Faster than the Speed of BRAM: In-Memory Computing for Next Generation FPGAs

Nathaniel Fredricks*, Abraham Mitchell*, Jason D. Bakos‡, David Andrews*
*Department of Electrical Engineering and Computer Science, University of Arkansas,
‡Department of Computer Science and Engineering, University of South Carolina,
{njfredri, atm015, dandrews}@uark.edu, jbakos@cse.sc.edu

***Abstract*—Modern FPGAs integrate dedicated DSP blocks and block RAMs (BRAMs) as architecturally peripheral rather than foundational compute components. As a result, Machine Learning accelerators implemented on FPGAs suffer additional latencies and limited intra-chip bandwidth when transferring data between memory and compute units. To effectively support machine learning, FPGA architectures must evolve toward a more Processor-in-Memory aligned organization. This paper introduces MC^2-BRAM, a Memory-Centric Compute BRAM architecture for FPGAs that integrates arithmetic units and dataflow control directly within on-chip block memory, enabling Processor-in-Memory execution for machine learning workloads. MC^2-BRAM includes *intra-* and *inter*-connect networks that enable concurrent low-latency data movement and execution within and between PIM tiles. As a result, MC^2-BRAM achieves the lowest inter-PIM accumulation latency among all proposed custom PIM tile designs. Additionally, MC^2-BRAM is the first custom PIM design to clock faster than the maximum frequency of BRAM, advancing the state of BRAM-based Processor-in-Memory designs. This enables MC^2-BRAM to deliver the best performance for low-precision GEMV operations. MC^2-BRAM achieves this performance while using 19% less area than the next smallest proposed PIM design, making it the smallest and fastest FPGA PIM architecture reported in literature to date.**

***Index Terms*—FPGA, Processor-in-Memory (PIM), Bit-serial, GEMV**

I. Introduction

Field-programmable gate arrays (FPGAs) have been widely explored as accelerators for machine learning (ML) workloads [1]–[4]. Despite their popularity, today's logic-centric reconfigurable fabrics are fundamentally mismatched with the memory-dominated nature of modern ML computations. DSP blocks are sparsely distributed and connected through the same general-purpose routing fabric as lookup tables (LUTs), while block RAMs (BRAMs) are treated as passive storage macros rather than active participants in computation. As a result, ML accelerators on FPGAs incur energy costs and delays moving data between memory and compute units through the FPGAs programmable interconnect network [5], [6]. The interconnect network itself can account for up to 60% of a chip's total dynamic power consumption and 80% of a circuit's total path delay [7].

By contrast, Modern ML workloads exhibit significant data reuse within localized regions of their computation graphs, making them well-suited to near-memory and in-memory execution [8]–[10]. Supporting these workloads efficiently requires FPGAs to evolve toward a Processor-in-Memory (PIM) aligned organization — one that replaces LUT-centric logic with arithmetic-first compute primitives tightly coupled to memory arrays, substitutes general-purpose routing with structured, locality-aware interconnect, and elevates on-chip memory from passive storage to an active computational substrate.

Native support for low-precision arithmetic and static dataflow scheduling is essential to fully exploit the characteristics of ML workloads. Incorporating PIM principles into reconfigurable architectures offers a path to ASIC-like efficiency for machine learning while preserving the programmability that distinguishes FPGAs from fixed-function accelerators [11].

In this paper, we present MC^2-BRAM, a memory-centric compute BRAM IP that realizes these principles within a next-generation FPGA fabric. Rather than treating the BRAM as a storage peripheral, MC^2-BRAM tightly integrates arithmetic-first compute primitives and locality-aware interconnect directly within the BRAM IP, with native support for low-precision arithmetic and static dataflow scheduling. The contributions of this paper include:

- The first memory-centric compute BRAM that preserves the Fmax of the baseline M20K BRAM while adding in-memory compute capability.
- The first FPGA Processor-in-Memory design to operate at clock frequencies exceeding the native memory Fmax.
- Novel intra- and inter-MC^2-BRAM zero-copy data networks that enable direct data movement between compute-enabled BRAMs without traversing the general routing fabric.
- A comparative evaluation against state-of-the-art PIM BRAM designs, demonstrating that MC^2-BRAM achieves the lowest inter-PIM accumulation latency, best low-precision GEMV performance, and smallest area overhead across all evaluated designs.

The remainder of this paper is organized as follows: Section II-A reviews relevant background on FPGA Processor-in-Memory designs; Section III-A describes the architecture, design, and implementation of MC^2-BRAM; Section IV-A evaluates MC^2-BRAM against state-of-the-art PIM designs; and Section V concludes with directions for future work.

II. Background and Related Work

A. PIM Architectures

Processor-in/near-Memory Architectures have origins dating back to Kautz's 1969 Cellular Logic-in-Memory Array [12] and Stone's 1970 Logic-in-Memory Computer [13]. Kautz proposed a 2D array of identical elementary cells, each with a small amount of logic and storage, such that the same array could be regarded as either a logically enhanced memory array or as a logic array that could be programmed to realize a desired logical behavior. Stone envisioned placing logic within a cache that could perform arithmetic and logical operations between elements of two cache blocks, as well as perform associative operations within a cache block. Stone posited that the cache would give the illusion to the processor that operations were occurring throughout the main memory, but at the speed of the cache.

The very definition of the term "Processor-in-Memory" has evolved and now covers two unique approaches. The first, termed Processing-Using-Memory (PUM), refers to architectures that enable the memory cells themselves to compute [14]. This approach is a modern-day extension of Kuatz's original Cellular Logic-in-Memory Array. The second term, Processor-Near-Memory (PNM), refers to architectures that place processing elements (PEs) within close proximity to, but not integrated within, the memory cell itself [14]. The nearness of the processing elements to the memory varies based on the systems and technologies used. The remainder of this paper focuses on this Processor-Near-Memory (PNM) approach. To maintain consistency with the existing literature, we will use the term Processor-In-Memory to refer to Processor-Near-Memory.

B. FPGA PIM Accelerators

Inspired by Neural Cache [8], Wang et al. [10] proposed Compute-Capable BRAM (CCB) that added in-memory compute capabilities to FPGA BRAMs. These BRAMs function as both storage and bit-serial arithmetic units. Results showed a 1.6x and 2.3x increase in peak multiply-accumulate throughput for a large Stratix 10 FPGA at a minimal cost of only a 1.8% increase in the FPGA die size. The proposed change did not affect the BRAM's interface to the programmable routing. A reconfigurable in-memory accelerator architecture (RIMA) was developed based on the CCB design for deep-learning inference. RIMA used the CCBs and the FPGA's reconfigurability to achieve 1.25x and 3x higher performance on average for 8-bit integer and block floating-point precisions, respectively, compared to the state-of-the-art Brainwave deep-learning soft processor. Results showed that RIMA implemented within a Stratix 10 FPGA could achieve an order of magnitude higher performance than a same-generation GPU.

Improving upon CCB, Arora et al. [15] proposed modifications to FPGA BRAMs to create CoMeFa (Compute-In-Memory Blocks for FPGAs) RAMs. CoMeFa RAMs use the true dual-port nature of FPGA BRAMs and include multiple programmable bit-serial processing elements. Two variations were proposed: the delay optimized CoMeFa-D, and the area optimized CoMeFa-A. By adding CoMeFa-D or CoMeFa-A RAMs to an Intel Arria-10-like FPGA, a geomean speedup of 2.55x or 1.85x was shown, respectively, with algorithmic improvements and efficient mapping, at the cost of 3.8% or 1.2% area, respectively.

Both CCB and CoMeFa used a transposed data layout for bit-serial computation, where operands are stored along the bitlines occupying multiple wordlines in a column-major format. This can lead to additional latency to the end application, due to conversion to and from the row-major format. To solve this problem, Chen et al. proposed a hybrid bit-serial and bit-parallel Multiply-Accumulate (MAC) dataflow in BRAMAC [16] and M4BRAM [17]. In contrast to CCB and CoMeFa, this approach avoids computing with the primary SRAM array, which is large and thus slow and power-intensive. Instead, it copies the operands from the SRAM array to a smaller, separate "dummy array" for MAC operations. This approach liberates the BRAM data ports for read/write tasks, facilitating the concurrent execution of MAC operations, data loading, and parallel DSP operations. BRAMAC's constraint lies in necessitating uniform precision (2-/4-/8-bit) for both weights and activations, restricting its use to uniform-precision DNNs exclusively. M4BRAM overcomes this limitation by enabling variable activation precision, spanning from 2 to 8 bits, while maintaining a MAC latency that scales linearly. Combining BRAMAC-2SA/BRAMAC-1DA with Intel's DLA [18] resulted in an average speedup of 2.05x/1.7x for AlexNet and 1.33x/1.52x for ResNet-34. M4BRAM surpassed BRAMAC by an average of 1.43x across diverse benchmarks.

In parallel with the proposed custom PIM designs, Kabhir et al. developed Da-VinCI, a Deep Learning Accelerator Overlay using In-Memory Compute [19]. Da-VinCI extended two earlier works: PiCaSO, a Processor in/near Memory Scalable and Fast Overlay architecture consisting of 4 x 4 PIM tiles implemented in LUTs that clocked at a BRAMs Fmax [20], and IMAGine, an In-Memory Accelerated GEMV Engine Overlay [11]. These works developed LUT-based design techniques to scale compute linearly with the full BRAM bandwidth of a device, while operating at the BRAM Fmax. Our work is inspired by several of the design techniques developed for IMAGine and Da-VinCI. Specifically, MC^2-BRAM follows IMAGine's approach of pipelining operands from the BRAM through bit-serial ALUs, and inclusion of hardware support for zero memory copies of partial products during reduction operations.

III. MC^2-BRAM Design

A. Design Goals

The source operands processed by the SIMD ALUs of a PIM architecture are read from the memory array. The clock frequency of the memory array thus sets the upper bound target for the computational latency of PIM architectures. Allowing the latency of the compute logic added to the BRAM to exceed the clock period of the BRAM results in a degradation of

TABLE I: Maximum Frequency (MHz) of Existing PIM BRAM Designs

PIM Design	Device	f_{BRAM}	f_{PIM}	Rel.
CCB	Stratix 10	1000	624	62%
CoMeFa-A	Arria 10	730	294	40%
CoMeFa-D	Arria 10	730	588	81%
BRAMAC-2SA	Arria 10	730	586	80%
BRAMAC-1DA	Arria 10	730	500	68%
M4BRAM-S	Arria 10	730	553	76%
M4BRAM-L	Arria 10	730	540	74%
MC^2-BRAM	Arria 10	730	730	100%

the theoretical upper performance bound. Table I lists the achieved clock frequencies (f_{PIM}) of state-of-the art custom PIM designs [11]. The relative frequency column (Rel.) shows that the achieved frequencies were 1.24x – 2.48x slower than the BRAM maximum frequency (f_{BRAM}). Although not shown in this table the achieved *system clock* frequencies (f_{Sys}) were 2.1x – 3.7x slower than the BRAM maximum frequencies (f_{BRAM}). This was attributed to the limitations of the soft logic and the routing resources of the FPGAs. Results also show the achievable system-level clock frequency decreased as the utilization of the chips' BRAMs increased [10], [15].

Memory access and data transfer times, not compute, can represent the dominating performance bottleneck in many ML applications. Google analysis showed an average of 53% of a TPUs clock cycles were spent waiting for weights for MLP and LSTM models [21]. The same analysis showed that, on average, only 10% of total clock cycles were spent computing for these applications. The ratio of cycles spent on memory versus compute is likely to widen as the noise tolerance of ML models continues to drive research towards low-precision computation-based models [22]. PIM architectures provide a convenient framework for addressing this memory wall.

Mapping large machine learning models across distributed BRAMs requires additional support for reduction operations. Within a PIM tile (BRAM + PEs), the presence of a low-latency zero-copy *intra*-connect network between the PEs can eliminate the overhead of sharing partial products through repeated memory writes and reads. The same support is needed in the form of a low-latency zero-copy *inter*-connect network between the distributed PIM tiles to reduce partial results from each PIM tile. This level of support was not addressed in the earlier custom PIM designs.

B. MC^2-BRAM Architecture

Fig. 2 shows the block diagram of the M20K BRAM [23] with our MC^2-BRAM modifications. The M20K BRAM has a physical memory array of 128 rows by 160 columns. The 160 columns pass through 4:1 multiplexers that select up to 40 bits to be read or written simultaneously on the BRAM's Port A and Port B.

As shown in Fig. 2, MC^2-BRAM adds a mode register that, when given a command word, enables switching between two operational modes:

1) **Memory Mode:** In this mode, MC^2-BRAM behaves as a normal FPGA BRAM. The outputs of the two 4:1 multiplexers are selected by the column decoders using the least significant address bits. Addresses can be changed at the BRAMs Fmax, rendering 40 of the 160 available bit lines accessible per clock cycle- only 25% of the memory array's available bandwidth.

2) **PIM Mode:** In PIM mode, the memory array provides source operands through the 4:1 MUXs, Port A and Port B, and into the pipelined SIMD processor shown in green at the bottom of Fig. 2. The SIMD processor contains 40 ALUs to match the 40-bit width of the M20K's Port A and Port B. The number of ALUs can be configured to match each BRAM architecture. A controller (not shown in Fig. 2) issues instructions to the array processor through data port A. Fig. 1 shows the layout of MC^2-BRAMs two instruction types. The instruction in Fig. 1(a) shows the layout for regular operations and Fig. 1(b) for determining the next operation for Booth's multiplication. Fig. 2 shows an additional counter to the left of the 4:1 MUXs to select inputs to the 4:1 multiplexer. The counter is clocked by the sub-clock shown in Fig. 3 at 4 times the Fmax rate. This allows four groups of 40 bits to be pipelined into Port A and Port B per Fmax clock cycle. The SIMD processor array is also clocked at the sub-clock rate, allowing arithmetic operations to occur at the BRAM's full bandwidth. SPICE timing analysis shows MC^2-BRAM's pipeline can clock at nearly 3 GHz, providing significant headroom for increasing the Fmax of the BRAM.

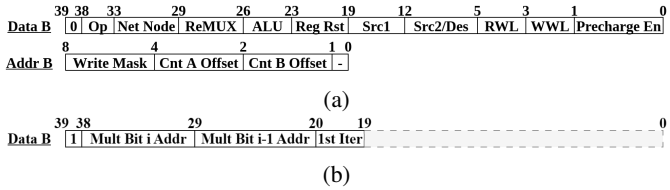


Fig. 1: Instruction format for (a) standard 3-phase operations (b) configuring the next iteration in Booth's multiplication

The data from Ports A and B are streamed through the 4-stage bit-serial pipeline shown in green in Fig. 2 at the sub-clock rate. The first pipeline stage allows data to be transferred between MC^2-BRAM through the network node. This is discussed in Section III-E below. In the second pipeline stage, data from Port A, Port B, and the Network Node are selected and routed into the ALU lines, X and Y. Computations are performed in the 40 bit-serial ALUs in the third pipeline stage. Data is written back to the memory array during the fourth pipeline stage.

C. Bit-Serial Operations

Similar to CCB [10] and CoMeFa [15], MC^2-BRAM's ALUs, shown in Fig. 4, implement bit-serial integer and fixed-point operations. Regular 2-operand addition and subtraction are performed 4 bits at a time in the 3 phases shown in the timing diagram in Fig. 3. In the read phase, the cells in the source rows of the memory array are activated and drive the column bit lines. While there are 160 columns, only

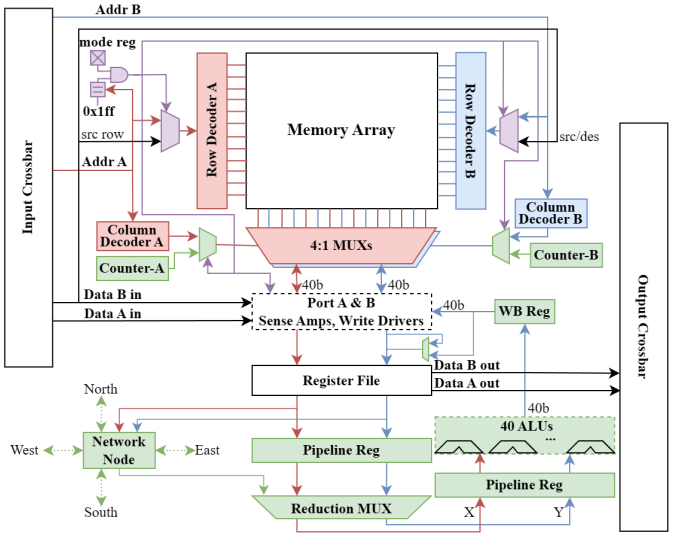


Fig. 2: High-level architecture of MC^2-BRAM. New components are highlighted in green. Additions common with previous works are highlighted in purple.

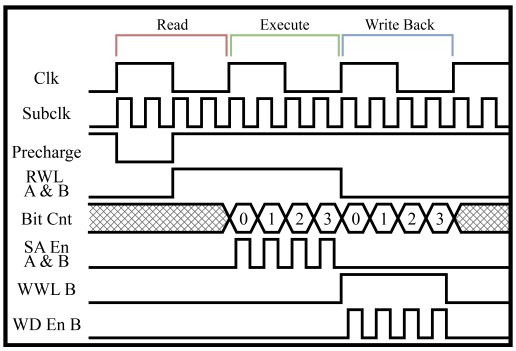


Fig. 3: Timing sequence for 2-operand addition/subtraction operations.

40 of those are output at a time from each port's column multiplexers.

In the execute phase, the Ports A and B column multiplexers stream the bit-serial data into the SIMD compute pipeline. To process all 160 bits in an Fmax clock cycle, we utilize the sense-amp cycling technique described in [24]. Every sub-clock cycle, the 4:1 column multiplexers switch to another column, and 40 new bits are latched into the register file. Data currently in the pipeline is processed and latched into the next register at the end of the pipe stage. After 4 sub-clock cycles, the first bit of data has passed through the ALUs, and the first bit of results (sum, difference, or copy) from those 40 ALUs is stored in the writeback registers.

At the beginning of the write-back phase, the pipeline is full (all 4 pipeline registers are populated). At this point, no more data is read, the row decoder for port B is changed to the destination address, and the results are ready to be streamed back to memory. When the write word-line signal is raised, the current bit of the results is read from the writeback register

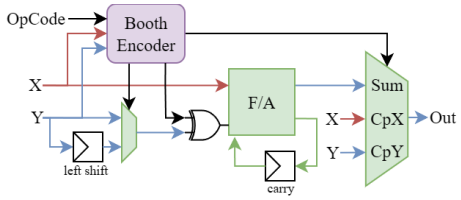


Fig. 4: Architecture of ALU used in MC^2-BRAM

and written to 1 of 4 columns (selected by the counter). The next bits of the two 40 operands are passed into the ALUs, and the next result bits are latched into the writeback register. This repeats until all 4 bits of the results are written to memory. To enable 4 writes in a single clock cycle, a technique similar to sense-amp cycling is used for the write drivers.

The use of bit-serial ALUs allows the precision to be tuned to the machine learning model and adjusted by an external soft-logic controller. Using the sense-amp cycling technique and the 4-stage pipeline, the latency for N-bit addition/subtraction is $4\times\lceil N/4\rceil$ (e.g., the latencies of 1, 2, 3, and 4-bit addition are the same). As mentioned earlier, a 4-bit width was chosen to be compatible with the M20K BRAM's existing column decoders (4:1 MUXs) and the pipeline length (4 stages). Thus, 2-operand addition and subtraction for N-bit numbers require $3N/4$ clock cycles at Fmax.

Booth's radix-4 algorithm is used for two's complement multiplication [25]. For each iteration in the multiplication, 2 bits of the multiplier are read by the ports and streamed to the ALUs. A third multiplier bit is saved from the previous iteration. Once in the ALUs, the multiplier bits are encoded to determine the next multiplicand operation: add, subtract, or copy. That operation is then carried out over the next 3N/4 + 3 clock cycles. This process is repeated for the remaining bits of the multiplier. Thus, N-bit multiplication takes $3N^2/8 + 2N$ clock cycles to perform.

D. Intra-BRAM Data Movement

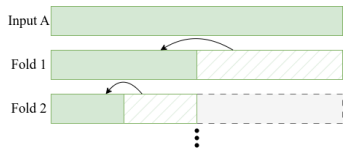


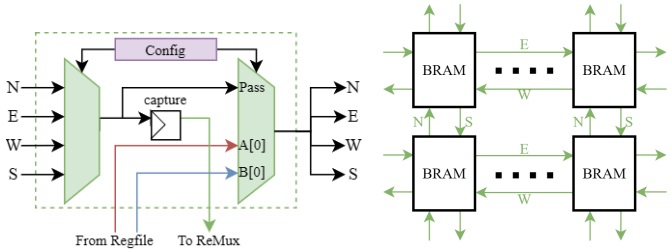
Fig. 5: Reduction MUX folding

The operations that underpin machine learning consist of multiply-accumulate (MAC) reduction operations of the form $y = w[0] * x[0] + w[1] * x[1] + ...w[n] * x[n] + b$ [26]. All multiplications can be performed concurrently, followed by the summation of the product terms and bias. To perform the summation, or reduction step, without additional hardware support, the partial sums must be copied between PEs by going through the fabric. CoMeFa [15] is the only existing design to provide an interconnect that serializes direct communication between PEs. However, this interconnect still requires copying partial sums back to memory before operating.

MC^2-BRAM introduces the Reduction Multiplexer (ReMUX) shown in Fig. 2 to bring zero-copy in-pipeline data movement between PEs. MC^2-BRAM design is a more efficient custom implementation of the OpMUX proposed in PiCaSO [20] extended to the full bandwidth of the internal multiplexers. This allows MC^2-BRAM's ReMUX to achieve a 4x speedup over PiCaSO's OpMUX.

The ReMUX implements a binary reduction of partial sums in $log_2(K)$ iterations to sum K words as shown in Fig. 5. The first iteration adds the upper 20 words to the lower 20 words on Port A. The next iteration adds the upper and lower halves of the 20-word group. This repeats until all 40 words have been accumulated. While the summation is occurring on Port A, the results can be written back on Port B. Read and write-back phases are overlapped, taking *N/2 + 1* clock cycles (running at the BRAM's Fmax) per iteration. Thus, the total execution time is $log_2(K) * (N/2+1)$ cycles to sum K words of N bits.

E. Inter-BRAM Data Movement



(a) (b)

Fig. 6: (a) Network node architecture (b) BRAM NEWS network

The partial sums computed concurrently within each MC^2-BRAM will need to be accumulated. To facilitate inter-BRAM accumulation, MC^2-BRAM's pipeline dedicates the first pipeline stage for inter-BRAM networking. Each MC^2-BRAM contains the Network Node, shown in Fig. 2, connected to its cardinal neighbors in a North-East-West-South (NEWS) network. The Network Node consists of a receiver MUX, a transmitter MUX, a capture register, and configuration registers. The receiver MUX selects inputs from the four neighboring BRAMs and latches the data within the capture register. The transmitter MUX selects a data source to transmit. The transmitter MUX can select from the local BRAM register file, the capture register, or act as a "pass-through", retransmitting the data directly from the receiver MUX. By traveling through chains of pass-through Network Nodes, data can traverse multiple BRAMs within a sub-clock cycle. Received data stored in the capture register can also be sent to the ReMUX. The ReMUX directs data from the capture register to the ALUs via the Y lines. Since an operand is streamed-in, each MC^2-BRAM only needs to dedicate one port to reading. Similar to intra-BRAM reduction, inter-BRAM addition/subtraction takes $N/2+1$ clock cycles.

CoMeFa [15] is the only other PIM design with inter-BRAM hardware support. CoMeFa's 1-dimensional movement chains together the outermost PEs of neighboring BRAMs. These data lines directly connect the output of one PE's sense amps to the next PE's write drivers. This configuration requires data to be written back into memory after each shift. Data moves 1 PE per clock cycle. While great for pipelined applications like FIR, the timing overhead grows greatly with distance and precision.

In contrast, MC^2-BRAM uses a 2-D network that eliminates all intermediate memory write-backs. Timing analysis of our 22nm models shows data can be moved through 9 BRAMs in a sub-clock period. To traverse greater distances, a "relay" BRAM can be used to pass data through its pipeline. After getting to the writeback register, the data is routed to the register file, skipping the memory write-back entirely. The data is then retransmitted by the relay's network node. Through relay chaining, MC^2-BRAM can perform inter-BRAM operations across arbitrary distances with zero data copying.

IV. Evaluation and Analysis

A. Tools and Methodologies

MC^2-BRAM was designed in Cadence's Virtuoso Studio [27]. COFFE [28] was used for area, timing, and power modeling and analysis using 22nm Predictive Technology Models [29]. COFFE does not directly support sizing for 20kb BRAMs. Similar to [15]–[17], the area was calculated by interpolating between 16kb and 32kb models. The estimated area for an M20K BRAM model was 5614.3 μm^2. SPICE simulations using the same 22nm technology were used to verify all timings. As done in [15], CCB's performance was scaled from Stratix 10 to Arria 10 for fair comparisons.

Resource	Count	Area Percentage
Logic Blocks	339,620	70.45%
DSP Slices	1,518	9.45%
BRAMs	2,423	20.10%
M20k RAM bits (kb)	48,460	
MLAB RAM bits (kb)	9,386	
Total RAM bits (kb)	57,846	

TABLE II: Properties of baseline Arria-10 GX900 FPGA

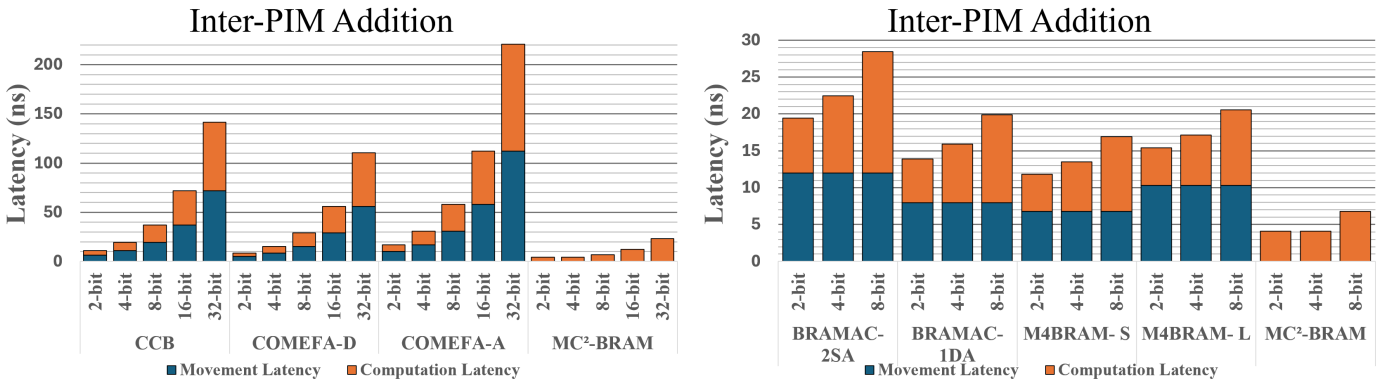
Table II describes properties of the Arria-10 FPGA [30] selected as our baseline architecture. Arria-10 devices are fabricated in a 20nm technology node, similar to the 22nm technology used in COFFE. The calculations for the area percentages are based on the area model as described in [31].

B. MC^2-BRAM Area and Frequency

Table III lists the total area overhead for MC^2-BRAM with other custom PIM designs. The first row shows the area increase to an M20K BRAM. MC^2-BRAM increases the area by 368.3 μm^2, equating to a 6.56% increase to the 5614.3 μm^2 M20K BRAM. The increase for the other custom PIM designs ranges from 34.4% for M4BRAM-L to 8.1% for the area optimized CoMeFa-A. Table III shows MC^2-BRAM introduces the smallest area overhead, and is 19% smaller than CoMeFa-A. The second row in Table III lists the additional

Property	CCB	Comefa-A	Comefa-D	BRAMAC-1DA	BRAMAC-2SA	M4BRAM-S	M4BRAM-L	MC²-BRAM
Area Overhead (BRAM)	16.80%	8.10%	25.40%	16.90%	33.80%	19.60%	33.40%	6.56%
Area Overhead (FPGA)	2.96%	1.77%	5.54%	3.49%	6.99%	4.06%	6.93%	1.36%
Clock Period Overhead	60%	150%	25%	46%	10%	32%	35%	0%
Supported Precision	Arbitrary	Arbitrary		2,4,8		2,4,8		Arbitrary
Inter-BRAM Networking	None	1D		None		None		2D
Arithmetic Dataflow	Serial	Serial		Hybrid		Hybrid		Serial
Operand Vector/Matrix Sizes	<160>,<160>	<160>,<160>		<2>,<2, 40/20/10>	<2>,<2,80/40/20>	<2, 1/2/4>,<2, 32/16/8/4/2>		<40>,<40>

TABLE III: Architectural Comparison Between MC^2-BRAM with Previous PIM Designs



(a) (b)

Fig. 7: Comparison between interBRAM accumulation latencies: (a) MC^2-BRAM vs COMEFA and CCB, (b) MC^2-BRAM vs BRAMAC and M4BRAM

area introduced by all PIM designs relative to the size of the Arria-10. With BRAMs occupying 20.1% of the Arria-10's area, MC^2-BRAM introduces a modest 1.36% increase in total chip area.

The execution pipelines of the previous PIM designs introduced critical paths that, when integrated with the BRAM, degraded the achievable Fmax operating frequency. The overhead penalties as a percentage increase in the Fmax clock period are listed in the third row of Table III. CoMeFa-A introduced the smallest area overhead of the previous designs, but the highest clock period overhead at 150%. CoMeFa-D optimized for delay reducing the clock overhead to 25%, but at a cost of increased area overhead. Of all previous designs, BRAMAC-2SA introduced the smallest clock overhead at 10%, but at the cost of the highest area overhead. In contrast, MC^2-BRAM introduces no clock overhead and runs at the BRAM Fmax. Timing analysis using our 22nm models showed the critical path through the MC^2-BRAM pipeline to be 128 ps. Instead of our design limiting the BRAM Fmax, the opposite is true: the maximum clock frequency of the BRAM itself (730MHz) is now the limiting factor. This allows MC^2-BRAM to scale performance as BRAMs get faster.

C. Accumulating between PIMs

Most realistic machine learning models are much too large to fit into a single PIM tile and will need to be partitioned throughout multiple PIM tiles. Partial results from each PIM tile will need to be summed. To evaluate this step, we measured the time to: (1) transfer a data word from the BRAM of one PIM Tile and (2) accumulate the data word in a second PIM Tile. Fig. 7 shows the results. Accumulation for the bit-serial designs is shown in Fig. 7(a-orange). All bit serial designs perform addition in the same number of clock cycles. MC^2-BRAM achieves an overall lower accumulation latency due to its higher clock frequency. CCB, CoMeFa-D, and CoMeFa-A incur the additional latency shown in Fig. 7(a-blue), due to a memory copy required to transfer data between PIM-Tile BRAMs. The transfer between the PIM Tiles is performed serially within the FPGA's programmable interconnect network at the design's system clock rate Fsys. MC^2-BRAM's Network Node shown in Fig. 6 eliminates the memory copy and allows data from a remote PIM-Tile BRAM to be input into a local PIM-Tile's ALU in a single clock cycle. The elimination of the memory copy and faster clock frequency allows MC^2-BRAM to achieve the low inter-PIM accumulation latencies shown in Fig. 7.

Fig. 7(b) shows the same accumulation latency analysis for MC^2-BRAM, BRAMAC, and M4BRAM. The hybrid bit-serial and bit-parallel dataflow designs of BRAMAC and M4BRAM result in lower computation latency than CCB and CoMeFa's bit-serial designs. Full bit-width partial sums are extracted from the dummy arrays in a single read operation in

fixed movement latencies that do not change with arithmetic bit precision. Because the dummy arrays are not fully bit-parallel, computation latencies still scale with bit-precision. Fig. 7(b) shows MC^2-BRAM, despite using bit-serial arithmetic, still outperforms these bit-hybrid PIM designs with 2.9x to 4.81x lower latencies. This is due to the increased clock frequency and the elimination of the memory copy step between the source and destination PIMs.

D. Benchmarks

We evaluated MC^2-BRAM using benchmarks from [10], [15], and [16].

1) GEMV: The GEMV benchmark was evaluated over a range of bit-precisions and matrix sizes. To make comparisons fair, all designs were given an equivalent slice of an Arria 10 FPGA containing 16 BRAMs. For visual clarity, only the lowest observed latencies among variants are shown for multi-variant designs. Fig. 8 shows results for 2-bit GEMV (with an 8-bit accumulator), 4-bit GEMV (with a 16-bit accumulator), 8-bit GEMV (with a 27-bit accumulator), and 16-bit GEMV (with a 36-bit accumulator) operations.

MC^2-BRAM achieves lower latencies across all bit precisions compared to the other bit-serial designs. Comparisons between MC^2-BRAM and BRAMAC and M4BRAM are more interesting. This is expected as MC^2-BRAM implements pure bit-serial arithmetic, whereas BRAMAC and M4BRAM implement hybrid bit-parallel arithmetic.

At 2-bit precision, MC^2-BRAM achieves 3.3x and 2.3x speedups over BRAMAC and M4BRAM, respectively. The additional clock cycles of the bit serial arithmetic operations in MC^2-BRAM are offset by the faster clock frequency. At 4-bit precision, the additional clock cycles of the bit serial arithmetic operations of MC^2-BRAM allow BRAMAC and M4BRAM to close the gap. However, MC^2-BRAM achieves 2x and 1.5x speedups over BRAMAC and M4BRAM, respectively.

At 8-bit precision, the additional clock cycles of the bit serial arithmetic operations allow BRAMAC to achieve a marginal speedup over MC^2-BRAM up to a matrix size of 300 x 300, after which MC^2-BRAM provides a marginal speedup that increases as the size of the matrix grows. M4BRAM provides a marginal speedup over MC^2-BRAM that diminishes around a matrix size of 600 x 600. MC^2-BRAM achieves a lower latency for larger matrices due to the fast inter-PIM tile reduction network and elimination of the need to do a memory copy for accumulating partial results between the PIM tiles.

GEMV at 16-bit precision represents less noise-tolerant applications. Only the bit-serial PIM BRAMs support this precision. MC^2-BRAM's Booths radix-4 multiplication and efficient reduction network combine to deliver 4x and 5x speedups over CoMeFa and CCB, respectively.

Results show MC^2-BRAM achieves the lowest latencies of all bit-serial designs. This results from the faster clock frequency and intra- and inter-BRAM data reduction networks. Results also highlighted tradeoffs between bit serial and bit parallel MAC operations and effects of faster clock speeds and data reduction hardware network support. Clock cycle counts for bit-parallel MAC operations grow linearly, whereas clock cycle counts for bit-serial MAC operations using Booth's radix-4 multiplication algorithm grow quadratically. MC^2-BRAM's faster Fmax and reduction network offsets the clock cycle count differences, enabling MC^2-BRAM to achieve lower latencies at smaller 2-bit and 4-bit precisions. MC^2-BRAM also achieves lower latencies for larger matrices at higher precisions. Performance at larger matrix sizes becomes more significant as real-world ML applications require ever larger GEMV operations. These results show that MC^2-BRAM is the fastest for low-precision GEMV and will also efficiently scale with the increasing size of next-generation machine learning models.

2) Elementwise Multiplication: Elementwise multiplication is a foundational operation in non-ML applications such as digital signal processing. To test multiplication performance in less noise-tolerant applications, we measure the latency to perform 16-bit multiplication. MC^2-BRAM's radix-4 multiplication is 3.79x, 2.96x, and 5.92x faster than CCB, CoMeFa-D, and CoMeFa-A, respectively.

3) Finite Impulse Response (FIR) Filter: We implement a FIR filter with 128 taps using the same load-compute-unload pipeline as described in [15]. Latency measures the time between loading a sample and unloading a result. MC^2-BRAM delivers a speedup of 3.51x, 2.72x, and 5.44x over CCB, CoMeFa-D, and CoMeFa-A, respectively.

4) Bulk RAID: We test bulk bit-wise operations with Redundant Array of Independent Disks (RAID) data recovery. Bit-wise XOR operations are carried out on 32 pairs of non-transposed words. As with other benchmarks, CCB, CoMeFa-D, and CoMeFa-A differ only by frequency. MC^2-BRAM outperforms CoMeFa-D in both frequency and cycle count, making MC^2-BRAM 1.67x faster than CoMeFa-D.

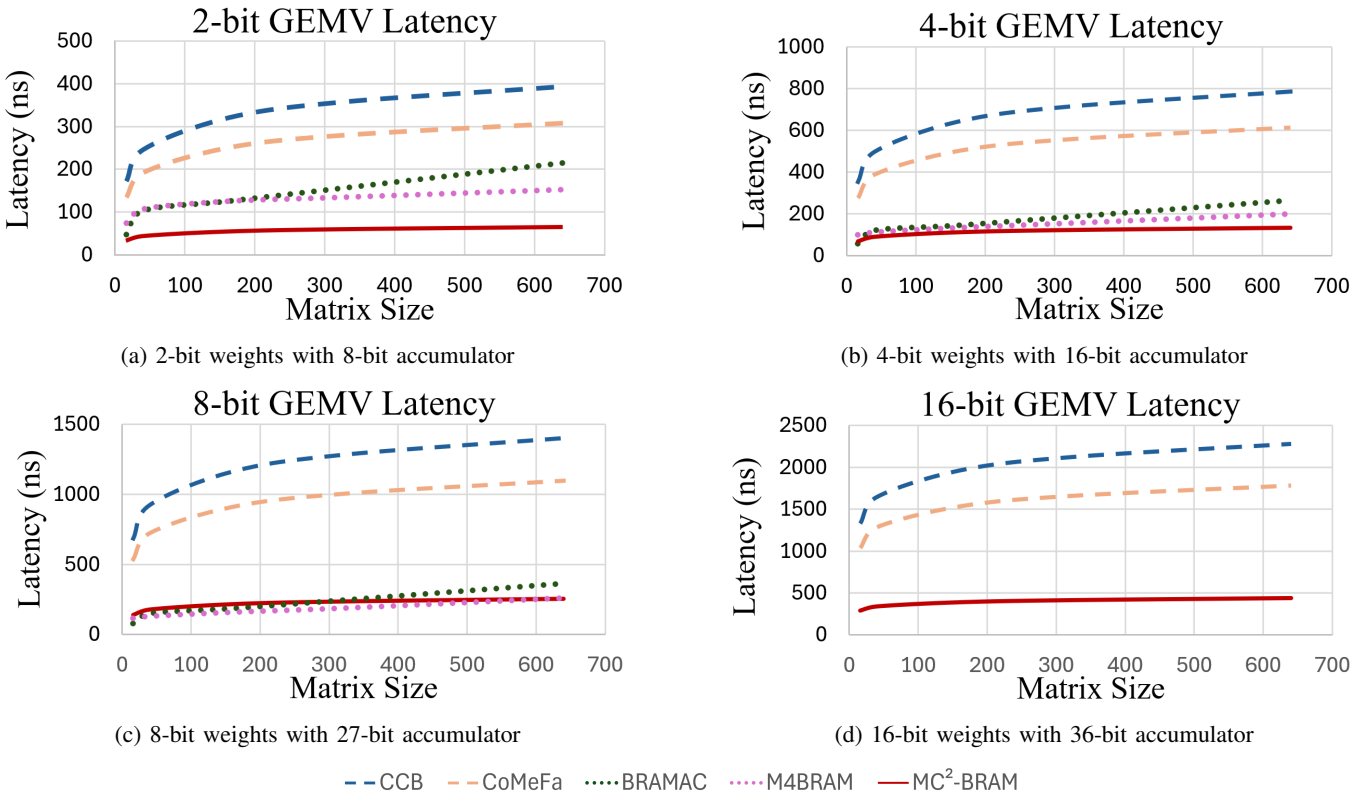
Fig. 10 shows the geometric mean of the three benchmarks, with MC^2-BRAM 3.05x, 2.38x, and 4.76x faster than CoMeFa-D, CCB, and CoMeFa-A, respectively.

E. Area-Delay Product (ADP)

We use the Area-Delay Product (ADP) as an overall figure of merit for design efficiency. The ADP is computed by multiplying GEMV latencies at each bit-precision and matrix size by the PIM area of each design. A lower ADP shows superior performance-per-area [32], [33]. Fig. 9 shows the ADP of MC^2-BRAM, BRAMAC, and M4BRAM for 8-bit precision. CCB and CoMeFa are omitted as their ADP is significantly higher than BRAMAC and M4BRAM across all GEMV precisions. We present the ADP for 8-bit precision as it represents the worst-case relative latency for our design. Even in this worst case, MC^2-BRAM achieves the best performance-per-area among all designs.

V. Conclusion and Future Work

This paper presented MC^2-BRAM, a new Processor-In-Memory BRAM architecture. MC^2-BRAM is the first FPGA-based PIM architecture to: (1) implement zero-copy intra and inter-BRAM networking, (2) enable concurrent data movement



(a) 2-bit weights with 8-bit accumulator

(b) 4-bit weights with 16-bit accumulator

(c) 8-bit weights with 27-bit accumulator

(d) 16-bit weights with 36-bit accumulator

Fig. 8: Comparison of GEMV latency at (a) 2-bit, (b) 4-bit, (c) 8-bit, and (d) 16-bit weight precisions

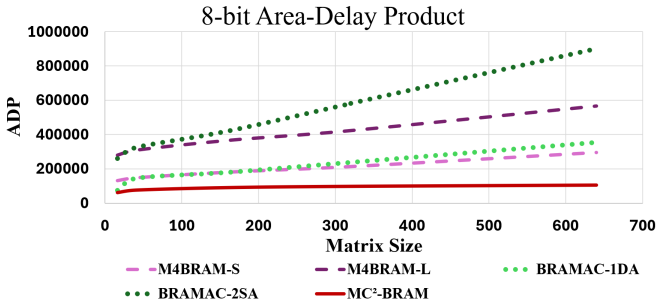


Fig. 9: ADP figure of Merit for MC^2-BRAM, BRAMAC, and M4BRAM using 8-bit GEMV latencies.

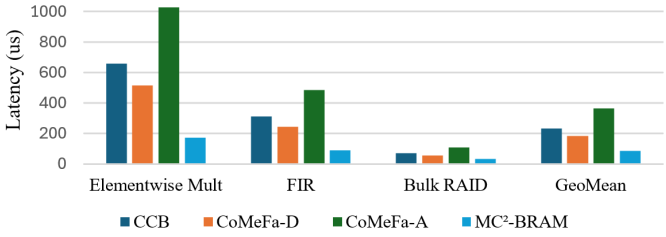


Fig. 10: Latencies of various bit-serial benchmarks. The geometric mean of latencies is also shown.

and execution, and (3) maintain and exceed the Fmax of the original M20K BRAM.

Analysis was presented showing MC^2-BRAM achieves the lowest latencies for both inter-BRAM accumulation and low-precision GEMV operations of all proposed BRAM PIM architectures. MC^2-BRAM performs inter-PIM accumulation at least 2.9x faster than previous PIM architectures. In GEMV operations, MC^2-BRAM outperformes the fastest prior PIM architecture by up to 2.45x at low bit-precisions while exhibiting superior scaling with larger matrix sizes. For broader workloads, MC^2-BRAM achieved geometric mean speedups of 2.38x to 4.76x across multiplication, signal processing, and bulk bit-wise benchmarks.

MC^2-BRAM achieved these performance results while maintaining the smallest footprint among all proposed custom FPGA PIM architectures. Area-Delay-Product (ADP) metrics showed in a worst-case scenario, MC^2-BRAM provided the highest speedup per unit area among all studied designs. Overall, integration of MC^2-BRAM into FPGA BRAMs promises substantial performance gains for low-precision machine learning and diverse workloads, enabling lower latencies for future FPGA architectures.

Future work includes developing programming models, compilation, and run-time support for PIM architectures, as well as integrating analog computing and emerging memory technologies, such as ReRAM, into PIM BRAMs. The development of new MLIR dialects for MC^2-BRAM is ongoing.

REFERENCES

[1] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," 2016. [Online]. Available: https://arxiv.org/abs/1602.04283

[2] J. Duarte, P. Harris, S. Hauck *et al.*, "Fpga-accelerated machine learning inference as a service for particle physics computing," *Computing and Software for Big Science*, vol. 3, no. 1, p. 13, 2019.

[3] M. B. Altman, W. Wan, A. S. Hosseini, S. Arabi Nowdeh, and M. Alizadeh, "Machine learning algorithms for fpga implementation in biomedical engineering applications: A review," *Heliyon*, vol. 10, no. 4, p. e26652, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844024026835

[4] X. Zhang, A. Ramachandran, C. Zhuge, D. He, W. Zuo, Z. Cheng, K. Rupnow, and D. Chen, "Machine learning on fpgas to face the iot revolution," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 894–901.

[5] A. Arora, B. Hanindhito, and L. K. John, "Compute rams: Adaptable compute and storage blocks for dl-optimized fpgas," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, pp. 1156–1163.

[6] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[7] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-d fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 216–229, 2007.

[8] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 383–396.

[9] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 367–379. [Online]. Available: https://doi.org/10.1109/ISCA.2016.40

[10] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. Iyer, and R. Das, "Compute-capable block rams for efficient deep learning acceleration on fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 88–96.

[11] M. A. Kabir, T. Kamucheka, N. Fredricks, J. Mandebi, J. Bakos, M. Huang, and D. Andrews, " IMAGine: An In-Memory Accelerated GEMV Engine Overlay ," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep. 2024, pp. 220–226. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/FPL64840.2024.00038

[12] W. Kautz, "Cellular logic-in-memory arrays," *IEEE Transactions on Computers*, vol. C-18, no. 8, pp. 719–727, 1969.

[13] H. S. Stone, "A logic-in-memory computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[14] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, M. Sadrosadati, and G. F. Oliveira, "A modern primer on processing in memory," 2025. [Online]. Available: https://arxiv.org/abs/2012.03112

[15] A. Arora, A. Bhamburkar, A. Borda, T. Anand, R. Sehgal, B. Hanindhito, P.-E. Gaillardon, J. Kulkarni, and L. K. John, "Comefa: Deploying compute-in-memory on fpgas for deep learning acceleration," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3603504

[16] Y. Chen and M. S. Abdelfattah, "Bramac: Compute-in-bram architectures for multiply-accumulate on fpgas," 2023. [Online]. Available: https://arxiv.org/abs/2304.03974

[17] Y. Chen, J. Dotzel, and M. S. Abdelfattah, "M4bram: Mixed-precision matrix-matrix multiplication in fpga block rams," 2023. [Online]. Available: https://arxiv.org/abs/2311.02758

[18] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An opencl(tm) deep learning accelerator on arria 10," 2017. [Online]. Available: https://arxiv.org/abs/1701.03534

[19] M. A. Kabir, N. Fredricks, T. Kamucheka, J. Mandebi, M. Huang, J. D. Bakos, and D. Andrews, "Da-vinci: A deep-learning accelerator overlay using in-memory computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 4, Nov. 2025. [Online]. Available: https://doi.org/10.1145/3770756

[20] M. A. Kabir, E. Kabir, J. Hollis, E. Levy-Mackay, A. Panahi, J. Bakos, M. Huang, and D. Andrews, "Fpga processor in memory architectures (pims): Overlay or overhaul?" in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, Sep. 2023, p. 109–115. [Online]. Available: http://dx.doi.org/10.1109/FPL60245.2023.00023

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[22] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 65–74. [Online]. Available: https://doi.org/10.1145/3020078.3021744

[23] D. M. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, "Architectural enhancements in stratix v™," in *Symposium on Field Programmable Gate Arrays*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:17639936

[24] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 259–272. [Online]. Available: https://doi.org/10.1145/3123939.3123986

[25] A. D. BOOTH, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 01 1951. [Online]. Available: https://doi.org/10.1093/qjmam/4.2.236

[26] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[27] C. D. Systems, "Virtuoso (ic6.1.7-64b.500.19)," 04 2018. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/cic/posts/virtuoso-ic6-1-7-isr19-and-icadv12-3-isr19-now-available

[28] S. Yazdanshenas and V. Betz, "Coffe 2: Automatic modelling and optimization of complex and heterogeneous fpga architectures," vol. 12, no. 1. [Online]. Available: https://doi.org/10.1145/3301298

[29] U. of Minnesota, "Predictive technology model," 2012. [Online]. Available: https://mec.umn.edu/ptm

[30] Intel, *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*, 2022. [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683461.pdf

[31] rafat. rashid, J. G. Steffan, V. Betz, and rafat. rashid, "Comparing performance, productivity and scalability of the tilt overlay processor to opencl hls," *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 20–27, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6032540

[32] A. Waris, A. Aziz, and B. M. Khan, "Area-time efficient pipelined number theoretic transform for crystals-kyber," *PLOS ONE*, vol. 20, no. 5, pp. 1–27, 05 2025. [Online]. Available: https://doi.org/10.1371/journal.pone.0323224

[33] A. Marquardt, V. Betz, and J. Rose, "Speed and area tradeoffs in cluster-based fpga architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 84–93, 2000.