# An Associativity-Agnostic in-Cache Computing Architecture Optimized for Multiplication

Marco Rios, William Simon, Alexandre Levisse, Marina Zapater and David Atienza
ESL, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland

***Abstract*—With the spread of cloud services and Internet of Things concept, there is a popularization of machine learning and artificial intelligence based analytics in our everyday life. However, an efficient deployment of these data-intensive services requires performing computations closer to the edge. In this context, in-cache computing, based on bitline computing, is promising to execute data-intensive algorithms in an energy efficient way by mitigating data movement in the cache hierarchy and exploiting data parallelism. Nevertheless, previous in-cache computing architectures contain serious circuit-level deficiencies (i.e., low bitcell density, data corruption risks, and limited performance), thus report high multiplication latency, which is a key operation for machine learning and deep learning. Moreover, no previous work addresses the issue of way misalignment, strongly constraining data placement not to reduce performance gains. In this work we drastically improve the previously proposed BLADE architecture for in-cache computing to efficiently support multiplication operations by enhancing the local bitline circuitry, enabling associativity-agnostic operations as well as in-place shifting inside local bitline groups. We implemented and simulated the proposed architecture in CMOS 28nm bulk technology from TSMC, validating its functionality and extracting its performance, area, and energy per operation. Then, we designed a behavioral model of the proposed architecture to assess its performance with respect to the latest BLADE architecture. We show a 17.5 and 22% area and energy reduction thanks to the proposed LG optimization. Finally, for 16bits multiplication, we demonstrate 44% cycle count, 47% energy and 41% performances gain versus BLADE and show that 4 embedded shifts is the best trade-off between energy, area and performances.**

## I. Introduction

Deep Neural Networks (DNN) are becoming increasingly complex and compute intensive, while simultaneously becoming more pervasive across all devices, including low power and area constrained devices on the so called "edge" [1].

In this context, energy efficiency as well as design and manufacturing costs become critical, calling for new accelerators and architectural innovations that provide both high efficiency and high scalability while remaining low-cost. Among the explored opportunities, in-memory computing (and particularly in-SRAM computing, or iSC) [2]–[6] appears as a promising solution, as it mitigates data movement in the cache hierarchy (i.e., reducing energy consumption) and enables ultra-wide Single Instruction Multiple Data (SIMD) operations while being compact and not making drastic changes in the architecture design and usage [7].

Current iSC solutions are based on BitLine (BL) computing, a technique consisting of the simultaneous activation of two or more WordLines (WL) in a SRAM array [3]–[8]. This simultaneous WL activation results in the corresponding BLs carrying out the bitwise *AND* and *NOR* operations between the bitcells of the accessed WLs. While bitwise operations can be useful in some cases, around 90% of the operations executed in DNNs are convolutions [9], which require word-level multiplications. Among the existing works, [7], [8] only support bitwise operations, while [3]–[6] propose support for addition (*ADD*) and *SHIFT* which, when chained, can be used to perform multiplication. One of the most promising architectures, BLADE [5], [6], exhibits the best trade-off between density and performance: (i) thanks to its Local Group (LG) organization, it does not require WL underdrive to mitigate data corruption risks [7], [8], and (ii) the use of LGs reduces array density by only 10 to 15%, making it more suitable than 8-10T bitcells arrays [3], [10].
However, among SRAM-based solutions none focus on multiplication optimization specifically, and all strongly rely on compile-time optimizations [7], [11] to deal with data miss alignment (i.e., data sharing different positions in regards with the BL multiplexers must be moved before being computed together).
In this work, we propose to enhance the BLADE architecture to enable associativity-agnostic operations as well as highly efficient multiplication operations.

Overall, the main contributions of this work are:

- We implement local BL multiplexer inside the LG, as opposed to global multiplexer inside the BL logic, enabling area and energy gain at constant performances and enabling associativity-agnostic operations between miss aligned data.
- We implement shift logic inside the LGs (embedded shifting) and assess the area and energy gains of the proposed modifications (17.5% and 22% respectively) through parasitic aware electrical simulations and layout using the 28nm bulk CMOS TSMC technology PDK.
- We propose an innovative multiplication scheme and designed an analytical behavioral model to demonstrate performance under various configurations of the proposed embedded shift logic. We show 44% cycle count, 47% energy and 41% performances gain for 16 bit multiplication compared to BLADE [5].
- We explore the design space of the proposed optimization and we demonstrate that a 4 bit embedded shift architecture provides optimal trade-off between performance, energy consumption and area overhead when performing

in-memory multiplication.

The paper is organized as follows: Section II introduces the concept of bitline computing and how the iSC multiplication is performed in-memory. Section III presents the proposed circuit optimizations within LGs that enable associativity-agnostic operation and enhance multiplication. Section IV details the circuit level validation performed as well as our behavioral model, while Section V analyzes the performances gains of the proposed architecture. Finally, Section VI concludes the paper.

## II. Background and Related Works

### A. In-SRAM Computing

A conventional SRAM read operation precharges the BL pair (BL and $\overline{\text{BL}}$) and activates one WL, in such way that either the BL and $\overline{\text{BL}}$ discharge depending on the data stored in the bitcell. A sense amplifier then issues a logic value for the read operation. This standard operation can be exploited to perform what is called bitline computing, which consists of accessing two WLs simultaneously. The resulting BL discharge results in a pair of logic operations between the bitcells of the activated WLs, namely *AND* and *NOR*. This basic in-SRAM Computing (iSC) logic can then be extended via a variety of methods.

Jeloka *et al.* [8] first demonstrated iSC architectures, which was then further enhanced via the addition of an extra logic gate to support *XOR* ops [7]. Unfortunately, these works are based on 6T SRAM bitcells and suffer from data corruption, due to short circuiting between bitcells when multiple WLs are accessed. Consequently, performance and voltage scaling are limited to secure reliable operation. To perform iSC operations at ultra low voltage, Dong *et al.* [12] rely on an unconventional 4T bitcell design, which suffers from instability and disturb risks. Neural Cache [4] and DRC$^2$ [3] overcome data corruption using 8T and 10T SRAM bitcells respectively resulting in an area overhead of at least 30% [13]. Finally, an intermediate solution, BLADE, using 6T SRAM bitcells and local bitlines was proposed in [5], [6]. BLADE divides the array into LGs, each one contains its own local bitline, suppressing the risk of data corruption and enabling voltage scaling while maintaining a low area overhead.

A common limitation of iSC computing is data miss-alignment regarding the BL multiplexer. In a cache, each multiplexed BL is named a *way*. In order to perform in-situ operations, operands must be aligned along the *ways*, i.e. share the same multiplexed BL as shown Figure 1-a. This is known as operand locality [7]. In this context, operations between miss-aligned data are not possible, leading to complex algorithm modifications and operand migration policies for iSC architectures [7], [11]. In this work, the introduction of local BL multiplexers per LG periphery enables operations between miss-aligned operands (i.e., associativity agnostic operations) when in two different LGs. It is worth to note that the proposed approach stays compatible with parallel TAG data access if included in a cache.

### B. BLADE Architecture

The BLADE architecture proposed in [5], [6] is an iSC architecture designed for low voltage edge devices. It performs bitline computing through an innovative memory array organization featuring LBLs. By ensuring that the operations are always performed between two different LGs, the risk of data corruption when accessing two WLs is eliminated.

Figure 1-a presents the BLADE block schematic. The architecture considers a *n-way* cache structure (0 to n-1 BL multiplexer), these n *ways* share the same BL logic through the Global Read BL (GRBL) Multiplexer. Each way consists of two LG with their private LBLs and peripheral circuitry (Local Group Periphery - LGP). The LGP in turn, shown figure 1-b, consists of the local read and write ports, the precharge circuit, and two Local Sense Amplifiers (LSA).

BLADE computes GRBLs instead of the LBLs. During a read operation, the LBLs are connected to GRBLs through the Local Read Port (LRP). This solution prevents the LBLs to be coupled and short circuiting two bitcells. During a write operation, the write amplifiers are connected to the LBLs through Global Write BL (GWBL) [6] not represented in figure 1-a.

The BLADE BL Logic contains a sense amplifier, a carry ripple adder, an operation multiplexer and a write back circuitry as shown Figure 1-c. It supports bitwise operations such as *NOR*, *XOR* and *AND*. With the addition of *ADD* and *SHIFT*, more complex operations can be performed, such as multiplication, subtraction and greater/less than.

### C. in-SRAM Multiplication and Challenges

Performing iSC multiplication requires the use of complex operations such as *SHIFT* and *ADD*. To this end, previous iSC works have proposed the use of carry ripple adder to enable array level multiplications [3], [5]; further, Simon *et al.* proposed to optimize addition via a Manchester carry chain adder [6]. However, among these works, multiplication itself is marginally studied [4], [5] and the effect of data structure are not discussed.

Multiplication is an operation between the multiplicand *A* and multiplier *B*, with the product *C* achieved through the summation of partial products. Traditionally, these partial products are shifted values of *A*. Therefore, multiplication is achieved by shift-and-adding *A*, according to the bit-values of *B*. When performed via iSC, each *SHIFT* or *ADD* operation requires two cycles, one to access the data and compute on the bitline, and one to Write Back (WB) the result to memory. Moreover, the total cycle count increases with the operand size.

In order to simplify binary multiplication, the iSC multiplication shifts *C* instead of *A*. In each shift/add cycle, the controller inspects one bit $b_n$ in the multiplier *B*, from the most (MSB) to the least significant bit (LSB). First, *C* is left-shifted. Then if $b_n = 1_2$, *A* is accumlated into *C*.

Table I demonstrates this technique via an example multiplication between the multiplicand ($A = 10_{10} = 01010_2$) and the multiplier ($B = 9_{10} = 01001_2$). The total number

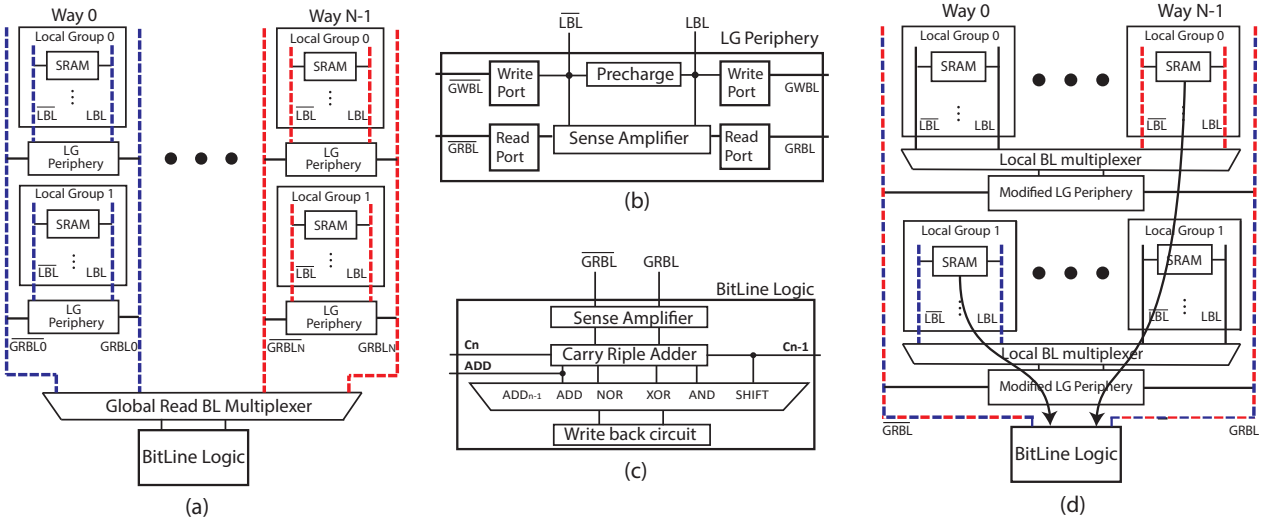


Fig. 1: (a) BLADE memory organization, (b) Local Group Periphery (LGP) block schematic and (c) BitLine Logic block schematic from [5], [6]. (d) Proposed associativity-agnostic memory organization with modified local group periphery and local BL multiplexer

| C - Binary | C - Decimal | $b_n$ | Operation | Cycle |
|---|---|---|---|---|
| 0 | 0 | 0 | Shift (C) | 2 |
| 0 | 0 | 1 | Shift (C) | 4 |
| 1 0 1 0 | 10 | | Add (C,A) | 6 |
| 1 0 1 0 0 | 20 | 0 | Shift (C) | 8 |
| 1 0 1 0 0 0 | 40 | 0 | Shift (C) | 10 |
| 1 0 1 0 0 0 0 | 80 | 1 | Shift (C) | 12 |
| 1 0 1 1 0 1 0 | 90 | | Add (C,A) | 14 |

TABLE I: Comprehensive example of a multiplication between $A = 10_{10}$ and $B = 9_{10}$ with detailed intermediate steps.

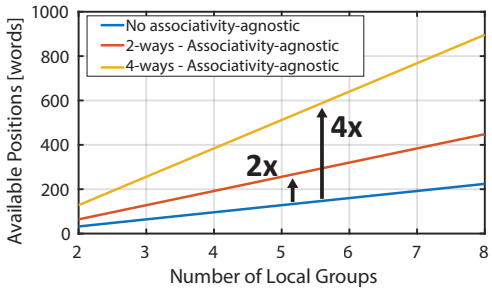


Fig. 2: Available positions for operands (i.e., misalignment mitigation) versus the number of LGs for the proposed and baseline BL computing memories.

of operations for the given example is 7 (2 *ADDs* and 5 $SHIFTs$), therefore resulting in a cycle count of 14.

BLADE [5], [6] integrates an $ADD_{n-1}$ between BL logic blocks in the operation multiplexer, as illustrated in Figure 1-c. This accelerates the multiplication by allowing addition results to be shifted during write back, thus only requiring two cycles per bit in the operand to complete a multiplication.

## III. Proposed Circuit Optimizations

In this section we present first the circuit innovations proposed in this work to enable associativity-agnostic operations. Then, we introduce the concept of embedded shifts inside the LGs and we show how such architecture can accelerate iSC multiplications.

### A. Associativity-Agnostic Local Group

Associativity-agnostic operations simplify the controller at the system level, as well as it mitigates one of the major drawbacks in iSC, namely, data miss-alignment. If we consider a 4-*ways*, 2 LG array with 32WLs per LG, each operand has only 32 potential available operands to which it can be multiplied with (it must be recalled that operands must occupy distinct LGs [5]). By including the BL multiplexer inside the LG, available positions for operands increases by the BL multiplexer width (4× for a 4-*way* cache associativity). More positions can be made available by increasing the number of LGs as shown Figure 2.

In this work, by moving the BL multiplexer from the BL logic to the LGP, all the *ways* share the same LGP. Controlling independently each local BL multiplexer, two different *ways* can couple into the same GRBLs, as depicted in figure 1-d.

This solution reduces the circuit complexity and improves the energy and area efficiency. By reducing the number of GRBLs and GWBLs by $n\times$, the array controller is simplified and the energy efficiency is improved as less GRBL demand less energy during the precharge phase. Furthermore, this work greatly enhances the area efficiency. Whereas BLADE needs $n$ LGP blocks per BL logic, we employ just one. The BL logic area reduces 34% thanks to the read and write multiplexers moved to the LGs.

### B. Efficient Multiplication With Embedded Shift

Thanks to the use of local BL multiplexer, we propose in this section to integrate what we call embedded shifting circuitry in the LG. As shown Figure 3, the output of the

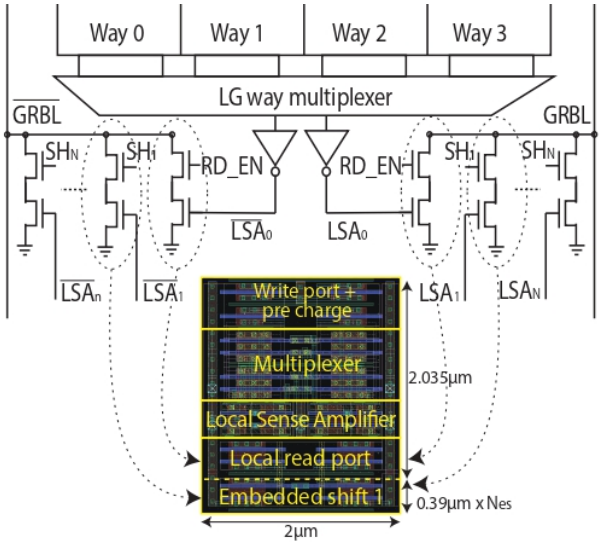


Fig. 3: Local Read Port extended for N embedded shifts. with corresponding circuit layout

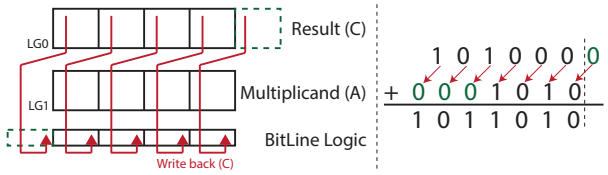


Fig. 4: Shifted addition path inside the memory array and its arithmetic representation.

Local Sense Amplifiers (LSA) are connected to a neighbour Local Read Port (LRP), i.e., to the neighbour GRBL. In this context, from the BL logic point of view, the data is shifted. During an iSC operation between two operands, this feature enables direct shifting of one of the operands without requiring any preliminary *SHIFT + WB* cycle.

We implemented the proposed modifications in a LG. Additional circuitry incurs a small area cost, with each embedded shift increasing the LG layout length by 0.39 $\mu$m when pitched on 2$\mu$m (4 bitcells width).

Table II shows the chain of operations that have to be executed for Embedded Shift Numbers ($N_{ES}$) going from 0 to 4. Column $N_{ES} = 0$ has no shift logic, and therefore the *SHIFT* operation count equals the size of the operands, while the number of *ADD*s to be performed equals the count of '1's in the multiplier. For $N_{ES} = 1$, *ADD* and *SHIFT* can be performed in one cycle by shifting the result inside the LG before performing an iSC operation with the multiplicand, as illustrated in Figure 4. Thus, $N_{ES} = 1$ reduces the operation count to just the number of bits in the operand, matching the performance of BLADE while simplifying the bitline logic.

It is possible to extend $N_{ES}$ to greater values, enabling operands to be shifted by more than one bit at a time. Such an architecture can be used to accelerate multiplication by analyzing the bit pattern of the multiplicand. Specifically, bit patterns with leading 0s can be accelerated. For example, for

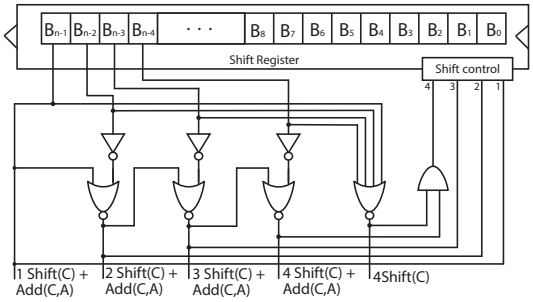


Fig. 5: 4-bits multiplication controller block schematic for n-bits word.

$N_{ES} = 2$, whenever the iSC controller detects a $0_2$ in the most significant bit, a 2 bit shift can be performed as opposed to 1. The two new available operations are:

1) $B_n = 00 \rightarrow 2 \times$ Shift(C) + WB(C).
2) $B_n = 01 \rightarrow 2 \times$ Shift(C) + Add(C,A) + WB(C).

For $N_{ES} = 3$, the patterns that are accelerated are:

1) $B_n = 000 \rightarrow 3 \times$ Shift(C) + WB(C).
2) $B_n = 001 \rightarrow 3 \times$ Shift(C) + Add(C,A) + WB(C).

So, for each LG shifting, the number of concatenated *SHIFT*s that is performed simultaneously increases. But as these occurrences are statistically less common, the gain saturates. For the given example, the operations executed for $N_{ES} = 2$ and 3 are 3 and 2, respectively. Representing 71% cycles count reduction in comparison with no embedded shift.

### C. Multiplication Controller

In order to perform data dependent operations, a dedicated control logic block must be considered. In this work we propose a multiplication controller and we assume that such logic is included inside the sub-array controller.

Figure 5 presents the block schematic of a $N_{ES} = 4$ controller and it can be used for a n-bits word size. The circuit is composed of one shift register and logic gates. Before any multiplication, as described in [5], one of the operands (ideally the one containing the least '1's) is stored inside the shift register. Then, at each cycle of the multiplication, depending on the MSBs (4 last bits in the case of a $N_{ES} = 4$ controller), the operation to be performed is calculated and processed as iSC operations in the memory. Then, before the next multiplication cycle, the register is shifted depending on the word structure. As an example, in Figure 5, if $B_{15}, B_{14}, B_{13}, B_{12} = 0000_2$, the issued operations are 4 shifts and then B stored in the shift register is shifted 4 times. It must be noted that here, we assume only one multiplication per subarray.

## IV. Experimental Setup

### A. Electrical validation

To validate electrically this work, and assess its performance, energy consumption and area efficiency, we implement a 256×64 (32WL per LG) SRAM array using a 28nm CMOS bulk technology PDK from TSMC, simulated at 1V. We follow a design methodology equivalent to [6] and only simulate the

| Result Vector (C) | $b_n$ | $N_{ES}=0$ | $N_{ES}=1$ | $N_{ES}=2$ | $N_{ES}=3$ |
|---|---|---|---|---|---|
| 0 | 0 | Shift (C) | Shift (C) | 2×Shift (C) + Add(C,A) | 2×Shift (C) + Add(C,A) |
| 0<br>1 0 1 0 | 1 | Shift (C)<br>Add (C,A) | Shift (C) + Add(C,A) | | |
| 1 0 1 0 0 | 0 | Shift (C) | Shift (C) | 2×Shift (C) | 3×Shift (C) + Add(C,A) |
| 1 0 1 0 0 0 | 0 | Shift (C) | Shift (C) | | |
| 1 0 1 0 0 0 0<br>1 0 1 1 0 1 0 | 1 | Shift (C)<br>Add (C,A) | Shift (C) + Add(C,A) | Shift (C) + Add(C,A) | |

TABLE II: Extended multiplication example, with the operations performed by the controller for $N_{ES}$ going from 0 to 4.

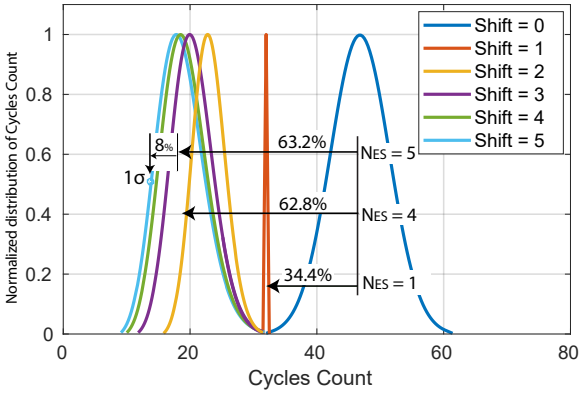


Fig. 6: Cycle count distribution of 16bits multiplications.

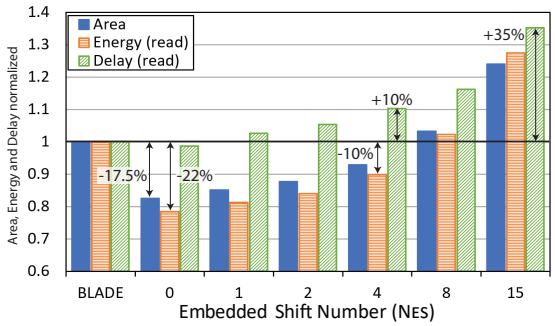


Fig. 7: Area overhead, read delay and energy evolution of the proposed work normalized with BLADE (lower is better).

critical paths with equivalent parasitics and gates to optimize the design and simulation time. In order to compare the proposed circuit optimizations to BLADE, we layout the LG as shown Figure 3.

### B. Behavioral model

To evaluate the performances of the proposed multiplication scheme, we designed an analytical behavioral model of the memory and multiplication controller. For a given set of parameters ($N_{ES}$, word length, multiplier value), the model calculates the amount of cycles required to perform the multiplication. We then extract statistical data considering all the possible values (i.e. from 0 to $2^{16}-1$ for 16 bits operands) to assess the cycles count distribution. Normalized representation are shown Figure 6.

For $N_{ES}=0$, each bit, if it is a '1', takes 4 cycles (*SHIFT, WB, ADD, WB*) while if it is a '0', it only takes 2 (*SHIFT, WB*). In the end, the distribution spans from 32 to 64 cycles.

For $N_{ES}=1$, the number of operations is equal to the size of the operand, regardless of the data structure, thus the distribution is concentrated in 32 cycles.

For $N_{ES}$ higher than 2, the right tail of the distribution always equals 32 cycles, representing the worst case of multiplier ($B=2^{16}-1$) when all bits are 1. The average, however, decreases accordingly to higher values of embedded shift. For each successive embedded shift, the accelerated patterns becomes more rare, lessening the potential gain. The difference between the average cycles count for $N_{ES}$ = 4 and 5 is less than half cycle.

While the average gain for $N_{ES}>2$ exceed 60%, it must be noted that the computation time is highly data dependant. For a neural network execution, forcing the weights to respect a given sparsity may enable additional gains with reduced performances drop. As a reference, for $N_{ES}=5$, considering a data structure where the accumulated cycle count of several multiplications shifts from the average to one sigma left, it represents an extra gain of 8%.

## V. Performance Results

In this section we analyze the effect of the proposed architecture modifications at both circuit and system level and combine them to gain a global picture of the enhancement. First, we perform an electrical characterization to assess the energy and delay gain of the operations in function of $N_{ES}$. Then, we combine these results with the full multiplication cycle count extracted from Section III-B. Finally, we identify area/energy/performances optimums.

### A. Electrical Characterization and Area Estimations

Figure 7 shows the specifications (area,. delay, energy) of the proposed solution compared to BLADE. By attaching more transistors in the LRP, the overall parasitic capacitance of the GRBL and of the LSA output increases, rising the energy consumption and reducing the performances. Thanks to the GRBL optimization enabled by the use of local BL multiplexer, this effect is compensated. Finally, for $N_{ES}=0$, the energy is reduced by 22% for one operation. Beyond $N_{ES}=7$, the energy consumption overcomes BLADE.

Concerning the read delay, for $N_{ES}=0$, the delay is similar to BLADE, less than 2% reduction. For higher values of $N_{ES}$, however, the value surpass BLADE because the path covered by the signal is longer, each embedded shift increases

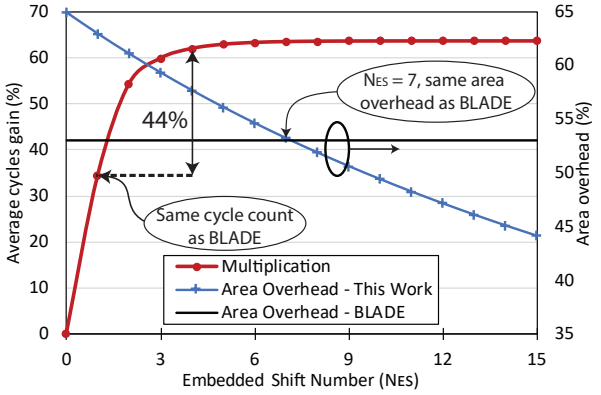


Fig. 8: Cycle gain and area overhead per embedded shift

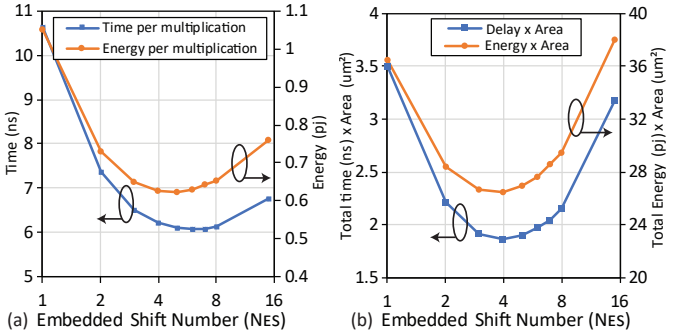


Fig. 9: Total energy and delay for multiplication performed under different values of embedded shift.

in $2\mu$m the signal propagation distance (it could be noted that increasing the LSA drive may mitigate this effect while increasing the LG area). The read delay is 10% higher for $N_{ES} = 4$ and it exceeds 35% for $N_{ES} > 15$.

Finally, we extract the corresponding area from the layout (as shown Figure 3) and we show a 17% density improvement for $N_{ES} = 0$. It must be noted that the proposed solution becomes larger than BLADE for $N_{ES} > 7$.

### B. System Level Assessment

Figure 8 shows the average cycles gain and area overhead in function of the embedded shift number. For the same multiplication performance as BLADE, ($N_{ES} = 1$), we show a 9% area overhead reduction. On the other hand, Figure 8 also showcase a gain saturation that can be explained by the fact that after a point, more embedded shifts only accelerate a marginal portion of the words possibilities, i.e., saturating the average gain. Compared to BLADE, we show 44% cycle count reduction.

Overall, we show several non-aligned trends: (i) the average multiplication performances gain (i.e., cycle count) tends to saturate with $N_{ES}$. (ii) The area overhead and energy are beneficial for a low amount of shifts (i.e., less than 7) but becomes disadvantageous beyond. (iii) The operation delay degrades with the number of embedded shifts.

Figure 9-a shows the time spent in $ns$ (blue) and the energy consumed in pJ (orange) to perform a complete 16bits multiplication (considering *ADD*, *SHIFT* and *WB* time and energy). The optimum values for energy and time are $N_{ES} = 5$ and 7, respectively. On the other hand, as shown Figure 9-b, when including area considerations (i.e., multiplying the time and energy by the total memory area), we show a Time$\times$Area and Energy$\times$Area optimum for $N_{ES} = 4$. Compared to BLADE, a $N_{ES} = 4$ enables 47% and 41% of average performance gain for energy and delay, respectively. Moreover, the area overhead is reduced by 4%.

## VI. Conclusion

In this work, we proposed circuit optimizations for iSC architectures enabling highly efficient associativity-agnostic multiplication. We apply these optimizations to the BLADE iSC architecture as it appears to be the most promising reported solution in terms of operation voltage range, performances, reliability and density. By including the BL multiplexer and adding embedded shifts inside the LGs, we show 44% cycle count, 47% energy and 41% performances gain for 16bits multiplication and show that multiplication can be highly accelerated if data structure is optimized. Finally, we show that the best area/energy/performances trade-off appears for 4 embedded shifts.


## Acknowledgements

This work has been supported by the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).


## References


[1] B. Reese. Ai at the edge: A gigaom research byte. *GigaOm*, 2019.

[2] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky. Efficient algorithms for in-memory fixed point multiplication using magic. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[3] K.-C. Akyel et al. DRC2: Dynamically reconfigurable computing circuit based on memory architecture. In *IEEE ICRC*, 2016.

[4] C. Eckert et al. Neural cache: Bit-serial in-cache acceleration of deep neural networks. *CoRR*, 2018.

[5] A.-W. Simon et al. Blade: Bitline accelerator for devices of the edge. *ACM GLSVLSI*, 2019.

[6] A.-W. Simon et al. A fast, reliable and wide-voltage-range in-memory computing architecture. *IEEE/ACM DAC*, 2019.

[7] S. Aga et al. Compute caches. In *HPCA*, 2017.

[8] S. Jeloka et al. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory. *IEEE JSSC*, 2016.

[9] M. Chang et al. Hardware accelerator for boosting convolution computation in image classification applications. In *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, 2017.

[10] A. Agrawal et al. X-sram: Enabling in-memory boolean computations in cmos static random access memories. *Trans. Circuits Syst. I*, 2018.

[11] M. Kooli et al. Smart instruction codes for in-memory computing architectures compatible with standard sram interfaces. *IEEE/ACM DATE*, 2018.

[12] Qing Dong et al. A 0.3 v vddmin 4+ 2t sram for searching and in-memory computing using 55nm ddc technology. In *2017 Symp. on VLSI Circ.* IEEE, 2017.

[13] L. Chang et al. A 5.3ghz 8t-sram with operation down to 0.41v in 65nm cmos. In *VLSI Symp.*, 2007.