Published in *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1349-1363 which should be cited to refer to this work.
DOI: 10.1109/TC.2020.2972528

BLADE: An in-Cache Computing Architecture for Edge Devices

William Andrew Simon, *Student Member, IEEE,* Yasir Mahmood Qureshi, *Student Member, IEEE,* Marco Rios, *Student Member, IEEE,* Alexandre Levisse, *Member, IEEE,* Marina Zapater, *Member, IEEE,* and David Atienza, *Fellow, IEEE*

Abstract—
Area and power constrained edge devices are increasingly utilized to perform compute intensive workloads, necessitating increasingly area and power efficient accelerators. In this context, in-SRAM computing performs hundreds of parallel operations on spatially local data common in many emerging workloads, while reducing power consumption due to data movement. However, in-SRAM computing faces many challenges, including integration into the existing architecture, arithmetic operation support, data corruption at high operating frequencies, inability to run at low voltages, and low area density. To meet these challenges, this work introduces BLADE, a BitLine Accelerator for Devices on the Edge. BLADE is an in-SRAM computing architecture that utilizes local wordline groups to perform computations at a frequency 2.8x higher than state-of-the-art in-SRAM computing architectures. BLADE is integrated into the cache hierarchy of low-voltage edge devices, and simulated and benchmarked at the transistor, architecture, and software abstraction levels. Experimental results demonstrate performance/energy gains over an equivalent NEON accelerated processor for a variety of edge device workloads, namely, cryptography (4x performance gain/6x energy reduction), video encoding (6x/2x), and convolutional neural networks (3x/1.5x), while maintaining the highest frequency/energy ratio (up to 2.2Ghz@1V) of any conventional in-SRAM computing architecture, and a low area overhead of less than 8%.

Index Terms—In-memory computing, in-SRAM computing, bitline computing, edge computing.

1 Introduction

As edge devices are being increasingly utilized to perform compute intensive tasks traditionally reserved for servers with access to standard accelerators such as GPUs [1], [2], a wide range of compute architectures that are specially suited for edge devices' area and power constraints have been proposed. One such innovation is the concept of near or in-memory computing [3], [4], which improves performance by computing at the point of data storage. Such architectures are placed near DRAM, within the cache hierarchy, or as separate accelerator units with dedicated memory arrays. In particular, in-SRAM Computing (iSC) shows promise by performing massive SIMD operations at a small area/energy cost [5]. Many iSC architectures have been proposed [6], [7], [8], with each demonstrating different aspects of the technology. However, each work also suffers from various shortcomings in relation to simulation methodology, electrical design, or application support. In particular, many of the works lack thorough analysis of system level integration implications, as well as demonstration of functionality within a full software stack. Also, data corruption within the SRAM array is a prominent problem for iSC architectures [3], [5], and the solutions proposed so far within literature either greatly reduce operating frequency [3] or area efficiency [6]. Finally, many proposed architectures do not support arithmetic operations such as addition, subtraction, or multiplication, and are therefore limited in accelerating arithmetic workloads [5], [6], [9].

Within this context, we present a BitLine Accelerator for Devices on the Edge (BLADE). BLADE is an iSC architecture targeted specifically for implementation in low-power edge devices. It performs massive SIMD bitwise and arithmetic computations required by emerging edge device workloads directly in the cache hierarchy, obviating the need for costly data movement or time-consuming CPU computation. BLADE addresses each of the previously described shortcomings prevalent in other iSC architectures. Design choices are motivated from the transistor, architectural, and system levels in order to demonstrate the architecture's energy and performance characteristics at all levels of abstraction. BLADE divides wordlines into isolated subgroups called local groups, eliminating the risk of data corruption, and utilizes a carry lookahead adder and operation pipelining to improve iSC operating frequency by 2.3x-2.8x compared to previous iSC architectures, while maintaining a low (8%) area overhead and functioning at the lowest operating voltage (0.6V) of any 6T bitcell iSC architecture. We integrate BLADE into an edge device cache hierarchy and benchmark it within a fully functioning Linux environment, enabling consideration of system level events such as cache misses, coherence requests, and CPU/cache hierarchy pipeline stalls.

- *The authors are with the Embedded Systems Laboratory, Swiss Federal Institute of Technology, Route Cantonale, 1008 Lausanne. E-mail: {william.simon, yasir.qureshi, marco.rios, alexandre.levisse, marina.zapater, david.atienza}@epfl.ch*

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 00.0000/TC.0000.0000000

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

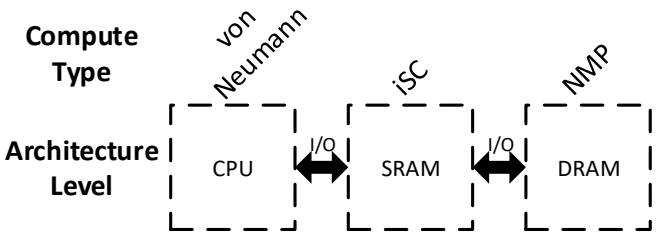


Fig. 1. Computing architecture showing the locations of in-SRAM and near memory computing extensions.

We therefore present the contributions of this paper as follows:

- We introduce BLADE, a holistically designed and simulated iSC architecture capable of arithmetic operations, designed specifically for edge devices.
- We utilize local bitlines, operation pipelining, and carry-lookahead addition to achieve the best voltage/frequency Pareto curve of any 6T iSC architecture (0.6V/416MHz-1V/2.2GHz for bitwise operations), validating our design in 28nm CMOS technology.
- We situate BLADE within the cache hierarchy, explain BLADE controller functionality, and propose Instruction Set Architecture (ISA) extensions that enable application support.
- We benchmark BLADE on the gem5-X architectural simulator using three edge device workloads, namely, cryptography, image processing, and neural networks, demonstrating 4x/6x, 6x/2x, and 3x/1.5x performance/energy gains respectively.

The remainder of the paper is organized as follows. Section 2 provides background on the design and challenges of iSC architectures. Section 3 details the subarray optimizations allowing BLADE to run at high frequency with a low energy consumption while avoiding data corruption. Section 4 explains how arithmetic operations are supported. Section 5 details BLADE's integration in the cache hierarchy and its interaction with the rest of the architecture. Section 6 provides information on BLADE's electrical validation as well as subarray design space exploration results. Section 7 details the methodology for evaluating BLADE at the system level. Section 8 illustrates our benchmark application results. Finally, Section 9 concludes this work.

2 Background and Challenges

When making design decisions for any novel architecture, balancing many competing factors is necessary, including functionality and generalizability, integration and implementation details, and area and energy costs. In relation to near and in-memory computing architectures, three such factors must be asked when considering the benefits and trade-offs of such architectures, namely, where to compute, what type of computation to be performed, and what challenges do architects face in performing said computation.

2.1 Where to compute: von Neumann vs. iSC vs. NMP

Traditional computer architectures utilize the well-known von Neumann architecture [10], which simplifies design and enables modularity by considering the CPU, memory hierarchy, and any other blocks or accelerators the architect includes as discrete blocks, connected by a bus. However, as CPU speed and memory size have both evolved much faster in relation to available interconnect bandwidth, the von Neumann architecture has increasingly faced challenges with data transfer, known as the von Neumann bottleneck [11]. Thus, two new fields of research have opened that attempt to alleviate this bottleneck, namely, Near Memory Processing (NMP) and in-SRAM Computing (iSC). The locations of these two new fields within the traditional von Neumann architecture are illustrated in Figure 1.

2.1.1 Near Memory Processing (NMP)

Near Memory Processing (NMP) is the practice of placing compute logic near memory, generally DRAM, in an effort to decrease access time [4]. Many architectures allocate compute blocks on the logic layer of HMC DRAM [12], [13], [14], [15]. Other works couple GPU architectures with 3D stacked memories [16], [17]. Still others utilize reconfigurable logic near the DRAM [18], [19], [20].

While NMP computing shows promise, there is still much research to be done to validate its feasibility. First, from an integration standpoint, NMP poses a challenge in regards to virtual-physical memory translation and managing cache coherency [4]. Second, from an application viewpoint, few works provide potential solutions for optimizing algorithms to utilize NMP units while accounting for data locality [17], [21]. Finally, the HMC logic layer area/power budget is very constrained, thus limiting NMP logic complexity [13].

2.1.2 in-SRAM Computing (iSC)

In contrast to NMP architectures, iSC architectures allocate compute logic immediately adjacent to SRAM bitcell arrays, either in dedicated iSC accelerators or in the preexisting cache of the memory hierarchy. iSC architectures take advantage of the SRAM array's BitLine/WordLine (BL/WL) structure to perform massive numbers of computations in a SIMD-like manner [5]. These architectures exploit the inherent data locality found in many applications to perform these operations, similar to other SIMD accelerators such as ARM's NEON [22] and Intel's AVX [23] architectures. Further, when integrated into a cache hierarchy, iSC architectures reduce energy consumption by (a) taking advantage of the cache's set/way allocation scheme to align data and avoid unnecessary data movement, as explained in Section 5.2, and (b) reducing data movement between the cache and the CPU. This is accomplished at a minimal area overhead, as the iSC architecture augments preexisting hardware. As BLADE is an iSC architecture, the scope of this paper will be limited to such architectures, and this work's contributions to the state of the art in this field.

2.2 What to compute: Simple vs. Arithmetic Operations

Choosing which operations to support in iSC architectures is not trivial, as there is a complex interrelationship between operation complexity, latency, throughput, and area overhead. Many iSC architectures support only bitwise operations, through a technique known as bitline computing, first demonstrated by Jeloka *et al.* [3], and illustrated in Figure 2.

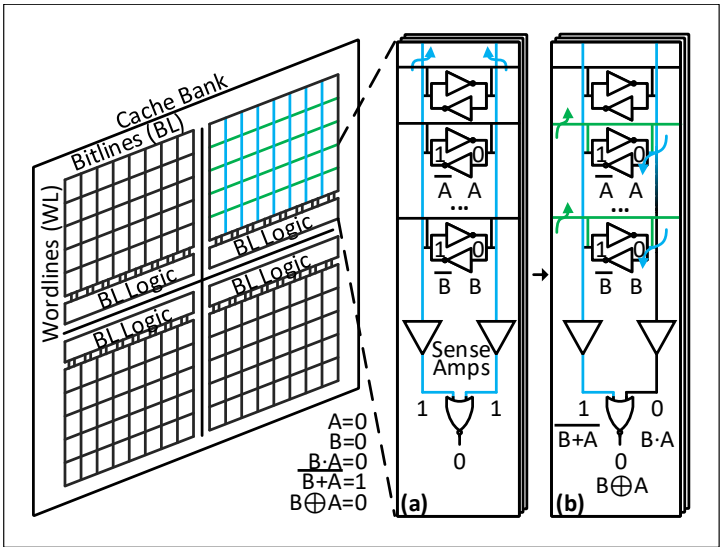


Fig. 2. Cache subarray with AND/NOR/XOR bitline computing on values A=0 and B=0. Bitwise operations are performed by first (a) precharging the bitlines, then (b) activating multiple wordlines, thus discharging the bitlines through the connected bitcells.

Bitline computing operates similarly to a standard cache read operation, which involves precharging a pair of BLs for each bitcell to be read (Figure 2-a), and then activating a single WL to connect the row of bitcells to the BLs, resulting in BL discharge according to the bitcell contents. In contrast, bitline computing involves the simultaneous activation of multiple WLs (Figure 2-b), with the resulting BL discharge computing two bitwise operations between the bits, with the BL producing an `and` operation and the BL Bar ($\overline{BL}$) producing an `nor` operation. Aga *et al.* extend this work in their iSC architecture by `nor`ing the two BLs, resulting in a `xor` operation [5]. This architecture forms the basis of many iSC publications [6], [9], [24], [25], [26], [27].

Conversely, multiple works support more complex workloads by implementing BL logic that performs arithmetic operations. Analog solutions such as those presented in [7], [28], [29], [30], [31] modulate BL and/or WL voltages to perform arithmetic operations and utilize analog circuitry implemented under the subarray to sense and convert BL voltages to approximate digital results. On the other hand, solutions such as [8], [32], [33] implement digital logic to compute exact solutions to arithmetic operations.

2.3 How to compute: iSC Architecture Challenges

The above iSC architectures have been successfully applied to a wide range of applications including query processing and in-memory checkpointing [5], cryptography [5], [24], [34], neural networks [9], [28], [29], [30], [31], [33], [34], finite state automaton [6], and video encoding [34], thus demonstrating their performance and energy benefits. However, iSC architectures face a variety of challenges at the system, architectural, and electrical level that must be overcome for such architectures to be feasible. These challenges include:

- Guaranteeing correct cache functionality at the electrical level.
- Supporting arithmetic operations such as addition and multiplication.
- Integration into the memory hierarchy while addressing issues such as memory translation, coherency, etc.
- Accurate benchmarking on a full software stack.

In the state-of-the-art literature, no single architecture succeeds in overcoming all of the aforementioned challenges. Through BLADE, we seek to address each challenge above via solutions that are validated at each level of abstraction through electrical design and simulation, architectural integration, and application level benchmarking.

3 Avoid Data Corruption while Maintaining High Operating Frequency

One of the biggest challenges in iSC architecture design is the avoidance of data corruption. When activating multiple WLs in a conventional 6T SRAM array, a short is produced between the activated bits. It is possible that, due to process variation during fabrication, the runtime content of the cells, or transistor aging and degradation, the contents of one bit can cause another bit to flip, as illustrated in Figure 3-a. Preventing data corruption has been achieved in previous literature via multiple methods. The first is to aggressively limit WL voltage, as done in [3], [5], [29], [32] and illustrated in Figure 3-b. However, such a technique greatly reduces operating frequency, reported to at 800MHz@1V [3] and 475MHz@1.1V [33]. Another method, demonstrated in Figure 3-c, is the introduction of 8T or larger bitcells, which isolate the bitcell's contents from the BLs [6], [8], [24], [31], at the cost of a significantly less area efficient SRAM subarray as 8T bitcells are up to 30% larger over 6T [35]. A third method is to use pulse width modulated WLs such that no two WLs are active simultaneously [28], removing the danger of data corruption, but at the cost of a 2.35x increase in periphery area. Finally, nonconventional technologies can be used, such as monolithic 3D integration [25], [26], or deeply depleted channel technology [36]. Emerging technologies present their own challenges however; for example, DDC technology demonstrates stability issues [37] and disturb risks, and results in poor performance/voltage scaling (100Mhz@0.6V.)

In this work, we present a novel iSC architecture that utilizes local bitlines to maintain a high operating frequency and low area overhead, avoid data corruption, and facilitate simple implementation in conventional 6T SRAM arrays.

3.1 What are Local Bitlines?

Local BitLines (LBLs), illustrated in Figure 3-d, are a cache optimization present in many architectures. LBLs divide groups of WLs into Local Groups (LGs), where all WLs in an LG share an LBL pair, which is connected to the Global BL (GBL) pair via I/O circuitry. LBLs reduce parasitic capacitance resulting from excessively long GBLs, thus improving cache energy efficiency and reducing delay. LBLs also improve static noise margin by isolating bitcells into small groups, reducing leakage interference. For these reasons, LBLs are already implemented in caches [38].

3.2 BLADE Methods of Operation

BLADE re-utilizes LBLs, taking advantage of the isolation provided to perform high frequency iSC operations reliably. A BLADE enhanced cache can perform three sets of memory operations, namely, standard read/write, slow same-LBL iSC operations, and fast multi-LBL iSC operations.

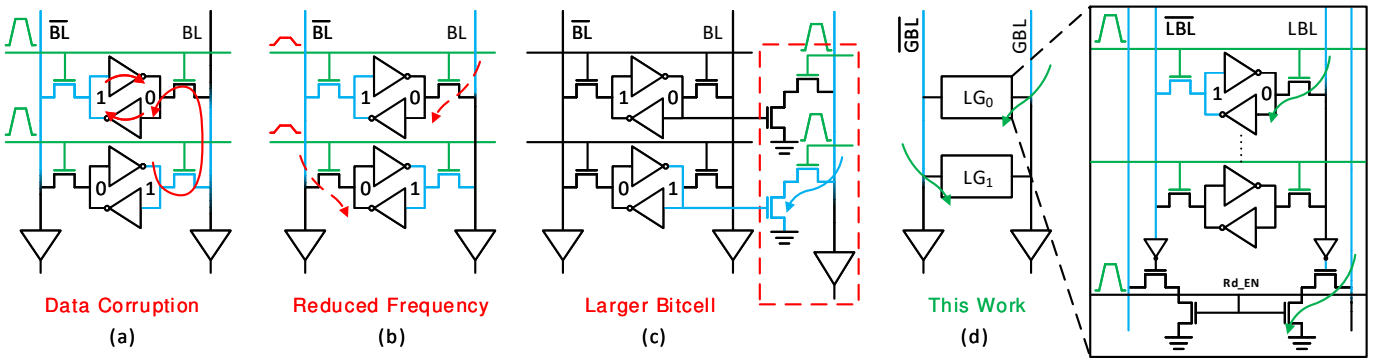


Fig. 3. Bitcell shorting may lead to one of the activated bitcells to flip, as demonstrated in (a). (b) avoids data corruption by lowering WL voltage, while (c) utilizes 8T bitcells to isolate bitcell contents. We propose Local Groups (LGs) to prevent data corruption (d).

3.2.1 Standard Read/Write Operations

Accessing a single WL of the cache performs a standard read or write operation. In order to perform a read operation, the GBLs and the LBLs of the LG to which the target set belongs are first precharged to V_{dd}. Then, when the WL is activated, one of the LBLs discharges. This discharge is propagated through the local read port to the GBLs (illustrated in Figure 3-d), where the discharge is sensed by one of the two single ended sense amplifiers attached to the GBLs. The addition of local read ports in fact transforms the standard 6T bitcells into pseudo two-port bitcells, in which simultaneous read + write operations are possible if the accessed words are located in different LBLs.

3.2.2 Standard iSC Operations

BLADE can also perform iSC operations in the manner described in Section 2.2. However, when performing operations between bitcells in opposing states, there is a risk of flipping a bitcell if the PMOS transistor of one bitcell is weaker than the access and pulldown transistors of the other bitcell, as shown Figure 3-a. To counteract this problem, it is necessary to greatly reduce the WL voltages, resulting in a low operating frequency (<1GHz) [3], [5].

3.2.3 LBL Enhanced iSC Operations

In order to avoid the aforementioned bitcell flipping while maintaining high operating frequency, BLADE reuses LBLs and their resulting LGs. When WLs belonging to different LGs are simultaneously activated, the bitcells are isolated from each other by the local read ports, eliminating the risk of bit flipping, as demonstrated in Figure 3-d. Activating each WL results in a simple read operation within the local group, which is then propagated to the GBLs by the local read ports, resulting in an iSC operation between the local groups, where the GBL represents an `and` operation, while the $\overline{\text{GBL}}$ represents a `nor`. LGs allow BLADE to maintain a high operating frequency without a reduction in WL voltage, while still utilizing small 6T bitcells.

LBLs also enable iSC operation in low voltage environments. Figure 4 illustrates the difference in functionality between (a) standard iSC operations and (b) LBL enhanced operations at a V_{dd} of 0.6V. LBL enhanced operations complete 4x faster than standard operations, and in fact the `and` operation fails to converge, due to excessive BL leakage.

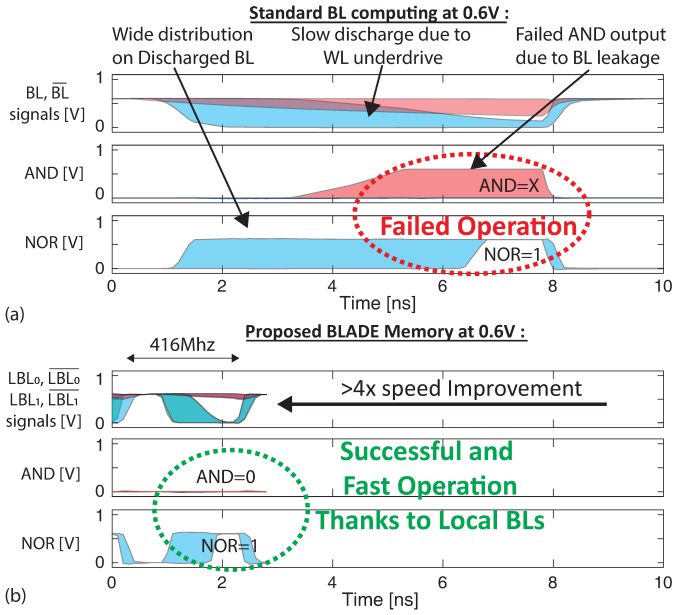


Fig. 4. Transient simulations of bitwise operations @0.6V on (a) previously published architectures, and (b) the proposed BLADE architecture.

It should be noted that introducing LBLs induce a data placement constraint, namely, that operands cannot share the same LBL, as discussed in Section 3.4.

3.3 LBL Enhanced iSC Cache Design Advantages

Utilizing LBLs to enable iSC operations provides multiple advantages over state-of-the-art iSC architectures. As discussed in Section 2, most iSC architectures require a significant redesign of either the bitcell array or periphery to enable computation. BLADE, on the other hand, introduces minimal changes to the cache architecture. The LBL design, as well as the use of digital computation in contrast to analog, allow BLADE to function over a large voltage range, thus easing implementation across a wide range of cache architectures and making BLADE suitable for low power edge devices. Ease of implementation is further facilitated by the fact that no modification to the bitcell array organization is necessary to implement BLADE, as all computation happens within the periphery. This greatly simplifies integration into existing SRAM fabrication design flows, where the most

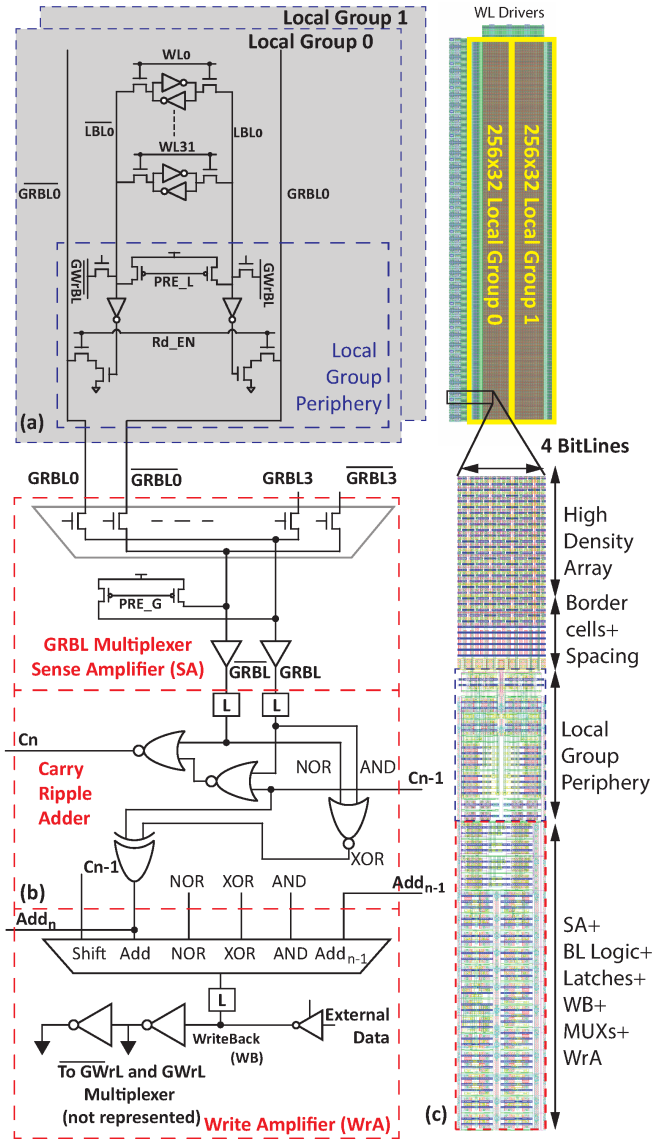


Fig. 5. Layout of a 256×2×32 BLADE memory with schematic diagram of the memory periphery.

aggressive and complex design steps relate to the bitcell array. Finally, BLADE scales easily to larger cache arrays/subarrays without major loss of performance or energy savings, as LBL length is invariant with regard to subarray size, as will be discussed in Section 6.2.

4 Logic to Support Arithmetic Operations

While bitwise operations enabled by simple bitline logic may be sufficient to support applications such as cryptography or binary neural networks, more complex workloads necessitate arithmetic operations.

As discussed in Section 2.2, iSC architectures can perform either simple bitwise operations or arithmetic operations such as addition and multiplication. Supporting arithmetic operations introduces trade-offs in area overhead and latency as the BL logic becomes more complex. Carry logic may have to cross multiple BLs [8]. Alternatively, unorthodox methods of computation may be introduced, such as bit serial computation as in [32], which stores and operates on data in

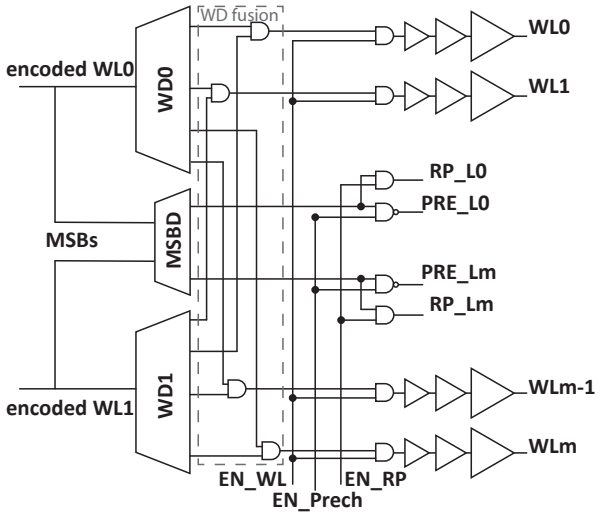


Fig. 6. Block schematic of the modified WL decoder used in this work.

a transposed manner, limiting extra BL logic to a couple of latches, while also greatly increasing throughput. However, latency is also drastically increased, even quadratically for multiplication. Further, transposition presents challenges, requiring either extra hardware in the form of transposition functional units in the cache controller, 8T bitcells with extra BLs/WLs [33], which decreases subarray area efficiency as discussed in Section 3, or software transposition, introducing challenging programming complexity. For analog solutions, area overhead is a significant limitation, with reported overheads of 19% [28] and >30% [31], and the acknowledgement that WL DACs double array periphery [29]. Analog designs also perform inexact computations, which may be unacceptable for high precision applications, limiting generalizability, and are very susceptible to process variation, temperature, and aging [30].

To accelerate the widest range of workloads possible, BLADE supports common arithmetic operations, such as `addition`, `subtraction`, `multiplication`, `greater/less than`, and `shift`. This is accomplished via digital logic, guaranteeing exactness of calculations, necessary for high precision workloads. We accomplish this by augmenting the standard BL logic explained in Section 2.2 with carry and shift logic.

4.1 Bitline Addition Architecture

Figure 5 illustrates BLADE's BL logic, as well as the transistor layout of a 4 way, 256x64 bitcell SRAM array with two local groups, each containing 32 WLs. In order to support addition, we implement a carry ripple adder underneath the array through the addition of two `nor` gates and a `xor` gate. Shift latches are also implemented within the BLADE controller to allow one cycle shifting. Each BL logic block receives a carry-in from the previous BL logic block, and provides a carry-out to the next block.

In order to drive multiple WLs as required by iSC operations, we utilize two WL decoders, which simultaneously decode two WL addresses. A bitwise `and` is performed on the decoded addresses before driving the WLs, as illustrated in Figure 6.

Algorithm 1 Modified add-and-shift multiplication for iSC computing in edge devices with BLADE.

Input: Op_0: Multiplier, Op_1: Multiplicand
Output: Res=$Op_0 \times Op_1$

1: Latch Op_0
2: $i = \#Bits_{Op_0} - 1$
3: **while** $i \geq 0$ **do**
4: $\quad Res << 1$
5: $\quad tmp = Op_1 + Res$
6: $\quad$ **if** $Op_0[i] = 1$ **then**
7: $\quad\quad Res \leftarrow tmp$
8: $\quad$ **else**
9: $\quad\quad Res \leftarrow Res$
10: $\quad$ **end if**
11: $\quad i - -$
12: **end while**

The implementation of addition logic enables many arithmetic operations, achievable through series of simple operations. For example, subtraction can be performed by negating the subtrahend by reading and storing the $\overline{GBL}$ value, then performing an addition between the operands with the first carry-in value set to 1. Greater than/less than can similarly be computed by subtracting the operands and sensing the MSB. Multiplication can be performed via a variation of the classic add and shift algorithm, as illustrated in Algorithm 1, in which the product is shifted left each step (line 4) and the multiplier is processed from MSB to LSB to evaluate which partial products must be accumulated(lines 6-9).

As memory I/O (sense amplifiers and writeback logic) and BL logic are pitched under the memory array, cache associativity influences physical design layout. During layout we found that a mux-4 array (4 way associative cache, $2\mu m$ per 4 multiplexed BLs) enables the most efficient design for the I/O and BL logic. However, as the BL logic is pitched on $1\mu m$, it is also possible to pitch the BL logic in a mux-2 ratio, allowing twice the number of iSC computations per cycle, in exchange for approximately a 2.5x area overhead increase (nonlinear due to increased interconnect complexity). The system level implications of such a configuration are explored in Section 8.

4.2 Improving Operation Throughput

When performing arithmetic operations at longer word lengths, (e.g. 32/64 bit), the performance gains of BLADE are significantly mitigated by both the delay of the ripple carry adder, as well as (in the case of multiplication) the add and shift strategy, as demonstrated in Figure 7-a. To solve this problem, we introduce two optimizations to BLADE's design; (1) multiplication stage pipelining to reduce latency, and (2) a Manchester Carry Chain adder implementation to reduce the addition critical path.

4.2.1 Manchester Carry Chain

At longer word lengths, the critical path of the ripple carry adder substantially outpaces cache access times, reducing pipeline effectiveness. We therefore propose a fast carry adder based on a dynamic Manchester Carry Chain (MCC) adder [39] implemented in buffered 4-bits configuration,

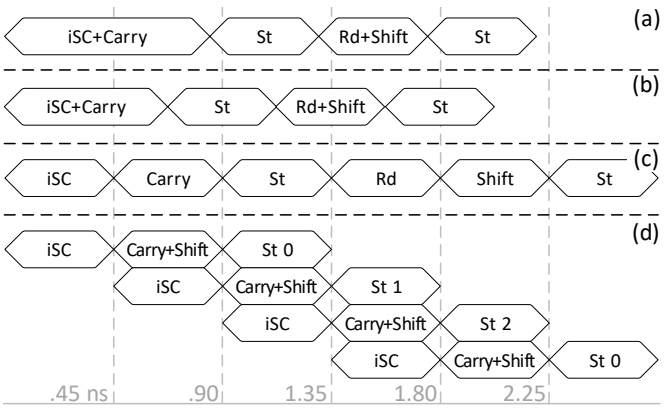


Fig. 7. Timing of multiplication cycle with (a) no optimization, (b) Manchester Carry Chain, (c) pipeline latches, and (d) add-forward line.

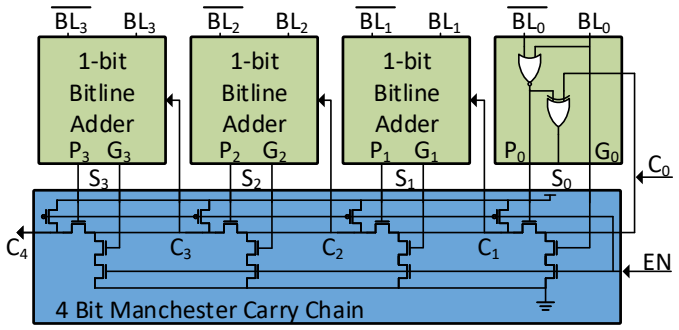


Fig. 8. Manchester carry chain architecture with reduced transistor count due to simplified Generate and Propagate signal generation.

illustrated in Figure 8. The $Generate_{0:3}$ and $Propagate_{0:3}$ signals are simply generated with a single *nor* gate thanks to bitline computing, greatly reducing the area overhead typically associated with such an architecture.

Four MCC blocks are needed per 16 bitcell columns, as columns are mux-4 multiplexed, with the remaining space used to fit inter-MCC signal buffers as well as decoupling capacitors. Such a design provides nearly 80% performance improvement versus standard carry ripple adder at 1V, and a 54% improvement for 64-bit additions at 0.6V. Figure 7-b represents the reduced carry time on a multiplication add-and-shift cycle with an MCC adder.

4.2.2 Arithmetic Operation Pipelining

Add and shift multiplication can seriously mitigate performance if not properly implemented. In order to improve operation throughput, we implement three optimizations that allow multiplication pipelining. First, we implement latches after the sense amplifiers. Without latches, the iSC and ripple carry operation must be completed in a single step before writeback can be performed. Latches isolate the carry logic from the read and writeback stages, enabling these stages to be pipelined, as illustrated in Figure 7-c. Second, we implement an add-forward line connecting the addition output of one BL logic block to the writeback stage of the next BL pair, allowing a combined add+shift operation. Finally, we observe that, as described in Section 3.2.1, BLADE-enhanced memory can perform a writeback and iSC operation simultaneously, if the writeback target block is in a different LG than those accessed for iSC. By first accumulating the product

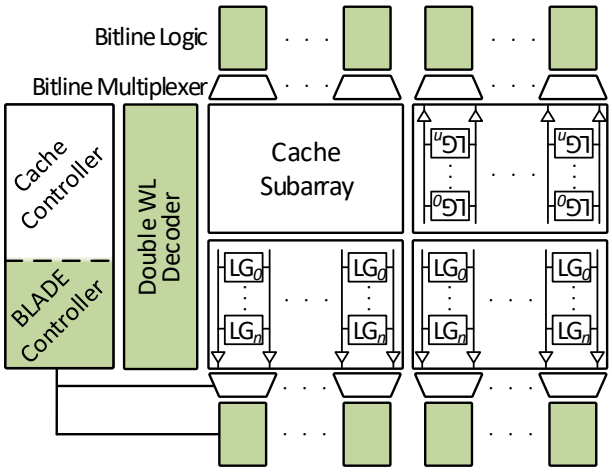


Fig. 9. BLADE implementation within the cache. Highlighted boxes are new additions to cache architecture.

in three partial sums, then summing these partial values, we can fully pipeline the iSC, carry, and writeback stages of multiplication, as illustrated in Figure 7-d. This strategy constrains the cache geometry to at least four LGs per subarray, one containing the multiplicand, which is accessed every cycle, and three containing the partial products.

5 BLADE System Level Integration and Functionality

The question of where to place an iSC architecture, how it integrates into the rest of the memory hierarchy, and how applications invoke it is a nontrivial problem. Problems such as virtual-physical memory translation, coherency, load/store consistency, interaction with standard memory functionality, and communication with the CPU must all be addressed. However, the majority of current literature focuses on the compute portion of the iSC architecture without discussing integration into the system architecture or system level simulation. While some works discuss programmer/ISA support [5], [6], [24], [32], [40], only a few simulate their work in a full software stack environment [5], [24].

BLADE is implemented within the cache hierarchy, as illustrated in Figure 9. Implementing BLADE within the cache reduces energy consumption due to data movement, as operand data will often already be loaded into the cache for utilization by the CPU. Also, in-cache implementation reduces area overhead as BLADE re-utilizes the existing cache SRAM arrays. Overhead is further reduced by integrating BLADE logic underneath any BL multiplexers that provide way interleaving, allowing the logic to be shared between BLs. Retaining way multiplexing functionality also enables parallel tag-data access, allowing the tag and data array to be read simultaneously to reduce read times.

Within the cache hierarchy, we specifically implement BLADE within the private L1 cache, as this provides a favorable trade-off between area footprint and functionality, as well as simplifying cache coherence considerations, in contrast to implementation in shared caches. BLADE can be implemented in lower level caches, providing an increase in the number of possible parallel operations, in exchange for a greater area overhead and coherence complexity. Figure 10 illustrates the instruction flow from issue by the processor to the BLADE cache controller until operation completion and return to the processor. The following sections provide greater detail on ISC operation functionality.

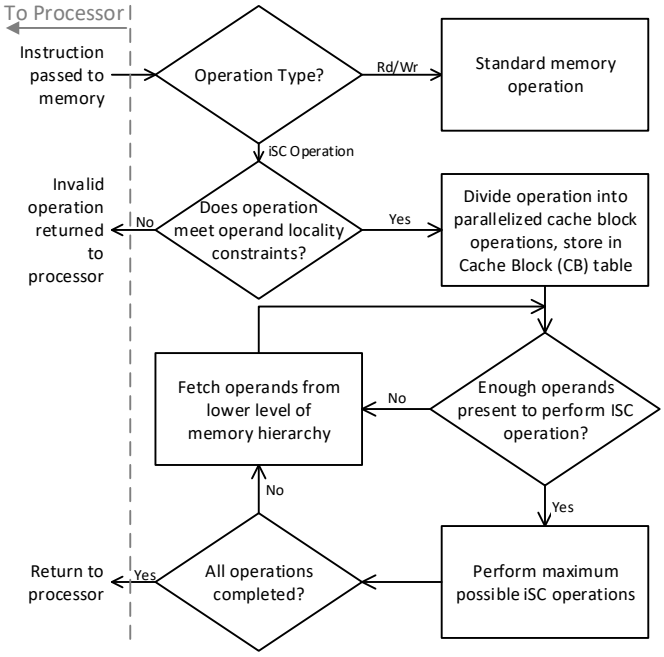


Fig. 10. Instruction flow in BLADE cache from time of issue by processor to operation completion.

5.1 iSC Instruction Passing and ISA Support

To enable BLADE support for applications, we extend the 64-bit ARMv8 ISA with BLADE opcodes, which allows the programmer to invoke BLADE computations from any application compiled for supported ARM devices. BLADE instructions are considered to be memory accesses, and are passed from the CPU to the cache controller, which in turn forwards the instruction to the BLADE controller for decoding and execution. The number of instructions required to invoke BLADE depends on the number of operands required for the desired operation, with up to three instructions passed in succession from the application. These instructions are transmitted to BLADE via the cache address bus, which transmits the operand and result addresses as discussed in Section 5.2, and data bus, which encodes the operand parameters as illustrated in Figure 11:

- The opcode of the requested operation.
- The width of the operands (8/16/32 bit).
- Data unique to specific operations (e.g., the number of bits to shift for a shift operation.)
- The number of successive operations to be performed.

In cases where the count of successive operations is different between operands, address ranges of shorter operands will loop, allowing for reuse of common operands in cases such as multiplying a large input range over a small filter.

Finally, as BLADE commands directly access the cache, it is necessary to place a memory barrier before and after each set of commands to ensure correct memory ordering.

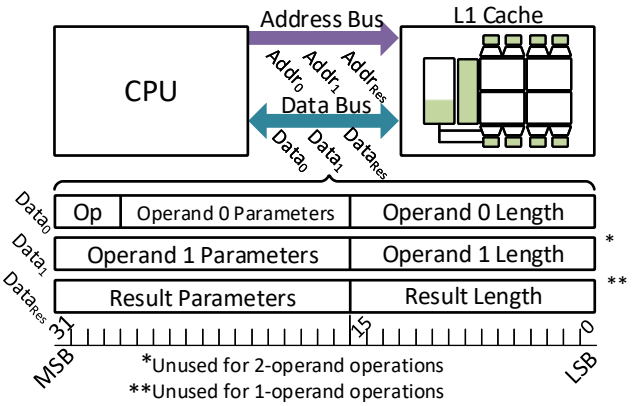


Fig. 11. iSC instruction format and transmission over address and data bus to L1 cache.

5.2 Operand Locality Constraints

As mentioned previously, iSC operands must share BLs to be eligible for iSC operations. Sets that can interact with each other are considered local, and therefore the requirements placed upon operands can be called operand locality constraints. These constraints depend on the geometry of the cache, as factors such as cache size, subarray size, and associativity affect which sets share BLs. However, all variations in cache geometry can be abstracted to three constraints on the operand memory addresses, as shown in Figure 12:

- The offset bits between two operands must match, guaranteeing operand alignment within a cache block.
- A certain number of set LSBs must match, guaranteeing that the operands share the same subarray.
- A certain number of MSBs must differ in order to avoid data corruption, as explained in Section 3.

To quantify the number of set LSBs that must match, we define a geometry value Val_{geo} that specifies the cache geometry properties over which different sets do not share BLs, and therefore cannot interact with each other. This value is calculated according to the following equation:

$$Val_{geo} = \#_{banks} * \#_{subbanks} * \#_{subarrays} * \#_{spwl} \quad (1)$$

where each value represents a cache parameter. $N_{subarrays}$ equals the number of subarray rows in a subbank, and N_{spwl} is the number of sets per WL. By interleaving cache sets across these cache structures, as illustrated in the example array in Figure 12, the number of set LSBs that must match between operands is $log_2(Val_{geo})$. Moreover, any program compiled to function within certain Val_{geo} will also function on any cache geometry of the value of Val_{geo} or smaller. As the operands of this work's benchmark applications are page aligned, they can function on any cache geometry with a Val_{geo} value of up to 64.

Next, it is necessary that a certain number of set MSBs do not match. This constraint results from BLADE's utilization of Local BitLines (LBLs) to run at high frequency (2.2GHz@1V) while preventing data corruption; the electrical functionality of this will be explained in greater detail in Section 3,

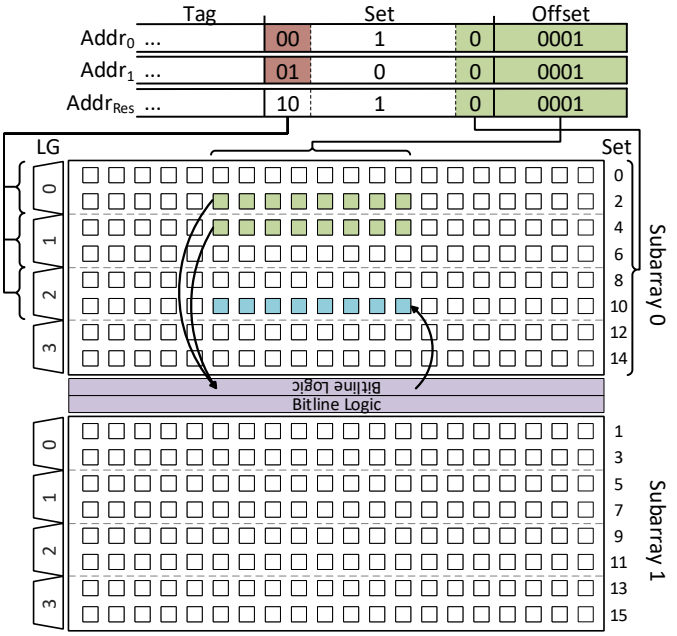


Fig. 12. The highlighted operands in sets 2 and 4 have matching offsets and set LSBs, and differing set MSBs, indicating that they share bitline logic, but not local groups.

however for now it is only necessary to say that a certain number of WLs N_{LBL} in each subarray share one LBL pair, and two operands in an iSC operation must not share the same LBL pair. This geometry constraint can be guaranteed when N_{MSBs} of the set bits of each operand are different, with N_{MSBs} defined as:

$$N_{MSBs} = log_2\left(\frac{N_{sets}}{Val_{geo} * N_{LBL}}\right) \quad (2)$$

where N_{sets} is the number of sets in the cache and N_{LBL} is the number of WLs sharing an LBL pair. Note that N_{MSBs} cannot fall below 1, as this would result in all WLs in a subarray sharing one LBL pair.

Figure 12 illustrates a simple cache with N_{sets}=16, N_{banks} = $N_{subbanks}$ = 1, $N_{subarrays}$=N_{LBL}= 2, and N_{spwl}=1, resulting in a Val_{geo} value of 2 and an N_{MSBs} value of 2. Therefore, one set LSB must match while one MSB cannot match.

We can also calculate the maximum number of simultaneous individual operations that can be performed per iSC operation as follows:

$$\#_{i\_ops} = \frac{Val_{geo} * size_{cb}}{w_{i\_op}} \quad (3)$$

where w_{i_op} is width of a single operation within a CB op in bytes and $size_{cb}$ is the size of a cache block in bytes. Using Equation 3, we calculate that the example cache in Figure 12 can perform 128 simultaneous operations, assuming a $size_{cb}$=64 bytes and w_{i_op}=1 byte.

Guaranteeing proper operand alignment is a challenge that all iSC architectures face, with different solutions being proposed. For example, Jeloka *et al.* [5] handle this by ensuring operands are page aligned and rely on future compiler extensions to guarantee this, while Eckert *et al.* [32] proposes the use of a special transpose unit in the cache

controller for transposing and allocating data in-cache, and assume for their micro-benchmark that application data is laid out in DRAM such that it maps to the proper locations in SRAM. BLADE guarantees alignment by reserving a portion of memory at kernel boot time that can be mapped by an application at runtime. Because memory accessed in this way is guaranteed to map to a specific location in physical memory, we can ensure operand locality by copying data into this reserved memory. The reserved memory is cacheable, contrary to standard reserved memory mapping.

5.3 Issuing ISC Operations to Cache Subarrays

In order to receive commands from the CPU and issue iSC operations to the cache subarrays, the cache controller is augmented to support these functions. iSC operations are limited to a page in length to simplify address translation. The controller breaks the iSC operation into Cache Block (CB) operations up to a cache block in length, and stores them in a CB table. The CB table is responsible for confirming the operation's eligibility for in-cache computation as described in Section 5.2, fetching missing operands, and tracking the status of the operation. Once all operands have been fetched and are present in the cache, the CB table issues the operation to the relevant cache subarrays. After all CB operations are complete, the CB table alerts the CPU that the iSC operation has completed.

5.4 Fetching and Allocating Operands

The CB table requests missing operation operands from memory. To simplify memory access, the BLADE control logic is implemented as a slave module to the standard cache controller. iSC operations received from the CPU are forwarded to the BLADE controller, and all memory requests from BLADE are forwarded to the cache controller, which handles them as standard requests in regards to considerations such as coherency updating, evictions, or MSHR coalescing.

When performing iSC operations, the way in which operand data is stored must be considered to meet operand locality constraints. Therefore, for any operation requiring more than one operand, such as bitwise, addition, or multiplication operations, operands are always stored in way 0 of a set. Any operands that are already loaded in a way other than 0 at the commencement of the iSC operation will be copied via an iSC swap way command to ensure correct functionality. Also, blocks containing iSC operands are tagged as such, and the cache controller will choose to evict other ways first in the case of a conflict miss.

In order to guarantee correct memory coherency functionality, cache blocks subject to a snoop request will be evicted, and the cache controller will inform BLADE of the eviction. The block must be re-requested by BLADE in order to resume functionality. However, multi-cycle operations such as multiplication must be completed atomically, and therefore any snoop requests occurring during such an operation are rejected and reissued.

5.5 Integration in the gem5-X Architecture Simulator

As this work is designed to accelerate edge level devices, we utilize gem5-X [41] to simulate the integration of BLADE into the cache hierarchy of an in-order CPU. It is also possible to integrate iSC architectures into out-of-order cores [5], albeit with more complicated memory access patterns.

In order to generate accurate performance statistics, we extract timing values from BLADE's transistor level design, and convert these into CPU cycle values based on the simulated CPU's target clock frequency. In the next few Sections we will discuss BLADE's circuit level design and optimizations.

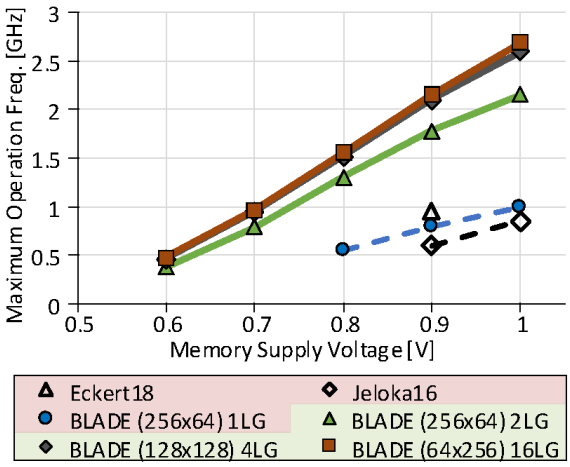


Fig. 13. Maximum frequency of bitwise operations vs. memory supply voltage at 28nm CMOS.

6 Electrical Validation and Design Space Exploration

In order to verify the functionality of BLADE and the aforementioned optimizations, we layout and simulate BLADE at the transistor level. We also perform a design space exploration by varying cache geometry parameters in order to illustrate energy, delay, and area trends.

6.1 Functional Validation of BLADE

Our layout methodology consists of implementation and simulation of the critical paths of the memory array (bitcells, WLs, global and local BLs), periphery (WL decoders and drivers), and iSC logic in 28nm bulk CMOS thin oxide transistors provided by TSMC's high performance technology PDK [42]. The periphery/iSC circuitry is implemented with the Low Voltage Threshold (LVT) technology flavor to optimize performance, while the memory array utilizes the Regular Voltage Threshold (RVT) technology flavor to limit static leakage. We design the bitcell based on the work presented in [43], [44], achieving a bitcell pitch of 0.127μm, and pitch the periphery along the bottom and sides of the array with a spacing of 500nm between the array/periphery to account for any spacing required between SRAM and logic design rules. The simulated netlist contains >8000 elements, is simulated at 300K for 10,000 Monte-Carlo runs, and accounts for CMOS variability and post-layout parasitics. Memory and periphery signal propagation time is modeled by equivalent circuits for the lines with corresponding gates and extracted RC networks.

TABLE 1
Worst Case iSC Energy/Frequency Values in a 256×64 array with 2 LGs.

Operation		Rd	Wr	iSC	Add			
Width		Bitwise			8b	16b	32b	64b
E/op[fJ]		23.5	25.9	23.8	20.7	41.6	83.3	167
MMC Carry Prop. Time [ps]		-	-	-	64	130	258	512
Array Leak./op [fJ]		88.9					137	163
Freq.[Ghz]	(CRA w/o pipeline)	2.2	2.2	2.2	2.2	2.2	1.7	1.2
	(CRA w/ pipeline)	2.2	2.2	2.2	2.2	2.2	2.2	1.0
	(MCC w/ pipeline)	2.2	2.2	2.2	2.2	2.2	2.2	2.2

Using this electrical characterization framework, we extract BLADE area, timing, and energy values for different subarray geometries. Figure 13 shows the maximum clock frequency of BLADE vs. other proposed iSC architectures at different supply voltages. The dotted blue line represents bitwise iSC operation within a single LG, as described in Section 3.2.2, with similar performance to other iSC architectures. The utilization of LGs, on the other hand, enables significantly higher iSC operating frequencies, as illustrated by the green, gray, and red curves. As can be seen, the utilization of LGs improves operation frequency by 2.5x-3x depending on subarray geometry vs. standard iSC architectures and extends the operating range down to 416MHz/0.6V.

Finally, Table 1 details the worst case energy/frequency values for different bitwise/addition iSC operations across different adder configurations. This table shows that, without optimization, 8/16-bit addition can be completed within 2.2GHz. However, reduced operating frequencies of 1.7/1.2GHz are necessary to complete 32/64-bit addition, respectively. By utilizing an MCC adder and carry logic pipelining, as explained in Section 4.2.2, a 2.2GHz operating frequency can be regained for 64-bit additions.

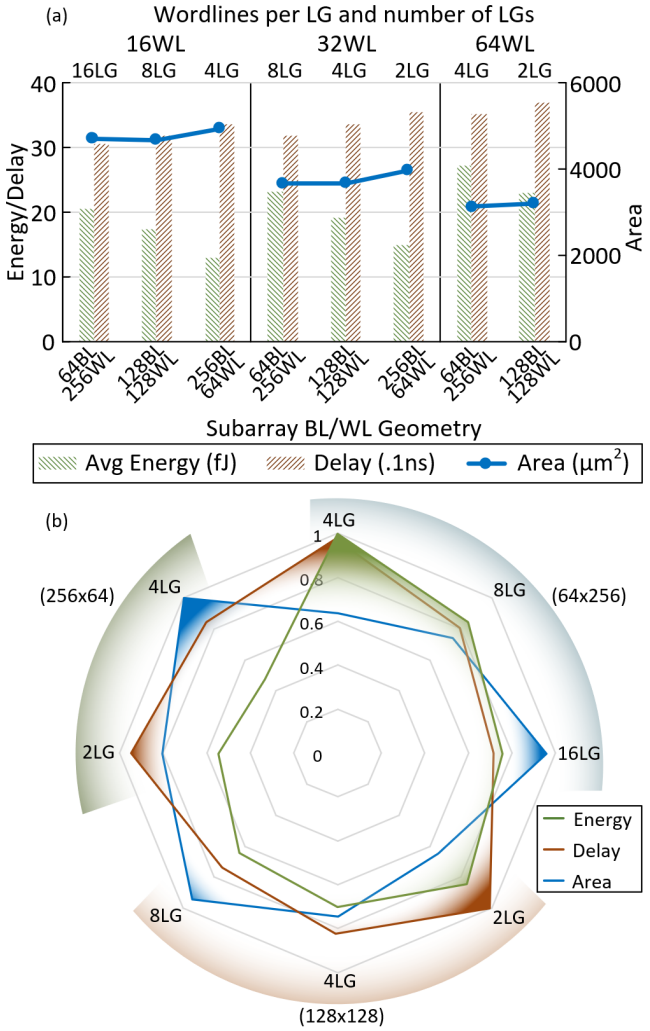


Fig. 14. (a) Energy, area, and delay variations for cache geometries. (b) The normalized energy, area, and delay design space, with maximum values for each metric equal to 1.

6.2 Subarray Design Space Exploration

There is a complex interrelationship between the parameters defining subarray geometry and the subarray's energy, delay, and area overhead. In order to demonstrate these relationships, we explore the design space defined by a subarray's number of bitlines, wordlines, and LGs, with our results displayed in Figure 14-a. To more clearly illustrate the design space, we normalize the design space over a range of various subarray geometries, as shown in Figure 14-b.

Area overhead is primarily influenced by the number of WLs belonging to each LG (LG size). In a 128x128 configuration, LG sizes of 16, 32, 64, and 128 result in area efficiencies of 55.6, 71, 84.4, and 91%, respectively. This is because larger LGs require less periphery per WL they contain, thus improving area efficiency.

Delay is influenced primarily by the LG size, and secondarily by the length of the WL. As LG size increases, parasitic capacitance increases and reduces switching time. Similarly, the parasitic capacitance of the WL increases delay, although this is partially offset by the reduced length of the GBL.

Finally, energy is also influenced by LG size and WL length. Similarly to delay, energy consumption decreases with smaller LGs due to parasitic capacitance. In contrast however, energy decreases as WL length increases, as only one WL is activated for any number of BLs, meaning that the energy per bit decreases with increasing numbers of BLs.

System level factors also play a role in deciding cache geometry. As discussed in Section 3.2.3, at least 2 LGs must be present in order to perform LBL-enhanced iSC operations. More generally, more LGs translates to less operand locality constraints on application data. Furthermore, 4 LGs are necessary to pipeline multiplication, as explained in Section 4.2.2.

As our simulations indicate, different use cases may necessitate different cache geometries, whether it be low-power, low-area, or high-performance designs. When benchmarking BLADE in this article, we consider a 128x128 subarray with 4 LGs. This subarray exceeds our target operating frequency of 2.2GHz while maintaining a low energy consumption and achieving an area efficiency of 71%, of which the BLADE architecture accounts for 8% area overhead.

7 System Level Benchmarking

In order to demonstrate BLADE's performance at the application level, we integrate BLADE into the gem5-X architectural simulator and profile a variety of emerging edge device workloads, allowing us to extract energy and runtime performance trends across a range of cache geometries [41].

7.1 Edge Device Workloads

As BLADE is targeted specifically for edge level devices, we select three benchmarks that are becoming increasingly prevalent on edge devices. These benchmarks demonstrate how BLADE is uniquely positioned for enhancing such devices.

7.1.1 Cryptography

One of the primary challenges the IoT industry faces as more and more sensitive data is stored and transmitted by edge devices is data privacy and security [45]. Indeed, lightweight cryptographic algorithms suitable for edge level devices are being developed [46], and many processors provide dedicated accelerators for cryptography [47]. Given these motivations, we utilize the Secure Hash Algorithm 3 integrity algorithm (SHA-3) [48] as a benchmark to illustrate that BLADE can provide low-power execution for such algorithms.

7.1.2 HEVC Video Processing

In 2018, Youtube and Snapchat combined accounted for over 20% of all mobile upstream traffic [1], and as more people broadcast their lives on social media, compression of upstream traffic will become a necessity on edge devices. The advent of 4K cameras on mobile devices will exacerbate this challenge, with more compression required to efficiently transmit. In order to demonstrate BLADE's capabilities in alleviating this problem, we benchmark Kvazaar [49], an application for High Efficiency Video Encoding (HEVC).

7.1.3 Convolutional Neural Networks

The ubiquity of Convolutional Neural Networks (CNNs) as a solution for a variety of problems has led to increasing interest in implementing effective CNN solutions on edge devices, with both algorithmic [50], [51] and hardware [2], [52] innovations proposed. We therefore include CNNs in our application benchmarks. We implement our benchmark with the Arm Compute Library (ACL) [53], an API from ARM designed to optimally utilize its NEON SIMD co-processor.

7.2 gem5-X Parameters

As stated in Section 5, iSC architectures interact heavily with other components of the computer architecture. In order to acquire accurate performance statistics while simulating such interactions, we utilize the gem5-X architectural simulator [41] with a full Linux software stack to benchmark BLADE. gem5 is calibrated with the parameters outlined in Table 2, which emulates the ARMv8 A53 in-order core found on the ARM Juno development board [54], and running an Ubuntu 18.04 LTS software environment that demonstrates less than 4% timing inaccuracy on profiling tests compared to physical hardware. We implement BLADE timing values by converting delays calculated Section 6.1 to cycle counts at 2GHz. The cache hierarchy utilizes a typical 64 byte WL length, with the cache geometry designed such that 1024 bitwise/128 8-bit iSC operations can be performed simultaneously. For performance comparison, we also simulate a NEON SIMD co-processor [22], a SIMD unit found on many edge devices.

TABLE 2
Simulator Parameters

Processor	2GHz, 4 stage pipeline, ARMv8 ISA in-order core, 7 entry LSQ
NEON Co-processor	128-bit registers 16 parallel 8-bit operations
L1-I Cache	32kB, 4-way, 1 cycle access
L1-D Cache	32kB, 4-way, 1 cycle access, BLADE
L2 Cache	mostly-exclusive, 1MB, 4-way, 6 cycle access
Memory	DDR3 2133MHz, 4GB
BLADE	Max 1024/128 bitwise/8bit simultaneous operations

7.3 McPAT Support

In order to estimate energy consumption of BLADE at a system level, we utilize McPAT [55], an architectural framework for estimating the area and energy of a specified architecture. For this work, we initialize McPAT with ARM Cortex-A53 architectural parameters [56]. We then augment this model using the energy statistics specified in Section 6.1 to estimate the added energy consumption of BLADE operations. gem5-X provides traces of all CPU, memory, NEON, and BLADE operations, which are subsequently provided to McPAT to compute application runtime energy consumption for NEON and BLADE benchmarks.

8 Benchmark Results

During benchmarking, we explore a wide range of hardware and algorithm parameters to demonstrate trends in performance and energy consumption that vary depending on architecture design choices. Specifically, we analyze effects on performance and energy in relation to the number of iSC operations performed, the associativity of the cache, and finally the size of the cache. Also, we analyze how optimization for arithmetic operations affects runtime of applications utilizing such operations.

8.1 Bitwise Operations/iSC Operation Count

In order to observe how the ratio between operations and memory accesses of an application affects BLADE performance, we benchmark the block permutation kernel of the SHA-3 algorithm. This kernel encrypts input data via a large number of bitwise operations, specifically `xor`, `shift`, and `and`, in a series of up to 24 rounds of permutation. By varying the count of bitwise operations being performed on input data, we can draw interesting conclusions about BLADE's effectiveness in accelerating such bitwise algorithms.

Figure 15 illustrates BLADE performance and energy improvement in comparison to NEON for bitwise operation/memory access ratios between 1 and 200 for 4096 bytes of data. As this figure shows, at lower ratios, memory

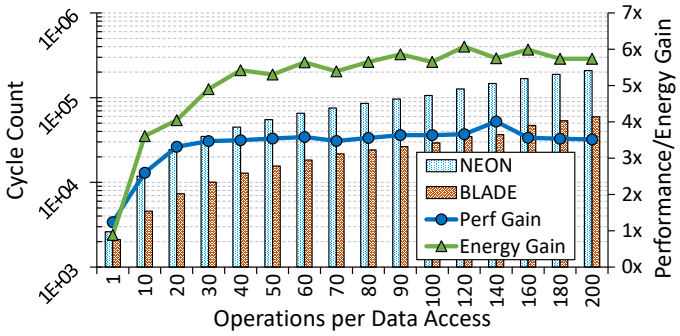


Fig. 15. Runtime/energy results for NEON/BLADE on bitwise operations.

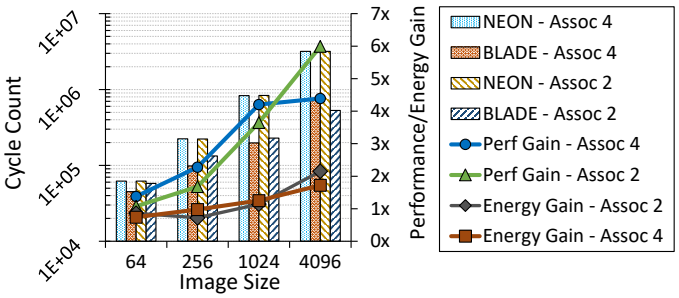


Fig. 16. Runtime/energy results for NEON/BLADE on FIR operations.

access dominates function time. However, as operation count per data access increases, BLADE provides correspondingly increasing acceleration, saturating at a ∼3.5x gain, achieved at 30 operations per access. As SHA3-256 performs over 400 operations per access, BLADE demonstrates strong applicability to such an application. BLADE's maximum demonstrated performance gain over NEON for bitwise operations is 4x.

Energy improvements follow a similar trend, with increasing energy gains that saturate at 40 operations per access. Energy improvement results from identical reasons as previously described; at higher operation/memory access ratios, energy consumption due to compute and L1-CPU data transfer become significant. BLADE eliminates L1-CPU transfers and reduces in-CPU operations, providing up to 6x energy improvement over NEON.

8.2 FIR Filter/Cache Associativity

In our next benchmark, we evaluate how cache associativity (and by extension the number of simultaneous operations BLADE can perform) affects performance. To accomplish this, we modify the associativity between 2 and 4 ways at an equivalent cache capacity, while benchmarking Kvazaar's FIR filter function. This function uses four separate 8-tap FIR filters, filtering an input image first horizontally, then vertically, to produce 16 filtered outputs. Input images of four different sizes are accepted, with progressively increasing memory and compute requirements.

Figure 16 illustrates performance and energy consumption differences resulting from the varying cache geometries. As this figure shows, a 4-way associative cache provides slightly higher acceleration over a 2-way associativity for tile sizes of up to 1024 pixels, as the wide 4-way cache results in less evictions of relevant data. However, at a tile size

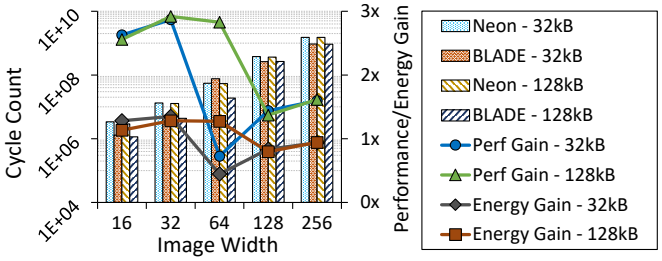


Fig. 17. Runtime/energy results for NEON/BLADE on convolution operations.

of 4096, compute requirements surpass those of memory, and therefore 2-way cache performance surpasses that of 4-way. This is because, assuming constant cache capacity, a 2-way cache can perform twice the number of parallel operations. Overall, BLADE provides up to a maximum of 6x performance gain over NEON for FIR filtering.

In contrast to the bitwise benchmark, FIR filter energy gain is more muted. This results from the fact that iSC multiplication is significantly more computationally complex than bitwise operations and therefore consumes more energy to execute. However, energy consumption due to L1-CPU data movement is still reduced, and BLADE still achieves a maximum of 2x energy reduction over NEON.

8.3 Convolution/Cache Capactiy

In the next set of experiments, we evaluate the effects of overall cache size on BLADE performance by benchmarking it with a convolutional layer of a CNN that we implement in ACL. The convolutional layer has 32 input planes and 32 output planes, and performs 3x3 convolution at a stride of 1 with a padding of 1 to maintain equivalent dimensions. We store input/output data in 32-bit fixed-point notation and weights in 8-bit fixed-point notation, and vary input layer width between 16 and 256 pixels. We utilize 2 cache sizes, 32kB and 128kB, to demonstrate effects of cache size on BLADE effectiveness.

Figure 17 illustrates how cache size impacts performance. Performance results for each cache follow similar trends, where both implementations see performance drop-offs at larger image widths. This is due to the fact that at larger input widths, the resulting output plane does not fit entirely within the L1 cache, requiring a more complex kernel loop to satisfy operand locality constraints as described in Section 5.2, and resulting in increased data movement. However, by increasing cache size to 128kB, larger input widths can be accommodated while avoiding performance drop-off; in this case, 128kB caches can accommodate an input width of up to 64 pixels, as opposed to 32 for a 32kB cache. Furthermore, these results demonstrate the benefits of moving BLADE to higher capacity, lower level caches, as discussed in Section 5. Overall, BLADE demonstrates a maximum performance gain of 3x over NEON for a convolutional layer of a CNN.

Similarly to performance gain trends, a drop to below 1x energy gain is seen at pixel widths of 64/128 for cache sizes of 32kB/128kB, due to increased data movement resulting from ill-fitting kernels. Such trends indicate that operand locality is an important factor in ascertaining the effectiveness of a particular iSC architecture and cache geometry, Thus, not all

TABLE 3
Cycles for 8/32-bit multiplication at different pipeline levels @2Ghz

Pipeline Level	Multiplication Cycle Count (8/32 bit)
No Pipeline	40/126
with Add-Forward	14/72
with Latches	24/66
Full Pipeline	15/39

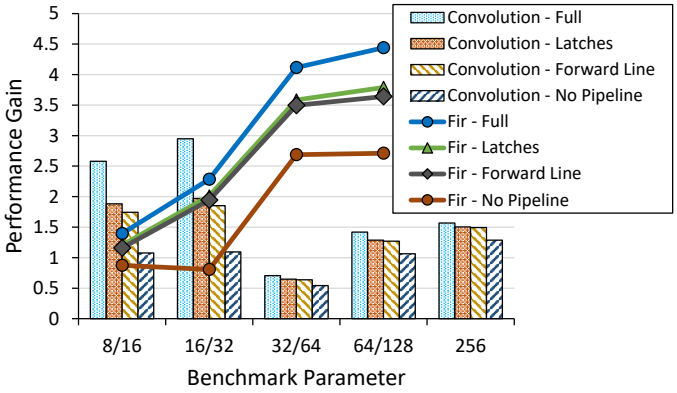


Fig. 18. Performance trends for different levels of arithmetic pipelining.

kernels or applications are well suited to such an architecture. Future work must be done to alleviate these constraints as much as possible. Ultimately, a maximum of 1.5x energy gain is achieved for the convolutional layer.

8.4 Arithmetic Logic Optimization

Lastly, we observe how the arithmetic operation optimizations described in Section 4.2 affect FIR filter and convolution benchmark performance. Table 3 enumerates cycle counts for 8 and 32-bit iSC multiplications for the benchmark architecture. These results show that 8-bit multiplication in fact completes in fewer cycles at 2GHz in an architecture without latches. However, the carry logic delay for 32 bits requires 2 cycles at 2GHz, resulting in a significant increase in cycle count for 32-bit multiplication. Figure 18 illustrates the effects of adding pipeline optimization to the BL logic for the FIR filter and convolution benchmarks, and demonstrates that pipelining provides significant performance improvement for a negligible area cost.

9 Conclusion

iSC architectures show great promise in accelerating a variety of workloads, and are particularly interesting for edge devices due to their area and energy constraints. In this context, we have presented BLADE. BLADE is an arithmetic iSC architecture whose utilization of industry standard 6T bitcell arrays enables easy integration into current SRAM fabrication flows, and its low power digital design makes it appropriate for accelerating emerging applications on edge devices. We validated BLADE's functionality from the system level down to the electrical level. At the system level, we integrated BLADE into the cache hierarchy of an in-order CPU, accounting for system level interactions such as coherency and load/store consistency. Then, at the electrical level, we laid out our enhanced cache design, demonstrating how the use of local bitlines provides the best voltage/frequency ratio (0.6V/415MHz-1V/2.2GHz) of any 6T iSC architecture while maintaining a low area overhead of 8%. Finally, we benchmark BLADE on a full software stack with three emerging edge device workloads, and demonstrated 4x/6x, 6x/2x, and 3x/1.5x performance/energy gains over a the NEON SIMD co-processor, thus validating our iSC design at the application level and demonstrating BLADE's effectiveness for edge level device acceleration.

Acknowledgments

This work has been partially supported by the EC H2020 RECIPE (GA No. 801137) project, the EC H2020 WiPLASH (GA No. 863337) project, and the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).

References

[1] Sandvine, "The mobile internet phenomena report," Tech. Rep., 02 2019.
[2] (2018). [Online]. Available: https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/
[3] S. Jeloka *et al.*, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory," *JSSC*, vol. 51, no. 4, pp. 1009–1021, 2016.
[4] G. Singh *et al.*, "A review of near-memory computing architectures: Opportunities and challenges," in *DSD*, 08 2018.
[5] S. Aga *et al.*, "Compute caches," in *HPCA*, 2017.
[6] A. Subramaniyan *et al.*, "Cache Automaton," *MICRO*, 2017.
[7] M. Kang *et al.*, "A 481pj/decision 3.4m decision/s multifunctional deep in-memory inference processor using standard 6t SRAM array," *CoRR*, vol. abs/1610.07501, 2016.
[8] K. C. Akyel *et al.*, "DRC2: Dynamically reconfigurable computing circuit based on memory architecture," in *ICRC*, 2016.
[9] W. Khwa *et al.*, "A 65nm 4kb algorithm-dependent computing-in-memory sram unit-macro with 2.3ns and 55.8tops/w fully parallel product-sum operation for binary dnn edge processors," in *ISSCC*, 02 2018, pp. 496–498.
[10] J. von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.
[11] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," *Commun. ACM*, pp. 613–641, 08 1978.
[12] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *HCS 23*, 08 2011.
[13] M. Drumond *et al.*, "The mondrian data engine," in *ISCA 44*, 06 2017, pp. 639–651.
[14] S. L. Xi *et al.*, "Beyond the wall: Near-data processing for databases," in *DaMoN 11*, 2015, pp. 1–10.
[15] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *ISCA 42*, 06 2015, pp. 105–117.
[16] D. Zhang *et al.*, "Top-pim: Throughput-oriented programmable processing in memory," in *HPDC 23*, 2014.
[17] K. Hsieh *et al.*, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," in *ISCA 43*, 06 2016, pp. 204–216.
[18] M. Gokhale, S. Lloyd, and C. Hajas, "Near memory data structure rearrangement," in *MEMSYS*, 2015, pp. 283–290.
[19] M. Gao and C. Kozyrakis, "Hrl: Efficient and flexible reconfigurable logic for near-data processing," in *HPCA*, 03 2016, pp. 126–137.
[20] A. Farmahini-Farahani *et al.*, "Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules," in *HPCA 21*, 02 2015, pp. 283–295.
[21] S. F. Yitbarek *et al.*, "Exploring specialized near-memory processing for data intensive operations," in *DATE*, 03 2016.
[22] ARM, "Introducing neon," ARM, Tech. Rep., 2009.
[23] C. Lomont, "Introduction to intel advanced vector extensions," Intel, Tech. Rep., 05 2011.
[24] A. Agrawal *et al.*, "X-sram: Enabling in-memory boolean computations in cmos static random access memories," *Trans. Circuits Syst. I*, vol. 65, no. 12, pp. 4219–4232, 12 2018.

[25] F. Hsueh *et al.*, "Tsv-free finfet-based monolithic 3d+-ic with computing-in-memory sram cell for intelligent iot devices," in *IEDM*, 12 2017, pp. 12.6.1–12.6.4.

[26] S. Srinivasa *et al.*, "A monolithic-3d sram design with enhanced robustness and in-memory computation support," in *ISLPED*, 2018.

[27] W. Simon *et al.*, "A fast, reliable and wide-voltage-range in-memory computing architecture," in *DAC*, 2019.

[28] M. Kang *et al.*, "A multi-functional in-memory inference processor using a standard 6t sram array," *JSSC*, pp. 642–655, 02 2018.

[29] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *JSSC*, vol. 52, no. 4, pp. 915–924, 04 2017.

[30] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pj/decision 3.12tops/w robust in-memory machine learning classifier with on-chip training," in *ISSCC*, 02 2018, pp. 490–492.

[31] A. Jaiswal *et al.*, "8t SRAM cell as a multi-bit dot product engine for beyond von-neumann computing," *CoRR*, vol. abs/1802.08601, 2018.

[32] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA 45*, 2018, pp. 383–396.

[33] J. Wang *et al.*, "A compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *ISSCC*, 02 2019, pp. 224–226.

[34] W. A. Simon *et al.*, "Blade: A bitline accelerator for devices on the edge," *GLSVLSI 29*, p. 6, 05 2019.

[35] L. Chang *et al.*, "A 5.3ghz 8t-sram with operation down to 0.41v in 65nm cmos," in *VLSI*, 06 2007, pp. 252–253.

[36] Q. Dong *et al.*, "A 4 + 2t sram for searching and in-memory computing with 0.3-v v_{ddmin}," *JSSC*, vol. 53, no. 4, 04 2018.

[37] R. Boumchedda *et al.*, "High-density 4t sram bitcell in 14-nm 3-d coolcube technology exploiting assist techniques," *VLSI*, 2017.

[38] M. E. Sinangil, H. Mair, and A. P. Chandrakasan, "A 28nm high-density 6T SRAM with optimized peripheral-assist circuits for operation down to 0.6V," *ISSCC*, 2011.

[39] M. Schlag and P. Chan, "Analysis and Design of CMOS Manchester Adders with Variable Carry-Skip," *TC*, 1990.

[40] M. Kooli *et al.*, "Smart instruction codes for in-memory computing architectures compatible with standard sram interfaces," in *DATE*, 03 2018, pp. 1634–1639.

[41] Y. M. Qureshi *et al.*, "Gem5-x: A gem5-based system level simulation framework to optimize many-core platforms," *HPC '19*, p. 12, 04 2019.

[42] (2011). [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/technology/28nm.htm

[43] M. Chang *et al.*, "A 28nm 256kb 6T-SRAM with 280mV improvement in VMINusing a dual-split-control assist scheme," *ISSCC*, 2015.

[44] H. Pilo *et al.*, "A 64Mb SRAM in 22nm SOI technology featuring fine-granularity power gating and low-energy power-supply-partition techniques for 37% leakage reduction," *ISSCC*, 2013.

[45] M. Elkhodr, S. A. Shahrestani, and H. Cheung, "The internet of things: New interoperability, management and security challenges," *CoRR*, vol. abs/1604.04824, 2016.

[46] S. Singh *et al.*, "Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions," *JAIHC*, pp. 1–18, 2017.

[47] ARM, "Arm cortex-a53 mpcore processor cryptography extension," ARM, Tech. Rep., 12 2014.

[48] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," *FIPS*, 2015.

[49] M. Viitanen *et al.*, "Kvazaar: Open-source hevc/h.265 encoder," in *ACMMM 24*, 2016, pp. 1179–1182.

[50] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," in *ISCA 43*, 06 2016, pp. 243–254.

[51] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018.

[52] V. Gokhale *et al.*, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPR*, June 2014.

[53] (2018). [Online]. Available: https://developer.arm.com/technol-ogies/compute-library

[54] ARM, "Arm versatile express juno r2 development platform," ARM, Tech. Rep., 11 2015.

[55] S. L. Xi *et al.*, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *HPCA 21*, 02 2015, pp. 577–589.

[56] K. Krewell, "Cortex-a53 is arm's next little thing," The Linley Group, Tech. Rep., 11 2012.

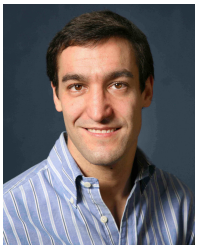
William Andrew Simon received his Master degree in Electrical Engineering, specialization in micro and nanoelectronics, from the Swiss Federal Institute of Technology Lausanne (EPFL) in 2017. He is currently a Ph.D. student in Electrical Engineering in the Embedded Systems Laboratory at EPFL. His research interests are in-memory computing, neural networks, emerging memory architectures, and FPGAs.

Yasir Mahmood Qureshi received his Master degree in Embedded Computing Systems from NTNU, Trondheim in 2013. He is currently a Ph.D. student in Electrical Engineering in the Embedded Systems Laboratory at the Swiss Federal Institute of Technology Lausanne (EPFL). His research interests are energy efficient servers, heterogeneous compute and hybrid memory architectures.

Marco Rios received his Master Degree in Computer Science and Electronics for Embedded Systems from Université Grenoble Alpes in 2018. He is currently a Ph.D. student in Electrical Engineering in the Embedded Systems Laboratory at the Swiss Federal Institute of Technology Lausanne (EPFL). His research interests are design of integrated systems and circuits, in-SRAM computing, 3D stacked technologies and the system impact of emerging memories.

Alexandre Levisse received his Ph.D. degree in Electrical Engineering from CEA-LETI, France, and from Aix-Marseille University, France, in 2017. He is currently a post-doctoral researcher in the Embedded Systems Laboratory at the Swiss Federal Institute of Technology Lausanne (EPFL). His research interests include circuits and architectures for emerging memory and transistor technologies, 3D stacked architectures, and in-memory computing and accelerators.

Marina Zapater received her Ph.D. degree in Electronic Engineering from Universidad Politécnica de Madrid, Spain, in 2015. She is currently a post-doctoral researcher in the Embedded System Laboratory at the EPFL, Switzerland, and has been appointed Associate Professor in the School of Management and Engineering Vaud (HEIG-VD) in the University of Applied Sciences Western Switzerland (HES-SO) in 2020. Her research interests include thermal and power optimization of complex heterogeneous systems, and energy efficiency in novel architectures, servers and data centers.



David Atienza (M'05-SM'13-F'16) is associate professor of electrical and computer engineering, and director of the Embedded Systems Laboratory at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. He received his Ph.D. in Computer Science and Engineering from UCM, Spain, and IMEC, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multiprocessor system-on-chip (MPSoC) and low-power Internet-of-Things systems, including new 2-D/3-D thermal-aware design for MPSoCs and many-core servers, ultra-low power edge AI architectures for wireless body sensor nodes and smart consumer devices. He has co-authored over 300 papers in peer-reviewed international journals and conferences, several book chapters, and seven patents. Dr. Atienza received the 2018 DAC Under-40 Innovators Award, 2018 IEEE TCCPS Mid-Career Award, a 2016 ERC Consolidator Grant, the 2013 IEEE CEDA Early Career Award, the 2012 ACM SIGDA Outstanding New Faculty Award, and a Faculty Award from Sun Labs at Oracle in 2011. He served as DATE 2015 Program Chair and DATE 2017 General Chair, is an IEEE Fellow, an ACM Distinguished Member, and has served as IEEE CEDA President (period 2019-2020).