Enabling Full Associativity with Memristive Address Decoder

Leonid Yavits, Roman Kaplan, and Ran Ginosar
Technion – Israel Institute of Technology

Address decoders are typically built using regular logic gates. A novel Memristive Perfect Induction gate replaces standard NAND, allowing for storing the address alongside data and comparing it to the input address, thus transforming the address decoder into CAM and enabling fully associative access. Applications include fully associative TLB, cache, and virtually addressable memory.

Consider a typical address decoder (see Figure 1) in which the addresses of a memory block, row, and column are hardwired by AND-ing either direct or inverted address bits. A memory block, row, and column where the address matches the hardwired pattern are selected. We suggest adding a pair of memristors to each input of an address decoder in a voltage-dividing manner, achieving a two-fold effect: The address pattern in each block, row, and column becomes programmable rather than hardwired, and a memristor pair forms a XNOR—allowing for comparison of the input address bit to the bit "programmed" into the memristors. These effects enable content addressability, effectively turning an address decoder into a CAM.

A memristor pair at each input transforms a standard logic gate (see Figure 2(a)). We call this new gate a Memristive Perfect Induction (MPI) gate since all possible input combinations (see Figure 2(d)) can be programmed using memristors. In this paper, we present and evaluate a programmable memristive address decoder based on a MPI gate. We show that it exhibits read latency and energy consumption similar to those of a hardwired address decoder. We propose three memristive address decoder applications. A fully associative TLB can be implemented with similar read delay and energy consumption as one-way associative TLB. Likewise, a fully associative or many-way associative cache can be implemented incurring read delay and energy close to that of a direct-mapped cache. Third, a memristive address decoder enables virtually addressable main memory.

This work makes the following contributions:

- An MPI gate, inputs of which can be programmed to be direct or inverted, enabling all possible combinations of inputs

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

- A programmable memristive address decoder, which enables associative access at timing and energy costs close to those of a constantly addressed memory

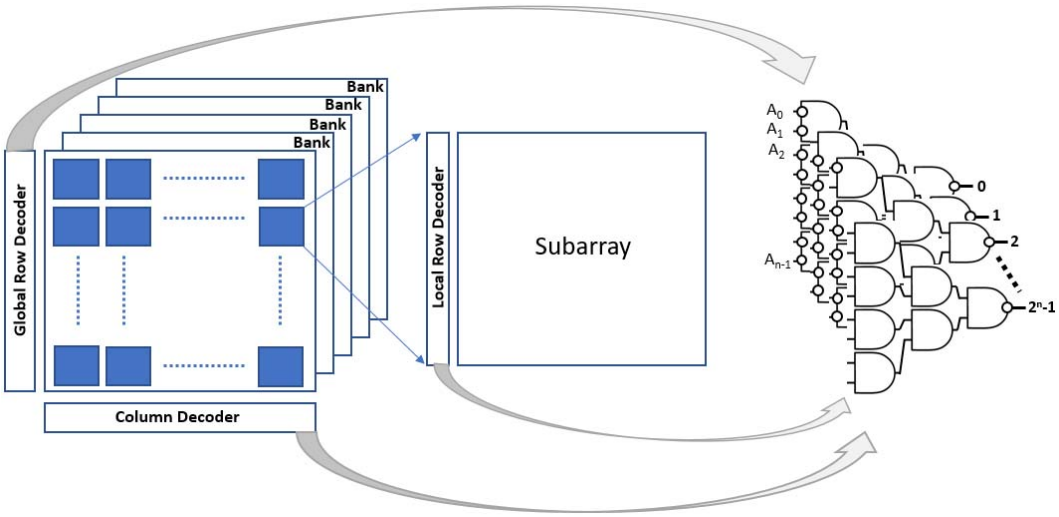


Figure 1. (Left) Block, row, and column address decoding in RAM. (Right) Example of a hardwired address decoder optimized for speed.

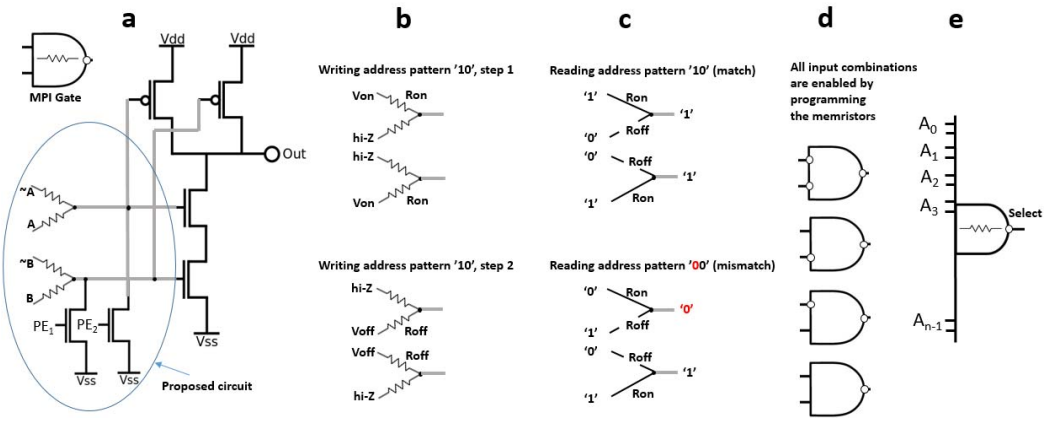


Figure 2. (a) Two-input MPI gate, (b) example of write, (c) examples of read with match and mismatch, (d) logic combinations enabled by memristors, and (e) programmable memristive address decoder.

MEMRISTIVE ADDRESS DECODER

A conventional address decoder is depicted in Figure 1. Typically, address decoding is decomposed into block address decoding, row address decoding, and column selection (see the left image in Figure 1). The block address decoder usually decodes the higher bits of an address, selecting a memory block. The row address decoder selects a memory row in each memory block, and the column selector chooses the memory columns in each memory block.

An address decoder can be implemented in a number of ways. For example, it can be designed as a series of AND tree segments (see the right image in Figure 1). This implementation is faster but occupies more silicon area than resource-sharing solutions, and it might be limited by memory pitch requirements. While random logic decoders are typically used in block address

decoding, other types of address-decoding schemes can be implemented in row or column address decoding, especially in dense nonvolatile memory.[1] For example, row address decoders are often implemented using dynamic NAND decoders.[2]

Conventional address decoders are hardwired. A constant address pattern is hardcoded in each address row by connecting each input of the decoder to either an address bit or inverted address bit, respectively. For example, all inputs of the first address row, associated with address "00…0," are hardwired to inverted address bits $\sim A_{n-1}$, $\sim A_{n-2}$, …, $\sim A_0$. All inputs of the last address row, associated with address "11…1," are hardwired to address bits A_{n-1}, A_{n-2}, …, A_0.

Memristors are two-terminal devices, and their resistance changes by changing the direction of the current through them. That resistance is bounded by a minimum resistance R_{ON} (low resistive state, logic "1") and a maximum resistance R_{OFF} (high resistive state, logic "0").

While a variety of resistive elements exist, one that seems well-suited for the address-decoder design is a memristor. It has an off/on ratio of up to 10^{11}, endurance of up to 10^{12}, and switching speed of 100 ps to 1 ns.[3]

Figure 2(a) presents the concept of an MPI gate. It is formed by adding a pair of memristors to each input of a conventional NAND gate, where the first memristor connects a direct input bit to both NMOS and PMOS gates, while the second memristor connects an inverted input bit to both NMOS and PMOS gates. Two separate NMOS transistors (PE_1 and PE_2) are added to program those memristors. Figure 2(d) shows all possible logic combinations of an MPI gate.

When an MPI gate is used in an address decoder (see Figure 2(e)), one of the memristors should be programmed to R_{ON}, while the second (complementary) memristor should be programmed to R_{OFF}. If the memristor connected to the input bit is R_{ON} (and the other one is R_{OFF}), the gate is controlled by the direct input. Alternatively, if the other element is R_{ON}, the gate is controlled by the inverted input. The former is considered address bit "1," and the latter is address bit "0."

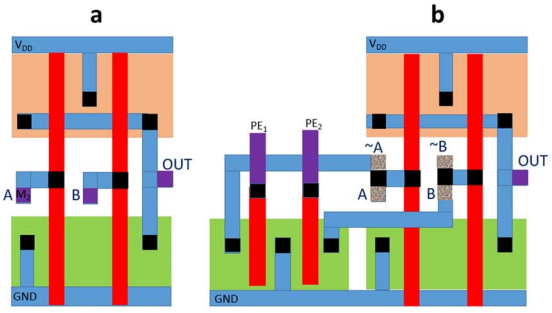
Memristors enable programming (writing) address alongside data block (which is written into a memory row, while the address is written into the associated row of address decoder). To write an address into a memristive decoder row, the PE_1 and PE_2 (see Figure 2(a)) are asserted, connecting the second terminals of the memristors to the ground. Address write occurs in two steps (see Figure 2(b)). In the first step (top of Figure 2(b)), address bit "1" is written into the applicable bits of the address row. A proper positive voltage level V_{ON} is applied only to those inputs in which an address bit "1" is written, as well as to those inverted inputs in which an address bit "0" is written, while the rest of inputs and inverted inputs are kept disconnected (hi-Z). In the second step (bottom of Figure 2(b)), address bit "0" is written into the remaining bits of the address row. A proper negative voltage level V_{OFF} is applied only to those inputs in which an address bit "0" is written, as well as to those inverted inputs in which an address bit "1" is written, while the rest of inputs and inverted inputs are kept disconnected. To enable the ternary implementation, a "don't care" state can be encoded by programming both memristors to R_{OFF} (not shown).

During data read (see Figure 2(c)), a pair of memristors functions as a XNOR. If an address bit matches the memristor value ("1" and R_{ON} or "0" and R_{OFF}, as shown in the top of Figure 2(c)), logic "1" is asserted to the gates of the relevant NMOS-PMOS pair of the NAND gate of Figure 2(a). Otherwise, XNOR outputs logic "0," as shown in the bottom of Figure 2(c). Only the address row in which the address pattern programmed into memristors match the address pattern placed on the address decoder input generates select signal. Clearly, at most one address row should be programmed with a given address. If no matching address is found, a "no match" is signaled (generated, for example, by wired-ORing of all address rows, not shown in Figure 2). Such a "no match" signal can be used to generate cache miss or page fault.

This read operation is functionally identical to a search in a CAM. In other words, the programmable memristive address decoder functions as CAM, allowing associative access to the memory array. With the memristive address decoder, memory addresses no longer need to be consecutive to enable efficient memory utilization, unlike in hardwired address decoders. Data originated by different applications with sparse addresses and random size memory allocations can be written continuously in physical memory.

EVALUATION

A two-input MPI gate was designed using the 28-nm CMOS High-k Metal Gate library from Global Foundries and SPICE-simulated using the TEAM model,[4] obtaining the timing and energy figures that are presented below. To compare the memristive address decoder to a hardwired CMOS one, we synthesized two 8-bit address decoders (hardwired and memristive-programmable ones) using the Synopsys Design Compiler with the 45-nm FreePDK open cell library and scaled the area and power figures to 28 nm to incorporate the MPI gate figures obtained by SPICE. The synthesis was optimized for timing. The comparison table in Figure 3(c) summarizes the area, power, and timing of the memristive-programmable and hardwired CMOS decoders.



CMOS - Memristor Decoder Comparison	Hardwired CMOS Decoder	Memristive Decoder	Over-head
Area, Cells, μm^2	288	354	23.1%
Area, Interconnect, μm^2	295	340	15.3%
Total Area, μm^2	583	694	19.1%
Dynamic power, μW	22.51	24.79	10.1%
Static power, μW	1.94	4.56	135.3%
Total power, μW	24.45	29.34	20.0%
Read Timing, ps	34.85	35.72	2.5%

Figure 3. (a) Layout example of a two-input standard NAND gate, (b) layout example of a two-input MPI gate, and (c) comparison of memristive and hardwired CMOS address decoders.

Silicon Area

A two-input MPI gate adds two NMOS transistors and four memristors to a regular NAND gate (Figure 2(a)). Memristors are created as vias between two metal layers and can at least partially be placed above CMOS logic. Figures 3(a) and 3(b) show the layout of a regular NAND and an MPI gate, respectively, presented in a standard cell row placement fashion. The MPI gate occupies almost twice the area of a regular NAND. However, the overall area overhead of the memristive address decoder is 19.1 percent (see Figure 3(c)).

Timing

Read access through the memristive address decoder is only slightly longer than a read through a hardwired address decoder. A small overhead of less than 1 ps, or 2.5 percent (see Figure 3(c)), is due to the input signal propagation through a memristor and the drain of the $PE_{1/2}$ transistor, as compared with propagation over wire.

Memory write is preceded by the address lookup. If the address exists (programmed into the address decoder), the data is written into that memory row. Otherwise, the new address is programmed into an available address row of the address decoder, simultaneously with writing the data in the same row of the memory array, to reduce the write latency. Programming delay could be up to 1 ns depending on the exact type of memristor.[3] This could substantially increase the write latency relative to hardwired address decoded memory. However, for most potential applications of the memristive address decoder, writes are quite infrequent compared to reads. Moreover, such increased write latency could be mitigated by dividing the memory into separate modules. If a write is followed by a read but they address different modules, then read and write can be executed in parallel. Another mechanism of the write latency mitigation is a write buffer. It uses a simple queuing mechanism to write data to memory during its free cycles. If a read comes before the data is written in memory, it is read from the write buffer instead.

Energy Consumption

Read dynamic energy consumption remains fairly close to that of a hardwired address decoder. The read energy overhead is between 0.05 and 0.1 fJ per gate, which translates to 10.1-percent overhead for the memristive address decoder (see Figure 3(c)). Write dynamic energy may also include the memristor programming energy, which may reach 1 pJ for memristors.[3]

Static power is consumed by current leaking through the memristor pairs and the programming transistors PE_1 and PE_2. A typical R_{OFF} for memristors embedded in digital logic is in $10^9\Omega$ range.[3] For an 8-bit address decoder, the leakage through additional hardware amounts to a static power consumption overhead of 2.62 μW, which leads to the total memristive address decoder power-consumption overhead of 20 percent (see Figure 3(c)).

Endurance

Endurance (namely, the number of times the memristor may be programmed until it stops functioning correctly) could limit the usage of the programmable memristive address decoder. The endurance of memristors is probably limited to 10^{12} times.[3] To mitigate such endurance, the frequency of write to each memory address must be low.

The probability of a write to a certain memory address equals the probability of a memory write times the probability of a specific entry to be selected; in the case of uniform memory, utilization equals $\frac{1}{|number\ of\ entries|}$. Given the typical size of L2 TLB or L2/L3 cache and typical write frequency in such devices, such probability can be quite low; for a memory structure with the memristive address decoder (with endurance of 10^{12}) to perform for 10 years at 1 GHz, the average frequency of write to each address should be ~ $1/315{,}000$ cycles^{-1}, so as not to exceed 10^{12} writes. For a 4-Mbyte L2 cache with the line size of 64 bytes and 4 Mbytes/64 bytes = 8,192 entries (yielding the probability of a certain address entry selection of $1/8{,}192$), assuming the fraction of memory access instructions is 20 percent and the L1 miss rate is 10 percent (yielding the L2 write probability of $1/200$), this condition is safely met:

$$1/200 \times 1/8{,}192 < 1/315{,}000.$$

If a memory structure has only a few entries (for example, some L1 DTLBs or L2 DTLBs with a 1-Gbyte page size), a contemporary memristor (with 10^{12} endurance) might not be a suitable building block. Such small memory structures, however, are set to benefit very little from the memristive address decoder anyway, since making them fully associative using conventional approaches (comparators) is quite cost-effective.

Effects of Process Variation

There are a number of device parameters potentially impacting the functionality and timing of the memristive address decoder, which can be affected by process variation. Such parameters include the R_{OFF} to R_{ON} ratio, which affects the behavior of a voltage divider at the address decoder input; R_{OFF}, which affects the static power; and memristor programming time, which affects memory write access timing. According to recent findings,[5] the worst-case R_{OFF}-to-R_{ON} ratio swing due to process variation is typically ±13 percent. The worst-case static power swing is ±6.8 percent. The worst-case program time and energy rise are 8.7 percent and 9 percent, respectively, due to voltage variation and 17.8 percent and 12 percent, respectively, due to process variation. The worst-case endurance drop is 9 percent due to voltage and

While process and voltage variation do affect the timing and energy consumption of a memristive address decoder, its functionality remains quite process variation-tolerant.

7.5 percent due to process variation, respectively. According to our simulations, while process and voltage variation do affect the timing and energy consumption of a memristive address decoder, as well as its endurance, its functionality remains quite process variation-tolerant.

APPLICATIONS

In this section, we suggest three potential applications of the memristive address decoder.

Translation Lookaside Buffer (TLB)

The hit ratio of TLB is important since an L2 TLB miss may result in a costly page walk, especially in a virtual-machine environment. The obvious way of improving the hit ratio is increasing the associativity of the TLB. However, associativity incurs larger silicon area, higher complexity, longer access delay, and higher energy consumption.

We propose replacing the TLB CMOS CAM with a programmable memristive address decoder, which provides an "affordable" full associativity. The concept of such a TLB is presented in Figure 4(a).

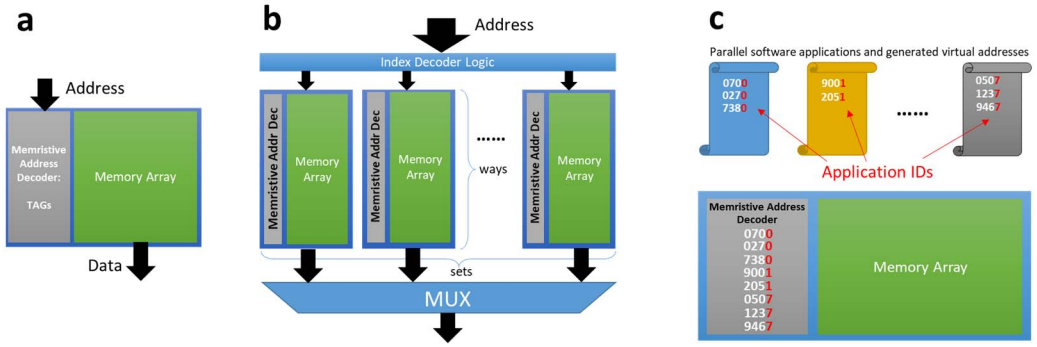


Figure 4. (a) Fully associative TLB, (b) many-way associative cache, and (c) virtually addressable memory concept.

Read access delay of a fully associative TLB using a memristive address decoder is similar to that of a one-way associative TLB, which is shorter than the access delay of a four-way or eight-way associative CMOS TLB. Read energy consumption of a fully associative TLB using a memristive address decoder is also similar to that of a one-way associative TLB. Writing energy of the memristive address decoder is higher than that of the hardwired TLB due to the need to program memristors. However, low write frequency in TLBs (around 1,000 or fewer writes per million instructions) is typical for many workloads.[6] In that case, the added energy required for programming the memristor may be negligible.

In summary, the memristive address decoder converts a one-way associative TLB into a fully associative TLB, improving hit ratio and reducing or eliminating page walks. The read latency and energy of such a fully associative TLB are very similar to those of a one-way associative TLB. The write energy of such a TLB is higher; however, since write is typically infrequent in TLBs, the impact on overall energy consumption may be minor.

The memristive address decoder converts a one-way associative TLB into a fully associative TLB, improving hit ratio and reducing or eliminating page walks.

Cache Memory

A fully associative cache generally delivers a higher hit ratio that a direct-mapped one. The memristive address decoder enables a fully associative cache with similar lookup time and energy as a direct-mapped cache that uses a hardwired address decoder. The silicon cost of a fully associative cache using the memristive address decoder is likely to be slightly higher than that of a direct-mapped cache, since memristive decoders are larger than hardwired ones.

Caches are usually too large to be designed as a single memory array. They are typically partitioned into a number of separate memory blocks, with the higher bits of address selecting the block and the lower bits selecting the memory row within the bank. There are two design possibilities. The first is to make both the block address and the row address decoders memristive-programmable, thus creating a fully associative cache similar to the fully associative TLB in Figure 4(a). Another (more silicon area-efficient) possibility is to make programmable only the row address decoder inside the memory block. This way, we create a many-way set associative cache in which each memory row of a memory block is a way and each memory block is a set. This many-way set associative cache concept is presented in Figure 4(b). A large number of ways is possible—for example, 256 or 512. The hit ratio of such many-way set associative caches is likely to be similar to that of a fully associative cache.

Read (lookup) timing and dynamic energy of a many-way associative cache are similar to those of a direct-mapped cache. The only difference is due to the discrepancy in the number of index bits in a direct-mapped cache versus the number of tag bits in the many-way set associative cache, which affects the number of NAND gates in, and hence the propagation delay of, the memristive row decoder. Cache replacement could be somewhat costly energy-wise, since per-cell programming energy of a memristor could reach 1 pJ.

In summary, the memristive address decoder converts a direct-mapped cache into a fully associative one. The lookup latency and dynamic energy of such a fully associative cache are similar to those of a direct-mapped cache. The write energy of such a cache is higher. However, as we move further in the cache hierarchy, the miss rate drops, and with it drops the frequency of write. Therefore, the impact of programming the memristors at each write on the overall energy consumption should not be critical, especially in higher-level caches.

The impact of programming the memristors at each write on the overall energy consumption should not be critical, especially in higher-level caches.

Virtually Addressable Memory

Virtual memory decouples a software application's view of its memory resources from their physical layout. This decoupling enables the operating system to eliminate memory fragmentation, as well as to improve performance and memory utilization. This is possible by managing physical memory resources while providing applications with a view of a contiguous address space that is isolated from all other applications. Virtual-to-physical address mapping (translation) often results in performance degradation, especially in virtual environments.[7]

Introducing the memristive address decoder to the main memory may enable the elimination of physical addressing altogether. In a write access, the virtual addresses—possibly extended by the application's ID (see Figure 4(c))—are transferred to the memory along with the data and are programmed into the memristive address decoder. Such virtual addresses, no matter how sparse they are and regardless of how much virtual space is allocated by each application, are programmed continuously into the physical memory so that they occupy only as much space as is actually required, allowing for efficient memory utilization.

The delay and energy impact of programming memristors can be mitigated by lower write frequency (which could be the case if most memory accesses hit in the cache and there is no write-through). Data is read using the virtual rather than physical address (which no longer exists). An

address is associatively looked up in the address decoder, and a matching row is selected. Although functionally equivalent to a search in content-addressable memory, the read is very similar in terms of access delay to a read using a hardwired address-decoded memory. If an address that is not programmed in the memristive address decoder is accessed, the "no match" signal is generated, signaling a page fault to the operating system.

RELATED WORK

Resistive element-based logic gates and combinational circuits have been introduced in prior work. Borghetti *et al*. introduced a memristor-based imply gate.[8] James *et al*. developed a memristor-based universal gate.[9] Gao *et al*. proposed a programmable CMOS/memristor threshold logic gate.[10] Zhao *et al*. designed an STT-MRAM and memristor-based nonvolatile full adder.[11] The purpose of our work is to extend the functionality of existing logic designs rather than replacing CMOS logic with a memristor-based one.

Resistive NOR CAM technologies have also been explored, such as memristor-based[12-14] and STT-MRAM-based[15,16] CAM. Our design bears similarity with resistive NAND CAM;[2] however, it extends its functionality beyond the NAND address decoder into general logic design.

A CMOS-based programmable address decoder to support a variable-width RAM (VaWiRAM) was introduced by L.K. John.[17] In contrast, our design combines memristors with CMOS logic to enable fully associative access.

CONCLUSION

We propose a programmable memristive address decoder in which the address patterns are programmed rather than hardwired. It allows per-bit comparing of the input address with the programmed address pattern, effectively turning the address decoder into a CAM. At the core of the memristive address decoder is a new MPI gate, which enables programming of any input combination using memristors.

The read latency and energy consumption of the memristive address decoder are similar to those of a hardwired decoder. Its silicon area overhead is limited. Thus, the memristive address decoder enables fully associative memory structures at a price close to that of direct-mapped ones.

The read latency and energy consumption of the memristive address decoder are similar to those of a hardwired decoder.

We discuss potential applications of the memristive address decoder. One such application is a fully associative TLB at similar silicon area, read delay, and energy consumption as a one-way associative TLB. Another application is a many-way associative cache with read access timing similar to that of a direct-mapped cache. Introducing the programmable memristive address decoder to main memory may potentially enable elimination of physical addresses and virtually addressable memory.

REFERENCES

1. G. Campardo, R. Micheloni, and D. Novosel, *VLSI-design of non-volatile memories*, Springer, 2015.
2. L. Yavits, U. Weiser, and R. Ginosar, "Resistive Address Decoder," *IEEE Computer Architecture Letters*, vol. 16, no. 2, 2017, pp. 141–144.
3. J. Yang, D. Strukov, and D. Stewart, "Memristive Devices for Computing," *Nature Nanotechnology*, vol. 8, no. 1, 2013, pp. 13–24.
4. S. Kvatinsky et al., "TEAM: threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, 2013, pp. 211–221.

5. M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1T1R resistive RAM," *IEEE Transactions on Very Large Scale Integration Systems*, 2014.
6. A. Bhattacharjee and M. Martonosi, "Characterizing the TLB behavior of emerging parallel workloads on chip multiprocessors," *18th International Conference on Parallel Architectures and Compilation Techniques*, 2009.
7. A.H. Zang et al., "Do-It-Yourself Virtual Memory Translation," *44th Annual International Symposium on Computer Architecture*, 2017.
8. J. Borghetti et al., "Memristive switches enable 'stateful' logic operations via material implication," *Nature*, 2010.
9. A. James et al., "Resistive Threshold Logic," *IEEE Transactions VLSI Systems*, 2014.
10. L. Gao et al., "Programmable CMOS/memristor threshold logic," *IEEE Transactions on Nanotechnology*, 2013.
11. W. Zhao et al., "Synchronous non-volatile logic gate design based on resistive switching memories," *IEEE Transactions on Circuits and Systems I*, 2014.
12. F. Alibart, T. Sherwood, and D.B. Strukov, "Hybrid CMOS/nanodevice circuits for high throughput pattern matching application," *IEEE Conference on Adaptive Hardware and Systems*, 2011.
13. J. Li et al., "1Mb 0.41 µm 2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," *IEEE Symposium on VLSI Circuits*, 2013.
14. O. Kavehei et al., "An associative capacitive network based on nanoscale complementary resistive switches for memory-intensive computing," *Nanoscale*, 2013.
15. G. Qing et al., "AP-DIMM: Associative Computing with STT-MRAM," *40th International Symposium on Computer Architecture*, 2013.
16. S. Matsunga et al., "Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices," *Applied Physics Express*, 2009.
17. L.K. John, "VaWiRAM: a variable width random access memory module," *Ninth International Conference on VLSI Design*, 1996.

ABOUT THE AUTHORS

Leonid Yavits is a postdoctoral fellow in electrical engineering at Technion – Israel Institute of Technology. He has a PhD in electrical engineering from the same university. He is the cofounder of VisionTech where he co-designed a single-chip MPEG-2 codec; the company was acquired by Broadcom in 2001. Yavits has co-authored a number of patents and research papers on SoC and ASIC. His research interests include non-von Neumann computer architectures, accelerators, and processing in memory. Contact him at leonid.yavits@nububbles.com.

Roman Kaplan is a PhD candidate in electrical engineering at Technion – Israel Institute of Technology under the supervision of Prof. Ran Ginosar. He has a master's degree from the same university. Previously, he worked as a software engineer. Kaplan's research interests are parallel computer architectures, in-data accelerators for machine learning, and bioinformatics. Contact him at romankap@gmail.com.

Ran Ginosar is a professor in the Department of Electrical Engineering and serves as head of the VLSI Systems Research Center at Technion – Israel Institute of Technology. He has a PhD from Princeton University in electrical and computer engineering. Previously, he was a visiting associate professor with the University of Utah and a visiting faculty with Intel Research Labs. His research interests include VLSI architecture, manycore computers, asynchronous logic and synchronization, networks on chip, and biologic implant chips. He has co-founded several companies in various areas of VLSI systems. Contact him at ran@ee.technion.ac.il.