm does, which was discussed in Section V-C. The area overhead of incremental allocation algorithm drops with more flexible interconnect options introduced. It is because, with extra reconfiguration flexibility introduced, the following instructions are easier to be allocated into the original shape and will not introduce extra AU usage. Overall, compared with the exhaustive search, the simultaneous allocation algorithm always provides the same solution while incremental allocation shows a maximum overhead of 11% with ∼2× runtime improvement.

2) Optimal Choice of the Interconnects: Fig. 14(a) shows the final configurable AU array area change allocated by the simultaneous algorithm with different interconnects choices. The x-axis represents the interconnect choice level: 1) level 1 contains interconnect option 1; 2) level 2 contains options 1 and 2; and 3) level 3 options 1, 2, and 3, etc. Note that some of AUs are turned into bypass mode to allow signal passing because only limited routing options are supported. The area consumption initially rises with more routing options, because the interconnect is not sufficient to reduce the AU usage numbers. But after more flexibility in routing is added, i.e., the optimal level of 4, the total shape of the AU array changes from 3 × 4 into 2 × 5 leading to the decrease of AU numbers and total area. Fig. 14(b) shows

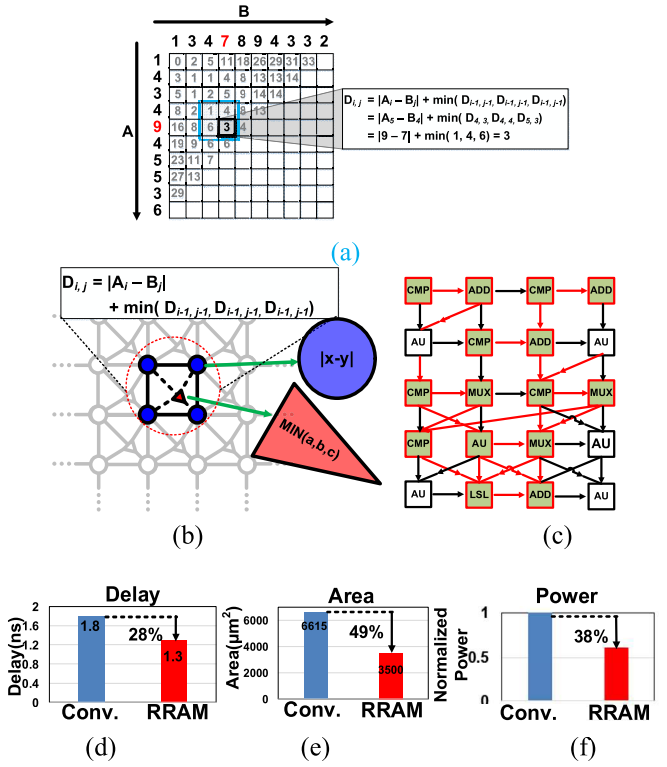


Fig. 15. (a) DTW algorithm. (b) DTW circuit diagram. (c) AU array floor plan and configuration of a DTW processing unit. (d) Delay comparison. (e) Area comparison. (f) Power comparison with the conventional design.

the area comparison between conventional and RRAM-based AU array (R-accelerator) with different topologies of the array. More than 45% reduction in area is observed in each case.

VI. Results and Analysis

A. Case Study on Time-Series Analysis

In order to evaluate the proposed R-accelerator design, we performed a case study using a DTW algorithm which is commonly used in time series classifications [24]. Fig. 15(a) shows the basic principle of DTW, which detects similarities among temporal signals despite the variable speed. As shown in Fig. 15(a), for two time series A and B, $D_{i,j}$ can be formulated as the summation of absolute difference $|A_i - B_j|$ and minimum value of its three ancestor nodes $\min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1})$ where A_i and B_j denotes the ith and jth elements of A, B, and $D_{i,j}$ denotes the DTW value at node (i, j). The instruction set of the DTW algorithm contains the following instructions: WTA/MIN, ABS, and ADD. The core operation of the algorithm is to accumulate the minimum value from its three ancestor nodes. The algorithm diagram is shown in Fig. 15(a) and the final AU array floor plan (5×4 AU array) is shown in Fig. 15(b) for a single processing unit of DTW. A configuration example of a DTW operation is also shown in Fig. 15(b) with the operation and interconnect marked in red. The final layout of the configurable AU array is shown in Fig. 16 in comparison with conventional ASIC design showing a total of 49% area saving of which, 27% comes from proposed logic contraction, 11% comes from

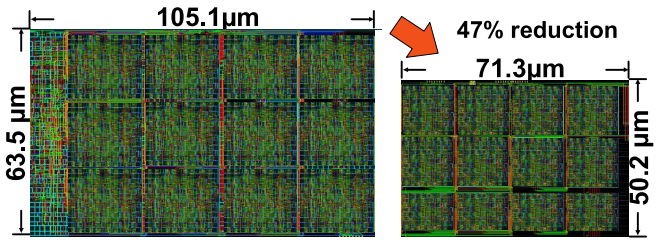


Fig. 16. AU array layout comparison: conventional AU array (left) and RRAM-based AU array (right).

TABLE II
Delay Reduction of Different Operations in AU

Function	ADD	SUB	LSL	LSR	CMP	XOR	MUX
Reduction	34%	32%	23%	18%	41%	21%	36%

TABLE III
Power Saving of Different Operations in AU

Function	ADD	SUB	LSL	LSR	CMP	XOR	MUX
Saving	38%	36%	27%	27%	27%	25%	36%

allocation algorithms, 4% comes from interconnections, and 7% comes from associated memory components.

To estimate the performance (delay) improvement, we performed "case-based" static timing analysis (STA) where the delay is reported under the constraint of the configuration of the design rather than general worst-case critical path. The "case-based" STA reported by Cadence Encounter tool provides a more accurate estimation of the specific operation of the accelerator. The same analysis is applied to the conventional design on the same reconfigurable module. Table II shows the delay in improvement for each configuration of the RRAM-based AU compared with conventional AU. Interestingly, CMP has the largest delay improvement, possibly due to high delay contribution of the selection logic in conventional AU design. Table III shows the power saving of under different configuration using similar "case-based" power analysis in Cadence Encounter tool.

Fig. 15(c) and (e) show the total improvement of delay and power from the proposed R-accelerator design, the processing unit of DTW accelerator compared with conventional ASIC design. Overall, 28% delay and 38% power saving are achieved as a result of the significantly smaller design in the proposed R-accelerator.

B. Case Study on General Benchmarks

1) Benchmarks: To evaluate the performance in more general use cases, we selected a set of six benchmark programs from [26]. The selected benchmarks are fft, adpcm, basicmatch, bitcount, qsort, and rigndael, all of which have repetitive computing loops suitable for mapping into R-accelerators. For each benchmark, we first used the low level virtual machine (LLVM) [27], [29] to obtain the DFGs and representative computing loops. We then identify custom instructions from the DFGs of each benchmark using instruction decomposition described in Section V-A. The characteristics of representative loops are shown in Table IV.

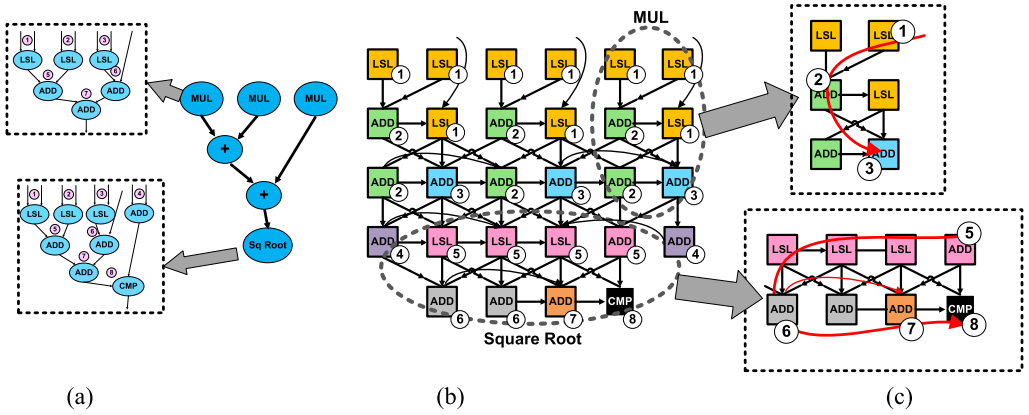


Fig. 17. (a) DFG of the representative loop in "qsort." (b) AU array and interconnection configuration diagram for "qsort." (c) Detailed configuration road map.

TABLE IV
BENCHMARK CHARACTERISTIC

Bench mark	Nodes	Edges	Domain
fft	28	41	Telecom
adpcm	27	47	Telecom
basicmatch	18	25	Automotive
bitcount	16	28	Automotive
qsort	27	42	Automotive
rigndael	16	28	Security

2) Experiment Setup: In our experiments, we compare the proposed RRAM-based reconfigurable architecture with conventional reconfigurable architecture. To obtain an optimal reconfigurable architecture used for the different benchmarks mentioned in Section VI-B, we take the DFGs of each representative loop as input and utilize the simultaneous allocation algorithm proposed in Section V-B to find out the optimal topology of the AU array.

3) Performance Results: Fig. 17 shows the detailed implementation of the example benchmark "qsort." The DFG of the representative loop in "qsort" is shown in Fig. 17(a), which contains three multiplications, two additions, and a square root. Fig. 17(b) shows the final configuration floor plan of the R-accelerator, which contains a 5×6 AU array. Fig. 18(a) and (b) show the power and delay comparison between proposed R-accelerator and conventional reconfigurable architectures through different benchmarks. The overall geometric mean of power-saving and delay improvement are 33% and 32%, respectively. For power saving, "fft" has the highest energy saving of 37%. This can be explained by the fact that "fft" has the largest number of multiplication (addition) operations, which have higher saving compared to other operations. The highest delay improvement is 38% achieved by "adpcm" due to dominant MUX operations, which have higher delay reduction compared to other operations. Fig. 18(c) shows the total area-saving of 45% achieved by the proposed R-accelerator design. Of the 45% area reduction, 25% comes from proposed logic contraction, 9% comes from the allocation algorithm, 4% comes from interconnections, and 7% comes from associated memory components.

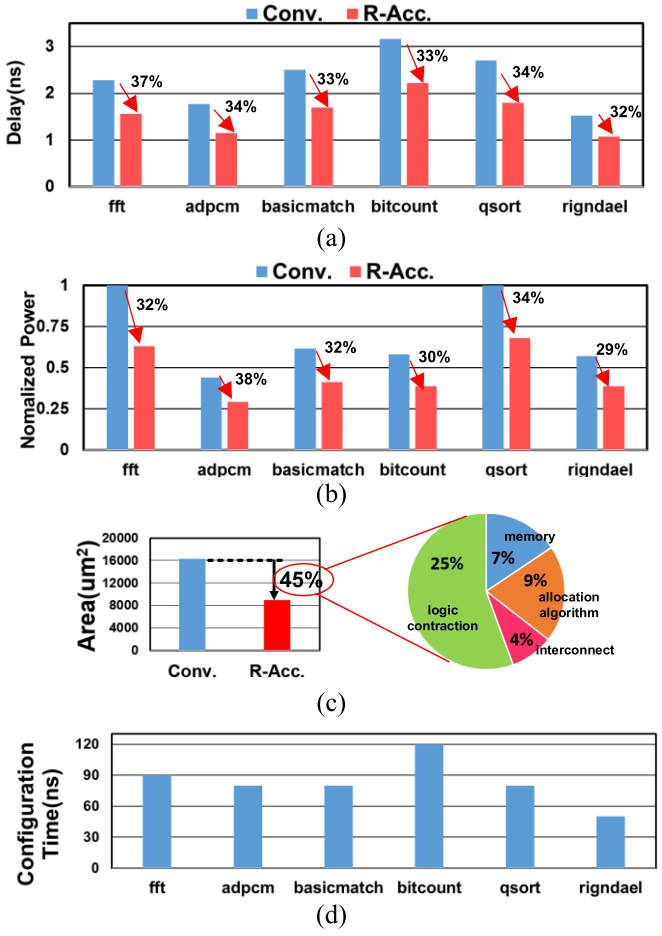


Fig. 18. (a) Power comparison. (b) Delay comparison. (c) Area comparison. (d) Configuration time comparison.

4) Configuration Time Analysis: One major difference of the proposed reconfigurable R-accelerator is that the RRAM needs to be written for each configuration. Compared with standard CMOS logic, the written speed of RRAM is slower, and thus, leads to the drawback of the proposed design. In addition, a special sequence needs to be enforced for configuration to ensure deterministic writing to each RRAM device. We study the impact of RRAM write in this section. The write time is assumed to be a moderate speed of 10 ns,

as listed in Table I. Fig. 17(b) and (c) show the configuration flow of the accelerator for "qsort" program. The different color indicates the order of reconfiguration: 1) the AUs with the same color are reconfigured at the same time and 2) the configuration order of the RRAM-based AU is marked by the number at the right-bottom corner of the AU. The detailed reconfiguration of multiplication and square root are shown in Fig. 17(c).

Fig. 18(d) shows the configuration time for different benchmarks. The configuration time varies from 50 to 120 ns and among them "bitcount" has the longest configuration time due to a more complex and longer DFG it has. Although the longer configuration time of the proposed R-Accelerator is a drawback compared with conventional design, the overhead of reconfiguration only happens once at the beginning of the program and can often be hidden through careful scheduling with CPU's operation. As RRAM technology improves, the configuration time due to RRAM write is expected to be significantly shortened.

5) Endurance and Variation Impact: As described in Section III-A, some practical issues of RRAM/RRAM devices have been reported including: 1) variability of the cell resistance especially at high resistance state and 2) endurance of resistance value during writing and reading [16], [17]. We performed variability test by varying HRS and LRS by a factor of 300% and evaluated the performance, i.e., cell delay impact. Our evaluation shows the performance variation is within 10% of the nominal value because the logic cell delay is strongly influenced by intrinsic transistors from driving buffers rather than the interconnect variation.

Such a performance variation indeed impacts the writing time of the RRAM device. However, the proposed reconfiguration operation using the RRAM device is only performed at the beginning of each program, and thus, can be repeatedly tuned to eliminate the potential variation of the device resistance. Overall, less than 1% delay degradation of the proposed R-accelerator is observed among all the experiments due to the RRAM variation. In addition, compared with the previous proposed neuromorphic application where high resolution and sturdy resistance of RRAM device is required, the proposed circuits have significantly relaxed requirement on the variability and durability of the device, and thus, can be easily utilized from commonly used RRAM/RRAM devices.

VII. Conclusion

We proposed a novel design methodology for creating reconfigurable application-specific accelerator using the RRAM device. A logic contraction technique along with a modeling technique was developed to significantly reduce the area cost of reconfigurable AUs. Two heuristic resource allocation algorithms were developed to achieve the optimal solution for AU placement and interconnect choices. Study on an example of DTW and general benchmark programs show that compared with conventional design methodology, a total of 49% area reduction, 28% of delay improvement, and 38% dynamic power saving are achieved.

References

[1] M. Borgatti, F. Lertora, B. Forêt, and L. Calí, "A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA, and customizable I/O," *IEEE J. Solid-State Circuits*, vol. 38, no. 3, pp. 521–529, Mar. 2003.

[2] T. Thanh-Hoang, A. Shambayati, and A. A. Chien, "A data layout transformation (DLT) accelerator: Architectural support for data movement optimization in accelerated-centric heterogeneous systems," in *Proc. EDAA*, 2016, pp. 1489–1492.

[3] T. Nakagawa and T. Shibata, "A real-time image feature vector generator employing functional cache memory for edge flags," in *Proc. IEEE ISCS*, May 2009, pp. 3026–3029.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Comput. Learn. Theory*, 1992, pp. 144–152.

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

[6] D. Rossi, F. Campi, S. Spolzino, S. Pucillo, and R. Guerrieri, "A heterogeneous digital signal processor for dynamically reconfigurable computing," *IEEE J. Solid-State Circuits*, vol. 45, no. 8, pp. 1615–1626, Aug. 2010.

[7] F.-L. Yuan, C. C. Wang, T.-H. Yu, and D. Marković, "A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 137–149, Jan. 2015.

[8] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-S. Kim, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *Proc. PAC*, 2008, pp. 166–176.

[9] H. Park, K. Fan, M. Kudlur, and S. Mahlke, "Modulo graph embedding: Mapping applications onto coarse-grained reconfigurable architectures," in *Proc. CASES*, 2006, pp. 136–146.

[10] Crossbar. Accessed: Dec. 2017. [Online]. Available: http://www.crossbar-inc.com/assets/resource/whitepaper/Crossbar-RRAM-Technology-Whitepaper.pdf

[11] Y. Shim, A. Sengupta, and K. Roy, "Low-power approximate convolution computing unit with domain-wall motion based 'Spin-Memristor' for image processing applications," in *Proc. DAC*, Jun. 2016, pp. 1–6.

[12] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proc. AISCA*, 2016, pp. 27–39.

[13] Y. Chen and H. Zhou, "Resource-constrained high-level datapath optimization in ASIP design," in *Proc. DATE*, 2013, pp. 198–201.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability*. Murray Hill, NJ, USA: Bell Laboratories, Jan. 1979.

[15] K. Tsunoda *et al.*, "Low power and high speed switching of Ti-doped NiO ReRAM under the unipolar voltage source of less than 3 V," in *IEDM Tech. Dig.*, Dec. 2007, pp. 767–770.

[16] A. Kawahara *et al.*, "An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2012, pp. 432–434.

[17] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor SPICE model and crossbar simulation based on devices with nanosecond switching time," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2013, pp. 1–7.

[18] R. Huang *et al.*, "Resistive switching of silicon-rich-oxide featuring high compatibility with CMOS technology for 3D stackable and embedded applications," *Appl. Phys. A, Solids Surf.*, vol. 102, no. 4, pp. 927–931, 2011.

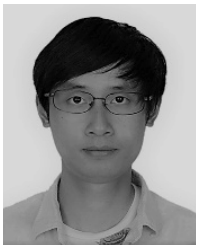
[19] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Adv. Mater.*, vol. 23, no. 47, pp. 5633–5640, 2011.

[20] T. Diokh *et al.*, "Investigation of the impact of the oxide thickness and RESET conditions on Disturb in HfO_2-RRAM integrated in a 65 nm CMOS technology," in *Proc. IRPS*, Apr. 2013, pp. 5E.4.1–5E.4.4.

[21] J.-J. Huang, Y.-M. Tseng, W.-C. Luo, C.-W. Hsu, and T.-H. Hou, "One selector-one resistor (1S1R) crossbar array for high-density flexible memory applications," in *IEDM Tech. Dig.*, Dec. 2011, pp. 31.7.1–31.7.4.

[22] X. Tang, E. Giacomin, G. De Micheli, and P.-E. Gaillardon, "Circuit designs of high-performance and low-power RRAM-based multiplexers based on 4T(ransistor)1R(RAM) programming structure," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 5, pp. 1173–1186, May 2017.

[23] X. Tang, P.-E. Gaillardon, and G. De Micheli, "A high-performance low-power near-Vt RRAM-based FPGA," in *Proc. ICFPF*, Dec. 2014, pp. 207–214.
[24] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.
[25] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Proc. DAC*, Jun. 2017, pp. 1–6.
[26] *Mibench*. Accessed: Dec. 2018. [Online]. Available: https://www.eecs.umich.edu/mibench/
[27] *LLVM*. Accessed: Dec. 2018. [Online]. Available: https://www.llvm.org
[28] M. A. Lastras-Montaño, A. Ghofrani, and K.-T. Cheng, "A low-power hybrid reconfigurable architecture for resistive random-access memories," in *Proc. HPCA*, Mar. 2016, pp. 102–113.
[29] P. Biswas, S. Banerjee, N. Dutt, L. Pozzi, and P. Ienne, "ISEGEN: Generation of high-quality instruction set extensions by iterative improvement," in *Proc. IEEE DATE*, Mar. 2005, pp. 1246–1251.
[30] Sylvain DUBOSI. *Crossbar Resistive RAM (RRAM): The Future Technology for Data Storage*. [Online]. Available: http://www.snia.org/sites/default/orig/-DSI2014/presentations/HotTopics/SylvainDuBoise_Future_Technology_final.pdf
[31] Z. Chen, H. Zhou, and J. Gu, "R-accelerator: A reconfigurable accelerator with RRAM based logic contraction and resource optimization for application specific computing," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2018, pp. 163–170.



Zhengyu Chen (S'16) received the B.S. degree in electrical engineering from Southeast University, Nanjing, China, in 2013, and the M.S. degree in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2015. He is currently working toward the Ph.D. degree in electrical and computer engineering at Northwestern University, Evanston, IL, USA.

He is an Aspiring Researcher doing research in the area of ultralow power design/algorithm for VLSI, mixed-signal ICs, and emerging device. His current research interests include the low-power algorithm design such as time-domain signal processing and accelerator design of machine learning algorithms.

Hai Zhou (SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from the University of Texas at Austin, Austin, TX, USA, in 1999.

He is currently an Associate Professor of electrical and computer engineering with Northwestern University, Evanston, IL, USA. His current research interests include VLSI computer-aided design, algorithm design, and formal methods. He has authored more than 150 papers in flagship journals and conferences in these areas.

Dr. Zhou received the CAREER Award from the National Science Foundation.

Jie Gu (SM'19) received the B.S. degree from Tsinghua University, Beijing, China, in 2001, the M.S. degree from Texas A&M University, College Station, TX, USA, in 2003, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2008.

He was an IC Design Engineer with Texas Instruments, Dallas, TX, USA, from 2008 to 2010, focusing on ultralow-voltage mobile processor design and integrated power management techniques. He was a Senior Staff Engineer with Maxlinear, Inc., Dallas, TX, USA, from 2011 to 2014, focusing on low-power mixed-signal broadband system-on-chip (SoC) design. He is currently an Assistant Professor with Northwestern University, Evanston, IL, USA. His current research interests include ultradynamic clock and power management for microprocessor and accelerators, emerging mixed-signal computing circuit, and the design of machine learning capable edge devices.

Dr. Gu was a recipient of the NSF CAREER Award. He has served as a Program Committee and Conference Co-Chair for numerous low-power design conference and journals, such as ISPLED, DAC, ICCAD, and ICCD.