# Computational Memory Architecture Supporting in Bit-Line Processing

**Driss Azougagh, Ahmed Rebbani, and Omar Bouattane**

SSDIA Laboratory, ENSET Mohammedia, Hassan II University of Casablanca, Morocco

**Summary**
Processing In Memory (PIM) is in demand more than ever to cope with the growth of Big Data, memory wall and power wall. It eliminates the overhead of data movement between processing unit and memory resulting in high bandwidth, massive parallelism, and high energy efficiency. Most existing PIM works are concentrated on near-memory processing (NMP) and/or in-memory processing (IMP). In this paper we present a computational memory architecture supporting in bit-line processing, or simply compute-line, using Directed dual bit-line Keeper. In one clock cycle and through bit-line selection, the compute-line allow multi-rows read, bitwise logic compute and multi-rows write, simultaneously. The bit-line keeper projects the outcome in the selected bit-line to the non-selected one before write operation. The architecture is backward compatible with conventional Static Random Access Memory (SRAM) but it has advantage of not using bit-line pre-charging and sensing for read and write operations. It reduces bit-line activities by more than half. When used as an in-memory computing, it also eliminates overhead of data movement demonstrating considerable reduction in bandwidth and energy consumption.
The proposed compute-line is validated and tested to show considerable performance and effectiveness according to the new capabilities offered. This architecture can support a variety of interconnect topology between multiple compute-lines which will benefit many parallel applications.

***Key words:***
*SRAM Memory, Built-in Computing, bit-line keeper, in-place processing.*

## 1. Introduction

The growth of Big Data, memory wall and power wall are posing unprecedented demand for Processing In Memory (PIM) [1]. The PIM solutions proposed to move processing logic near the cache [2], [3] or main memory [4], [5]. 3D stacking can make this possible [6]. Compute Caches significantly push the envelope by enabling in-place processing using existing cache elements, where at least one of the operands used in computation has cache locality for data-centric applications. Several Processor-in-Memory architectures have been proposed [7], [8]. Authors in [9] proposed Compute Caches architecture that uses an emerging SRAM circuit technology, which can be referred to as bit-line computing [10], [11].

Today, in [12], RRAM was viewed as a promising candidate that can meet future storage and computing needs. Author discussed potential computing applications enabled by RRAM devices within both conventional and emerging computing paradigms and introduced a concept of RRAM based Memory Processing Unit (MPU). In their investigation, due to the sneak-path effect and the tradeoff between data retention and endurance, device-level and system level innovations are still needed for large-scale implementation and storage systems. RRAM device tends to be applicable for LUT-based circuitry to store truth table of a Boolean function and for programmable switch to link other CMOS sub-circuits in Field Programmable Gate Arrays (FPGAs).

Static Random Access Memory (SRAM) has been the predominant technology used to implement memory cell in computer systems [13]. In conventional SRAM, bit-lines need to be pre-charged prior any read operation and requires differential sensing or amplification of its voltage levels.

In this paper, we introduce a computational memory architecture based on a compute-line with directed dual bit-line Keeper. It’s main purpose is to bring parallel computation as close as possible to the location of the stored data/bits. The compute-line is reflexive in the sense that read, logic, and write operations can be executed locally. In each cycle, a compute-line fulfills one bit-wise operation in the following sequences. It first selects and sets up an operation, reads bit information from external inputs and/or local memory cells in parallel and simultaneously, stabilizes the bit-lines throughout the keeper, and finally writes the data to external outputs and/or local memory cells.

Next is organized as follow. In the next section we start with an introduction of the computational memory architecture. In the third section we describe it functionality and analyze the results. And finally, we conclude with perspective remarks.

## 2. Computational Memory Architecture

A Computational Memory Architecture is composed of m parallel compute-lines as shown in Fig. 1(a). Each

Manuscript received July 5, 2018
Manuscript revised July 20, 2018

compute-line is composed of d blocks, a pair of bit-lines (XBL and YBL) and an extra pair of select-lines (XSL and YSL), as depicted in Fig. 1(b). Each block can be either an output (OUTPUT), an input (INPUT), a memory cell (MCELL) or a bit-line Keeper (KEEPER) as illustrated in Fig. 1(c). For functional compute-line, all block components and their dimension d×m need to be carefully chosen in order to allow stable multiple read/writes from/to MCELLs to/from candidate bit-line(s), simultaneously and efficiently.

The architecture has a control unit with extra control column lines (XSL and YSL) for orchestration and synchronization of in-place processing operations. To save circuitry overhead, one global pair of select-lines XSL and YSL is sufficient and can be shared in an interleaved manner between each two adjacent compute-lines by replacing, for each column k, XSLk with XSL and YSLk with YSL. Furthermore, horizontal blocks BLOCKj* of the same kind from different compute-lines can share the same word line commands from the control unit. In such a setup, computational memory will act as a bit-wise Single Instruction Multiple Data (bit-wise SIMD).

The circuitry of the blocks OUTPUT, INPUT, MCELL and KEEPER are described in details in the Fig. 2(a), (b), (c), and (d), respectively. For a given particular compute-line, all blocks share the same bit-lines and selected lines. We use the initial letters "X" and "Y" to emphasize the symmetry in the compute-line between left and right bit-lines XBL and YBL, the directed/inverted read word lines XR/YR, the direct/inverted write word line XW/YW, storage nodes XB and YB, inputs XI and YI and outputs XO and YO, respectively. Similar analogy applies for nMOS/pMOS transistors and AND logical gates. However, during simulation, AND logic gates in the figure are replaced with pass transistors. The KEEPER, select-line(s) and bit-lines play major role in supporting in-place/in-line computing when operand locality constraint is satisfied. Similarly to [14], our Computational Memory does not require a pre-charging cycle for read. However, unlike [14], it does not use separated extra read-line (RL) for read operation, instead it uses the existing bit-lines for read operation and in addition it uses extra select-line(s) for operation control.

## 2.1 OUTPUT Block

The OUTPUT block comprised of storage nodes (XB and YB) tightly connected to the outputs (XO and YO) and four nMOS pass transistors (XT1, YT1, XT2 and YT2). At write command (when only XW or YW is set high), the OUTPUT stores the bit-lines state into it's corresponding storage nodes XB and YB directly (XW="1") or inversely (YW="1"). Similar to a buffer, OUTPUT's bit of information are continuously projected as an output XO and YO for external use by other compute-lines and/or devices.

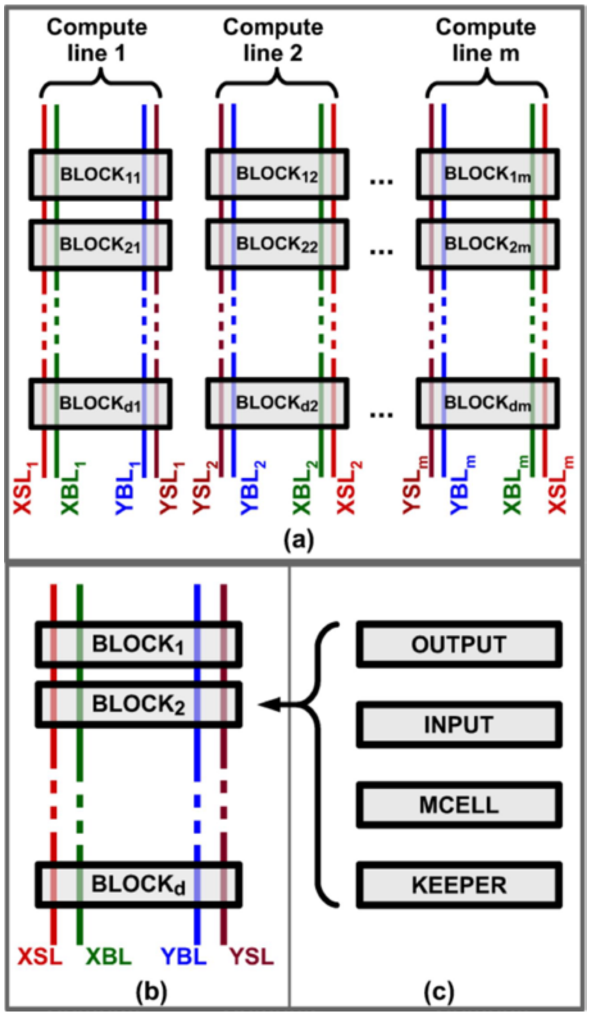

Fig. 1 The Computational Memory Architecture at (a), compute-line at (b) and possible kinds of block at (c).

## 2.2 INPUT Block

The INPUT block comprised of two read word lines (XR and YR), two inputs (XI and YI) and four groups of a couple of AND logic gate and nMOS transistor; (XG1,XN1), (YG1,YN1), (XG2,XN2) and (YG2,YN2). The nMOS transistors XN1 and XN2 are connected to the bit-line XBL, symmetrically; transistors YN1 and YN2 are connected to the bit-line YBL. When one of the AND logic gate Gj outputs a "1", the corresponding nMOS transistor Nj is activated and pulls-down the bit-line connected the it's drain.

## 2.3 MCELL Block

The MCELL structure combines and couples the above two blocks (INPUT and OUTPUT) by simply wiring XO and YO with XI and YI, respectively. In the following, MCELL will be addressed as an INPUT or an OUTPUT whenever fit. The MCELL intermediates between it's storage nodes XB and YB and the bit-lines XBL and YBL throughout read/write word lines (XR/XW, and YR/YW) and select-lines (XSL and YSL).

## 2.4 KEEPER Block

The KEEPER block is composed of two capacitors XC and YC, two bit-lines XBL and YBL, four pulling up transistors XK, XP, YK and YP, two AND logic gates XG and YG and two pulling-down nMOS pass transistors XN and YN. All these components are used internally except for the bit-lines that are shared with the remaining blocks in the compute-line. KEEPER's role is to keep the pair of bit-lines superposed using a controlled cross-coupled inverter. Through the control line BK, the KEEPER may translate the inverted state in the selected bit-line to the non-selected one.
When BK is "1", the nMOS transistors XK1 and YK1 are activated and supply weak voltage to charge the selected bit-line if no block is pulling down the bit-line. At the same time, the KEEPER forces the state inverse of selected bit-line to the other. The direction where to keep the bit-lines' voltage levels inverted depends on which select-line is activated. When XSL and BK are set high (with YSL="0"), the logic gate XG forwards the XBL state to the inverter composed of YP and YN transistors causing YBL to a state opposite to that of XBL.
In case no pull down is exercised on XBL, the XBL's state is forced to high by charging the capacitor XC, through XK and XP transistors commanded by AND logic gate YG with output is "0" since YSL is "0", and consequently the capacitor YC is discharged. Otherwise, if XBL is strongly pulled down and forced to "0", the capacitor XC is discharged and the capacitor YC is charged afterward.

## 3. Functionalityand Results

The compute-line supports all bit-wise basic logic operations summarized in the equation 1, including NOT, NOR and NAND. It is worth mentioning that when both XSL and YSL and/or both XW and YW (of the same OUTPUT) are set high, the states of bit-lines XBL and YBL become undetermined. When both XR and YR (of the same INPUT) are set high both bit-lines are pulled down to "0". In the following, we try to avoid the use and omit investigation details of these cases.

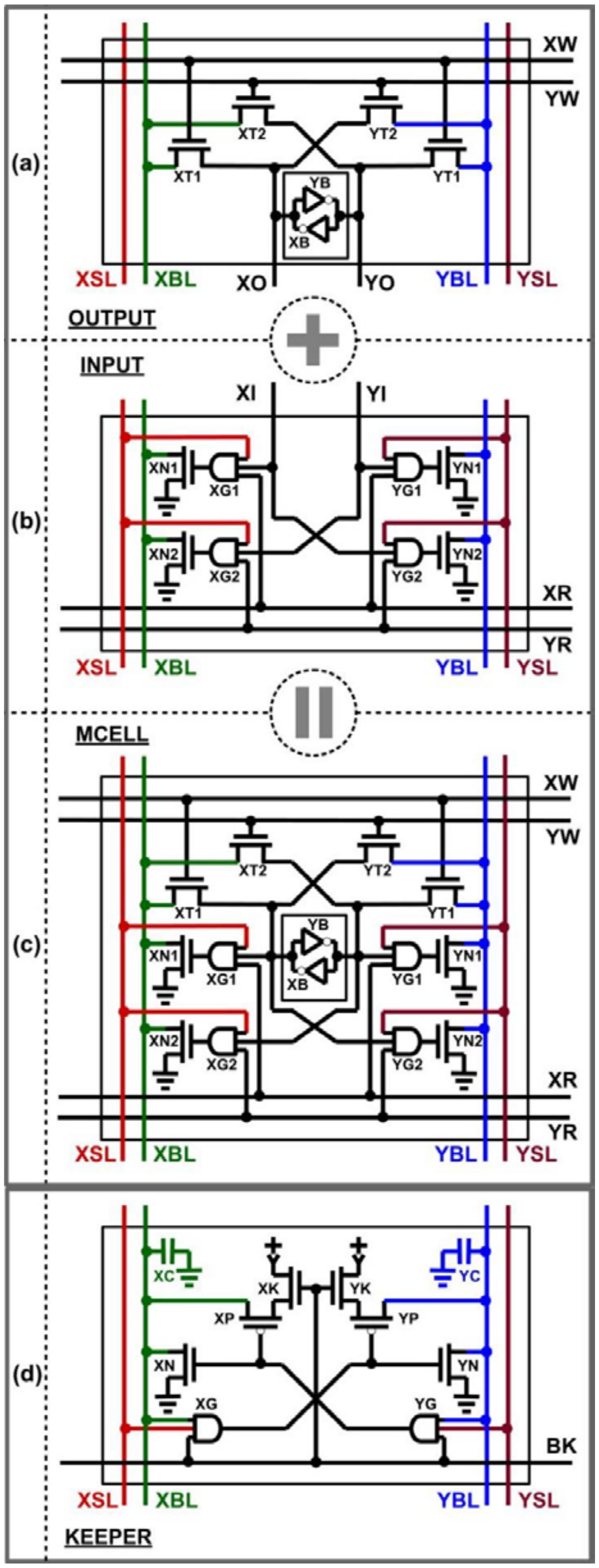

Fig. 2 Compute-line circuitry for (a) OUTPUT, (b) INPUT, (c) MCELL and (d) KEEPER blocks.

$$XB_i = XW_i * XBL + YW_i * YBL \quad ;where$$

$$\overline{XBL = \sum_{j=1}^{d} XSL * (XR_j * XB_j + YR_j * YB_j)} \quad and$$

$$\overline{YBL = \sum_{j=1}^{d} YSL * (XR_j * YB_j + YR_j * XB_j)}. \tag{1}$$

This section provides a timing diagram exemplary of a memory controller for the proposed compute-line. We then apply the timing diagram sequence to read and write a bit of information. We used a setup that uses a compute-line with one KEEPER, four MCELLs, one INPUT and one OUTPUT while sharing the same bit-lines and select-line(s).

To reduce the circuitry overhead in the compute-line, we omitted all commands starting with initials Y and their related logics as in Fig. 3. Therefore, we excluded line YW and transistors XT2 and YT2 in all OUTPUTs, and line YR, transistors XN2, YN1 and YN2, and the AND logic gates XG2, YG1, and YG2 in all INPUTs. In the KEEPER block, we removed the transistor XN and the logical gate YG and wired the gate of XP to the ground. To differentiate between INPUT and OUTPUT from MCELLs commands we renamed read word-line XR of INPUT with RD and write word-line W of OUTPUT with WR, and we numbered from 1 to 4 the ones of MCELLs.

For a case study, we propose the implementation of a bitwise Full adder to compute an addition of two n-bits values a and b with an initial carry bit c0. This is the arithmetic operation used the most in any computation unit. For demonstration purpose, our implementation takes two input operands ai and bi and one carry ci, and returns an output si and a carry ci+1.

On a compute-line, there is an order in which a set of operations will be conducted in time to fulfill one full adder operation. At the beginning, three cycles are consumed to read one carry c0 and two bits of information a0 and b0 from a streamed input XI. These cycles can be omitted when the data are available locally. It then follows nine cycles for nine basic logical operations required by the full adder to compute the outcome addition s0 and carry c1. For further use of the outcome carry c1, we might need to just fetch the next significant bits of information a1 and b1 if not available locally.

For spice simulation, we used 45nm PTM model, as cited in [15], for high-performance application (PTM-HP), incorporating high-k metal gate and stress effect (level=54 & version=4.0). Other tested transistor models provided similar results with noticeable difference in the choice of the working configuration for the capacitor and transistors.

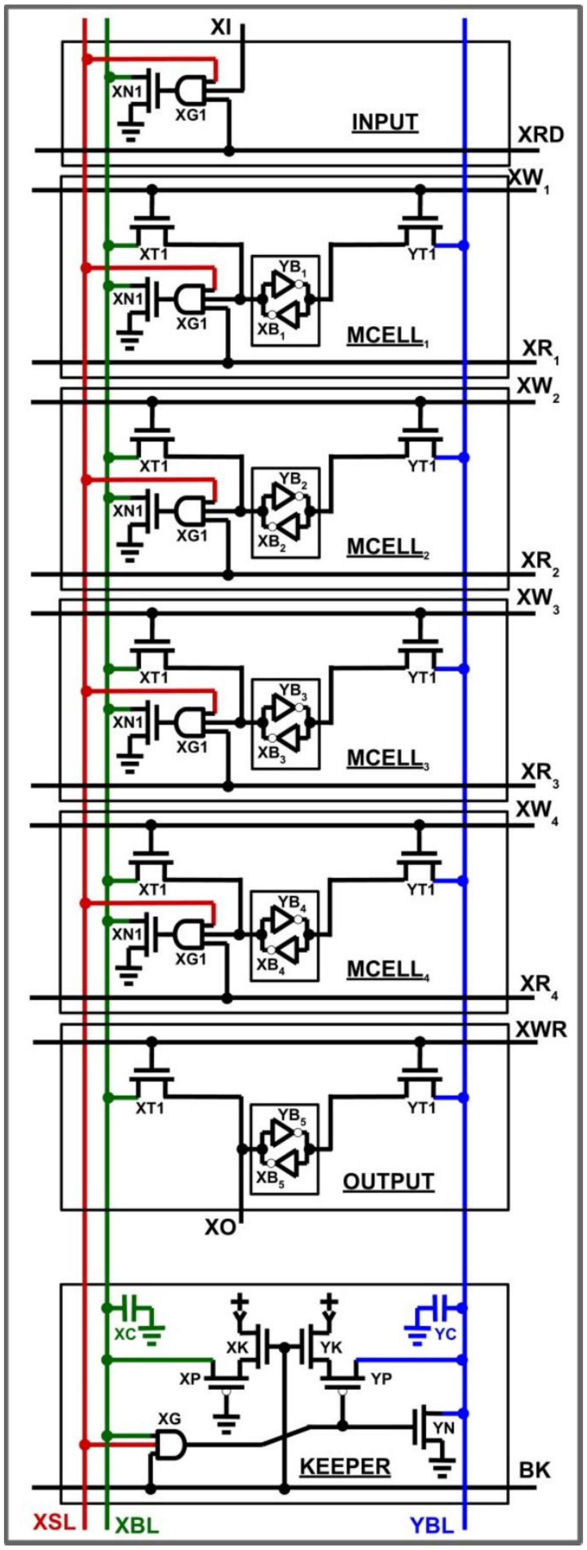

Fig. 3 Compute-line circuitry for (a) OUTPUT, (b) INPUT, (c) MCELL and (d) KEEPER blocks

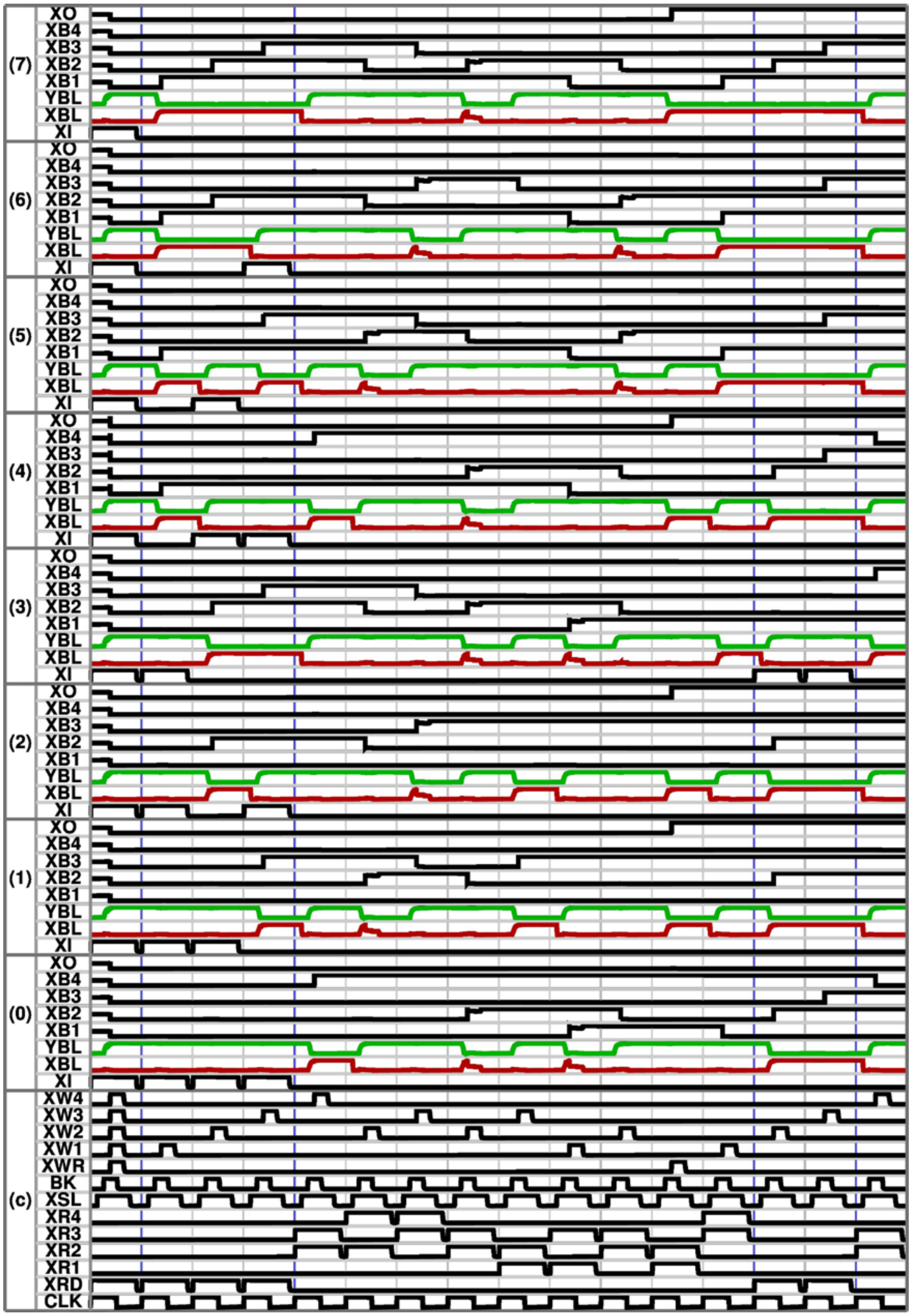

Fig. 4 Voltage waveforms for simulated Full Adder for (c) control commands and, for i  {0,..,7}, row (i) shows XI, XBL, YBL, XB1, XB2, XB3, XB4, XO for input operands value extracted from the binary representation b0|a0|c0 of i

## 2.1 Memory Controller

The role of controller is to orchestrate one or many operations to be computed in the bit-lines. The controller can receive a stream of operations to apply in a form of command words. For one cycle, a command word can be decoded and applied to a circuitry clocked to generate an output command, as in Fig. 4(c), and apply the generated commands to the compute-line(s). For a number $i\in\{0,..,7\}$ with its equivalent 3-bit binary representation i = b0|a0|c0, the Fig. 4(i) represents the voltage evolution in time of the compute-line running the full adder of a0 and b0 with a carry c0. Initially c0 is fetched from the inverse of XI input and stored into XB1. Then, for each iteration k, when ak and bk are not available locally, they are fetched to XB2 and XB3 sequentially from the inverse of XI input, respectively.
In the figure, a cycle is determined by a clock (CLK). Within a clock CLK, we can distinguish eight intervals or phases. The first phase is reserved for operands selection when setting read word lines XRD, XR1, XR2, XR3 and XR4 to high and before setting the select-line XSL to high. The second phase start from the time XSL is set high until before the bit-line keeper command BK is set high. At this stage, if any participant INPUT pulls-down the bit-line XBL, the capacitor XC is discharged and maintains it new state.
As BK is set high, in the third phase, both XBL and YBL are adjusted as explained before. The KEEPER might charge XBL and then transfers the inverse of XBL to YBL. Right after, follows the fourth phase where write word-lines XWR, XW1, XW2, XW3 and XW4 are set high and all storage nodes of the selected OUTPUTs will be updated. The remaining phases, five to eight, start by setting to low BK, WR, XSL and XR, in the given order. These phases are used to keep consistency and stability of stored data. Any changes applied in the input operands after sixth phase has no effect and is not advised to do so for efficiency purpose. The critical zone that might temper with the operation outcome is localized between the third phase and the fifth. Hence, any change in the INPUTs at this zone might result into an inconsistent outcome.

## 2.2 Analysis and Results

In the diagram, the first CLK cycle-1 is reserved to reset storage nodes of all OUTPUTs to zero in one single row read (XRD="1") and multiple-rows write (XWR=XW1=XW2=XW3=XW4="1"). The next clock cycle0 is reserved to load the initial carry c0 in XI's inverse and store it in XB1. The clocks cycle1 and cycle2 are reserved to load the sequential content of a0 and b0 read from the inverse of the INPUT's XI to be stored in XB2, and XB3, respectively. Nine cycles from cycle3 to cycle11 are used to achieve the full adder based only on a NOR logic operation. At first, the contents in XB2 and XB3 are retrieved by read word-lines XR2 and XR3 and their NOR result is stored into the XB4 using XW4. Similarly, full sequence of operations is summarized in the following list:

- ❖ load initial carry :
  $cycle_0$ : XB1 = NOT(XI) ; for XI = NOT($c_0$)
- ❖ first bit-wise operation :
  $cycle_1$ : XB2 = NOT(XI) ; for XI = NOT($a_0$)
  $cycle_2$ : XB3 = NOT(XI) ; for XI = NOT($b_0$)
  $cycle_3$ : XB4 = NOR(XB2, XB3)
  $cycle_4$ : XB2 = NOR(XB2, XB4)
  $cycle_5$ : XB3 = NOR(XB3, XB4)
  $cycle_6$ : XB2 = NOR(XB2, XB3)
  $cycle_7$ : XB3 = NOR(XB1, XB2)
  $cycle_8$ : XB1 = NOR(XB1, XB3)
  $cycle_9$ : XB2 = NOR(XB2, XB3)
  $cycle_{10}$ : XO = NOR(XB1, XB2) ; ➔ $s_0$ = XO
  $cycle_{11}$ : XB1 = NOR(XB3, XB4) ; ➔ $c_1$ = XB1
- ❖ second bit-wise operation :
  $cycle_1$ : XB2 = XI ; where XI = $a_1$
  $cycle_2$ : XB3 = XI ; where XI = $b_1$
- ❖ first bit-wise operation :
  … (2)

After cycle11, in the kth iteration, the next significant bits ak+1 and bk+1 can be fetched into XB2 and XB3 respectively, and proceed with the next bit-wise full adder using the carry remained from the previous computation in XB1. As the proposed computational memory architecture can have m compute-lines, the time to compute an addition of two n-bits values can be amortized to $11\times n/m$ cycles when data are locally available. As m increases, the average overall time of an n-bits addition becomes promising and exempted from the overhead of exchanging data with CPU in conventional systems.
Consistent results are obtained in the OUTPUTs XO and XB1 for all combination of i = b0|a0|c0 and they are depicted in the corresponding row (i) in the figure. Cycles cycle4, cycle5, cycle6, cycle5 and cycle6 experience recursive bit-wise NOR operation, where one of the involved operands is used as well to store the outcome result. Before storing the outcome results, as the operand voltage level is high, a slight impact on the bit-line XBL can be noticed. However, Recursion consistency is guaranteed due to the stability enforced by the KEEPER on the bit-lines.
To analyze performance, we estimated the count of charging capacitors and bit-line activities when the conventional SRAM is used instead and compared the results with the proposed compute-line for conducting one

full-adder. The Fig. 5 summarizes the performance gained by using compute-line and shows an improvement of 60% for bit-line activities and 68% capacitor charging count. Although the operands are symmetric for a full-adder, the performance varies depending also on the content of the involved operands. Therefore, the sequence in which the bit-wise NOR operation is applied does impact the bit-line activities and capacitor charging count. In the x axis in the curve, the content of the operands (the input i = b0|a0|c0) are ordered so that the performance keeps increasing. Overall, as the number of input values with high voltage value is large as the performance increase for the full adder.

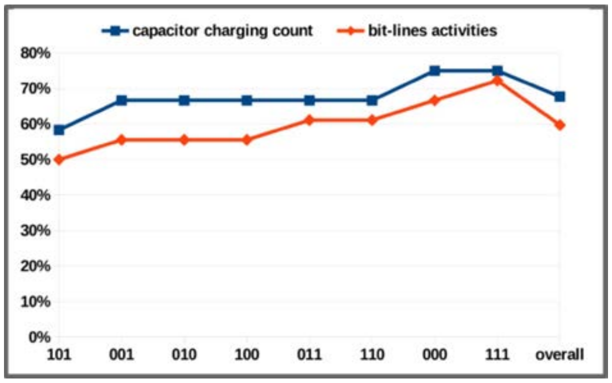

Fig. 5 Performance improvement for capacitor charge count and bit-line activities

## 3. Conclusion

In this paper we introduced a computational memory for in-place storage processing based on compute-line concept with support of extensive data locality. It is a memory with built-in computing capabilities designed to support bit-wise basic binary operations (including NOR, NAND and NOT) on multiple local and/or remote data simultaneously. A compute-line was centralized by KEEPER that keeps directing bit-lines to stay distinguished and stable right before every write operation. KEEPER was able to correct the result in the selected bit-line and apply the inverse to the non-selected one after reading from INPUTs and before writing the outcome result to OUTPUTs.

We introduced new computational memory architecture with little overhead compared to conventional SRAM while having extra computational power that reduces data movement and allows designer to massively harness parallelism it can offer. Generally, depending on the program code of interest, a suitable connection topology between multiple compute-lines can be investigated to achieve massive parallelism at execution. An illustrator example was introduced and showed how data movement can be reduced considerably while executing built-in bit-wise operations. In compute-line, the bit-lines do experience little activities offering great potential of reducing power consumption. In our testing example, the bit-line charging was reduced by about 68% and bit-line activity by 60%.

## References

[1] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” SIGARCH Comput. Archit. News, vol. 23, no. 1, pp. 20–24, Mar. 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[2] P. A La Fratta and P. M Kogge, “Design enhancements for in-cache computations,” Workshop on Chip Multiprocessor Memory Systems and Interconnects, 01 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.577.4395&rep=rep1&type=pdf

[3] F. Duarte and S. Wong, “Cache-based memory copy hardware accelerator for multicore systems,” IEEE Transactions on Computers, vol. 59, no. 11, pp. 1494–1507, Nov 2010. [Online]. Available: https://doi.org/10.1109/TC.2010.41

[4] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, “A case for intelligent ram,” IEEE Micro, vol. 17, no. 2, pp. 34–44, Mar 1997. [Online]. Available: https://doi.org/10.1109/40.592312

[5] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, “Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization,” in 2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), ser. MICRO-46. New York, NY, USA: ACM, Dec 2013, pp. 185–197. [Online]. Available: https://doi.org/10.1145/2540708.2540725

[6] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture,” in 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). ACM, June 2015, pp. 336–348. [Online]. Available: https://doi.org/10.1145/2749469.2750385

[7] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick, “Scalable processors in the billion-transistor era: Iram,” Computer, vol. 30, no. 9, pp. 75–78, Sep 1997. [Online]. Available: https://doi.org/10.1109/2.612252

[8] M. Oskin, F. T. Chong, and T. Sherwood, “Active pages: a computation model for intelligent memory,” in Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235). Washington, DC, USA: IEEE Computer Society, Jun 1998, pp. 192–203. [Online]. Available: https://doi.org/10.1109/ISCA.1998.694774

[9] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, “Compute caches,” in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2017, pp. 481–492. [Online]. Available: https://doi.org/10.1109/HPCA.2017.21

[10] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," IEEE Journal of Solid-State Circuits, vol. 51, no. 4, pp. 1009–1021, April 2016. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2515510
[11] M. Kang, E. P. Kim, M. s. Keel, and N. R. Shanbhag, "Energy-efficient and high throughput sparse distributed memory architecture," in 2015 IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, May 2015, pp. 2505–2508. [Online]. Available: https://doi.org/10.1109/ISCAS.2015.7169194
[12] M. A. Zidan and W. D. Lu, "Rram fabric for neuromorphic and reconfigurable compute-in-memory systems," in 2018 IEEE Custom Integrated Circuits Conference (CICC), April 2018, pp. 1–8.
[13] A. Valero, S. Petit, J. Sahuquillo, P. LÃ¸spez, and J. Duato, "Design, performance, and energy consumption of edram/sram macrocells for l1 data caches," IEEE Transactions on Computers, vol. 61, no. 9, pp. 1231–1242, Sept 2012. [Online]. Available: https://doi.org/10.1109/TC.2011.138
[14] N. Verma and A. P. Chandrakasan, "A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy," IEEE Journal of Solid-State Circuits, vol. 43, no. 1, pp. 141–149, Jan 2008. [Online]. Available: https://doi.org/10.1109/JSSC.2007.908005
[15] "Predictive technology model." [Online]. Available: http://ptm.asu.edu/

**Driss Azougagh** received his B.S. degree in Computer Science in 1995 from Mohamed ben Abdellah University, Fes, Morocco. He received his Master degree in Computer Science in 2002 from Korea Advanced Institute of Science and Technology, Deajeon, Korea. He is a Ph.D. student at the University Hassan II Mohammedia, ENSET Institute. His research is focused on computer architecture. His research interests include (Massively Distributed and Parallel) Computer Architecture and Processing.

**Ahmed Rebbani** received the B.S. degree in Electronics in 1988 the M.S. degree in Applied Electronics in 1992 from the ENSET Institute, Mohammedia, Morocco. He received the DEA diploma in information processing in 1997 from the Ben Msik University of Casablanca MOROCCO. He is now a teacher of computer networks, and researcher at the University Hassan II Mohammedia, ENSET Institute. His research is focused on renewable energy.

**Omar Bouattane** has his Ph.D. degree in 2001 in Parallel Image Processing on Reconfigurable Computing Mesh from the Faculty of Science Ain Chock, CASABLANCA, Morocco. He has published more than 30 research publications and brevets in various National, International conference proceedings and Journals. His research interests include Massively Parallel Architectures, cluster analysis, pattern recognition, image processing and fuzzy logic.