2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) | 979-8-3503-1205-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/FCCM57271.2023.00015

BRAMAC: Compute-in-BRAM Architectures for Multiply-Accumulate on FPGAs

Yuzong Chen and Mohamed S. Abdelfattah
Department of Electrical and Computer Engineering, Cornell University
{yc2367, mohamed}@cornell.edu

***Abstract*—Deep neural network (DNN) inference using reduced integer precision has been shown to achieve significant improvements in memory utilization and compute throughput with little or no accuracy loss compared to full-precision floating-point. Modern FPGA-based DNN inference relies heavily on the on-chip block RAM (BRAM) for model storage and the digital signal processing (DSP) unit for implementing the multiply-accumulate (MAC) operation, a fundamental DNN primitive. In this paper, we enhance the existing BRAM to also compute MAC by proposing BRAMAC (Compute-in-BRAM Architectures for Multiply-Accumulate). BRAMAC supports 2's complement 2- to 8-bit MAC in a small dummy BRAM array using a hybrid bit-serial & bit-parallel data flow. Unlike previous compute-in-BRAM architectures, BRAMAC allows read/write access to the main BRAM array while computing in the dummy BRAM array, enabling both persistent and tiling-based DNN inference. We explore two BRAMAC variants: BRAMAC-2SA (with 2 synchronous dummy arrays) and BRAMAC-1DA (with 1 double-pumped dummy array). BRAMAC-2SA/BRAMAC-1DA can boost the peak MAC throughput of a large Arria-10 FPGA by 2.6×/2.1×, 2.3×/2.0×, and 1.9×/1.7× for 2-bit, 4-bit, and 8-bit precisions, respectively at the cost of 6.8%/3.4% increase in the FPGA core area. By adding BRAMAC-2SA/BRAMAC-1DA to a state-of-the-art tiling-based DNN accelerator, an average speedup of 2.05×/1.7× and 1.33×/1.52× can be achieved for AlexNet and ResNet-34, respectively across different model precisions. Our code is available at: https://github.com/abdelfattah-lab/BRAMAC.**

I. Introduction

Deep neural networks (DNNs) have become ubiquitous in many important fields such as computer vision, speech recognition, and natural language processing. However, a well-trained DNN model for complicated tasks has a huge model size ranging from several hundreds of megabytes (e.g., AlexNet classifying ImageNet) to several hundreds of gigabytes (e.g. GPT3 producing human-like text) [1]–[3]. Accordingly, many researchers have been exploring reduced numerical precisions to represent DNN model weights and activations, especially during inference where reduced-precision arithmetic incurs little or no accuracy loss compared to full-precision floating-point (FP) [4], [5]. This low-precision property allows better utilization of on-chip memory and computation resources for improved performance. For example, Nvidia GPUs can obtain a 4-8× inference speedup using INT8 precision compared to FP32 precision [6], and an additional 1.6× speedup using INT4 precision compared to INT8 [7].

In the meanwhile, FPGAs are becoming an increasingly popular platform for DNN acceleration due to their hardware programmability that enables customized datapaths and numerical bit-widths suitable for low-precision inference [8]–[11]. FPGA-based DNN accelerators heavily rely on block random access memory (BRAM) for model storage and digital signal processing (DSP) units for implementing multiply-accumulate (MAC)—the fundamental primitive in DNNs. Nevertheless, most FPGA vendors' DSP blocks do not natively support precisions lower than 18 bits, making them sub-optimal for implementing low-precision MAC [12]–[14]. For DNNs to better utilize FPGA's on-chip resources, researchers have proposed novel DSP architectures for low-precision MAC [15], [16]. More recently, some works have proposed to add compute capability inside BRAMs and enable them to perform various Boolean and arithmetic operations [17], [18]. This computing in-memory (CIM) approach does not sacrifice the performance of existing logic resources on FPGA but rather complements them to further boost the FPGA's computing throughput. In addition, CIM can reduce the routing associated with data movement between memory and logic units, hence saving energy and area. This is especially true in DNN accelerators where model parameters and activations are frequently transferred between BRAMs and DSPs to perform massive computations.

In this paper, we further enhance the FPGA's compatibility with low-precision DNNs by proposing BRAMAC, an efficient compute-in-BRAM architecture for multiply-accumulate. Unlike previous CIM architectures that compute directly on the main BRAM array [17], [18], BRAMAC first copies the data from the main BRAM array to an additional, separate memory array and then computes on this "dummy" array, which is a true dual-port BRAM with the same number of columns as the main BRAM array but only 7 rows. This 7-row dummy array can be accessed fast with low power consumption due to a much smaller parasitic load on its bitlines compared to the main BRAM array which typically has >100 physical rows. Furthermore, the dummy array allows BRAMAC to function like a normal BRAM even during CIM operations—the main BRAM array's read and write ports are available for use by the application logic. Finally, BRAMAC is optimized for DNN MAC operations by performing shared-input multiplication and in-place accumulation. We enumerate our contributions below:

1) We propose new peripheral circuits that enable BRAMAC to compute two MACs (or one MAC2), $P = (W_1 I_1 + W_2 I_2)$, simultaneously using a hybrid bit-serial & bit-parallel dataflow.
2) We propose two BRAMAC variants with different area-throughput trade-offs: BRAMAC with 2 synchronous dummy arrays (2SA) and BRAMAC with one double-pumped dummy array (1DA).

2576-2621/23/$31.00 ©2023 IEEE
DOI 10.1109/FCCM57271.2023.00015

3) We design an embedded finite-state machine (eFSM) to free up the main BRAM ports during MAC2 computation and to allow simultaneous main BRAM access, thus enabling efficient tiling-based DNN acceleration.
4) We quantify the benefits of employing BRAMAC in a tiled FPGA DNN accelerator, which achieves up to 2.04× and 1.52× performance improvements for AlexNet and ResNet-34, respectively over the baseline accelerator without BRAMAC.

II. Related Work

In this section. we discuss previous work that targeted efficient MAC implementation on FPGAs including logic block, DSP, and BRAM enhancements.

A. Logic Block with Fast Arithmetic

To efficiently implement arithmetic operations in soft logic, modern FPGAs contain hardened adder circuitry in their logic blocks (LBs) [19]. These adders range from simple ripple-carry adders to more complex variants such as carry-bypass adders and carry-lookahead adders. In order to reduce the carry propagation delay, dedicated routing is used to propagate carry signals between different LBs. Inspired by the superior efficiency of adopting low-precision in DNN, recent research started to investigate adding more hardened arithmetic in LBs. For example, Boutros *et al.* [20] proposed three LB architectural enhancements to improve the performance of MAC implemented in soft logic. Their most promising proposal increases the MAC density by 1.7× while simultaneously improving the MAC speed.

B. Low-Precision DSP Architectures

Modern commercial FPGAs include DSP blocks that implement efficient multiplication with additional features such as pre-addition and accumulation commonly used in signal processing applications [19]. Nevertheless, most FPGA vendors' DSP multipliers have a minimum precision of 18-bit, making them less competitive in accelerating low-precision DNNs. To address this limitation, researchers have proposed new DSP architectures to support low-precision MAC. Boutros *et al.* [15] introduced an enhanced Intel DSP (eDSP) that supports four 9-bit or eight 4-bit multiplications without using additional routing ports. Rasoulinezhad *et al.* [16] presented a modified Xilinx DSP, called PIR-DSP, that can carry out six 9-bit, twelve 4-bit, or twenty-four 2-bit multiplications. Regarding industry DSP trends, the recent Xilinx Versal and Intel Agilex devices added support for 8-bit multiplication in their DSP blocks [21], [22]. In addition, Intel's latest Stratix-10 NX device added a new DSP (called AI tensor) block that contains 30 INT8 multipliers and can also be configured as 60 INT4 multipliers [23].

C. Computing In-BRAM

With the emergence of CIM to overcome the von-Neumann bottleneck [24], some FPGA researchers suggest augmenting existing BRAM architectures with compute capability. Wang *et al.* [17] proposed a compute-capable BRAM (CCB) that uses bit-serial arithmetic to enable a high degree of computation parallelism. However, the circuit implementation of CCB requires an additional voltage supply to mitigate the read-disturb issue associated with activating two word-lines from one BRAM port, which is challenging to implement in practice. Arora *et al.* [18] later designed a new compute-in-BRAM architecture called CoMeFa to overcome some limitations of CCB. CoMeFa also relies on bit-serial arithmetic but exploits the dual-port nature of BRAM to read out two operands from two ports, respectively instead of activating two word-lines from one port, thus eliminating the read-disturb issue.

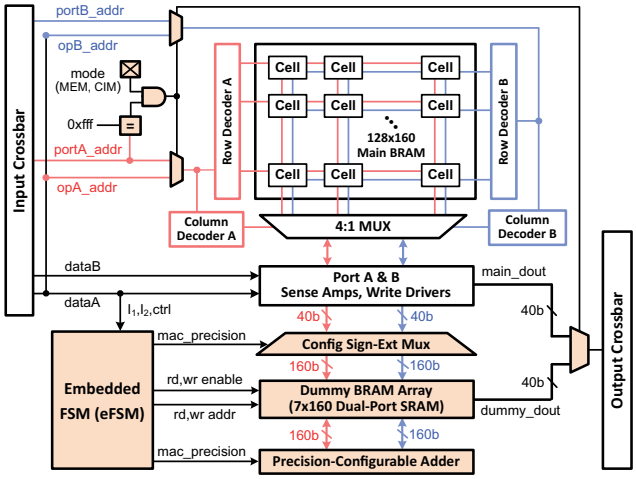


Fig. 1: Top-level block diagram of BRAMAC modified from Intel's M20K BRAM. New circuit blocks are orange-shaded.

Both CCB and CoMeFa require transposed data layout for bit-serial computation, i.e., each word occupies one column and multiple rows instead of one row and multiple columns in a conventional data layout. However, transposing data is expensive in both latency and additional hardware cost (e.g. a swizzle module in CoMeFa) for online execution. Furthermore, these two BRAM architectures compute directly on the main BRAM array and receive the CIM instruction through a BRAM write port—this prevents tiling. As a result, these two works are limited to accelerating only persistent-style DNN inference where the model weights are transposed offline and remain persistent in the on-chip memory. Different from CCB and CoMeFa, BRAMAC adopts a hybrid bit-serial & bit-parallel MAC dataflow that eliminates the requirement of transposed data layout. In addition, BRAMAC doesn't compute on the main BRAM array which is typically large and therefore, slow and power-hungry. Rather, it copies the main BRAM's data to a special, separate dummy BRAM array for computation. This dummy array has only 7 rows and therefore can be accessed much faster compared to the main BRAM array. It can also free up the read and write ports of the main BRAM during CIM to allow tiling-based DNN acceleration.

III. BRAMAC Architecture and Dataflow

A. Overall Architecture

Fig. 1 shows the top-level block diagram of BRAMAC modified from Intel's M20K BRAM [25] with added circuit blocks orange-shaded. The routing interface (i.e., input and

output crossbar) of BRAMAC is the same as that of M20K. The main BRAM array's dimension is 128-row × 160-column, i.e., 20 kb memory capacity. The 4:1 column multiplexing feature of M20K is preserved. One additional SRAM cell is added to select one of the two operation modes of BRAMAC:

1) MEM: In this memory mode, the behavior of BRAMAC is identical to that of a conventional M20K. The input crossbar sends the address and data to portA and portB. For memory reads, the two addresses are decoded by the row and column decoders. The 40-bit BRAM output data from sense amplifiers is sent to the output crossbar. For memory writes, the data is sent to the write drivers for updating the main BRAM.

2) CIM: This is the compute mode where BRAMAC can compute MAC2, $P = (W_1 I_1 + W_2 I_2)$, using 2-bit, 4-bit, or 8-bit operand precision. The two groups of operands, (W_1, W_2) and (I_1, I_2), can be thought of as weights and inputs of DNN in the remainder of this paper, respectively. At a high level, BRAMAC computes MAC2 by keeping weights inside BRAMAC while streaming inputs from outside.

The main BRAM is automatically configured as a simple dual-port memory with a maximum data width of 40-bit, and a depth of 512 to maximize the read/write throughput. A special address (0xfff) is reserved and compared with the portA address, and if equal, the 40-bit portA data is treated as a CIM instruction. The CIM instruction contains two addresses for reading two 40-bit data from the main BRAM, respectively. Each 40-bit data is a vector that contains multiple low-precision W_1/W_2 elements. The configurable sign-extension mux sign-extends the 40-bit vectors to 160-bit before copying them to a dummy BRAM array which is a 7-row × 160-column true dual-port BRAM without the column multiplexing feature. The CIM instruction also contains the two inputs, I_1 and I_2, and several control signals that are sent to an eFSM to trigger and control the MAC2 operation. The precision-configurable adder can read two 160-bit vectors from the dummy array, performs a single-instruction-multiple-data (SIMD) add, and writes the sum back to the dummy array. Since the dummy array has the same number of columns as the main BRAM array, it can read out 40-bit data similar to the main BRAM. A 2-to-1 mux is added to select the data between the main BRAM and the dummy array.

B. Hybrid Bit-Serial & Bit-Parallel MAC Dataflow

BRAMAC computes 2's complement MAC2 by adopting a hybrid bit-serial & bit-parallel dataflow [26] as described in Algorithm 1. The for-loop in line 2-11 iterates through two inputs bit-by-bit. Each iteration involves multiplying the entire W_1 and W_2 by a single bit from I_1 and I_2, respectively, followed by a bit-parallel addition to obtain the partial sum (psum) as shown in line 3. If the current input bit is the most-significant bit (MSB), then psum is subtracted from P (line 5) since the MSB is negative in 2's complement representation. If the current input bit is not the least-significant bit (LSB), then P also needs to be shifted left by 1-bit after adding psum (lines 6, 9).

The hybrid bit-serial & bit-parallel MAC2 algorithm is efficient for computing matrix-vector multiplication (MVM) where the n^{th} vector element is multiplied by all elements

Algorithm 1: Hybrid Bit-Serial & Bit-Parallel MAC2

```
Require : All numbers are integers in 2's complement
Input   : W ∈ Z^2, I ∈ Z^2, precision n ≥ 2
Output  : P ∈ Z
1  Initialization P = 0
2  for i = (n − 1) downto 0 do
3      psum = W1 * I1[i] + W2 * I2[i]
4      if i == (n − 1) then
5          P = P + inv(psum) + 1
6          P = P << 1
7      else if i ≠ 0 then
8          P = P + psum
9          P = P << 1
10     else
11         P = P + psum
12 return P
```

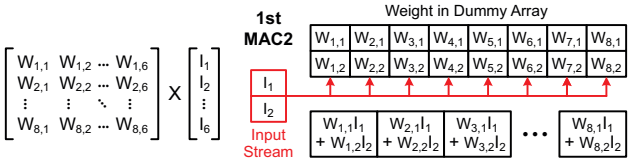


Fig. 2: Example of MAC2 to compute matrix-vector multiplication.

of the n^{th} matrix column. To exploit this input-sharing in BRAMAC, two inputs are packed into the CIM instruction that is sent to BRAMAC, then multiplied by all elements of the corresponding two matrix columns copied to the dummy array, respectively. Copying a matrix column requires the weight matrix to be transposed so that matrix columns correspond to a BRAM row. This can be easily done offline for DNNs. Fig. 2 illustrates an example of using MAC2 to compute MVM where the matrix dimension is 8×6. For the first MAC2, the first and second matrix columns are copied from the main BRAM to the dummy array. Two vector elements I_1 and I_2 are streamed to BRAMAC through the CIM instruction and multiplied by all 8 elements of the first and second matrix columns to obtain 8 partial sums. For large matrices, the number of matrix elements that can be loaded to the dummy array depends on the MAC precision. Since the two read ports of the main BRAM have a total data width of 80-bit, they can copy ten 8-bit, twenty 4-bit, or forty 2-bit weights to a dummy array for one MAC2, providing a parallelism of 10, 20, or 40 MACs, respectively.

C. Circuit Design to Support MAC2

We now describe the new circuit blocks in BRAMAC to support MAC2. These circuit blocks are shown in Fig. 3, including a dual-port "dummy" BRAM array (Fig. 3(a)), a configurable sign-extension mux (Fig. 3(b)), a 160-bit SIMD adder implemented using 1-bit full adders, and read/write circuits (Fig. 3(c)).

1) Dual-Port Dummy BRAM Array: The dual-port dummy BRAM array is 7-row × 160-column without column multiplexing as shown in Fig. 3(a). Its SRAM cell is identical to that used in the main BRAM. Each column contains two sense amplifiers and two write drivers to allow true dual-port access. Its 1st row is hard-coded to always store **0**. The 2nd and 3rd rows store the $\mathbf{W_1}$ and $\mathbf{W_2}$ vectors, respectively that

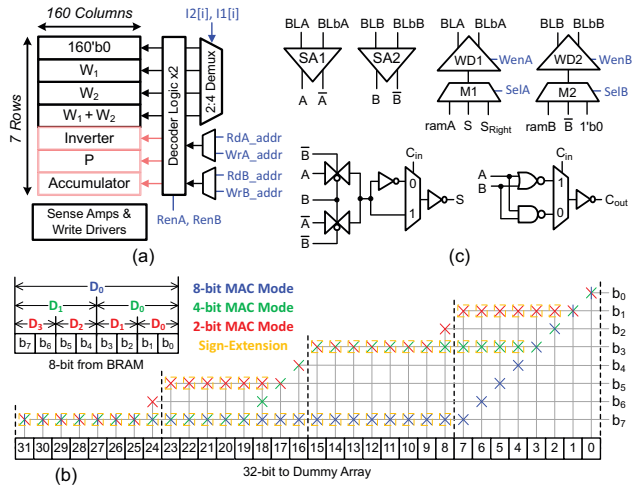


Fig. 3: BRAMAC circuit blocks for computing MAC2: (a) dual-port dummy BRAM array, (b) configurable sign-extension mux (here we are displaying one out of five identical blocks), (c) 1-bit full-adder with read/write circuits.

are copied from the main BRAM array. The 4th row stores a **(W_1 + W_2)** vector. The 5th **Inverter** row is used to store the temporary inverted psum required by the binary subtraction (line 5 of Algorithm 1). The 6th row stores the MAC2 result **P**. The 7th row is a wide **Accumulator** to accumulate multiple MAC2 results that form a large dot product.

The read and write operations of the dummy array are controlled by address and enable signals (blue signals in Fig. 3(a)) sent from the eFSM as described in Section III-A2. The access to 1st – 4th rows during MAC2 is managed by both the decoder logic and a 2-to-4 demux. The 2-bit selection signal of the demux comes from the current two processing bits of the two inputs $\mathbf{I_1}$ and $\mathbf{I_2}$, respectively. This allows calculating psum (line 3 of Algorithm 1) using a look-up table [27]. If **{I_2[i], I_1[i]}** is 2'b00, then the 1st zero row will be read out and added to the 6th row **P**. If **{I_2[i], I_1[i]}** is 2'b11, then the 4th row **(W_1 + W_2)** will be read out and added to **P**. If **{I_2[i], I_1[i]}** is 2'b01 or 2'b10, then then the 2nd row $\mathbf{W_1}$ or the 3rd row $\mathbf{W_2}$ will be read out and added to **P**.

Since the dummy array copies data from the main BRAM array for computation, a coherency issue may arise where the main BRAM is being updated while the dummy array is still computing using the stale data. We leave it for the programmer/compiler to explicitly ensure the memory coherency similar to the explicit handling of the read-during-write behavior of Intel's BRAM [28].

2) Configurable Sign-Extension Mux: Although not reflected in Algorithm 1, the $\mathbf{W_1}$ and $\mathbf{W_2}$ vectors from the main BRAM need to be sign-extended before being copied to the dummy array in order to prevent overflow during MAC2. To support this, two configurable sign-extension muxes are added between the main BRAM and the dummy array. Each mux has five identical blocks, one of which is shown in Fig. 3(b). Since the main BRAM has a data width of 40 bits, it can copy five 8-bit, ten 4-bit, or twenty 2-bit elements to the dummy array simultaneously. Each of the five identical mux blocks can sign-extend one 8-bit element to one 32-bit element (blue crosses in Fig. 3(b)), or two 4-bit elements to two 16-bit elements (green crosses in Fig. 3(b)), or four 2-bit elements to four 8-bit elements (red crosses in Fig. 3(b)). Moreover, since a 2/4/8-bit MAC2 only requires a maximum bit-width of 5/9/17 bits to store the result, the proposed sign-extension mux can provide a higher bit-width required by MAC2. This allows multiple sequential MAC2 results to be accumulated by adding the 6th row (that stores the MAC2 result **P**) and the 7th row (that stores the **Accumulator**) of the dummy array.

3) Bit-Parallel SIMD Adder with Read/Write Circuits: The 160-bit SIMD adder in BRAMAC is designed using the conventional 1-bit full adder as shown in Fig. 3(c). It supports bit-parallel SIMD addition by configuring itself to twenty 8-bit adders, ten 16-bit adders, and five 32-bit adders for 2-bit, 4-bit, and 8-bit MAC2, respectively, giving a worst-case delay equal to 32-bit addition. The two operands **A** and **B** of the SIMD adder come from two sense amplifiers, **SA1** and **SA2** that compare the voltage differential of two bit-line pairs, (**BLA**, **BLbA**) and (**BLB**, **BLbB**). To support the addition followed by 1-bit shift-left operation (required in lines 6 and 9 of Algorithm 1), a write-back mux **M1** before the write driver **WD1** is used to select either sum **S** from the current full adder or sum from the right full adder $\mathbf{S_{Right}}$. **M1** can also select **ramA** to copy the first data $\mathbf{W_1}$ from the main BRAM. Similarly, a write-back mux **M2** before the write driver **WD2** is used to select between three signals: **B-bar** to perform inverting, **ramB** to copy the second data $\mathbf{W_2}$ from the main BRAM, and 1'b0 to initialize either **P** (line 1) or the **Accumulator**. Both **M1** and **M2** are controlled by the eFSM.

IV. BRAMAC Variants

A. BRAMAC with Two Synchronous Dummy Arrays (2SA)

This variant, called BRAMAC-2SA, has two synchronous dummy arrays that share the same clock domain as the main BRAM. In this architecture, each dummy array is fed by one port of the main BRAM during weight copy. Since BRAMAC intrinsically supports multiplying the same input with many weights as discussed in Section III-B, this variant adopts an input-sharing approach to balance the data reuse. Specifically, in each MAC2 iteration, the two dummy arrays copy the same weights but process different inputs. The first dummy array receives two inputs I_1, I_2 and calculates $W_1I_1 + W_2I_2$, while the second dummy array receives another two inputs I_3, I_4 and calculates $W_1I_3 + W_2I_4$.

An example 4-bit MAC2 operation for one dummy array of BRAMAC-2SA is illustrated in Fig. 4. Note that we are displaying 2 out of 10 lanes with 10-bit sign-extension due to space limitation (instead of 16-bit sign-extension as described in Section III-C2). In **Cycle 1** and **Cycle 2**, $\mathbf{W_1}$ and $\mathbf{W_2}$ are sign-extended and copied to the dummy array. During these two cycles, the two inputs for each dummy array are also sent to BRAMAC-2SA through the CIM instruction and latched for further processing. In **Cycle 3**, $\mathbf{W_1}$ and $\mathbf{W_2}$ are read out and added. The sum is written back to the 4th row to store **(W_1 + W_2)**. Simultaneously, the 6th row **P** can also be initialized to zero. In **Cycle 4**, the MSB of two inputs is streamed to the dummy array. The selected row **W1** is inverted to prepare for

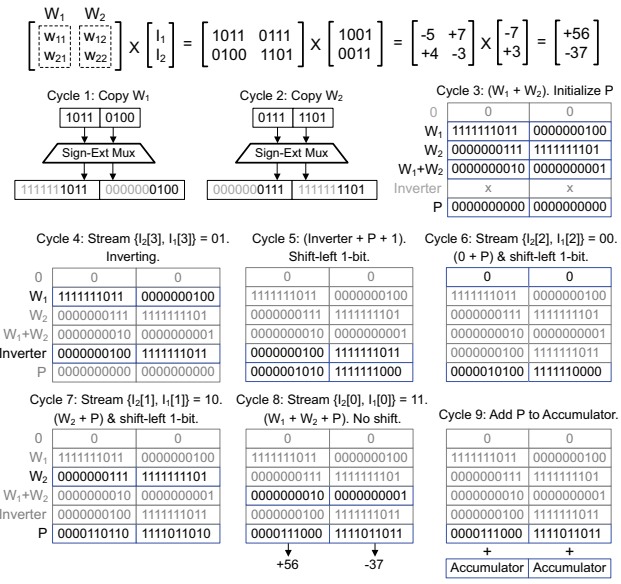


Fig. 4: Example operation of one dummy array in BRAMAC-2SA for 4-bit MAC2. We are displaying 2 out of 10 lanes with 10-bit sign extension instead of 16 bits (due to space limitation).

the binary subtraction. In **Cycle 5**, **Inverter** is added to **P**. The sum is shifted left by 1-bit and written back to **P**. The input streaming continues to **Cycle 8** where the LSB of two inputs is processed and the correct MAC2 result **P** is obtained. In **Cycle 9**, **P** is added to the 7th **Accumulator** row. Then it can be initialized for the subsequent MAC2.

The above example indicates that BRAMAC-2SA can complete a 4-bit MAC2 using 9 cycles. However, during the write-back phase of the last two cycles, i.e., **Cycle 8** and **Cycle 9**, the current two weights $\mathbf{W_1}$ and $\mathbf{W_2}$ are no longer needed in the dummy array since the current MAC2 result **P** is already obtained at the bit-parallel adder's output. As a result, these two cycles can also be used to copy the next two weights $\mathbf{W_3}$ and $\mathbf{W_4}$, respectively as illustrated in Fig. 5(a). Therefore, the 4-bit MAC2 in BRAMAC-2SA only requires 7 cycles to complete. This pipelining can also be applied to 2-bit and 8-bit MAC2. The only difference between 2-bit, 4-bit, and 8-bit MAC2 is the number of cycles spent for processing every input bit as described in line 2-11 of Algorithm 1. Thus, 2-bit and 8-bit MAC2 can take 5 and 11 cycles to complete, respectively.

B. BRAMAC with One Double-Pumped Dummy Array (1DA)

This variant, called BRAMAC-1DA, has only one dummy array to reduce the area overhead. Using one dummy array degrades the MAC throughput by 2× compared to BRAMAC-2SA, however, we propose to double-pump the dummy array with a 2× main BRAM clock frequency. Memory multi-pumping is a commonly used technique in FPGA design to improve the system throughput [29], [30]. The double-pumped dummy array doesn't add any additional area overhead compared to a synchronous dummy array. Rather, it only requires a separate clock routing during compilation.

Because the main BRAM and the dummy array only interact during weight copy, synchronization between them can be

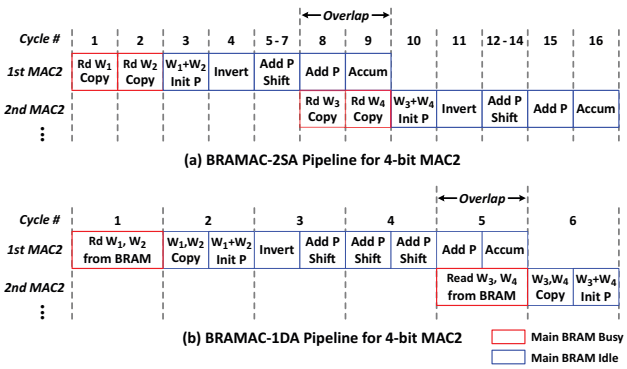


Fig. 5: Pipeline diagram of 4-bit MAC2 in (a) BRAMAC-2SA and (b) BRAMAC-1DA.

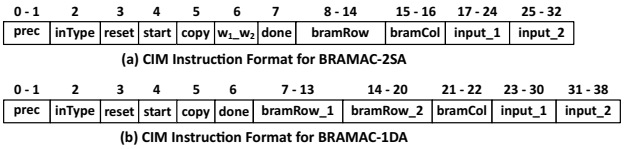


Fig. 6: CIM instruction format for (a) BRAMAC-2SA and (b) BRAMAC-1DA.

easily handled. Fig. 5(b) shows the pipeline diagram of 4-bit MAC2 for BRAMAC-1DA. In **Cycle 1**, the main BRAM reads out two weights $\mathbf{W_1}$ and $\mathbf{W_2}$. In the first half of **Cycle 2**, the dummy array copies $\mathbf{W_1}$ and $\mathbf{W_2}$ using its two write ports. Then the dummy array can compute the MAC2 using the same operation flow as BRAMAC-2SA, except that every cycle in BRAMAC-2SA is now half a cycle in BRAMAC-1DA. Similar to the pipelining optimization for BRAMAC-2SA, the main BRAM can start to read the next two weights $\mathbf{W_3}$ and $\mathbf{W_4}$ in **Cycle 5** while the dummy array is computing. As a result, the 4-bit MAC2 can be completed using 4 cycles. This pipelining can also be applied to 2-bit and 8-bit MAC2. Hence, 2-bit and 8-bit MAC2 can take 3 and 6 cycles to complete, respectively.

C. Embedded FSM to Free Up BRAM Ports

Since the dummy array's behavior is deterministic for computing MAC2, we propose to control it using an eFSM. This eFSM receives a CIM instruction to trigger the MAC2 computation and control the dummy array's read/write access. The CIM instruction is only required when the main BRAM needs to send data to the dummy array (indicated by the red boxes in Fig. 5). As a result, the main BRAM is busy for 2 cycles in BRAMAC-2SA and 1 cycle in BRAMAC-1DA. When the main BRAM is idle, it can perform normal read operations to feed LBs/DSPs or write operations to load the next tile of weights from off-chip DRAM, allowing tiling-based DNN acceleration. This is different from CCB and CoMeFa whose BRAM ports are always busy during CIM.

Fig. 6(a) and (b) show the proposed CIM instruction format for BRAMAC-2SA and BRAMAC-1DA, respectively. For BRAMAC-2SA, **bramRow** and **bramCol** are combined to form one BRAM address during each copy operation. On the other hand, BRAMAC-1DA needs to receive two BRAM

TABLE I: Resource Counts and Area Ratio of the Baseline Arria 10 GX900 FPGA.

Resource	Count	Area Ratio
Logic Blocks (LBs)	33920	70.4%
DSP Units	1518	9.5%
BRAMs (M20K)	33920	20.1%

addresses at the same time. This is achieved by using two BRAM row addresses **bramRow1** and **bramRow2** with a shared column address **bramCol**.

The two BRAMAC variants share some common control signals. The 2-bit **prec** specifies one of the three supported MAC2 precisions. The **inType** is used to indicate whether the two inputs are signed or unsigned. If the inputs are unsigned, then the inverting cycle can be skipped to improve performance. The **reset** resets the dummy array to the initial state and writes zero to its accumulator. When the **start** is enabled, BRAMAC is triggered to perform MAC2. The **copy** tells BRAMAC to copy the data read from the main BRAM to the dummy array, and an additional $\mathbf{w_1\_w_2}$ signal is needed for BRAMAC-2SA to indicate the currently copied data is W_1 or W_2. These two signals also allow the efficient pipelining optimization in Fig. 5 where the weight copy of the next MAC2 can be overlapped with computing the current MAC2. The **done** indicates whether to read out the dummy array's accumulator. When it's enabled, the **bramCol** is used to select 40-bit data from the dummy array's accumulator row every cycle. As a result, between every two dot products, the main BRAM needs to be busy for 8 and 4 cycles to read out the accumulator in BRAMAC-2SA and BRAMAC-1DA, respectively. However, as the dummy array's accumulator has a size of 8/16/32-bit for 2/4/8-bit MAC precisions, it can process a maximum dot product size of 16/256/2048 before being read out to amortize this cost.

V. Circuit-Level Evaluation

A. Tools and Baseline FPGA

We use COFFE [31], an automatic FPGA transistor sizing tool, to model and optimize the area and delay of all BRAMAC components except for the eFSM which is implemented in SystemVerilog to verify its functionality. We use Synopsys Design Compiler with TSMC 28-nm technology to synthesize and get the area of the eFSM, which are 137 μm^2 and 81 μm^2 for BRAMAC-2SA and BRAMAC-1DA, respectively after scaling to 22-nm. We get the area of an M20K block from COFFE by interpolating between 16 kb and 32 kb BRAMs. For delay estimation, COFFE runs Hspice simulations using the 22 nm Predictive Technology Model [32].

For the baseline FPGA in the remainder of this paper, we use an Arria-10 GX900 device [33] at the fastest speed grade (10AX090H1F34E1SG) whose resource information is shown in Table I. The Arria-10 device family is fabricated using 20-nm technology similar to COFFE's simulation setup. The area ratio for each resource type is estimated based on the area model in [34]. The proposed BRAMAC architecture enhances the baseline FPGA by replacing all M20K blocks with either BRAMAC-2SA or BRAMAC-1DA.

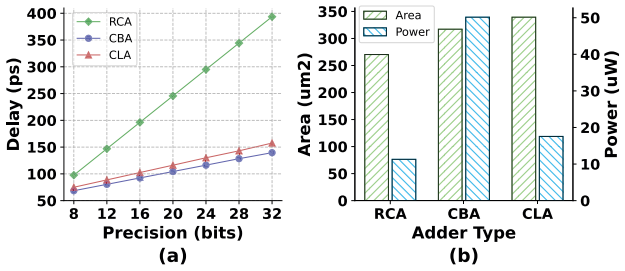


Fig. 7: Comparison between RCA, CBA, and CLA: (a) Delays vs. precision. (b) Area and power at 32-bit precision.

B. Design Choice for Adder

As the SIMD adder in BRAMAC has a worst-case delay of 32-bit addition during 8-bit MAC2, a ripple-carry adder (RCA) can significantly increase the critical path delay of the dummy array and become the frequency bottleneck of BRAMAC. Hence, we also explore two variants of fast adders [35]: Carry Lookahead Adder (CLA) with a 4-bit carry lookahead generator using mirror implementation, and Carry Bypass Adder (CBA) with 4-bit Manchester carry chain using dynamic logic. We use COFFE to automatically size the carry-out generator, the carry lookahead generator, and the Manchester carry chain to obtain the best area-delay trade-off for RCA, CLA, and CBA, respectively.

Fig. 7 illustrates the performance, area, and power of three different adders RCA, CBA, and CLA based on COFFE simulations. As shown in Fig. 7(a), the performance gap between RCA and two other fast adders CBA/CLA becomes larger as the adder precision increases. At the highest adder precision, i.e., 32-bit accumulation during 8-bit MAC, RCA has a delay of 393.6 ps, which is 2.8× slower than CBA (139.6 ps) and 2.5× slower than CLA (157.6 ps). As illustrated in Fig. 7(b), all three adders have similar areas, but CBA has the highest power consumption of 50.2 μW, which is 4.44× and 2.86× higher than RCA (11.3 μW) and CLA (17.6 μW), respectively. This is because that CBA employs the dynamic Manchester carry chain which is faster but more power-hungry than static CMOS logic. Overall, CLA has the best trade-off between delay, area, and power. Hence, we adopt CLA in BRAMAC for the remainder of our evaluation.

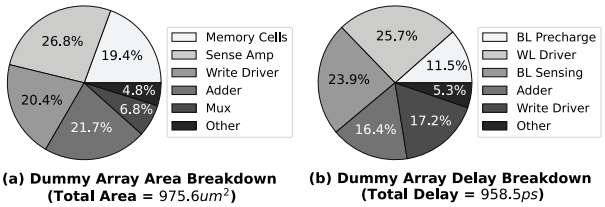
C. BRAMAC Area and Frequency

Fig. 8(a) illustrates the area breakdown of BRAMAC's dummy array. The total area of a dummy array is 975.6 μm^2, which represents an area increase of 16.9% compared to the baseline M20K. Since M20K constitutes 20.1% area of the baseline FPGA, this area overhead is equivalent to only 3.4% increase in the FPGA core area. Note that we ignore the area overhead of eFSM in our later evaluation because COFFE's area model doesn't include any BRAM control logic and some M20K components such as error correction circuits [25]. Given that the eFSMs of BRAMAC-2SA/BRAMAC-1DA are equivalent to only 1.4%/2.4% of the baseline M20K area, it's expected that the area overhead of BRAMAC doesn't change compared to the baseline M20K when a more accurate area model is adopted.

TABLE II: Key Features of BRAMAC and Prior State-of-the-art MAC Architectures for FPGA

Architecture		eDSP [15]	PIR-DSP [16]	CCB [17]	CoMeFa-D [18]	CoMeFa-A [18]	BRAMAC-2SA	BRAMAC-1DA
Modified FPGA Block		DSP	DSP	BRAM	BRAM	BRAM	BRAM	BRAM
Supported MAC Precision (-bit)		4, 8	2, 4, 8	Arbitrary	Arbitrary	Arbitrary	2, 4, 8	2, 4, 8
Area Overhead (Block)		12%	28%	16.8%	25.4%	8.1%	33.8%	16.9%
Area Overhead (Core)		1.1%	2.7%	3.4%	5.1%	1.6%	6.8%	3.4%
Clock Period Overhead over the Baseline FPGA Block		0%	30%	60%	25%	150%	10%	46%
# of MACs in Parallel / MAC Latency (Cycles) [1]	2-bit	8 / 1	24 / 1	160 / 16	160 / 16	160 / 16	80 / 5	40 / 3
	4-bit	8 / 1	12 / 1	160 / 42	160 / 42	160 / 42	40 / 7	20 / 4
	8-bit	4 / 1	6 / 1	160 / 113	160 / 113	160 / 113	20 / 11	10 / 6
Design Complexity		Very Low	Very Low	High	Low	Medium	Low	Medium

[1] For DSP architectures, the accumulator size for each MAC precision is the same as that in the baseline DSP. For BRAM architectures, the accumulator sizes for 2-bit, 4-bit, and 8-bit MACs are 8-bit, 16-bit, and 27-bit, respectively. The MAC latency is reported based on unsigned multiplication for CCB and CoMeFa, and 2's complement multiplication for BRAMAC.



(a) Dummy Array Area Breakdown (Total Area = 975.6um^2) **(b) Dummy Array Delay Breakdown (Total Delay = 958.5ps)**

Fig. 8: (a) Area and (b) delay breakdown of the dummy array.

Fig. 8(b) shows the critical path delay breakdown of BRAMAC's dummy array. With only 7 rows, the dummy array's bitline parasitic load is significantly reduced compared to the main BRAM. As a result, it can precharge and discharge much faster, giving less than 1 ns critical path delay. This suggests that the dummy array itself is able to run at a maximum frequency (F_{max}) of 1 GHz independent from M20K whose F_{max} is 730 MHz in Arria-10 [28]. For BRAMAC-1DA, this limits the F_{max} of M20K to 500 MHz in CIM mode. While this is less than the typical BRAM F_{max}, realistic FPGA delays are usually constrained by soft logic and routing, and it is unlikely that a design on Arria-10 will achieve a frequency higher than 500 MHz. For BRAMAC-2SA, the critical path occurs during the weight copy where the write-back phase can only start after reading out data from the main BRAM. Hence, the F_{max} of BRAMAC-2SA is dependent on M20K. Specifically, the dummy array's write driver has a delay of 165 ps, leading to a 1.1× lower F_{max} compared to the baseline M20K.

D. Comparison with Other MAC Architectures on FPGA

We compare BRAMAC with other state-of-the-art architectures for MAC on FPGA, including eDSP [15], PIR-DSP [16], CCB [17], and CoMeFa [18]. All architectures use the same baseline Arria-10 FPGA as described in Section V-A. Each architecture replaces the corresponding FPGA block in the baseline with its proposed new block. The key features for each studied architecture are summarized in Table II.

Due to bit-serial arithmetic, CCB and CoMeFa have the highest flexibility in the supported precision. However, their proposed bit-serial algorithms for fixed-point multiplication only work for unsigned numbers, while eDSP, PIR-DSP, and BRAMAC can support 2's complement MAC. Although BRAMAC-2SA has the highest area overhead, it achieves the highest frequency compared to other BRAM architectures. The two DSP architectures have the lowest design complexity as they can be implemented in digital CAD flow, while BRAM design typically involves analog components and manual layout effort [31]. Among all BRAM architectures, CCB has the highest design complexity as it needs an extra voltage supply. CoMeFa-A and BRAMAC-1DA have medium design complexity since they require novel timing design techniques—sense amplifier cycling and a double-pumped clock, respectively.

VI. Application-Level Evaluation

A. Peak MAC Throughput Comparison

We compare the peak MAC throughput of the baseline FPGA with those of enhanced FPGAs that employ BRAMAC and other MAC architectures studied in Section V-D. We consider three MAC precisions: 2-bit multiply (with an 8-bit accumulator), 4-bit multiply (with a 16-bit accumulator), and 8-bit multiply (with a 27-bit accumulator). The peak MAC throughput of each resource type is determined as follows:
(1) LB: We synthesize, place, and route one MAC unit using only LBs in Quartus to obtain its F_{max} and resource utilization. We then follow the same methodology as [17], [18] to calculate the total MAC throughput by optimistically assuming that all LBs can be used at the same F_{max}.
(2) DSP: The Arria-10 DSP has two 18×19 multipliers, each can implement one 8-bit MAC, two 4-bit MACs, or four 2-bit MACs using DSP packing described in [36]. We run Quartus to generate a DSP in m18x18_sumof2 mode and find its F_{max} to be 549 MHz. We use the same F_{max} for eDSP but a 1.3× lower F_{max} for PIR-DSP based on its reported F_{max}.
(3) BRAM: We use Quartus to generate the baseline M20K in simple dual-port mode and find its F_{max} to be 645 MHz. BRAMAC-2SA and BRAMAC-1DA would run at 586 MHz (1.1× lower) and 500 MHz, respectively, while CCB, CoMeFa-D, and CoMeFa-A would run 1.6×, 1.25×, and 2.5× slower, respectively based on their reported F_{max} degradation.

Fig. 9 shows the peak MAC throughput breakdown in TeraMACs/sec for different architectures and MAC precisions. Compared to the baseline Arria-10 device, BRAMAC-2SA/BRAMAC-1DA can improve the peak throughput by

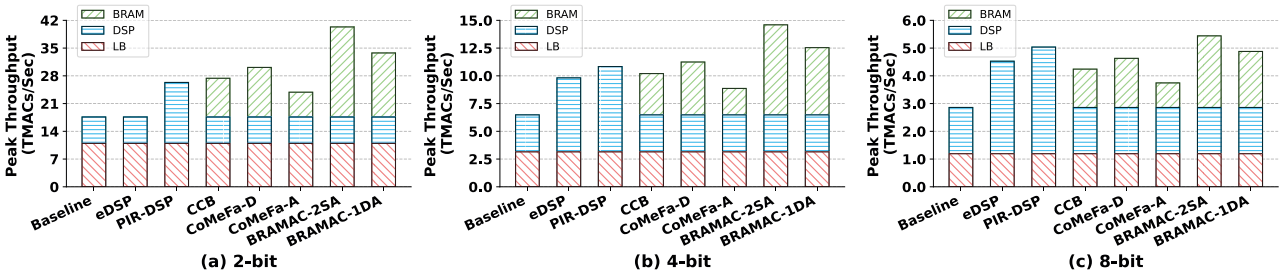


Fig. 9: Peak MAC throughput of different architectures for various MAC precisions: (a) 2-bit, (b) 4-bit, (c) 8-bit.

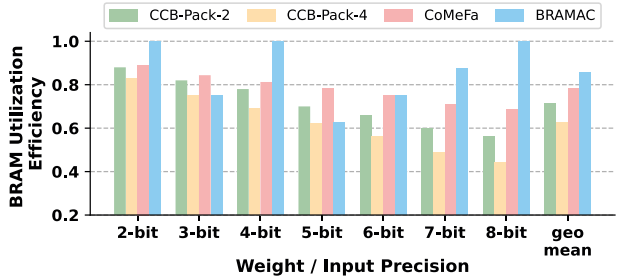


Fig. 10: Comparison of BRAM utilization efficiency for DNN model storage at different precisions.

2.6×/2.1×, 2.3×/2.0×, and 1.9×/1.7× for 2-bit, 4-bit, and 8-bit MAC, respectively. Although CCB and CoMeFa can compute 160 MACs in parallel, they suffer from long-latency bit-serial arithmetic, leading to lower throughput than BRAMAC. Compared to low-precision DSP architectures, BRAMAC-2SA can deliver higher MAC throughput across all precisions, while BRAMAC-1DA's throughput is only slightly lower than PIR-DSP for 8-bit MAC. Note that BRAMAC is an enhanced BRAM architecture, therefore doesn't preclude the use of eDSP or PIR-DSP on the same FPGA. The combination of BRAMAC and eDSP/PIR-DSP can further boost an FPGA's MAC throughput.

B. BRAM Utilization Efficiency for DNN Model Storage

Since BRAMAC computes MAC in a separate dummy array that is fully decoupled from the main BRAM, it can store a DNN model efficiently. Fig. 10 compares the BRAM utilization efficiency between BRAMAC, CCB, and CoMeFa for storing DNN models with different precisions from 2- to 8-bit. Here, utilization efficiency is defined as the effective capacity ratio of a BRAM that can be used to store weight. A higher utilization efficiency can store the DNN model using fewer BRAM blocks, saving both area and power consumption. For CCB, we examine two variants, CCB-Pack-2 and CCB-Pack-4, that map 2 and 4 sequential bit-serial MACs to the same BRAM column, respectively.

BRAMAC can achieve 100% utilization for 2-bit, 4-bit, and 8-bit precisions. Other precisions can be stored in BRAMAC with lower efficiency by sign-extending them to 4-bit or 8-bit. Despite this, BRAMAC still achieves the highest average BRAM utilization efficiency which is 1.3× and 1.1× better compared to CCB and CoMeFa, respectively. This is because CCB and CoMeFa use extra BRAM space to store temporary products and partial sums, while BRAMAC stores temporary results only in the dummy array. For CCB, a higher packing factor computes more sequential MACs before a slow in-memory reduction, giving a higher performance at the cost of more BRAM usage to save a copy of the input vector. On the other hand, CoMeFa offers a one-operand-outside-RAM mode that streams the input vector, avoiding a copy to BRAM which improves utilization efficiency when compared to CCB.

C. Performance Improvement over CCB and CoMeFa

We use general matrix-vector multiplication (GEMV) to benchmark and compare the application performance of BRAMAC, CCB, and CoMeFa. We choose BRAMAC-1DA for this experiment because it has a similar BRAM area and frequency overhead as CCB/CoMeFa. We assume that there is only one BRAM block available to perform the computation. This approach captures the performance of an architecture normalized to BRAM utilization. We consider both persistent and non-persistent (tiling-based) computations that exclude and include the cycles needed for loading the matrix data to the single BRAM block, respectively. Since the data mapping and computation flow of the three studied architectures are deterministic, we use a detailed analytical model to map a given GEMV workload to each architecture and count the number of cycles required. In addition to the latency of MAC, our analytical model accounts for latency associated with copying the input vector and reading out the accumulation results in each architecture.

Fig. 11 illustrates the speedup of BRAMAC-1DA over CCB and CoMeFa when performing GEMV with different matrix sizes, precisions (2-bit, 4-bit, 8-bit), and computation styles (persistent and non-persistent). Overall, BRAMAC-1DA achieves up to 3.3×/2.8×/2.4× (and 4.1×/3.4×/2.8×) speedups for 2/4/8-bit persistent (and non-persistent) GEMV.

At the same precision, BRAMAC-1DA achieves higher speedup for non-persistent computation thanks to its eFSM that allows loading the next matrix tile while computing on the current tile. Regarding different precisions, the speedup of BRAMAC-1DA decreases as the precision increases. This is because a higher precision directly reduces the computation parallelism of BRAMAC-1DA by 2×, and it takes more cycles to process more input bits. On the other hand, CCB/CoMeFa only sacrifice latency but not parallelism at higher precision. Nevertheless, BRAMAC-1DA still achieves better performance for all cases due to its overall MAC throughput improvement over CCB/CoMeFa as discussed in

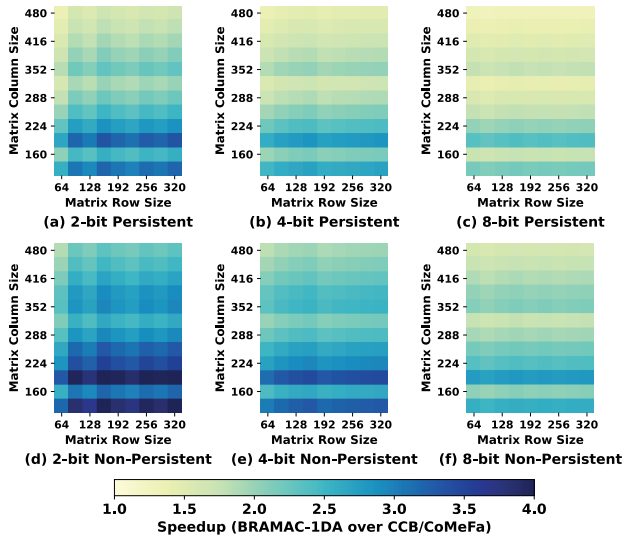


Fig. 11: Speedup (based on cycles) of BRAMAC-1DA over CCB/CoMeFa for GEMV with different matrix sizes, precisions, and computation styles.

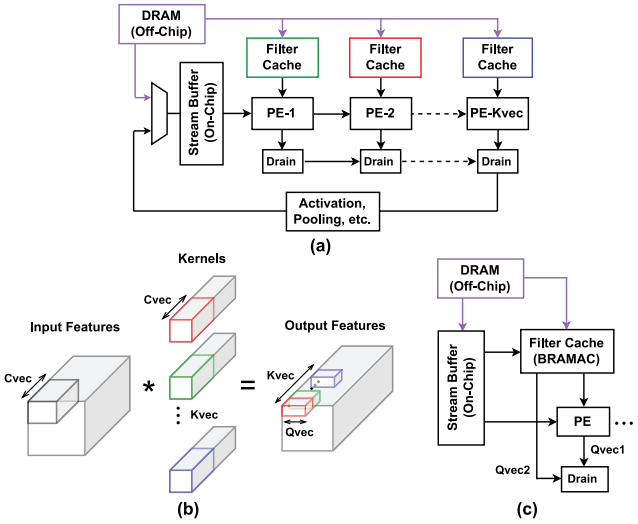


Fig. 12: DLA's (a) architecture and (b) computation parallelism across different axes for CNNs. (c) The architecture of DLA-BRAMAC (with one PE shown).

Section VI-A. Note that CCB/CoMeFa's bit-serial algorithms for fixed-point multiplication only support unsigned numbers. It's expected that they require much higher latency when supporting 2's complement MAC.

Along the matrix row size, the speedup of BRAMAC-1DA is mainly affected by the vectorization efficiency, and this effect is more pronounced at a lower precision. For example, consider the 2-bit persistent case in Fig. 11(a), where BRAMAC-1DA can compute 20 outputs simultaneously. If the matrix row size is 64, i.e., the first column in Fig. 11(a), then at least 4 iterations are required to compute an output vector of size 64, with only $64/80 = 80\%$ useful computation in BRAMAC-1DA. On the other hand, if the matrix row size is 160, i.e., the fourth column in Fig. 11(a), then the output vector divides perfectly into 8 iterations at 100% efficiency, thus giving better speedup as indicated by the darker color of the fourth column compared to the first column. Similar trends exist in 4-bit and 8-bit cases but are less pronounced.

Along the matrix column size, the speedup of BRAMAC-1DA is determined by not only the vectorization efficiency but also the achievable packing factor of CCB/CoMeFa. For example, consider the 8-bit non-persistent case in Fig. 11(f). If the matrix column size is 480, i.e., the top row in Fig. 11(f), then CCB/CoMeFa can perform 3 sequential MACs on the same BRAM column before a slow in-memory reduction to amortize the reduction's latency cost. On the other hand, if the matrix column size is 128, i.e., the bottom row in Fig. 11(f), then a reduction is necessary for CCB/CoMeFa after every bit-serial MAC, resulting in much longer latency. On the contrary, BRAMAC's dummy array doesn't require a special reduction operation. Rather, it performs in-place accumulation at the end of every MAC2.

D. Case Study: Employ BRAMAC to Intel's DLA

To demonstrate the feasibility of BRAMAC for tiling-based DNN inference with non-persistent weight storage, we employ BRAMAC to Intel's Deep Learning Accelerator (DLA) [9], [10] and develop a cycle-accurate simulator to model DLA in both the baseline FPGA and the enhanced FPGA with BRAMAC (which we call DLA-BRAMAC). The original DLA is designed to accelerate convolutional neural networks (CNNs) as shown in Fig. 12(a). It has a processing element (PE) array organized in a 1D systolic structure, a stream buffer to store input and output features, and a filter cache to store weights. It can be parameterized by Cvec, Qvec, and Kvec which represent the computation parallelism per cycle in input depth, output width dimension, and output depth, respectively as illustrated in Fig. 12(b). For DLA-BRAMAC, the stream buffer can send different input features to the PE array and the BRAMAC-based filter cache simultaneously as shown in Fig. 12(c). In this way, BRAMAC can complement the PE array to calculate different outputs along the Qvec dimension.

Similar to the approach used in the original DLA [9], we conduct design space exploration to find the optimal DLA and DLA-BRAMAC configurations (i.e., Cvec, Qvec, and Kvec) for two popular CNN models: Alexnet and ResNet-34. Our analytical model is set to optimize the target function $perf * (perf/area)$ to balance performance and area cost. It assumes that all multipliers are implemented using DSPs, and each DSP can pack one 8-bit, two 4-bit, or four 2-bit multiplications using the DSP-packing technique in [36]. For area modeling, we use the DLA area model from [9] to estimate the number of DSPs and BRAMs required for a specific configuration. We ignore the number of ALMs in our area modeling since they are mainly used to implement non-compute-intensive operations and are expected to be similar in DLA and DLA-BRAMAC. To evaluate the performance, our cycle-accurate simulator accounts for the latency associated with the MAC2 computation and the dummy array's accumulator readout. Note that BRAMAC's eFSM can effectively pipeline adjacent MAC2 operations to hide the latency of the weight copy, except for the first MAC2 of every CNN layer

TABLE III: Optimal Configurations of DLA and DLA-BRAMAC for AlexNet and ResNet-34

Model	Accelerator	DLA			DLA-BRAMAC-2SA			DLA-BRAMAC-1DA		
		Config[1]	DSPs	BRAMs	Config[2]	DSPs	BRAMs	Config[2]	DSPs	BRAMs
AlexNet	2-bit	(2, 16, 96)	1152	352	(1+2, 24, 140)	1260	1128	(2+2, 16, 100)	1200	816
	4-bit	(3, 16, 32)	1152	544	(1+2, 16, 100)	1200	1600	(1+1, 12, 130)	1170	1080
	8-bit	(3, 12, 24)	1296	868	(2+2, 10, 50)	1500	1740	(1+1, 8, 100)	1200	1664
ResNet-34	2-bit	(4, 12, 72)	1296	792	(1+2, 16, 140)	840	832	(2+2, 22, 80)	1320	924
	4-bit	(3, 8, 64)	1152	736	(2+2, 12, 70)	1260	972	(1+1, 16, 90)	1080	1056
	8-bit	(3, 4, 64)	1152	1452	(2+2, 6, 65)	1170	1530	(1+1, 12, 65)	1170	1788

[1] The configuration value for DLA has the form of (Qvec, Cvec, Kvec).
[2] The configuration value for DLA-BRAMAC has the form of (Qvec1+Qvec2, Cvec, Kvec), where Qvec1 and Qvec2 are the numbers of output features computed by DSP and BRAMAC, respectively.

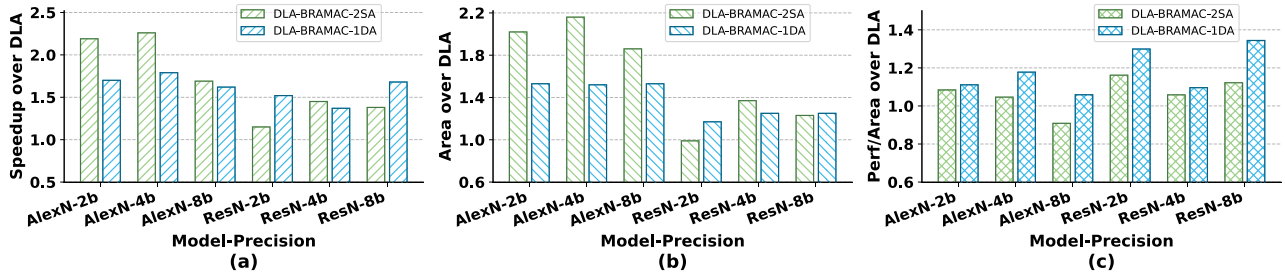


Fig. 13: Comparison between DLA and DLA-BRAMAC for accelerating AlexNet and ResNet at different precisions: a) performance, (b) utilized DSP-plus-BRAM area, (c) performance per area.

where an additional 2 cycles are required to start the initial weight copy. However, this overhead is negligible given that each CNN layer takes thousands of cycles to complete.

Table III summarizes the optimal configuration for each (accelerator, model, precision) case. The performance and utilized DSP-plus-BRAM area of DLA-BRAMAC, normalized to those of DLA, are shown in Fig. 13. The utilized DSP-plus-BRAM area is calculated based on the area overhead of BRAMAC and the area model from [34]. On average, compared to the baseline DLA for AlexNet, employing BRAMAC-2SA/BRAMAC-1DA achieves 2.05×/1.7× speedup at the cost of 2.01×/1.52× DSP-plus-BRAM area, giving 1.01×/1.12× performance gains per utilized area. For ResNet-34, employing BRAMAC-2SA/BRAMAC-1DA achieves a lower speedup of 1.33×/1.52× on average at the cost of 1.2×/1.22× DSP-plus-BRAM area, which corresponds to 1.11×/1.25× performance gains per utilized area. The larger DSP-plus-BRAM area is mainly attributed to more BRAM usage for computation and BRAMAC's area overhead.

In general, BRAMAC-2SA and BRAMAC-1DA achieve higher speedup for AlexNet compared to ResNet-34 as shown in Fig. 13(a). This is because that BRAMAC is better at supporting a higher Kvec that allows the same input feature to be multiplied by many kernels. The early and most compute-intensive residual blocks of ResNet-34 only have an output channel depth of 64, while the first convolution layer of AlexNet has an output channel depth of 96. The latter gives more freedom for DLA-BRAMAC to optimize its configuration with high vectorization efficiency. However, a higher speedup for AlexNet comes with a larger utilized area as illustrated in Fig. 13(b). Comparing the two BRAMAC variants, BRAMAC-2SA has a lower performance gain per utilized area for all model-precision combinations as observed from Fig. 13(c). Although the MAC throughput of BRAMAC-2SA is slightly improved over BRAMAC-1DA, it has 2× BRAM area overhead compared to BRAMAC-1DA. While our results more than justify the area overhead of BRAMAC, we expect higher gains for a DNN accelerator that is: (1) purpose-built around the capabilities of BRAMAC, and (2) used to accelerate DNNs with more matrix multiplications such as transformers [37]—we will work on both aspects in the future.

VII. Conclusion

This paper proposes BRAMAC, a compute-in-BRAM architecture for MAC on FPGAs. To the best of our knowledge, BRAMAC is the first compute-in-BRAM architecture that: (1) adopts a hybrid bit-serial & bit-parallel dataflow to support variable-precision MAC using 2's complement representation, (2) computes in a separate dummy array which improves the main BRAM array's utilization efficiency, (3) employs an embedded finite-state machine to free up the main BRAM ports during in-memory computation. The two proposed variants, BRAMAC-2SA/BRAMAC-1DA, boost the peak MAC throughput of a large Arria 10 FPGA by 2.6×/2.1×, 2.3×/2.0×, and 1.9×/1.7× for 2-bit, 4-bit, and 8-bit precisions, respectively at the cost of 6.8%/3.4% increase in FPGA core area. BRAMAC also improves the BRAM utilization efficiency by 1.3× and 1.1× compared to two recent compute-in-BRAM architectures, CCB and CoMeFa, respectively while significantly outperforming both architectures on matrix-vector multiplications. Combining BRAMAC-2SA/BRAMAC-1DA with Intel's DLA, a tiling-based DNN accelerator, an average speedup of 2.05×/1.7× and 1.33×/1.52× can be achieved for AlexNet and ResNet-34, respectively. With its ability to support both persistent and tiling-based DNN acceleration, BRAMAC has the potential to be a highly practical and valuable addition to future AI-optimized FPGAs.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 248–255.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, 2020, pp. 1877–1901.

[4] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," *arxiv:abs/2106.08295*, 2021.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[6] H. Wu, "Low Precision Inference on GPU," 2019. [Online]. Available: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf

[7] Nvidia, "INT4 Precision for AI Inference," 2019. [Online]. Available: https://developer.nvidia.com/blog/int4-for-ai-inference/

[8] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, 2018.

[9] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ Deep Learning Accelerator on Arria 10," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[10] M. S. Abdelfattah, D. Han, A. Bitar, R. Dicecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling, and G. R. Chiu, "DLA: Compiler and FPGA Overlay for Neural Network Inference Acceleration," *28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 411–4117, 2018.

[11] Intel, "Intel Stratix 10 NX FPGA Overview," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/nx.html

[12] Xilinx, "UltraScale Architecture DSP Slice User Guide, (UG579 v1.11)," 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp

[13] Intel, "Intel Stratix 10 Variable Precision DSP Blocks User Guide (UG-S10-DSP)," 2021. [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683832.pdf

[14] Achronix, "Speedcore eFPGAs)." [Online]. Available: https://www.achronix.com/sites/default/files/docs/Speedcore_eFPGA_Product_Brief_PB028.pdf

[15] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," *28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 35–42, 2018.

[16] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. W. Leong, "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 35–44, 2019.

[17] X. Wang, V. Goyal, J. Yu, V. Bertacco, A. Boutros, E. Nurvitadhi, C. Augustine, R. R. Iyer, and R. Das, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," *IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 88–96, 2021.

[18] A. Arora, T. Anand, A. Borda, R. Sehgal, B. Hanindhito, J. Kulkarni, and L. K. John, "CoMeFa: Compute-in-Memory Blocks for FPGAs," *IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 1–9, 2022.

[19] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, vol. 21, pp. 4–29, 2021.

[20] M. Eldafrawy, A. Boutros, S. Yazdanshenas, and V. Betz, "FPGA Logic Block Architectures for Efficient Deep Learning Inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, pp. 1–34, 2020.

[21] Xilinx, "DSP58 Architecture," 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/am004-versal-dsp-engine/DSP58-Architecture

[22] Intel, "Intel Agilex Variable Precision DSP Blocks User Guide," 2021. [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683037.pdf

[23] M. Langhammer, E. Nurvitadhi, S. Gribok, and B. M. Pasca, "Stratix 10 NX Architecture," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, pp. 1 – 32, 2022.

[24] M. Horowitz, "Computing's Energy Problem (and what we can do about it)," *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

[25] D. M. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, "Architectural Enhancements in Stratix V," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2013.

[26] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2016.

[27] C.-F. Lee, C. Lu, C.-E. Lee, H. Mori, H. Fujiwara, Y.-C. Shih, T.-L. Chou, Y. D. Chih, and T.-Y. J. Chang, "A 12nm 121-TOPS/W 41.6-TOPS/mm2 All Digital Full Precision SRAM-based Compute-in-Memory with Configurable Bit-width For AI Edge Applications," *IEEE Symposium on VLSI Technology and Circuits*, pp. 24–25, 2022.

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

[29] J. Choi, K. Nam, A. Canis, J. H. Anderson, S. D. Brown, and T. S. Czajkowski, "Impact of Cache Architecture and Interface on Performance and Area of FPGA-Based Processor/Parallel-Accelerator Systems," *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–24, 2012.

[30] R. Shi, Y. Ding, X. Wei, H. Li, H. Liu, H. K.-H. So, and C. Ding, "FTDL: A Tailored FPGA-Overlay for Deep Learning with High Scalability," *57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[31] S. Yazdanshenas, K. Tatsumura, and V. Betz, "Don't Forget the Memory: Automatic Block RAM Modelling, Optimization, and Architecture Exploration," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 115–124, 2017.

[32] Arizona State University, "Predictive Technology Model," 2012. [Online]. Available: http://ptm.asu.edu/

[33] Intel, "Arria 10 Device Overview," 2022. [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683332.pdf

[34] R. Rashid, J. G. Steffan, and V. Betz, "Comparing performance, productivity and scalability of the TILT overlay processor to OpenCL HLS," *International Conference on Field-Programmable Technology (FPT)*, pp. 20–27, 2014.

[35] University of California, Berkeley, "ECE241, Lecture 18 Adders," 2003. [Online]. Available: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241_s03/Lectures/lecture18-adders.pdf

[36] J. Sommer, M. A. Özkan, O. Keszocze, and J. Teich, "DSP-Packing: Squeezing Low-precision Arithmetic into FPGA DSP Blocks," *arxiv.org/abs/2203.11028*, 2022.

[37] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," *arxiv:abs/1706.03762*, 2017.