High-Throughput Pattern Matching With CMOL FPGA Circuits: Case for Logic-in-Memory Computing

Advait Madhavan, *Member, IEEE*, Tim Sherwood, *Senior Member, IEEE*, and Dmitri B. Strukov, *Senior Member, IEEE*

***Abstract*— In this paper, we propose a novel CMOS+MOLecular (CMOL) field-programmable gate array (FPGA) circuit architecture to perform massively parallel, high-throughput computations, which is especially useful for pattern matching tasks and multidimensional associative searches. In the new architecture, patterns are stored as resistive states of emerging nonvolatile memory nanodevices, while the analyzed data are streamed via CMOS subsystem. The main improvements over prior work offered by the proposed circuits are increased nanodevice utilization and, as a result, substantially higher throughput, which is demonstrated by a detailed analysis of the implementation of pattern matching task on the new architecture. For example, our estimates show that the proposed CMOL FPGA circuits based on the 22-nm CMOS technology and one crossbar layer with 22-nm nanowire half-pitch allows up to 12.5% average nanodevice utilization, i.e., the fraction of the devices turned to the high conductive state, as compared to a typical ~0.1% of the original CMOL FPGA circuits. This in turn enables throughput close to 7.1 × 10^{16} bits/s/cm^2 at ~ 1 fJ/bit energy efficiency, for matching of ~ 10^7 250-bit patterns stored locally on a 1 cm^2 chip. These numbers are at least 2 orders of magnitude better throughput as compared to that of other state-of-the-art FPGA methods, and begin to approach ternary content-addressable memory -like performance at similar CMOS technology nodes. More generally, we argue that the proposed concept combines the versatility of reconfigurable architectures and density of the associative memories. It can be viewed as a very tight symbiotic integration of memory and logic functions for high-performance logic-in-memory computing.**

***Index Terms*— CMOS+MOLecular (CMOL), field-programmable gate array (FPGA), hybrid circuits, logic-in-memory computing, memristor, pattern matching, ReRAM, resistive switching, ternary content-addressable memory (TCAM).**

I. Introduction

RECONFIGURABLE circuits (RCs) are very efficient for information processing tasks [1], such as image and signal processing (e.g., filtering, edge detection, coding, and feature extraction [2], [3]), and string matching (e.g., in a context of network intrusion detection [4]–[7], deoxyribonucleic acid sequence matching [8], [9], database searching [10], and network packet routing [11], [12]). The common feature in these tasks is that they can be efficiently parallelized, and that the same basic operation is performed numerous times using one set of fixed data known in advance (which are allowed to change infrequently), such as a filter template in image processing or keyword in string matching, along with streaming input data. In this respect, RCs offer massively parallel, instant-specific computation customized for the needs of the particular application, and thus potentially offer real-time processing coupled with low-power consumption.

However, even contemporary RCs cannot provide enough computational power for future demands. For example, in network intrusion detection applications, deep packet information of computer networks is compared against a known sequence of data representing a computer virus or other malicious content [4], [13]. In order to provide real-time protection, the search engine should perform many comparisons in parallel, and simultaneously allow for updating of virus signatures. Earlier RC implementations were adequate to ensure a few Gbit/s/cm^2-scale sustained throughput for ~2000 100-B long patterns [4]. The throughput could be further significantly improved by employing dynamic reconfiguration and customized hardware, including dedicated ternary content-addressable memories (TCAMs) [5], [7], [13], [14]. However, even these techniques have limited benefits, largely due to excessive reconfiguration overhead for multicontext field-programmable gate arrays (FPGAs) [1], I/O limitations for dynamic reconfiguration, and/or rigid inefficient structure of content-addressable memories (CAMs), and thus may be insufficient for future needs. The deployment of faster 100-Gbit/s-scale data networks, as well as the continued increase in the number of patterns (e.g., the number of known viruses) makes real-time protection impossible even for the most advanced circuit implementations with CMOS technology.

The performance of RCs can be greatly improved using hybrid CMOS/nanoelectronic circuits [15]–[18]. One such example is CMOS+MOLecular (CMOL) FPGA [17], [19]–[23], where CMOL stands for CMOL scale hybrid circuit, which was conceived to seize the density advantages of emerging technologies, such as nanoimprint lithography and monolithically integrated self-assembled nanodevices,

Manuscript received September 7, 2017; revised January 16, 2018; accepted February 17, 2018. Date of publication March 20, 2018; date of current version November 30, 2018. This work was supported in part by RICARDO through NSF under Grant 1730309, in part by NSF under Grant 1563935, in part by CCF through NSF under Grant 1740352, and in part by AFOSR MURI under Grant FA9550-12-1-0038. *(Corresponding author: Dmitri B. Strukov.)*

A. Madhavan and D. B. Strukov are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA (e-mail: amadhavan@umail.ucsb.edu; strukov@ece.ucsb.edu).

T. Sherwood is with the Department of Computer Science, University of California, Santa Barbara, CA 93106 USA (e-mail: sherwood@cs.ucsb.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.2018.2809644

and to combine it with the flexibility and versatility of CMOS technology. Most of the configuration overhead in CMOL FPGAs, including all configuration memory and some routing circuitry, is lifted above the CMOS plane. Logic gates are based on a combination of nonlinear nanoscale resistive switching devices (which are also called "memristors" [24]) and CMOS logic, which improves the aggregate density of the logic circuitry [20], [21], [25]. In other CMOL-FPGA-like concepts, nanoscale devices are only utilized for routing purposes [22], [23].

Though the density advantage is significant, the nanodevice utilization in the previously reported works on CMOL FPGAs [19], [21], [23], [26]–[30] is well below 1% due to the limited benefits of utilizing high fan-in gates. In this paper, we present a modification to the original logic structure [19], [20] and show that some information processing tasks are uniquely suited for the high fan-in gates of CMOL FPGA circuits. The TCAM-like cell architecture allows for a more efficient use of memristive devices resulting in much higher performance, while still being able to maintain the reconfigurability, hence blending the best of both worlds. The close proximity of the nanodevices to CMOS, by virtue of the vertical integration, allows for synergistic interaction between memory and computation, hence resulting in state-of-the-art performance.

Some of the architecture details and one application study have been reported earlier in [30]–[32]. The major contribution of this paper is as follows.

1) A performance analysis of the proposed circuit. Our estimates account for the sizing of CMOS circuits, which was generally neglected in previous CMOL FPGA work, though crucial for providing correct functionality for the considered circuits.
2) An optimization procedure that considers architectural, topological, and circuit-level constraints to maximize the throughput of the proposed circuits.
3) An additional application case study.

The rest of this paper is organized as follows. In Section II, we briefly review the background material on pattern matching, the considered resistive switching devices, and CMOL circuits. In Section III, we introduce modified CMOL FPGA architecture. Section IV discusses two applications mapped on a new architecture, while performance modeling results are provided in Section V. Finally, the results are discussed and summarized in Section VI.

II. Background

A. Pattern Matching

At a high level, contemporary high-performance pattern matching approaches can be divided into two groups. The first approach makes use of the reconfigurable nature of FPGA, exploiting the fine-grain configurability of the devices to implement a dense pattern matching structure [1], [33]–[36]. For example, many FPGA schemes make use of the configurable interconnect to stream data through a series of basic pattern matching operations performed by lookup tables inside logic blocks (Fig. 1). Going a step further, the reconfigurable nature of the hardware can be exploited to optimize matching

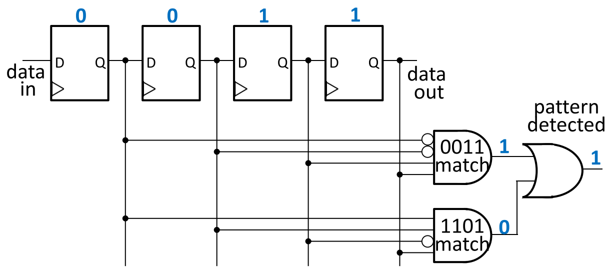

Fig. 1. Example of pattern matching circuitry, which is designed to detect two patterns, "0011" and "1010" in data streaming along the chain of DFFs. The bit values alongside wires illustrate a specific example of the data values in the pipeline and resulting logic values, i.e., detection of "0011" pattern. Note that three logic gates (2 AND and 1 OR) can be realized with one 4:1 lookup table in the FPGA implementation.

structures for the particular set of patterns/expressions being searched [37]–[39], e.g., through a technique analogous to common expression elimination [4], [36] or by constructing deterministic and nondeterministic finite-state automata for recognition [40]–[42]. The flexibility and bit-level configurability of FPGAs make them a natural platform for instance-specific highly parallel implementations in which both memory functions (i.e., storage of patterns and logic operations) are performed locally. On the other hand, reconfigurability comes at a high price, typically with $\sim 40\times$ larger area and $\sim 3\times$ longer delay as compared to custom circuit implementations [43].

The second approach is based on TCAMs [44]–[56], which allows bit-level comparisons of streaming data against stored patterns in massively parallel fashion [Fig. 2(a)]. The relatively dense structure of CAMs, which are roughly $2\times$ sparser than conventional static random access memories (SRAMs), allows more patterns to be stored in the same unit of silicon as compared to FPGA approaches. The ternary aspect of the TCAM allows it to match do not care conditions as well. The downside of this approach is that the long memory lines used for matching must be charged and discharged on each and every search cycle, even when no matches are to be found.

The principle of operation of a conventional SRAM-based CAM is shown in Fig. 2(b). It consists of several CAM memory cells arranged along a match line. Each CAM cell has a dual-inverter memory element (which comprises 4 transistors), and 4 match and pull-down transistors. (The read/write circuitry of each memory cell, which is another 4 transistors, has been left out for clarity.) Once the data have been stored in the memory element, the search operation is initiated by precharging the match line to a logical "1." The data to be searched are then presented along the search lines. Depending upon the data stored in the memory element, on a mismatch, there will be a clear discharge path from the match line to ground and on a match, there will be no discharge.

Fig. 2(c) shows an SRAM TCAM cell with the ability to store do not cares by splitting the memory cell into two, which can now both store zeros, thereby always keeping the discharge path off. Similar to the T/CAM cells shown in Fig. 2(b) and (c), there were many proposals of TCAM cell implementations with other memory technologies. For example, Fig. 2(d)–(g) shows TCAM cells based on flash

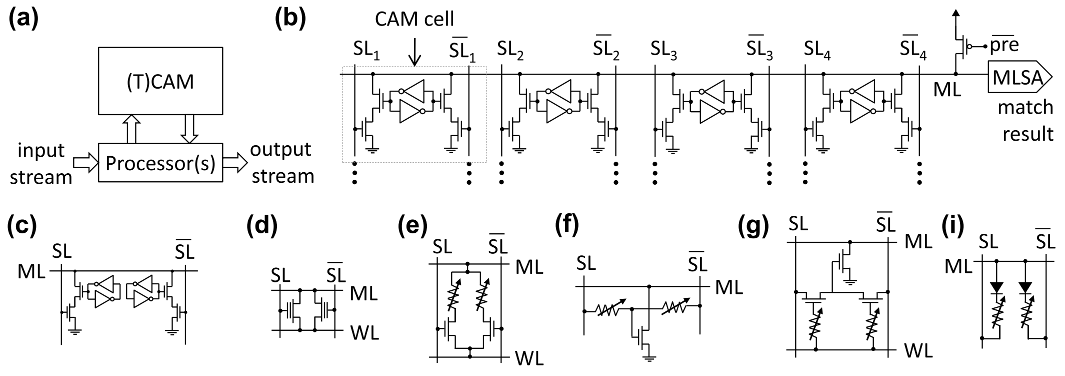

Fig. 2. Pattern matching with CAMs. (a) General idea. (b) Example of one row of SRAM-based CAM memory implemented in OR style [14]. (c) SRAM-based TCAM memory cell [14]. (d)–(i) TCAM cells based on nonconventional memory technologies. (d)-(g) TCAM cells based on flash memory [48], hybrid CMOS/MRAM [49], CMOS/STT-RAM [50], and CMOS/memristors [51] technologies, respectively. (i) Memristor-based implementation.

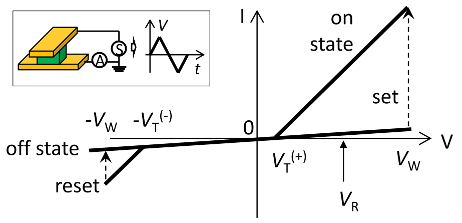

Fig. 3. Idealized I–V for bipolar memristors. (Inset: Cartoon of a crosspoint device.).

memory [48], hybrid CMOS/magnetic random access memory (MRAM) [49], CMOS/spin torque transfer (STT)-RAM [50], and CMOS/memristor [51] circuits. In this paper, we consider the implementation of TCAM cell with a pair of two memristors [Fig. 2(i)] [30]–[32].

More recently, pattern-matching implementations were suggested based on hyper-dimensional memory [57] and micrometer automata processor [58], [59]. In spite of algorithmic differences, the operation of hyper-dimensional memory circuit is somewhat similar to that of TCAM with the added functionality of measuring the sense current which is representative of the distance of the mismatched query pattern with the stored patterns. On the other hand, automata processor approach is more similar to FPGA computing in the ability to perform fine grain, massively parallel operations on a stream of input data. It is essentially a "sea-of-gates" fabric with Boolean logic gates and counters interconnected with a reconfigurable routing network, but is more catered toward implementations of high-throughput nondeterministic finite-state machines.

B. Resistive Switching Devices

Resistive switching devices [24] are a key ingredient of the proposed CMOL FPGA pattern matching circuits. (In this paper, we also use terms "crosspoint device," "nanodevice," or simply "device" to describe memristors.) In its simplest form, a memristor consists of three layers: top and bottom (metallic) electrodes, and a thin film of some insulating material (inset of Fig. 3), most typically transition metal oxide, which can undergo resistive switching [24]. Specifically, by applying a relatively large ("write") voltage bias (V_W) across the electrodes of such a nanodevice, the thin film can be switched reversibly between high ("ON") and low ("OFF") conductive states, characterized by R_{ON} and R_{OFF} resistances, respectively. For properly engineered nanodevices, the conductive state can be retained indefinitely and probed without disturbing it by applying relatively small ("read") voltage bias ($\leq V_R$). Because of an ionic memory mechanism [24], and a simple structure, which is conducive to aggressive lithographic and other patterning techniques, memristors have excellent density prospects. For example, several groups have recently shown metal oxide memristors with nanodevice area below 15×15 nm^2 [60], [61], which is defined by the overlap area of bottom and top electrodes.

Switching the bipolar nanodevice between high conductive and low conductive states is accomplished by applying write voltages of opposite polarity. For example, Fig. 3 shows hysteretic I–V curve for idealized bipolar nanodevice for which applying $V \geq +V_W$ across the device would switch it into the ON state (so-called set transition), while applying negative voltage $V \leq -V_W$ would switch it back (reset) to the OFF state.

The very high density of individual memristors can be sustained at the circuit level by employing passive crossbar structures, which consists of mutually perpendicular nanowires with nanodevices formed at their crosspoints. Crossbar integration imposes additional requirements for the memristors, such as the need for low forming voltage [24]. (The forming process is a one-time application of a relatively large voltage or current pulse to turn an as-fabricated "virgin" nanodevice to operational memristor.) Other major challenges of passively integrated crossbar circuits are state disturbances of half-select devices during write operation, sneak-path currents during read operation, and the common problem of currents running via half/unselected nanodevices.

Currents via unselected and half-selected devices, which are much higher for the write operation because of larger voltage ranges, can lead to undesirable voltage drops across nanowires. One of the solutions to this problem is to utilize nanodevices with strongly nonlinear electron transport presented as (diode-like) nonlinear I–V with threshold voltage V_T for the current flow (Fig. 3), which suppresses any unwanted currents in the crossbar circuit [24], [62], [63]. For example,

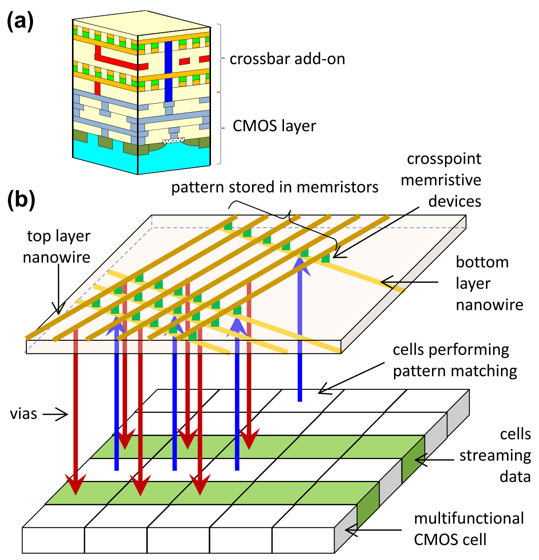
(a)
crossbar add-on
CMOS layer
(b)
pattern stored in memristors
crosspoint memristive devices
top layer nanowire
bottom layer nanowire
cells performing pattern matching
vias
cells streaming data
multifunctional CMOS cell

Fig. 4. (a) Cartoon of CMOL circuits. (b) Functional partitioning of the CMOL circuit for pattern matching applications by the area-distributed interface (red and blue arrows), between the CMOS layer at the bottom and passive nanowire crossbar on top.

the sneak-path is naturally cut by suppressing reverse current for the devices with very asymmetric I–Vs with $V_T^{(+)} \ll V_R < V_T^{(-)}$.

For simplicity, in our analysis, we assume bipolar memristors described by asymmetric idealized I–V curve with precisely defined write voltage V_W, i.e., with no device-to-device and cycle-to-cycle variations.

C. CMOL Structure

In CMOL structures one [17], [25] or several [64]–[66] crossbar layers are vertically integrated on top of conventional CMOS circuits [Fig. 4(a)]. One of the key characteristics of the CMOL architecture are an ability of accessing (reading or writing) every crosspoint nanodevice from much sparser CMOS circuitry without sacrificing crossbar integration density. The crosspoint memristors can be programmed to either high or low resistive states, and together with the CMOS layer create a reconfigurable fabric that can perform information processing (i.e., pattern matching for the considered applications) and interconnect duties.

In particular, the CMOS layer is arranged as an array of "atomic" CMOS cells [Figs. 4(b) and 5], which are connected to the nanoscale crossbar circuit via an area-distributed interface. Each cell houses CMOS circuits that provide unique access to each of the cell's two vias from the cell array periphery, and also CMOS circuitry specific to the implemented CMOL circuit. The nanowire crossbar is rotated with respect to the array of atomic cells underneath, and provides high fan-in and fan-out connectivity between them. For example, each (output) blue via connects to a certain quasi-horizontal nanowire, which in turn connects to multiple quasi-vertical nanowires through crosspoint devices. These quasi-vertical wires each connect to (input) red vias of other surrounding cells. The rotation of the crossbar naturally breaks nanowires into segments, and as a result, every nanowire segment is

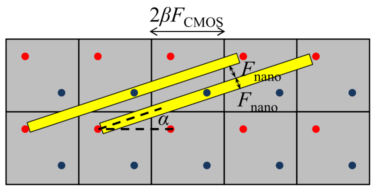

Fig. 5. Crossbar rotation in CMOL shown for specific value of $r = 3$.

connected by only one via (Fig. 5). (Note that more readily manufacturable CMOL structures, with nanowires running strictly in vertical and horizontal directions, are also possible. For example, the effective rotation of the nanowire array can be implemented by adjusting positions of the cells' vias [23] or by using zig-zag shaped nanowires [65].)

Selection of any crosspoint device, to perform read or write operation, is implemented with double-decoding scheme. The first level of CMOS decoders, implemented by the peripheral CMOS decoders and pass gates/transistor of the atomic CMOS cells, is used to select a pair of vias, one blue and one red, which connect to the corresponding mutually perpendicular nanowire segments that lead to the crosspoint device in question. The second level of decoding is implemented with half-biasing approach, which utilizes memristor nonlinearities in switching kinetics and electron transport to enable unique access to the specific crosspoint device.

The angle of the crossbar rotation α depends upon several parameters such as cell complexity, CMOS process, and pitch of the nanowires. Specifically, assuming that F_{nano} and F_{CMOS} are the minimum half-pitch of the nanowire crossbar array and the feature size of CMOS circuitry, respectively, and that the side length of one atomic CMOS cell is $2\beta F_{\text{CMOS}}$, where β is a parameter representing cell's size, the CMOL topology is described by set of equations [17]

$$\tan(\alpha) = \frac{1}{r}, \quad \sqrt{r^2+1} = \frac{\beta F_{\text{CMOS}}}{F_{\text{nano}}}. \tag{1}$$

It is also very convenient to characterize CMOL architecture with parameter M_A

$$M_A = \frac{L - 2F_{\text{nano}}}{2F_{\text{nano}}} - 1 = r^2 - 1 \tag{2}$$

that defines the number of atomic cells connected to one atomic cell and is equal to the number of crossings (memristors) on one nanowire segment. For example, Fig. 7(a) shows a CMOL structure for $r = 6$ with its connectivity domain highlighted, and, in particular, shows that the given atomic cell can be connected to any of the other $M_A = 36$ atomic cells, including connection to itself, in its connectivity domain via the crossbar structure.

For a fixed complexity CMOS cell in a certain process, lowering the pitch of the nanowire allows higher density by increasing the relative angle of the crossbar with respect to the CMOS vias. This also explains how, while keeping nanowire half-pitch constant, maximum crossbar density can be preserved irrespective of the size of the atomic cell, while only affecting its connectivity. In addition, it is worth mentioning that the CMOL segmented crossbar structure is not only good for high fan-in fan-out computation, but also has

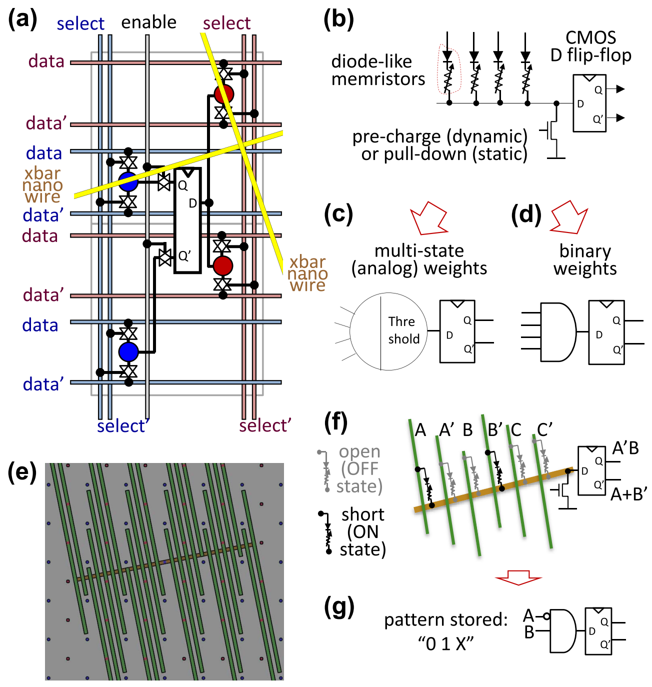

Fig. 6. Proposed CMOL FPGA for pattern matching. (a) Unit cell, which is comprised of two atomic cells, hosting CMOS DFF and pass gates. For clarity, Schmitt trigger is not shown. (b) Equivalent circuit of one unit cell with diode-like memristor having I–V characteristics shown in Fig. 3 suitable for (c) linear threshold logic, or (d) diode–resistor logic. (e) Fragment of CMOL fabric showing nanowires connected to one of the input nanowire of unit cell. (f) Example of unit cell operation. (f) Six inputs [out of 24 total in (e)]. (g) Equivalent gate representation of pattern matching operation for the specific pattern stored in memristors shown in (f).

a large amount of parallelism embedded in it. Two adjacent atomic cells share a considerable portion of their connectivity domains. These shared cells, in spite of sharing quasi-vertical nanowires, interact with the adjacent atomic cells through different memristors as their crosspoints lie on different quasi-horizontal lines.

III. Pattern Matching Circuit Architecture

Fig. 6 shows the proposed CMOL FPGA circuits for pattern matching. The CMOL fabric is a uniform array of "unit" cells, each comprised of two atomic cells. The unit cell implements a CMOS D-flip-flop (DFF) connected via pass gates to cell's vias. To improve voltage margins, we assume that each unit cell also hosts a Schmitt trigger, which is inserted between the cell's input vias and the input of the DFF. Note that the DFFs' inputs and outputs are connected to each other only via crossbar circuit and not via CMOS subsystem.

Similar to the originally proposed circuits [19], [20], memristors at the nanowire crosspoints can be programmed to perform logic as well as interconnect functions. In order to configure the CMOL crossbar circuit to implement custom logic, first, the CMOS block is disabled in all cells by deasserting "enable" line [Fig. 6(a)]. This is equivalent to tristating the output of the CMOS cell such that applied write voltages do not short circuit the output of the DFF's drivers. As a result, any crosspoint device in the crossbar structure can be programmed to the ON or OFF state by utilizing the double-decoding scheme of CMOL memory. Note that unlike the original concept discussed in [19], [20], here, we assume that each (red or blue) via is connected with pass gates to two select and two data CMOS lines [Fig. 6(a)]. Connecting each via to a pair of data lines allow different voltages to be applied independently, which is more desirable for a half-biasing scheme, while in the original concept some of the nanowires were always floated. Also, in principle, pass transistors for connecting data and select lines to vias can be utilized instead of pass gates, however, as our estimates below show, this does not help much in reducing the cell area.

After the programming stage, logic operations are implemented with diode–resistor logic formed by the ON-state nanodevices and CMOS pass transistors [20], while the signal restoration, inversion, and latching are performed by the CMOS subsystem [Fig. 6(b) and (f)]. Similar to conventional implementations, two flavors, static and dynamic, diode–resistor logic are possible with only subtle modification of the underlying hardware (though with a different requirement of the nanodevices). In the static case, the cell's input nanowires are pulled down to the ground via pass transistor. Within the course of a single operation, i.e., performed in one clock cycle, the final output voltage value is determined by the resistive divider formed by the diode–resistor logic. The dynamic case, on the other hand, is similar to TCAM-based implementations with the circuit operation divided into a precharge and an evaluate phases. In the precharge phase, the cell's input nanowires behave like a match lines [similar to Fig. 2(i)] which are precharged low using a pull-down transistor, while the output nanowires are decoupled from their corresponding DFF outputs by deasserting the pass gate inputs. In the evaluation phase, the output enable lines are asserted which enable the proper logic functionality of the dynamic cell by pulling the output voltage high in the case of a mismatch and leaving it low in the case of a match.

The specific logic functionality of each unit cell and its connectivity is governed by the state of memristors connected to its quasi-horizontal nanowire [Fig. 6(e)]. For instance, Fig. 6(f) shows a particular example of implementing function A'B, where signals A, B, C, and their complements are routed from the output of the surrounding unit cells. Both true and complementary values of the signal are available at the output of the DFF, so that each bit of a pattern is represented by 2 memristors. Fig. 6(g) shows the equivalent custom logic gate, which performs the exact pattern matching corresponding to the specific pattern stored in memristors [Fig. 6(f)]. It is worth mentioning that the state of the memristors remains unchanged during logic operation stage, because the voltage drop across memristors are always less or equal to V_R. Also note that while Fig. 6(f) shows matching of two 3-bit patterns, i.e., one pattern comprised by the state of flip-flop cells and another one by the state of memristors, the number of bits in a pattern that can be compared by one unit cell is typically much larger (>100) for practical values of topological parameter r.

The unit cell can be also configured to perform approximate pattern matching when analog properties of memristors are utilized to implement linear threshold gates [Fig. 6(c)] [67]. Such linear threshold gate can implement matching of two

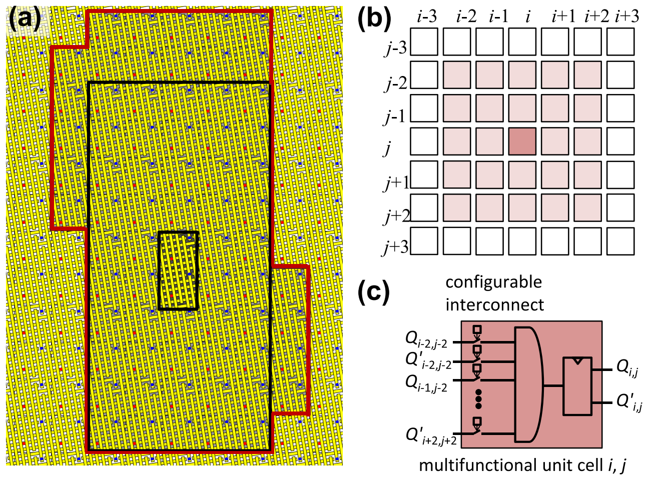
(a)
(b)
i-3 i-2 i-1 i i+1 i+2 i+3
j-3 j-2 j-1 j j+1 j+2 j+3
(c)
configurable interconnect
$Q_{i-2,j-2}$
$Q'_{i-2,j-2}$
$Q_{i-1,j-2}$
$Q'_{i+2,j+2}$
$Q_{i,j}$
$Q'_{i,j}$
multifunctional unit cell i, j

Fig. 7. (a) Top view of crossbar circuit and input connectivity domain for CMOL with $r = 6$, which is considered for application studies. (b) Equivalent sea-of-gate circuit architecture. (c) Functionality of one logic tile. (a) Set of atomic cells bounded by red lines is a physically permitted domain (of atomic cells), while the somewhat smaller domain of unit cells surrounded by black lines is used in mapping examples. In (b), the central tile can be connected, i.e., directly pass output to or take inputs from, to any of the surrounding shaded tiles. In (c), AND gate should be replaced with linear threshold gate when analog properties of memristors are exploited.

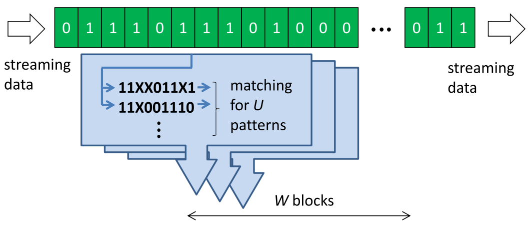

Fig. 8. General idea for pattern matching with 1-D streaming data. Here, we assume that adjacent blocks of pipeline data are shifted by one bit position and the block length matches the pattern length.

patterns based on the Hamming distance between them and the specific threshold for Hamming distance can be programmed infield by setting appropriate analog resistance states of the memristors [57], [68].

Since a unit cell consists of two atomic cells and has two input crossbar nanowires, its connectivity domain, i.e., the domain of atomic cells to which the given one can be connected, is larger [Fig. 7(a)]. The largest unit cell connectivity domain M is achieved by having the least overlap between individual connectivity domains of two atomic cells comprising the unit cell, and in this case $M \approx M_A$. For example, this could be implemented with the blue via having contacts at the edges of the crossbar nanowire segments as shown in Fig. 7(a), which is achieved by choosing appropriate relative position of vias inside the cell.

To simplify mapping of the applications to the proposed CMOL fabric and to provide a simple abstracted view of the logic and routing architecture, we will further use artificially smaller (than physically permissible) rectangular shaped domains—for example, domain of 5×5 unit cells for topological parameter $r = 6$ [Fig. 7(a)]. Therefore, the proposed CMOL FPGA architecture can be thought of as an array of multifunctional unit cells [Fig. 7(b) and (c)]. Every unit cell can pass its outputs to or accept inputs from any of the unit cells in 5×5 connectivity domain, which is always centered with respect to a given unit cell [Fig. 7(b)]. Moreover, as discussed above, every unit cell can be configured to perform AND (or linear threshold functions) with normal or complimented outputs of unit cells in its connectivity domain. Naturally, due to De-Morgan's law and the presence of complementary output, Boolean OR functions can also be realized for every unit cell.

As we will show next, the patterns will be stored as the state of memristors, some unit cells will be configured to store incoming data that needs to be searched for patterns, while the remaining unit cells will be used to process and store data indicating successful matches [Fig. 4(b)]. In this respect, the proposed architecture is similar to conventional TCAM circuits. Indeed, TCAM cells are implemented with differential pair of memristors [Fig. 2(i)], search lines are supplied from unit cells streaming data [Fig. 2(a)], while the pattern matching in a row of cells [Fig. 2(b)], and latching of the match result is implemented within a single unit cell dedicated for TCAM operation. The major advantage of the proposed architecture, however, is very high density of TCAM-like cells and flexible, FPGA-like allocation of unit cells for streaming and processing the data, which can be tailored for a particular application.

IV. Application Mapping Case Studies

A. 1-D Pattern Matching

Let us first consider 1-D stream of data pushed through a very deep pipeline, which is representative of network intrusion detection, bioinformatics, network routing, and various other string processing applications [1]. Conceptually, the idea of pattern matching for 1-D streaming data is simple (Fig. 8). To improve throughput it is natural to perform multiple pattern matching operations in parallel. Because of fan-out restrictions (i.e., limited connectivity domain) pattern matching is performed simultaneously with several (W) blocks of the pipeline data as shown in Fig. 8, with U operations done concurrently for each block. (Here, we assume that block length matches the length of pattern being searched.) With such a scheme, the total number of pattern matching operations performed in a given cycle is $U \times W$ and W cycles are needed to check a certain portion of the streaming data against all ($U \times W$) programmed patterns. Therefore, it is natural to allocate (configure) some unit cells in the homogeneous array to perform streaming data by forming long pipelines, while others to implement pattern matching. For simplicity, we assume that the results of pattern matching operations are logically summed together so that the circuit generates a single bit on the output at every cycle. (A more sophisticated processing would be straightforward given universality of the unit cells and flexibility in mapping.)

Fig. 9 shows an example of the mapping where white, green, and blue flip-flops represent unit cells performing pattern matching, data streaming, and processing of pattern matching results, respectively. More specifically, in this example, the streaming data are passed via two independent pipelines

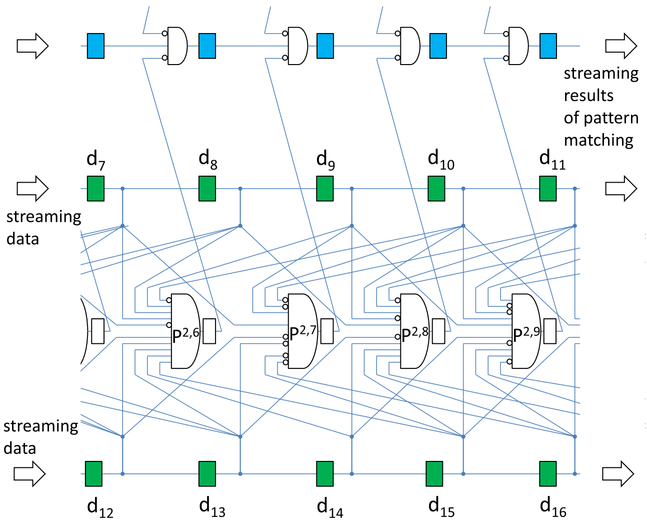

Fig. 9. Example of pattern matching operations for the data streamed via green flip flops (unit cells), with the first half of the data coming from the top pipeline and the second half from the bottom one. The white unit cells are programmed to match patterns "0111011111," "1X10000X00," "1011001111," and "1100110110," while the results of pattern matching operations are summed in a pipeline comprised by a chain of blue cells. Here, d denotes data bit stored by a unit cell and the index for d denotes the order of data in 1-D stream, while p denotes a pattern being matched by a given unit cell and pair of indexes for p represents specific pattern index within the block (out of total U patterns) and block id (out of total W blocks), respectively—see Fig. 8. Note that the summation is performed with AND gate based on De-Morgan's Law.

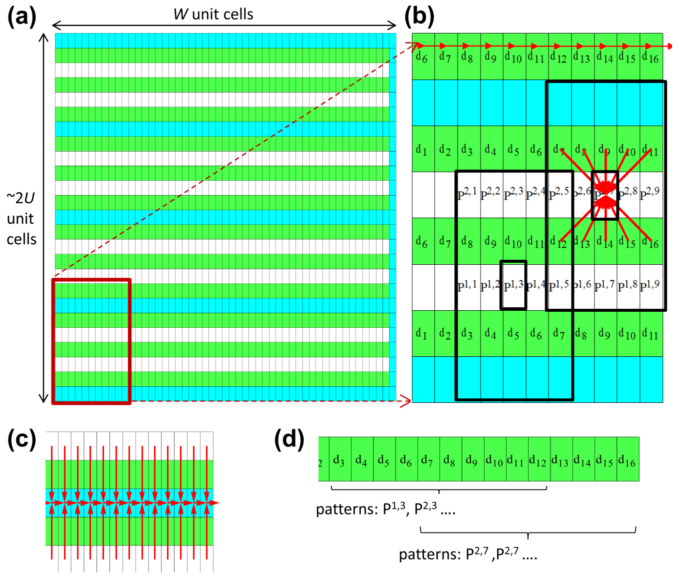
(a)
W unit cells
~2U unit cells
(b)
(c)
(d)
patterns: $P^{1,3}$, $P^{2,3}$ ....
patterns: $P^{2,7}$,$P^{2,7}$ ....

Fig. 10. Mapping of 1-D pattern matching task to CMOL FPGA with $r = 6$. (a) General mapping scheme. (b) Zoomed-in view showing mapping of the streaming data and unit cells performing pattern matching. (c) Scheme for getting logical summation operation of pattern matchings and data propagation in a pipeline. (d) Relative window for pattern matching operations in the data stream. Red arrows show schematically data movement/logical operation performed by each type of unit cell.

formed by green unit cells, and four exact pattern matching operations are performed with streaming data at each cycle by white unit cells. The results of pattern matching operations are summed in a pipeline comprised by blue unit cells.

In general, the largest number of bits in a pattern that can be matched with one unit cell is $(N_{\text{bit}})_{\max} = M - 1$. This would correspond to the case when all unit cells in the connectivity domain of the given one are configured to stream data. In this case, only $\chi = \chi_{\min} = 1/M$ fraction of the unit cells are performing the pattern matching operation. At the other extreme case, i.e., when all unit cells in the connectivity domain are configured to perform pattern matching except for one, $\chi = \chi_{\max} = (M - 1)/M$ and $N_{\text{bit}} = (N_{\text{bit}})_{\min} = 1$. More generally, the number of bits in a pattern which can be compared by one unit cell is

$$N_{\text{bit}} = (1 - \chi)M. \quad (3)$$

It is trivial to show that the largest total number of bits in all patterns matched per one cycle (χN_{bit}) is achieved when $\chi = 0.5$, i.e., when half of the unit cells in the connectivity domain are configured to perform pattern matching and the other half to stream data. (Note that this conclusion assumes that the length of pattern is not fixed but rather a parameter which is optimized. Also, the unit cells configured to process the results of pattern matching are neglected in this analysis, which is justified due to their relatively small number, at least in our considered case.) A similar observation for balancing streaming and processing resources has been made when mapping network processing tasks on conventional FPGA circuits [34].

Fig. 10 shows an example of one such mapping. The streaming 1-D data are duplicated and pushed through several pipelines. To ensure that any connectivity domains of white cells consists of always unique and contiguous streaming data from the green cells, the relative position of data in every second pipeline is shifted by five positions [Figs. 9(b) 10(b)] for the considered connectivity domain size (Fig. 7). The unit cells that should be allocated to duplicate the streaming data are not shown though their overhead is negligible.

Because of the limited size of the connectivity domain, the logical summation from all of the pattern matching cells is done in several steps. As Figs. 9 and 10(c) show, at each cycle a particular blue cell latches the logical summation of the values from the two nearest white cells (which hold the results of pattern matches from the previous cycle), and one blue cell to the left of the given one. The partial sum is fully pipelined and propagates along a row of blue cells at the rate of one cell position per cycle, i.e., as fast as the streaming data. Once partial sums from different rows are propagated to the right edge of the chip, they are summed up in the similar fashion to get one single value. This value represents the logical summation of all results of the comparisons performed within the array at specific time window.

Finally, let us note that for a considered value of $r = 6$, each white cell performs $N_{\text{bit}} = 10$ bit-wide pattern matching. With such mapping, white cells in the same column are performing matching within the same block of data at any given cycle [Fig. 10(b) and (d)]. Therefore, the number of pattern matching operations per block (U) is given by the number of white cells in a column, which is roughly equal to the half of the total number of unit cells per column. The number of different blocks (W) is given by the number of unit cells in a row,

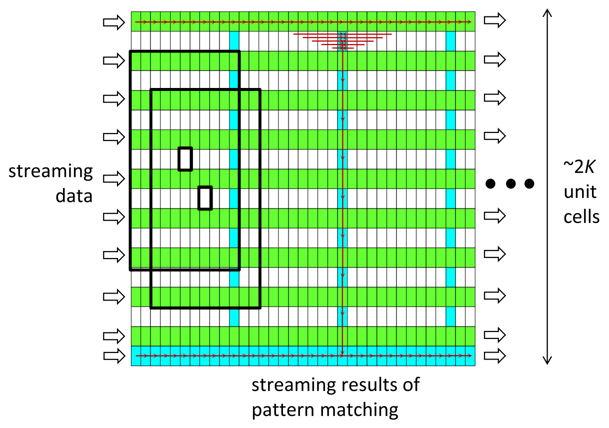

Fig. 11. General idea for mapping image processing tasks to the proposed CMOL FPGA architecture.

i.e., width of the unit cell array, so that the total number of patterns which can be matched by an array is roughly half of the total number of unit cells in the array.

B. 2-D Pattern Matching

The main difference for mapping of image processing tasks is that both streaming data and patterns are 2-D. For example, in automatic target recognition (ATR) systems 128 × 128 array of 8-bit pixels is searched for potential targets which are described by 16 × 16 binary pixel templates [36]. The bottleneck operation in ATR algorithm is 1-bit correlations between input image and template which has to be done for all possible relative offsets and for rather large number of templates. (It should be noted that contemporary ATR systems work with much larger input and template image sizes and higher data rate, e.g., required for hyperspectral image processing [69].)

Naturally, a correlation operation which produces multibit output value cannot be done with a single unit cell in digital CMOL FPGA circuits. On the other hand, a combined operation of correlation with thresholding, which is effectively an approximate pattern matching, is straightforward if unit cell is configured to implement a linear threshold gate. Such an operation might be sufficient for eliminating bottleneck processing in ATR and other related image processing tasks.

Similar to the previously considered mapping scheme, the maximum utilization is achieved with balanced number of white and green cells (Fig. 11). Let us assume that the connectivity domain is large enough that each unit cell performs matching between streaming data of input image and the whole template. (The proposed scheme can be extended to the case when the domain is smaller, by performing matching operations for the parts of the template instead.) Let us also assume that a $K \times K$ 2-D image is pushed through a pipeline formed by green cells, e.g., from left to right, by one unit cell position each cycle. To perform correlation for one template, for all possible vertical offsets in one cycle requires programming a column of K white cells to perform matching with the same template. Evaluation of all horizontal offsets would just require K cycles to push data (from left to right) past the column of a particular white cells. The next column of white cells can be programmed to perform matching for a second template and so on. The result of the pattern matching operations might be summed all together and pushed to the bottom of the array and then propagated to the right.

Many different alternative implementations are also possible. For example, for faster processing (though sacrificing the total number of templates) the streaming data can be pushed by several unit cell positions in one cycle. In this case, more than one column of white cells should be allocated per one template, but the number of cycles to check for all possible offsets is proportionally less.

V. Performance Modeling

We have modeled the performance of a generic 1-D pattern matching task and found the optimal parameters to maximize the possible pattern matching throughput per unit area. Before discussing the details of an optimization procedure, let us first outline some common assumptions for the devices, diode–resistor logic and its performance modeling while focusing on the more promising dynamic logic counterpart of this architecture.

A. Nanowires

The resistance of $2F_{\text{nano}}$-long nanowire segment is approximated using Matthessian formula [70], which accounts for the increase in the resistivity ρ due to surface scattering effects in nanoscale wires, i.e.,

$$R_{\text{wire}} \approx \rho \frac{2F_{\text{nano}}}{\mathscr{A}(F_{\text{nano}})^2} \approx \frac{2\rho_{\text{bulk}}}{\mathscr{A}F_{\text{nano}}}\left(1 + \frac{\lambda}{F_{\text{nano}}}\right). \tag{4}$$

Here, $\mathscr{A}$ is the relative thickness of the nanowires with respect to its width (i.e., the cross-sectional aspect ratio), while λ is the mean free path of the electrons.

The capacitance of $2F_{\text{nano}}$-long nanowire segment is approximated analytically by using the equation

$$C_{\text{wire}} \approx \epsilon_0\epsilon_1 \frac{F_{\text{nano}}^2}{2d} + \epsilon_0\epsilon_2 \frac{F_{\text{nano}}^2}{2d} + \mathscr{A}\epsilon_0\epsilon_2 4F_{\text{nano}} + \mathscr{A}\epsilon_0\epsilon_2 \frac{4_{\text{nano}}}{\log\left[\frac{F_{\text{nano}}}{d} + 10\right]} \tag{5}$$

which was verified using COMSOL simulations. Here, d is the thickness of the thin film, i.e., the distance between two, mutually perpendicular sets of crossbar nanowires, ϵ_0 is vacuum permittivity, ϵ_1 and ϵ_2 are dielectric constants of the nanodevices and surrounding insulator, respectively, and a constant 10 was determined by fitting (5) to the numerical simulations. Note that on the right-hand side of (5), the first two terms crudely correspond to parallel plate capacitance, the third term is the interlayer side wall capacitance, and the last term is the side wall capacitance between crossbar nanowires in the same layer.

In our performance analysis, we assume copper crossbar nanowires with $\mathscr{A} = 0.1$ and $\rho_{\text{bulk}} = 1.7 \times 10^{-8}$ Ω-m. Using $\lambda = 40$ nm, (4) yields an accurate approximation for both grain and surface scattering as reported in international technology roadmap for semiconductors (ITRS) [71]. For capacitance estimates, we assume $\epsilon_1 = 3.9, \epsilon_2 = 2.5$, and $d = 5$ nm, which is representative of SiO_2 memristive devices.

B. Diode–Resistor Logic

Let us assume that at least γ fraction of voltage applied from the CMOS cell is dropped across nanodevice and at most $1-\gamma$ fraction is dropped on nanowire and pass gate connecting outputs of DFF and the corresponding vias. This can be satisfied by choosing R_{ON} and R_{pass} according to

$$R_{\text{ON}} \geq (1-\chi)\gamma M(MR_{\text{wire}} + R_{\text{pass}})/(1-\gamma) \tag{6}$$

where the factor $(1-\chi)M$ is effectively the maximum fan out for each output of the DFF cell (3), i.e., the largest permitted fraction of the devices in the ON state on each of the two output nanowires.

For dynamic logic operation, the slowest (worst case) mismatch operation corresponds to charging via single nanodevice in the ON state. For the considered asymmetric I–V characteristics, this charging time will be always faster than that of the match operation and the worst case voltage margins are given by

$$\triangle V \approx V_R/(1 + 2MR_{\text{ON}}/R_{\text{OFF}}). \tag{7}$$

The safe margins can be in principle calculated from noise and CMOS variations analysis [19]. Equation (7), however, shows that such analysis can be simplified by selecting large enough R_{OFF} so that the margins are comparable with V_R. For example, requiring $R_{\text{OFF}}/R_{\text{ON}} > 2M$, which is very reasonable assumption as we show below, results in $\triangle V \approx V_R/2$, and provides justification of neglecting leakages via OFF state devices in (6), as well as delay and power estimates.

In our simulations, we assume $V_R = 1$ V and $\gamma = 0.9$.

C. Area, Delay, and Power

According to Section III-C, area of unit cell is

$$A_{\text{cell}} = 2(2\beta F_{\text{CMOS}})^2 \approx 2M(2F_{\text{nano}})^2. \tag{8}$$

To calculate $\beta_{\min}$, the minimum cell area is estimated similar to [72], i.e., by counting the number of transistors in the cell, and modeling the area of each transistor according to its driving strength.

Specifically, out of the total 54 transistors in a cell, we assume that there are 22 minimum-size transistors, including those used for configuration circuits, which do not have to support high currents because of the diode-like asymmetric I–V characteristics of memristors. There are also 20 transistors that compose the DFF which are sized according to 3 input and 2 input NAND gates, as well as a Schmitt trigger that is composed of 6 transistors, 2 of which are minimum sized, 2 of size 2 and 2 of size 4. On the other hand, we assume 4 transistors, which are used in the output drivers, and 4 transistors of the two pass gates controlled by enable lines (Fig. 6) are scaled up to accommodate the current driving requirements of memristor layer for proper operation of diode–resistor logic. For simplicity, all nonminimum-size transistors are scaled up equally and their area is estimated as

$$A_{\text{tran}} \approx (0.5 + 0.5(R_{\text{pass}})_{\max}/R_{\text{pass}}) \times 25F_{\text{CMOS}}^2 \tag{9}$$

where $(R_{\text{pass}})_{\max}$ is effective drain–source resistance of the minimum-size transistor at the saturation for the specific F_{CMOS} node. Note that using $R_{\text{pass}} = (R_{\text{pass}})_{\max}$ in (9) assumes that area for the minimum-size transistor is $25F_{\text{CMOS}}^2$.

For the dynamic logic, the delay is estimated as

$$\tau \approx 2(2MC_{\text{wire}} + C_{\text{gate}})R_{\text{ON}} \tag{10}$$

where 2 MC_{wire} is capacitance of two nanowire segments, while C_{gate} is a total capacitance of CMOS circuitry at the input of DFF, including its gate capacitance and drain capacitances of the configuration and pull-down pass gates. The additional factor of 2 is to account for both precharging and evaluation phases, which is rather conservative assumption given that precharging currents are not limited by R_{ON} value. Note that for all studied parameters C_{gate} is typically much smaller than 2 MC_{wire}.

The average power per unit cell is dominated by the dynamic component, for which the upper bound is evaluated as

$$P_{\text{cell}} \approx (2 \times 2MC_{\text{wire}} + C_{\text{gate}})V_R^2/\tau. \tag{11}$$

Equation (11) implies activity factor of 1 for DFF's input and output nanowires and input CMOS circuitry, i.e., their charging and discharging within a single clock cycle. This is quite a pessimistic assumption given that matching events can be assumed to be rare on average and hence outputs of all cells configured to perform pattern matching will not change. Still, as we show later, the total power, which should be less than the maximum allowable power density $p_{\max}$, i.e.,

$$P_{\text{cell}} \leq p_{\max}A_{\text{cell}} \tag{12}$$

is rarely a limiting factor for performance.

In our simulations, we assume $p_{\max} = 200$ W/cm^2, and $(R_{\text{pass}})_{\max} = 27.3/13.3/6.6/4.6$ kΩ and $C_{\text{gate}} = 7.5/22.5/76.2/135$ fF for the considered $F_{\text{CMOS}} = 22/45/90/130$ nm nodes, respectively—all typical values specified by ITRS [71].

D. Throughput and Energy Per Bit

Given the area of the chip A_{chip}, the total number of cells is $N_{\text{cell}} = A_{\text{chip}}/A_{\text{cell}}$, while the total number of patterns N_{pattern} that can be stored in memristors and compared in one cycle is

$$N_{\text{pattern}} \approx \chi N_{\text{cell}}. \tag{13}$$

The total number of pattern bits stored in a chip is, therefore,

$$\begin{aligned} N_{\text{total}} &= N_{\text{bit}}N_{\text{pattern}} \approx (1-\chi)\chi MA_{\text{chip}}/A_{\text{cell}} \\ &\approx (1-\chi)\chi A_{\text{chip}}/(8F_{\text{nano}}^2) \end{aligned} \tag{14}$$

and the aggregate pattern matching throughput is

$$T = \frac{N_{\text{total}}}{\tau}. \tag{15}$$

Another metric of interest is the consumed energy during pattern matching operations per single bit, which is simply

$$E_{\text{bit}} = P_{\text{cell}}N_{\text{cell}}\tau/N_{\text{total}}. \tag{16}$$

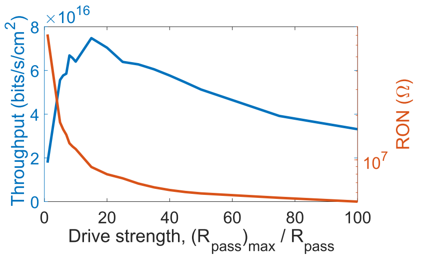

Fig. 12. Optimal value of R_{ON} and the pattern matching throughput as a function of driving strength for a particular case $F_{\mathrm{CMOS}} = F_{\mathrm{nano}} = 22$ nm and $\chi = 0.5$.

E. Optimization and Simulation Results

In general, our goal is to find the maximum possible pattern matching throughput per unit area, i.e., the largest T/A_{chip}, for a particular choice of mapping scenario χ and technology feature sizes F_{CMOS} and F_{nano}, by optimizing R_{pass} and R_{ON}. In particular, we use a brute-force approach, by first sweeping via broad range of R_{pass} values, starting with its maximum possible value $(R_{\mathrm{pass}})_{\mathrm{max}}$. Naturally, the technology parameters, mapping scenario, and R_{pass} impact the area of the cell and hence all dependent topological and application mapping parameters, i.e., $r, \beta, \beta_{\mathrm{min}}, A_{\mathrm{cell}}, M_A, M, N_{\mathrm{bit}}, N_{\mathrm{pattern}}, N_{\mathrm{total}}$ defined by (1)–(3), (13), and (14). We then sweep via realistic (implementable) range of R_{ON}, which is constrained from below by proper operation of diode–resistor logic defined by (6) and (in very few cases) power budget, i.e., (11) and (12).

For a fixed technology parameters and mapping scenario, an optimal throughput peaks at certain R_{pass} value and, for any specific R_{pass}, always obtained using the smallest permissible value of R_{ON} (Fig. 12). The detailed results of an optimization for $\chi = 0.5$ and the impact of χ on throughput are shown on Figs. 13 and 14, respectively.

VI. Discussion and Summary

To get intuition behind the optimization procedure, let us first note that for a fixed chip area, the total number of stored bits depends only on F_{nano} and χ (14), (15), so that the largest throughput is achieved by minimizing the delay τ. In turn, the delay, which depends on the product of R_{ON} and M, is minimized by tuning value of R_{pass}. Indeed, for relatively large values of R_{pass}, the cell area weakly depends on pass transistor scaling and decreasing R_{pass} results in smaller R_{ON} because of (6). However, when scaled pass transistors start dominating the cell area, the further decrease in R_{pass} is counterproductive because of increase in M due to (1) and (2). As a result, there is a certain optimum value of R_{pass} corresponding to the smallest delay and largest throughput (Fig. 12). Interestingly, the optimal driving strength, i.e., $(R_{\mathrm{pass}})_{\mathrm{max}}/R_{\mathrm{pass}}$, is always close to ~ 15 for all studied cases of F_{CMOS}, F_{nano}, and χ, at which the area of the pass gates and drive circuits is comparable to that of the remaining circuitry in a cell.

The results of optimization as shown in Fig. 13, show that though throughput is reaching its maximum value at around $F_{\mathrm{nano}} = F_{\mathrm{CMOS}}$, it remains relatively constant across changes in F_{nano} for fixed values of F_{CMOS}. This can be attributed to two simultaneous factors that have opposing effects on throughput. On one hand, increasing F_{nano} reduces the size of the connectivity domain, which in turn, reduces the value of R_{ON}, since the pass gates have to support a smaller number of devices, and lead to faster operation speeds. On the other hand, reducing the size of the connectivity domain reduces the number of bits matched and hence reduces the throughput. The simulation results also show that the throughput is roughly proportional to $1/F_{\mathrm{CMOS}}$. This is because for a given F_{nano}, the decrease in CMOS feature size leads to proportionally smaller connectivity factor, and, in turn, proportionally smaller optimal values of R_{ON}.

Another interesting result from the optimization procedure [as shown in Fig. 13(d)] is the independence of the energy per bit metric on F_{CMOS}. Intuitively, this means that E_{bit} is determined by dynamic energy of a single, $2F_{\mathrm{nano}}$-long segment of the crossbar wire (match line).

Furthermore, though $\chi = 0.5$ corresponds to the maximum number of stored pattern matching bits, as discussed in Section IV, Fig. 14 shows that the largest throughput is achieved with $1 - \chi < 0.5$. Indeed, the reduction of $1 - \chi$ allows a smaller R_{ON} to be supported by the same size pass gates. This reduction in R_{ON} in turn allows for a reduction in τ that outweighs the loss in throughput as a result of a lowering in number of pattern matching bits (N_{bit}). This reduction in $1-\chi$ is also met with a commensurate increase in the number of streaming cells hence throughput does not drop drastically. (Note that the further decrease in $1-\chi$, beyond what is shown in Fig. 14, will eventually lead to the drop in the throughput due to power density constraint.)

The considered values of R_{ON} and R_{OFF} are quite realistic for the most cases, which indicates the practicality for manufacturing such circuits. For example, at $\chi = 0.5$ and $F_{\mathrm{nano}} = F_{\mathrm{CMOS}}$, the optimal value of R_{ON} is always around 10 MΩ [Fig. 13(b)], while the corresponding ON/OFF ratio from (7) and [Fig. 13(a)] is $R_{\mathrm{OFF}}/R_{\mathrm{ON}} \geq 2 \times 10^3$, which are not uncommon for memristors [24]. It should be also noted that while the considered nanowire aspect ratio $\mathscr{A} = 0.1$ is rather conservative choice, the throughput is not very sensitive to $\mathscr{A}$ and would actually slightly improve further by considering even smaller $\mathscr{A}$. This is because wire resistance is rarely a limiting factor in our optimization and $\mathscr{A}$ only impacts fringe capacitance of the crossbar wires.

The maximum throughput for $F_{\mathrm{CMOS}} = F_{\mathrm{nano}} = 22$ nm and $\chi = 0.5$ is close to 8×10^{16} bits/s/cm^2 for matching of $\sim 10^7$ 250-bit patterns, assuming practical power consumption density [Fig. 13(f)]. This number compares very favorably with the reported state-of-the-art FPGA performance (Table I). (Note that because the largest throughput in our circuit is always achieved at $F_{\mathrm{nano}} = F_{\mathrm{CMOS}}$, we report only one feature size for our work.) Our reported throughput vastly exceeds FPGA only implementations and rivals state-of-the-art TCAM based implementation such as [44] and [49]. We expect that algorithmic improvements, in particular, the use of common subexpression elimination techniques will increase the throughput of the proposed circuits even further.

Even more important is the fact that the proposed circuits could potentially offer much higher pattern capacity without any performance penalty. Because the number of storage

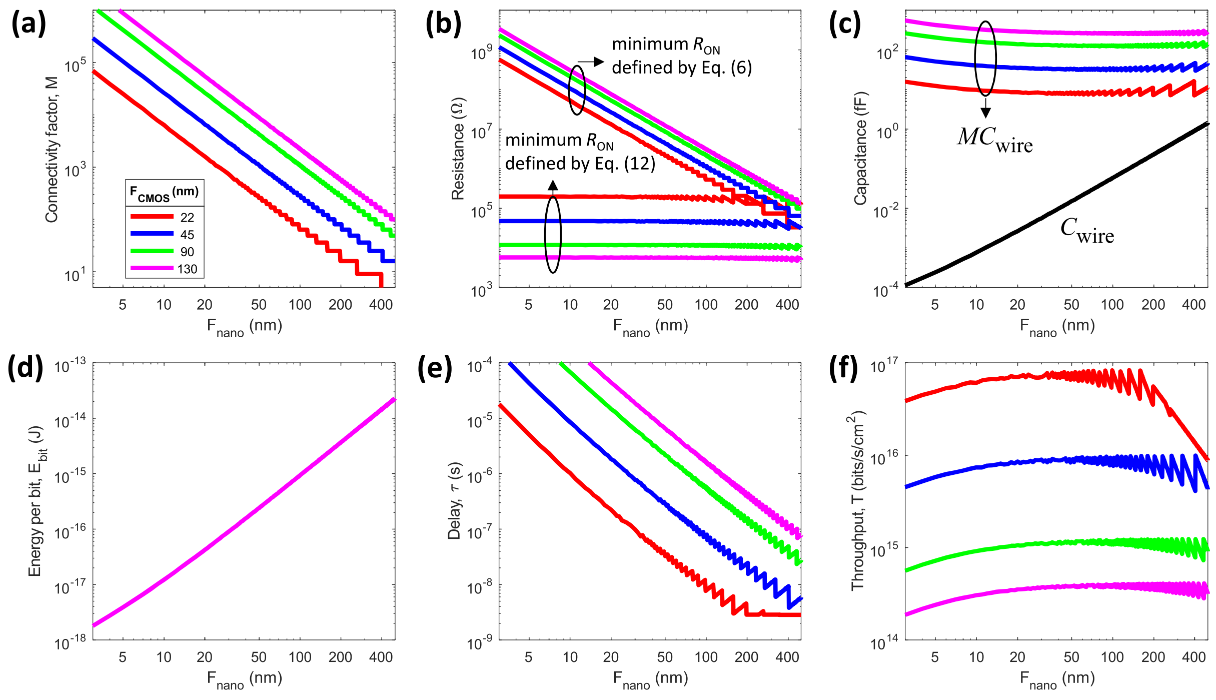

Fig. 13. Simulation results for the optimal value of R_{pass}, $\chi = 0.5$, and specific values of F_{CMOS} and F_{nano} [73]. (a) Connectivity factor of a unit cell. (b) Optimal value of R_{ON} determined as a minimum resistance satisfying two constrains: proper operation of diode–resistor logic (6) and power density budget (12). (c) Capacitance of a segment and the whole wire. (d) Energy per bit (which is the same for all values of F_{CMOS}). (e) Diode–resistor logic delay (clock cycle time). (f) Aggregate pattern matching throughput per unit area. Note that for the shown case, R_{ON} is always determined by (6).

TABLE I

PERFORMANCE OF VARIOUS PATTERN MATCHING ARCHITECTURES. (# THE NUMBERS FOR BOTH TCAM AND FPGA IMPLEMENTATIONS ARE RATHER OPTIMISTIC. ONLY AREA OF MEMORY CELLS IS TAKEN INTO ACCOUNT FOR THE FORMER, WHILE FPGA NUMBERS ARE ESTIMATED BASED ON THE REPORTED LOGIC UTILIZATION.)

Architecture		CMOS process (nm)	Match time (ns) / V_{DD} (V)	Through put (Gbps)	Chip area (cm²)	Cell area (μm²)	Throughput per unit area (bits/s/cm²) #	Refer ence	Comments
FPGA	Virtex 7	28	1.4 / 1	5.56	12.25	-	-	[40]	merged-state pipelined NFA
	Spartan 3	90	5.2 / 1.2	1.54	2.89	-	-	[41]	DFA, failureless pipelined Aho-Corasick algorithm
	Virtex 7	28	4.6 / 1	6.69	12.25	-	-	[42]	4-character per cycle, memory-based Aho-Corasick algorithm
	Virtex 6	40	6.9 / 1	0.14	12.25	-	-	[39]	byte to token information reduction, reg-ex feature detection
	Virtex 7	28	- / 1	29.6	12.25	-	-	[37]	Pi-DFA, multi-stride regular expression matching
	Virtex 5	65	6.25 / 1	11	18.06	-	-	[38]	modified McNaughton-Yamada algorithm
TCAM	36 Kb static CMOS	180	3 / 1.2	1.2×10^4	-	17.52	1.8×10^{15}	[52]	144 bit match line
	4.5 Mb dynamic CMOS	130	6.99 / 1.5	6.7×10^5	-	3.59	3.9×10^{15}	[53]	144 bit match line
	1 Mb phase change memory	90	1.9 / 1.2	5.5×10^5	-	0.41	1.2×10^{17}	[44]	64 bit match line
	144 Kb charge-recycle	180	10 / 1.8	1.4×10^4	-	22.46	4.4×10^{14}	[54]	144 bit match line
	72 Kb range-match	130	4.8 / 1.2	1.5×10^4	-	3.39	6.1×10^{15}	[55]	144 bit match line
	32 Kb PF CDPD	180	2.1 / 1.8	1.5×10^4	-	23.8	2×10^{15}	[56]	128 bit match line
	2 Kb magnetic tunnel junction	90	0.29 / 1.2	7×10^3	-	10.35	3.3×10^{16}	[49]	144 bit match line
High-throughput CMOL FPGA		130	~ 100 / 1	5×10^5	~ 1	0.135	5×10^{14}	This work	~250 bit match line
		90	~ 70 / 1	2×10^6	~ 1	0.065	2×10^{15}		~250 bit match line
		45	~ 40 / 1	1.5×10^7	~ 1	0.016	1.5×10^{16}		~250 bit match line
		22	~ 20 / 1	1.1×10^8	~ 1	0.004	1.1×10^{17}		~250 bit match line

elements in existing hardware-based pattern matchers is limited by the 2-D chip area, they must be dynamically reconfigured to accommodate additional patterns that are beyond their storage capabilities and rely on OFF-chip storage. Dynamic reconfiguration is rather slow and very energy inefficient, thus we expect that the throughput for a fixed area for higher capacity pattern matching tasks will be considerably smaller than the ideal value. On the other hand, it should be possible to support larger bit capacity in the proposed circuits by integrating more crossbar layers, e.g., similar to 3-D CMOL circuits [64], [65] without large penalty in throughput.

Even though our performance analysis is somewhat simplified, we believe that we accounted for the most important factors. For example, CMOS process variations, critical for diode–resistor logic operation can be effectively dealt with by appropriate scaling of the CMOS transistors. Its additional overhead, as well as additional area due to bulky programming circuitry which might be required for memristors with large write voltages [74] should not change much our simulation results because of already large pass gates/drive circuit transistors for the optimal cases. Though, the dynamic logic is susceptible to capacitive coupling noise, it should be possible

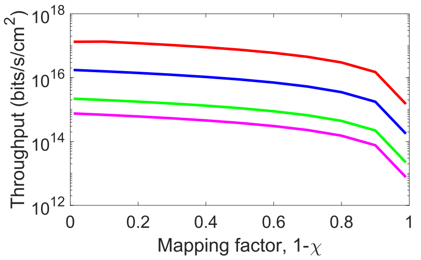

Fig. 14. Maximum throughput as function of mapping scenario for several values of F_{CMOS} (similar to the ones used in Fig. 13).

to minimize its effect by balancing signal transitions. For example, the CMOL topology and application mapping can be further optimized to avoid Miller effects. Based on our experience with CMOL design [75] and quite large area of CMOL cells in this paper, the area overhead of peripheral decoders and clock distribution network should be also insignificant. As it is evident from the choice of the optimal values of R_{ON} [Fig. 13(b)], the dynamic power consumption of our circuits is always well below 200 W/cm^2. Our estimates shows that the neglected dynamic power of the clock distribution network (especially considering relatively slow cycle times at optimal $F_{\mathrm{nano}} = F_{\mathrm{CMOS}}$) and static power are also always limited to the sub watt range.

Perhaps the most critical challenge toward practical realization of the proposed circuits is that fabrication technology for the memristive devices, especially for their passive back-end-of-line integration, is in need of improvement. A particular concern is memristor nonidealities, such as current fluctuations due to drift in the memristor state and random telegraph noise, and variations in the switching threshold voltages. One possibly strategy to deal with these issues is to identify defective devices during test stage and avoid them during application mapping [19], [76]. On the other hand, variations in the ON and OFF resistance states would be less problematic due to, e.g., possibility of fine-tuning memristor conductances using simple tuning algorithm setup. Also, though the cycling endurance for many memristors is generally much less as compared to that of volatile memories, it should be still adequate for many FPGA applications, given that the memristor states are only switched during reconfiguration stage and remain unchanged during logic operation.

In summary, in this paper, we proposed new CMOL FPGA circuits for high-throughput computation. The performance advantage of novel circuits is mainly due to very high density of nanoscale devices and very tight and synergetic integration of memory and logic functions. The tight integration is enabled by high communication bandwidth of the area-distributed interface between the nano and CMOS subsystems, while the synergy is due to flexible resource allocation that allows nanodevices to be used either as a TCAM cell or to implement programmable logic/interconnect. Though CMOL circuits are essential for getting high bandwidth between memory and logic subsystems, other stacking schemes with area-distributed connectivity, such as through silicon via technology, and different memory devices, such as flash memory might be suitable for the proposed concept. Understanding the prospects of using such mature and readily available device and integration technologies is one of the important future research directions.

Acknowledgment

The authors would like thank R. Brayton, A. Mishchenko, and K. K. Likharev for their useful discussions.

Conflicts of Interest

The presented work is built upon our previous results reported in [30]–[32]. The material from these papers is included in Sections III and IV-A. However, Sections IV-B, V, and VI elaborate on the new results, which were never published before.

In particular, in this paper we present the following, for the first time.

1) A performance analysis of the proposed circuit. Our estimates also account for the sizing of CMOS circuits, which was generally neglected in all previous CMOL FPGA works but, as we show in this paper, is crucial for providing correct functionality.
2) An optimization procedure that considers architectural, topological and circuit level constraints to maximize the throughput of the proposed circuits.
3) An additional application case study.

References

[1] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, vol. 1. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[2] Q. Gu, T. Takaki, and I. Ishii, "Fast FPGA-based multiobject feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 30–45, Jan. 2013.
[3] C. Gentsos, C.-L. Sotiropoulou, S. Nikolaidis, and N. Vassiliadis, "Real-time canny edge detection parallel implementation for FPGAs," in *Proc. ICECS*, Dec. 2010, pp. 499–502.
[4] Y. H. Cho and W. H. Mangione-Smith, "Deep network packet filter design for reconfigurable devices," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 2, Feb. 2008, Art. no. 21.
[5] L. Tan and T. Sherwood, "Architectures for bit-split string scanning in intrusion detection," *IEEE Micro*, vol. 26, no. 1, pp. 110–117, Jan. 2006.
[6] T. N. Thinh, T. T. Hieu, V. Q. Dung, and S. Kittitornkun, "A FPGA-based deep packet inspection engine for network intrusion detection system," in *Proc. 9th Int. Conf. Elect. Eng./Electron., Comput., Telecommun. Inf. Technol. (ECTI-CON)*, May 2012, pp. 1–4.
[7] H. Le and V. K. Prasanna, "A memory-efficient and modular approach for large-scale string pattern matching," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 844–857, May 2013.
[8] Y. Xin *et al.*, "Parallel architecture for DNA sequence inexact matching with Burrows-Wheeler transform," *Microelectron. J.*, vol. 44, no. 8, pp. 670–682, Aug. 2013.
[9] D. Lavenier, G. Georges, and X. Liu, "A reconfigurable index FLASH memory tailored to seed-based genomic sequence comparison algorithms," *J. VLSI Signal Process.*, vol. 48, no. 3, pp. 255–269, 2007.
[10] Q. Zhang, R. D. Chamberlain, R. S. Indeck, B. M. West, and J. White, "Massively parallel data mining using reconfigurable hardware: Approximate string matching," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Apr. 2004, p. 259.
[11] M. B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster, "SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware," *ACM SIGCOMM Comput. Commu. Rev.*, vol. 40, no. 4, pp. 183–194, Oct. 2010.
[12] T. Sherwood, G. Varghese, and B. Calder, "A pipelined memory architecture for high throughput network processors," in *Proc. 30th Annu. Int. Symp. Comput. Archit.*, Jun. 2003, pp. 288–299.
[13] C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu, "Fast regular expression matching using small TCAMs for network intrusion detection and prevention systems," in *Proc. 19th USENIX conf. Secur.*, Aug. 2010, p. 8.

[14] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[15] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, Nov. 2003.

[16] A. DeHon, "Nanowire-based programmable architectures," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, Jul. 2005.

[17] K. K. Likharev and D. B. Strukov, "CMOL: Devices, circuits, and architectures," in *Introducing Molecular Electronics*. New York, NY, USA: Springer-Verlag, 2006, pp. 447–477.

[18] X. Tang, P.-E. Gaillardon, and G. De Micheli, "A high-performance low-power near-Vt RRAM-based FPGA," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2014, pp. 207–214.

[19] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, p. 888, 2005.

[20] D. B. Strukov and K. K. Likharev, "A reconfigurable architecture for hybrid CMOS/nanodevice circuits," in *Proc. ACM/SIGDA 14th Int. Symp. FPGAs*, 2006, pp. 131–140.

[21] D. B. Strukov and K. K. Likharev, "Reconfigurable hybrid CMOS/nanodevice circuits for image processing," *IEEE Trans. Nanotechnol.*, vol. 6, no. 6, pp. 696–710, Nov. 2007.

[22] Q. Xia *et al.*, "Memristor-CMOS hybrid integrated circuits for reconfigurable logic," *Nano Lett.*, vol. 9, no. 10, pp. 3640–3645, 2009.

[23] D. Strukov and A. Mishchenko, "Monolithically stackable hybrid FPGA," in *Proc. Conf. Design, Autom. Test Eur.*, Mar. 2010, pp. 661–666.

[24] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.

[25] K. K. Likharev, "Hybrid CMOS/nanoelectronic circuits: Opportunities and challenges," *J. Nanoelectron. Optoelectron.*, vol. 3, no. 3, pp. 203–230, 2008.

[26] A. M. Arafeh and S. M. Sait, "Cells reconfiguration around defects in CMOS/nanofabric circuits using simulated evolution heuristic," in *Proc. ISQED*, Mar. 2015, pp. 581–588.

[27] W. N. N. Hung, C. Gao, X. Song, and D. Hammerstrom, "Defect-tolerant CMOL cell assignment via satisfiability," *IEEE Sensors J.*, vol. 8, no. 6, pp. 823–830, Jun. 2008.

[28] Z.-L. Pan, L. Chen, and G.-Z. Zhang, "Efficient design method for cell allocation in hybrid CMOS/nanodevices using a cultural algorithm with chaotic behavior," *Frontiers Phys.*, vol. 11, no. 2, p. 116201, Apr. 2016.

[29] S. M. Sait and A. M. Arafeh, "Cell assignment in hybrid CMOS/nanodevices architecture using Tabu search," *Appl. Intell.*, vol. 40, no. 1, pp. 1–12, Jan. 2014.

[30] D. B. Strukov, "Hybrid CMOS/nanodevice circuits with tightly integrated memory and logic functionality," in *Proc. Nanotechnol.*, vol. 11. 2011, pp. 9–12.

[31] F. Alibart, T. Sherwood, and D. B. Strukov, "Hybrid CMOS/nanodevice circuits for high throughput pattern matching applications," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jun. 2011, pp. 279–286.

[32] A. Madhavan and D. B. Strukov, "Mapping of image and network processing tasks on high-throughput CMOL FPGA circuits," in *Proc. IEEE/IFIP 20th Int. Conf. VLSI Syst.-Chip (VLSI-SoC)*, Oct. 2012, pp. 82–87.

[33] B. Schmidt, *Bioinformatics: High Performance Parallel Computer Architectures*. Boca Raton, FL, USA: CRC Press, 2010.

[34] Z. K. Baker and V. K. Prasanna, "Time and area efficient pattern matching on FPGAs," in *Proc. FPGA*, 2004, pp. 223–232.

[35] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *J. VLSI Signal Process.*, vol. 28, nos. 1–2, pp. 7–27, May 2001.

[36] K.-N. Chia *et al.*, "Configurable computing solutions for automatic target recognition," in *Proc. IEEE Symp. FPGAs Custom Comput. Mach.*, Apr. 1996, pp. 70–79.

[37] J. Yang, L. Jiang, Q. Tang, Q. Dai, and J. Tan, "PiDFA: A practical multi-stride regular expression matching engine based On FPGA," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[38] Y.-H. Yang and V. Prasanna, "High-performance and compact architecture for regular expression matching on FPGA," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1013–1025, Jul. 2012.

[39] N. L. Or, X. Wang, and D. Pao, "MEMORY-based hardware architectures to detect ClamAV virus signatures with restricted regular expression features," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1225–1238, Apr. 2016.

[40] H. Kim and K.-I. Choi, "A pipelined non-deterministic finite automaton-based string matching scheme using merged state transitions in an FPGA," *PLoS ONE*, vol. 11, no. 10, p. e0163535, 2016.

[41] H. J. Kim, "A failureless pipelined Aho-Corasick algorithm for FPGA-based parallel string matching engine," in *Information Science and Applications*. Berlin, Germany: Springer, 2015, pp. 157–164.

[42] X. Wang and D. Pao, "Memory-based architecture for multicharacter Aho–Corasick string matching," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 1, pp. 143–154, Jan. 2017.

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

[44] J. Li *et al.*, "1Mb 0.41 μm^2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," in *Proc. Symp. VLSI Circuits (VLSIC)*, Jun. 2013, pp. C104–C105.

[45] S. Matsunaga *et al.*, "Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control," in *Proc. Symp. VLSI Circuits (VLSIC)*, Jun. 2011, pp. 298–299.

[46] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," in *Proc. Micro*, Dec. 2011, pp. 339–350.

[47] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 189–200, 2013.

[48] T. Hanyu, N. Kanagawa, and M. Kameyama, "Non-volatile one-transistor-cell multiple-valued cam with a digit-parallel-access scheme and its applications," *Comput. Elect. Eng.*, vol. 23, no. 6, pp. 407–414, 1997.

[49] S. Matsunaga *et al.*, "Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices," *Appl. Phys. Exp.*, vol. 2, no. 2, p. 023004, 2009.

[50] W. Xu, T. Zhang, and Y. Chen, "Design of spin-torque transfer magnetoresistive RAM and CAM/TCAM with high sensing and search speed," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 1, pp. 66–74, Jan. 2010.

[51] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1407–1417, Aug. 2011.

[52] I. Arsovski, T. Chandler, and A. Sheikholeslami, "A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme," *IEEE J. Solid-State Circuits*, vol. 38, no. 1, pp. 155–158, Jan. 2003.

[53] H. Noda *et al.*, "A cost-efficient high-performance dynamic TCAM with pipelined hierarchical searching and shift redundancy architecture," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 245–253, Jan. 2005.

[54] T. Kusumoto, D. Ogawa, K. Dosaka, M. Miyama, and Y. Matsuda, "A charge recycling TCAM with Checkerboard Array arrangement for low power applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2008, pp. 253–256.

[55] Y.-D. Kim, H.-S. Ahn, S. Kim, and D.-K. Jeong, "A high-speed range-matching TCAM for storage-efficient packet classification," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 6, pp. 1221–1230, Jun. 2009.

[56] J.-S. Wang, H.-Y. Li, C.-C. Chen, and C. Yeh, "An AND-type match-line scheme for energy-efficient content addressable memories," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2005, pp. 464–610.

[57] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *Proc. HPCA*, Feb. 2017, pp. 445–456.

[58] I. Roy, A. Srivastava, M. Nourian, M. Becchi, and S. Aluru, "High performance pattern matching using the automata processor," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2016, pp. 1123–1132.

[59] K. Zhou, J. Wadden, J. J. Fox, K. Wang, D. E. Brown, and K. Skadron, "Regular expression acceleration on the micron automata processor: Brill tagging as a case study," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct./Nov. 2015, pp. 355–360.

[60] B. Govoreanu *et al.*, "$10 \times 10 nm^2$ Hf/HfO$_x$ crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *IEDM Tech. Dig.*, Dec. 2011, pp. 6–31.

[61] S. Pi, P. Lin, and Q. Xia, "Cross point arrays of 8 nm × 8 nm memristive devices fabricated with nanoimprint lithography," *J. Vac. Sci. Technol. B, Microelectron. Process. Phenom.*, vol. 31, no. 6, p. 06FA02, 2013.

[62] D. B. Strukov and H. Kohlstedt, "Resistive switching phenomena in thin films: Materials, devices, and applications," *MRS Bull.*, vol. 37, no. 2, pp. 108–114, Feb. 2012.

[63] G. W. Burr *et al.*, “Access devices for 3D crosspoint memory,” *J. Vac. Sci. Technol. B, Microelectron. Process. Phenom.*, vol. 32, no. 4, p. 040802, 2014.

[64] D. B. Strukov and R. S. Williams, “Four-dimensional address topology for circuits with stacked multilayer crossbar arrays,” *Proc. Nat. Acad. Sci. USA*, vol. 106, no. 48, pp. 20155–20158, 2009.

[65] B. Chakrabarti *et al.*, “A multiply-add engine with monolithically integrated 3D memristor crossbar/CMOS hybrid circuit,” *Sci. Rep.*, vol. 7, Feb. 2017, Art. no. 42429.

[66] G. C. Adam, B. D. Hoskins, M. Prezioso, F. Merrikh-Bayat, B. Chakrabarti, and D. B. Strukov, “3-D memristor crossbars for analog and neuromorphic computing applications,” *IEEE Trans. Electron Devices*, vol. 64, no. 1, pp. 312–318, Jan. 2017.

[67] G. Ligang, F. Alibart, and D. B. Strukov, “Programmable CMOS/memristor threshold logic,” *IEEE Trans. Nanotechnol.*, vol. 12, no. 2, pp. 115–119, Mar. 2013.

[68] D. Gavrilov, D. B. Strukov, and K. K. Likharev. (2017). “Capacity, fidelity, and noise tolerance of associative spatial-temporal memories based on memristive neuromorphic network.” [Online]. Available: https://arxiv.org/abs/1707.03855

[69] S. M. Chai, A. Gentile, W. E. Lugo-Beauchamp, J. Fonseca, J. L. Cruz-Rivera, and D. S. Wills, “Focal-plane processing architectures for real-time hyperspectral image processing,” *Appl. Opt.*, vol. 39, no. 5, pp. 835–849, 2000.

[70] C. Kittel, *Introduction to Solid State Physics*. Hoboken, NJ, USA: Wiley, 2005.

[71] *International Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 2013.

[72] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAS*, vol. 497. New York, NY, USA: Springer, 2012.

[73] (2018). *MATLAB Code for Optimal Throughput Calculation*. [Online]. Available: https://www.ece.ucsb.edu/~strukov/papers/2018/pm/code.m

[74] X. Tang, G. Kim, P.-E. Gaillardon, and G. De Micheli, “A study on the programming structures for RRAM-based FPGA architectures,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 4, pp. 503–516, Apr. 2016.

[75] M. Payvand *et al.*, “A configurable CMOS memory platform for 3D-integrated memristors,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 1378–1381.

[76] D. B. Strukov and K. K. Likharev, “CMOL FPGA circuits,” in *Proc. CDES*, 2006, pp. 213–219.

Advait Madhavan (M’12) received the M.S. and Ph.D. degrees from the Electrical and Computer Engineering Department, University of California Santa Barbara, Santa Barbara, CA, USA, in 2013 and 2016, respectively.

He is currently a Postdoctoral Researcher at the National Institute of Standards and Technology, Gaithersburg, MD, USA. His current research interests include novel methods for information processing, ranging from conceptualization of high-level architectures, analog and digital circuits to integration with emerging technologies and chip designs.

Dr. Madhavan was a recipient of the Micro Top Pick Award in 2015.

Tim Sherwood (M’03–SM’14) is currently a Professor of Computer Science and the Associate Vice Chancellor for Research at the University of California Santa Barbara, Santa Barbara, CA, USA. He specializes in the development of processors exploiting novel technologies, provable properties, and hardware-accelerated algorithms.

Prof. Sherwood is a seven-time winner of the IEEE Micro Top Pick Award, an ACM Distinguished Scientist, winner of the UCSB Academic Senate Distinguished Teaching Award, and is the 2016 SIGARCH Maurice Wilkes Awardee “for contributions to novel program analysis advancing architectural modeling and security.”

Dmitri B. Strukov (M’02–SM’16) received the M.S. degree in applied physics and mathematics from the Moscow Institute of Physics and Technology, Dolgoprudny, Russia, in 1999 and the Ph.D. degree in electrical engineering from Stony Brook University, Stony Brook, NY, USA, in 2006.

He is currently a Professor of Electrical and Computer Engineering at the University of California Santa Barbara, Santa Barbara, CA, USA. His current research interests include different aspects of computations, in particular addressing questions on how to efficiently perform computation on various levels of abstraction.