POLITECNICO DI TORINO
Repository ISTITUZIONALE

Memory-System Design for Precision-Scalable Systolic Arrays

Original
Memory-System Design for Precision-Scalable Systolic Arrays / Solimine, M., Manca, E., Lagostina, L., Lavagno, L., Casu, M.R.. - ELETTRONICO. - (2026), pp. 1-7. (2026 IEEE Nordic Circuits and Systems Conference (NorCAS) Tampere (Fin) 20-21 October 2026) [10.1109/norcas71164.2026.11791141].

Availability:
This version is available at: 11583/3018450 since: 2026-11-26T09:01:05Z

Publisher:
IEEE

Published
DOI:10.1109/norcas71164.2026.11791141

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

27 November 2026

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: https://doi.org/10.1109/NorCAS71164.2026.11791141

Memory-System Design for Precision-Scalable Systolic Arrays

Michele Solimine, Edward Manca, Lorenzo Lagostina, Luciano Lavagno, and Mario R. Casu
Department of Electronics and Telecommunications, *Politecnico di Torino*, Turin, Italy
{edward.manca, lorenzo.lagostina, luciano.lavagno, mario.casu}@polito.it

***Abstract*—Precision-Scalable Systolic Arrays (PS-SAs) increase low-precision throughput by executing multiple operations in parallel within each Processing Element. This capability, however, creates strong precision- and architecture-dependent bandwidth requirements, making the local memory system a first-order design constraint. This paper presents a design methodology and a design-space exploration flow for memory systems tailored to PS-SAs. BitFusion (BF), BitBlade (BB), and BitShare (BS) are integrated into a common weight-stationary baseline supplied by separate input, weight, activation, and output memories. The proposed memory architecture incorporates precision-dependent packing and unpacking logic and exposes two design knobs that define the design-space. We synthesize a total of 76 configurations in a 28-nm CMOS technology using foundry SRAM macros. Iso-performance area results show that the conventional monolithic architecture with register-based skewing organization is consistently dominated by partitioned hybrid solutions, while all minimum-area memory configurations encode the skewing inside the SRAM layout. Considering the overall system including memory and the SA, we found that the BB-based PS-SA achieves the smallest total area, 0.778 mm^2, compared with 0.946 mm^2 for BF and 2.968 mm^2 for BS. These results demonstrate that memory organization is decisive in system-level comparisons of PS-SA architectures.**

***Index Terms*—Memory System Design, Precision Scalable Multipliers, Systolic Arrays, Deep Neural Networks, Mixed Precision**

I. Introduction

Systolic Arrays (SAs) are widely used as computing accelerators for Matrix Multiplication (MatMul), an operation prevalent in Deep Neural Networks (DNNs) [1]. Their simple Processing Element (PE) structure and neighbor-to-neighbor communication enable high performance and efficiency. In recent years, several successful SA designs have been proposed by both industry [2] and academia [3], [4], [5].

Modern DNNs employ an increasingly large number of parameters. To balance accuracy and model size, Mixed-Precision DNNs (MP-DNNs) have recently emerged [6]. Since different layers may use different numerical precisions, *precision scalability* has been introduced to efficiently support MP-DNN execution on dedicated hardware. Precision-Scalable (PS) PEs can execute either a single operation on full-precision operands (e.g., 8 bits) or multiple parallel operations on lower-precision operands (e.g., 4 or 2 bits), by reusing and rearranging their constituent Sub-Units (SUs).

Many architectures have been proposed for PS-PEs [7], [8], [9], [10]. In [7], a 16-bit Baugh-Wooley multiplier reuses the same logic to perform 1, 2, 4, or 8 operations on 16-, 8-, 4-, and 2-bit operands, respectively, while keeping input and output bitwidth constant. This scheme has been adopted in CPU-centric designs [11], [12] and 2D parallel architectures [13]. BitFusion (BF) [14] instead builds an 8-bit multiplier from sixteen 2×2 SUs, called *bitbricks*, which can operate independently or be combined as eight 4×2, four 4×4 or 8×2, two 8×4, or one 8×8 multipliers. Similar approaches are adopted by BitBlade (BB) [15] and BitShare (BS) [16]; BB reorganizes the SUs to reduce area at low frequencies, while BS targets lower input and weight bitwidth. Moreover, BB was shown in [15] to outperform [13], based on the computing scheme of [7]. Compared with [7], BF-, BB-, and BS-like solutions provide greater hardware reuse and higher throughput, but require a quadratically increasing number of operands as precision decreases. At 2-bit precision, an 8-bit PS multiplier based on [7] requires $2\times4\times2 = 16$ input bits, whereas [14] requires $2\times16\times2 = 64$ bits. Thus, input bitwidth is four times larger, shifting the design challenge toward the memory system. To combine high throughput and an efficient memory system, we focus on the BF, BB, and BS approach.

In this paper, we propose a memory-system design methodology tailored to PS-SAs. This is challenging because PS-SA bandwidth requirements vary significantly with arithmetic precision and PS-PE parallelism. Previous works [3], [17], [18], [19] have proposed memory systems for conventional SAs. However, none of them systematically investigates the architectural trade-offs introduced by integrating PS-PEs into a SA. We integrate BF, BB, and BS architectures into a common baseline sharing implementation, control, and memory interfaces. This enables a fair comparison of their memory-system requirements under the same architectural assumptions. Although the BF, BB, and BS papers mention dedicated memory organizations, they do not provide sufficient details for reproduction. Rather than reproducing those memory implementations, our contribution is to systematically investigate how memory systems for PS-SAs should be organized.

A further aspect of SA design is the *skewing delay*, i.e., the temporal offset between data supplied to consecutive PEs along the SA boundary, required to ensure that operands meet at the right clock cycle in an SA. We found three strategies to implement it. The most common uses Skewing Registers (SRs) between memory and PEs [4], [5], [17], regardless of the memory banking. A second approach [19] partitions the memory so that each bank serves one PE and implements skewing through constant offsets between the banks read/write

TABLE I
PS-SAs bandwidth and tile size requirements for input, weights and partial sums at each precision. The PS-SA size is reported in the name, while the operations performed by the PS-PE are reported as (m, n, h).

	BF ($\mathbf{R{=}16, C{=}16, D{=}16}$)			**BB** ($\mathbf{R{=}16, C{=}1, D{=}16}$)			**BS** ($\mathbf{R{=}16, C{=}1, D{=}16}$)		
Prec.	(m, n, h)	bitwidth [bits] $(I,\ W,\ A/O)$	tile size [bytes] $(I,\ W,\ A/O)$	(m, n, h)	bitwidth [bits] $(I,\ W,\ A/O)$	tile size [bytes] $(I,\ W,\ A/O)$	(m, n, h)	bitwidth [bits] $(I,\ W,\ A/O)$	tile size [bytes] $(I,\ W,\ A/O)$
8x8	(1,1,1)	(128, 128, 512)	(256, 256, 1024)	(1,1,16)	(128, 128, 512)	(256, 256, 1024)	(1,1,16)	(128, 128, 512)	(256, 256, 1024)
8x4	(1,1,2)	(128, 256, 448)	(256, 512, 896)	(1,1,32)	(128, 256, 448)	(256, 512, 896)	(1,2,16)	(128, 128, 896)	(256, 256, 1792)
8x2	(1,1,4)	(128, 512, 416)	(256, 1024, 832)	(1,1,64)	(128, 512, 416)	(256, 1024, 832)	(1,4,16)	(128, 128, 1664)	(256, 256, 3328)
4x4	(1,1,4)	(256, 256, 384)	(512, 512, 768)	(1,1,64)	(256, 256, 384)	(512, 512, 768)	(2,2,16)	(128, 128, 1536)	(256, 256, 3072)
4x2	(1,1,8)	(256, 512, 352)	(512, 1024, 704)	(1,1,128)	(256, 512, 352)	(512, 1024, 704)	(2,4,16)	(128, 128, 2816)	(256, 256, 5632)
2x2	(1,1,16)	(512, 512, 320)	(1024, 1024, 640)	(1,1,256)	(512, 512, 320)	(1024, 1024, 640)	(4,4,16)	(128, 128, 5120)	(256, 256, 10240)

Algorithm 1 MatMul (a) before and (b) after tiling. L_x denotes the loops introduced by the x-th level of tiling.

(a) **for** $i \in [0, I)$ **do**
  for $j \in [0, J)$ **do**
    for $k \in [0, K)$ **do**
      $O[i][j] \leftarrow O[i][j] + I[i][k] \times W[k][j]$

(b) **for** $t_i \in [0, I/(D{\times}n))$ **do**
  for $t_j \in [0, J/(R{\times}m))$ **do**
    for $t_k \in [0, K/(C{\times}h))$ **do**
      L_1 { **for** $t_0 \in [0, D)$ **do** — (*temporal loop*)
        for $s_1 \in [0, R)$ **do** — (*spatial loop - systolic*)
          for $s_0 \in [0, C)$ **do** — (*spatial loop - systolic*)
            L_2 { **for** $d_2 \in [0, m)$ **do** — (*spatial loop - parallel*)
              for $d_1 \in [0, n)$ **do** — (*spatial loop - parallel*)
                for $d_0 \in [0, h)$ **do** — (*spatial loop - parallel*)
                  $(i, j, k) \leftarrow$ fn(loop indices)
                  $O[i][j] \leftarrow O[i][j] + I[i][k] \times W[k][j]$

addresses. A third option modifies the SRAM data layout so that each word contains the values required by the PE at the SA boundary at a given cycle. To the best of our knowledge, this approach has not been explored in any previous works.

We also explore hybrid combinations of these three strategies through a parametric memory architecture with two design-knobs. M defines the number of memory blocks (mem-blocks) into which each memory is partitioned; when $M{>}1$, the mem-blocks use the address-offset technique of [19]. S determines whether the remaining skewing within each mem-block is implemented using SRs or encoded in the SRAM data layout. By varying M and S, we explore the memory-system trade-offs of BF-, BB-, and BS-based PS-SAs.

The rest of the paper is organized as follows. Sec.,II describes the PS-SA architecture, Sec. III details the memory system and its design knobs, Sec. IV presents the design-space exploration, and Sec. V summarizes and concludes the paper.

II. Precision-Scalable Systolic Arrays

SAs exploit the *spatial* and *temporal reuse* of MatMul loop nests. The three-loop MatMul is shown in Alg.,1, omitting zero initialization for compactness. Spatial loops can be executed in parallel; therefore, for full spatial mapping, the product of their iteration counts determines the number of PEs. Instead, temporal loops define the computations performed by each PE over time. Spatial reuse occurs when a value is forwarded across neighboring PEs, whereas temporal reuse occurs when a PE retains and reuses a value across iterations. We adopt the most common weight-stationary (WS) dataflow, where weights are temporally reused, while inputs and accumulated results are spatially reused. In the WS dataflow, the spatial loops over j and k assign weight $W[k][j]$ to a PE, while time evolves over i; thus, $I[i][k]$ is used by the k-th PE column and $O[i][j]$ is communicated along the j-th PE row.

As shown in Alg.,1, the MatMul mapped onto the SA is transformed using *tiling*, which partitions the iteration space of Alg.,1a into smaller chunks that enable better data reuse. In Alg.,1b, a first tiling level ($L1$) partitions loops (I, J, K) into tiles of size $(D{\times}m, R{\times}n, C{\times}h)$, where R and C are the SA dimensions, D is the number of temporal steps per tile, and (m, n, h) defines the PS-PE parallelism, which varies according to the precision as shown below. When $(m, n, h) > (1, 1, 1)$, more than one operation is performed within the PE, which reduces the number of iterations and virtually increases the size of the tile being processed, as multiple low-precision data get packed in the same memory space. The parameter D determines for how long stationary values are reused. Loops $(s0, s1, t0)$ describe the computations mapped across the PS-PEs, with $t0$ representing the temporal dimension and $(s0, s1)$ the spatial ones. A second tiling level ($L2$), with loops $(d0, d1, d2)$ having iteration counts (m, n, h), describes the operations performed internally by each PS-PE. Hence, each PS-PE acts as a small MatMul unit computing $m{\times}n{\times}h$ iterations in parallel.

Tab. I summarizes, for each architecture and precision, the corresponding (m, n, h) parallelism, the per-cycle bitwidth requirements of inputs (I), weights (W), and partial sums/outputs (O), and the corresponding tile sizes. These tile sizes are later used, together with the interface bitwidths, to determine the required SRAM width and capacity. Specifically, a suitable SRAM macro must provide a data width sufficient to satisfy the worst-case bitwidth, and a sufficient capacity to store the worst-case tile size.

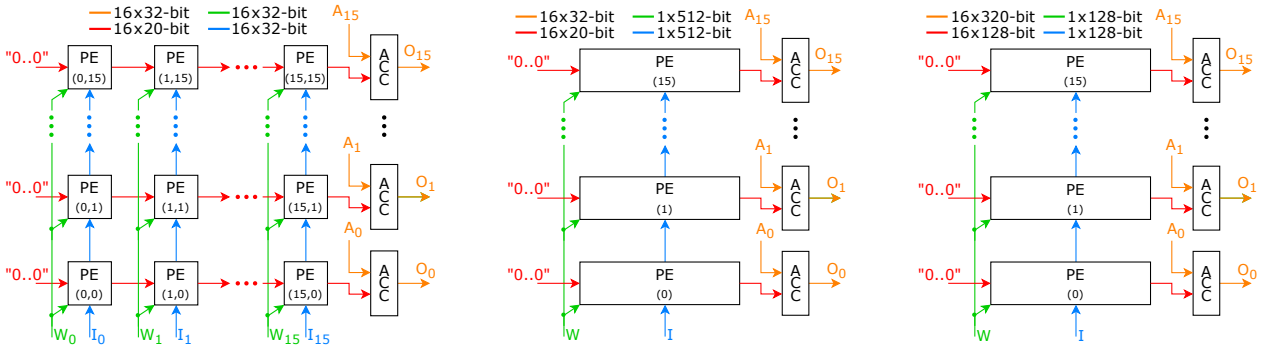

Fig. 1. (a) PS-SA for the BF PS-PE; (b) PS-SA for the BB PS-PE; (c) PS-SA for the BS PS-PE.

As a first example, consider the BF case in Tab. I. The PE inner parallelism is such that one multiplication is performed at 8×8 precision with $(m, n, h) = (1, 1, 1)$, while the parallelism increases as the precision scales down, reaching sixteen parallel multiplications at 2×2 precision with $(m, n, h) = (1, 1, 16)$. The BF-PE internal arrangement is such that only the K dimension of the original loop is affected by the increased parallelism, reducing the number of iterations as h increases from K/C down to $K/(C\times16)$.

The increased parallelism and throughput at lower precision determine an increased bitwidth for inputs, weights, and partial sums. This is clarified by Fig.,1a. Following [14], the BF-based PS-SA has R=C=16, for a 16×16 array of PS-PEs, each containing a precision-scalable unit and registers for inputs (I), weights (W), and partial sums (A). At each cycle, a PE receives a new input from below and a partial sum from the left, through the blue and red connections, while the weights, shown in green, are stored once and reused for the next D steps[1]. The input and weight bitwidth requirement of the PE scales from 8 bits at 8×8 precision to 32 bits at 2×2. As a result, to cover this last worst case, the sixteen blue and green connections carry 32 bits each, totaling 512 bits as shown in Tab.,I. Consequently, the tile sizes of I and W increase from 256 bytes at 8×8 to 1024 bytes at 2×2. As for the partial sums, the number of bits must also account for the accumulation range, which we discuss at the end of the section.

For the BB case, Tab. I shows that to achieve the same throughput of the BF case, a uni-dimensional array with R=16, C=1 with larger PEs is built. At 8×8 precision, each PE already performs 16 multiplications with (m, n, h)=$(1, 1, 16)$, which scale up to $(1, 1, 256)$ at 2×2 precision. Fig. 1b shows the architecture[2]. The same reasoning as BF leads to a total of 512 bits for inputs and weights, and a tile size of 1024 bytes.

[1]Weight distribution is the only difference in our BF implementation with respect to [14]. The weights are broadcast along dedicated column lines, with one PE per column loading a new weight each cycle, which is then reused for the next D steps.

[2]In our implementation, we transform the shared input connection of [15] into a more efficient systolic path, so that the input values are forwarded from one PS-PE to the next.

Tab. I shows that the implementation of BS is significantly different in terms of (m, n, h) parallelism of the PE. The overall SA has the same 1D arrangement of the BB case, as shown in Fig. 1c, but each precision has a different organization of the parallelism that guarantees a fixed total bitwidth and tile size equal to 128 bits and 256 bytes for inputs and weights.

In general, the number of bits of inputs, weights, and partial sums follow directly from the tiling parameters associated with the I, W, and O matrix indices in Alg. 1: their values are $n\times h\times C$, $m\times h\times C$, and $n\times m\times R$, respectively. These tiling parameters also determine the sizes of the I, W, and O tiles: that are equal to $n\times D\times h\times C$, $m\times R\times h\times C$, and $m\times D\times n\times R$, respectively.

The total bitwidth and tile size of inputs and weights of Tab. I are obtained by multiplying these numbers by the bit precision. Unlike inputs and weights, partial sums must account for the accumulation range. For operands of b_I and b_W bits, we add 16 accumulation bits to the product of b_I+b_W bits, giving a partial-sum width of b_I+b_W+16. Thus, e.g., 8×8 and 2×2 operations require 32- and 20-bit partial sums, respectively. Since accumulation bits depends on the dimensionality of the NN layers, we use Llama 3.1 405B [20] as worst-case model, whose layers require at most 53,248 reduction steps, thus at most 16 accumulation bits. This also matches the 32-bit accumulation width used by BF and BB for 8×8 operations in [14] and [15]; since BS does not specify its accumulation width in [16], we apply the same criterion for fairness. The total partial-sum bitwidth then depends on the number of values computed in parallel. For BB at 2×2, the number of partial sums is $m\times n\times R = 1\times1\times16 = 16$, and the resulting bitwidth is $16\times20 = 320$ bits. Conversely, BS requires $4\times4\times16 = 256$ partial sums, and a total bitwidth of $256\times20 = 5120$ bits. On the other hand, a partial sum tile of BB at 2×2 contains $m\times D\times n\times R = 1\times1\times16\times16 = 256$ elements, thus $256\times20 = 5120$ bits, while the same tile of BS contains $m\times D\times n\times R = 4\times4\times16\times16 = 4096$, for a total of $4096\times20 = 81920$ bits. Clearly, the BS advantage of fixing the bitwidth and tile size of input and weight, is largely compensated by the large bitwidth and tile size required for partial sums, which has a significant impact on BS memory.

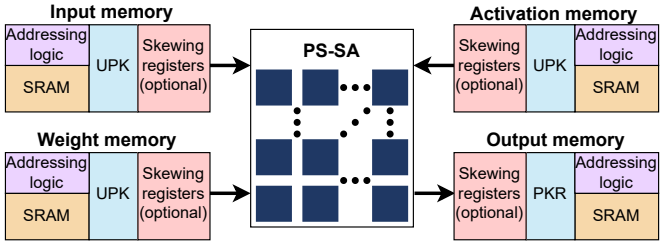

Fig. 2. Memory system overview. The PS-SA in the center is connected to three input memories (Input, Weight, and Activation), and an output memory.

TABLE II
Set of allowed values of the design-knobs for BF, BB, and BS PS-PEs, for input (I), weight (W), activation (A), and output (O) mem-blocks.

	BF	BB	BS
I: **W:**	$M=\{1,2,4,8,16\}$ $S=\begin{cases}\{\mathrm{T},\mathrm{F}\} & M\neq 16\\ \mathrm{F} & \text{else}\end{cases}$	$M=1$ $S=\mathrm{F}$	$M=1$ $S=\mathrm{F}$
A: **O:**	$M=\{1,2,4,8,16\}$ $S=\begin{cases}\{\mathrm{T},\mathrm{F}\} & M\neq 16\\ \mathrm{F} & \text{else}\end{cases}$	$M=\{1,2,4,8,16\}$ $S=\begin{cases}\{\mathrm{T},\mathrm{F}\} & M\neq 16\\ \mathrm{F} & \text{else}\end{cases}$	$M=\{1,2,4,8,16\}$ $S=\begin{cases}\{\mathrm{T},\mathrm{F}\} & M\neq 16\\ \mathrm{F} & \text{else}\end{cases}$

III. Memory System for PS-SAs

Our memory system is shown in Fig. 2. Similarly to [17], we use separate Input, Weight, Activation, and Output memories, connected to the I, W, A, and O interfaces of the PS-SAs in Figs. 1a-c. With respect to Alg. 1b, A provides previously accumulated partial sums, which are summed to the current $I \cdot W$ tile and the result is written to O. Each memory of Fig. 2 contains an SRAM element and related addressing logic. Input and Output memories also include an Unpacker (UPK) and a Packer (PKR), respectively, described below. Finally, memories may include SRs, depending on the design-knobs.

The memories can be partitioned into multiple mem-blocks, each serving a group of consecutive PS-PEs on both boundaries of the SA. We define M as the number of mem-blocks into which a memory is partitioned, representing the first design knob of our exploration. Its allowed values depend on the data type and PS-SA architecture, as reported in Tab. II, and generally range from 1 to 16.

Besides partitioning, a further aspect of SA memory design is the *skewing delay*, which in our architectures is equal to one cycle in our PS-SAs. We explore three implementation strategies. The first uses SRs, the second includes the skewing encoding in the SRAM data layout, while the third integrates the skewing as an offset between the read/write address of the memory-blocks. The design knob S selects the first two options, with S=T (i.e., *true*) enabling SRs and S=F (*false*) encoding skewing in SRAM. Allowed values for *S* are reported in Tab.,II. The address-offset strategy is determined by M. These approaches can be combined, defining a design-space of hybrid solutions that we explored, varying M and S knobs, to identify the least-area configuration for each memory.

Fig. 3 shows the different memory architectures obtained by varying M and S. For M=1, a single mem-block serves 16 PS-PEs (Fig. 3a and Fig. 3b), for M=4, four mem-blocks serve four PS-PEs each (Fig. 3c and Fig. 3d), and for M=16, each mem-block serves only one PS-PE (Fig. 3e). The SRAM bandwidth (BW) of each mem-block is selected to satisfy the maximum requirement of the PS-PE group it serves, across all precisions, according to Tab. I. When the memory is partitioned into multiple mem-blocks, this aggregate BW is equally distributed among them. Each mem-block SRAM is implemented using foundry 28-nm SRAM IP macros. We combine macros so that their aggregate data width matches the required BW. We found that selecting SRAM macros with the largest available data width allows the mem-block to be implemented with the smallest area. In Fig. 4 we show how the *skewing delay* is implemented when M>1. Specifically, Fig. 4a refers to the M=4, S=T configuration. Each of the four mem-blocks serves four PS-PEs, and includes six SRs. In the example, the address (*ADDR*) selects the third column of data (in blue), which are supplied to the PS-PEs. Fig. 4b shows the case with S=T and M=8, thus each of the previous mem-blocks is partitioned in two parts, each serving two PS-PEs. Part of the *skewing delay* is obtained by the constant offset that substracts 2 from the address. As a result, while the bottom mem-block supplies the PS-PEs with the third colunm, the top one selects the first column. The rest of the *skewing delay* is added by the two SRs, one per mem-block.

The design of a memory system for PS-SAs must account for the significant bitwidth variation across precision modes in Tab. I. At some precisions, the per-cycle data demand is smaller than the SRAM word width, which is designed to provide the data for the worst case. Consequently, a single SRAM word can contain data for multiple PS-SA cycles. Discarding the unused bits would waste memory accesses and reduce energy efficiency. Instead, we pack data for multiple cycles into one SRAM word and use an UPK, shown in light blue in Fig. 2, to extract and supply the required portions over consecutive cycles.

The internal structure of the UPK is shown in Fig. 5. For simplicity, we used the BF Input memory with M=16 mem-blocks and S=F, corresponding to Fig. 3e. Here, each mem-block and its UPK serve one PS-PE. When the UPK reads an SRAM word, it stores it in a local register, shown in the figure divided into four chunks for clarity. Depending on the precision, either part or all of this register is supplied to the PS-PE through MUX paths implementing the supported unpacking ratios. The control logic uses two parameters, *N* and *P*: *N* configures a counter from 0 to $N-1$, whose output *C* selects the proper register chunks, while *P* selects the MUX path associated with the current precision. Thus, the appropriate chunks are supplied to the PS-PE at cycle *C%N*, where % denotes the modulo operator. A new SRAM word is read when the counter reaches $N-1$. The output mem-blocks perform the inverse operation: a PKR combines data produced over multiple PS-SA cycles into a single SRAM word. Each mem-block has its own UPK or PKR; as a result, increasing

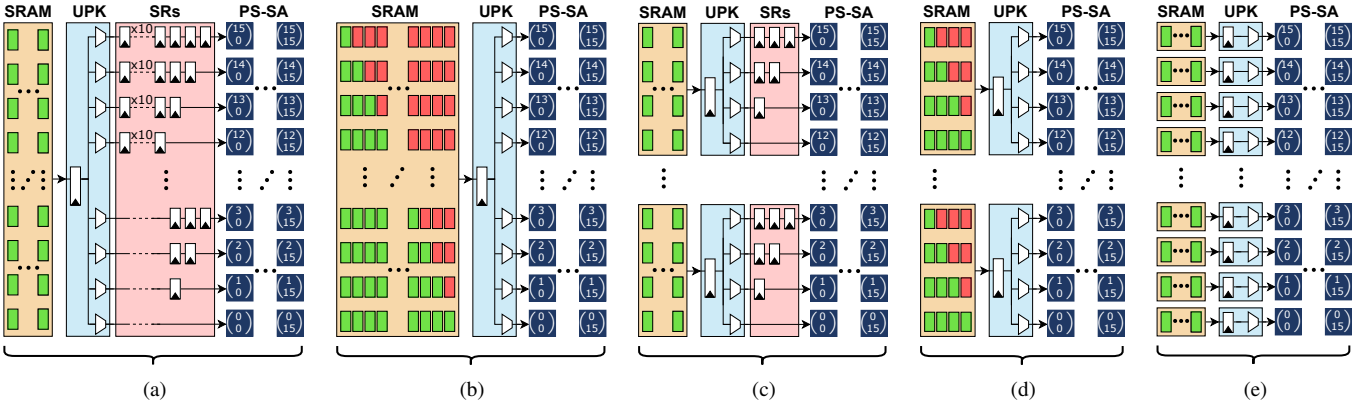

Fig. 3. Possible memories of Fig. 2 varying our design-knobs. (a) M=1, S=T, (b) M=1, S=F, (c) M=4, S=T, (d) M=4, S=F, (e) M=16, S=F.

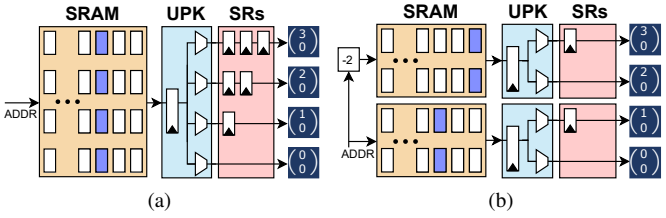

Fig. 4. *Skewing delay* in the memory address. (a) w/o skewing (b) w/ skewing.

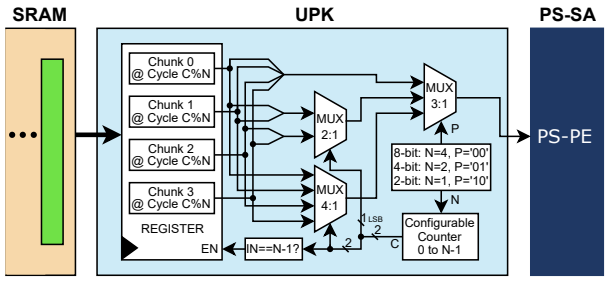

Fig. 5. Unpacker for the Input mem-block of BF with M=16, S=F.

M also increases their number in the memory.

We give details on how M and S affect memory organization. For M=1, a mem-block serves all the PS-PEs. With S=T, skewing is implemented through SRs, as in Fig.,3a: PE $(1, 0)$ receives one cycle of delay, and each successive PE along the vertical direction receives one additional delay. Since one mem-block serves all PS-PEs, this configuration requires the maximum number of SRs, while the SRAM remains *dense*, i.e., all entries are useful (green cells). With S=F, the organization becomes that of Fig.,3b, where SRs are removed and each SRAM word is arranged according to the values required by the PS-SA at a cycle. This introduces invalid entries (red cells), potentially requiring a larger SRAM to store the same amount of useful data. As M increases, more mem-blocks are used and part of the skewing is implemented through constant address offsets between them, reducing the skewing that each mem-block must handle. For example, with M=4 and S=T, the number of SRs decreases from 120 (Fig. 3a) to 24 (Fig. 3c). With S=F, the number of invalid SRAM entries similarly decreases from 120 (Fig. 3b) to 24 (Fig. 3d). Finally, with M=16, (Fig, 3e) skewing is implemented entirely through address offsets between mem-blocks; since S has no effect, we arbitrarily set it to F.

The same considerations apply to the output memory. PS-SA outputs can either be written while preserving their skewing (S=F), or realigned through SRs (S=T), before being stored in the SRAM. The O memory can also be partitioned into multiple mem-blocks, with address offsets accounting for the skewing partially, when M<16, or entirely when M=16.

IV. Experimental Results

We evaluated BF-, BB-, and BS-based PS-SAs, each comprising the four memories of Fig.,2 and the configurations defined by M and S in Tab.,II. In all cases, execution and tile transfer are overlapped using double-buffer memories storing two tiles, while performance is evaluated on one tile. BF has 9 configurations for each memory, summing to 36, while BB and BS have 20 configurations each. This yields 76 different configurations combined, which we fully explored by synthesizing each memory configuration with Synopsys Design Compiler using a 28-nm standard-cell library at 500 MHz clock frequency. Fig.,6 reports the area results in twelve plots arranged as a 3×4 matrix, with rows corresponding to architectures (BF, BB, and BS) and columns to memory types (I, W, A, and O). Each plot shows area versus M, with red and blue lines representing S=T and S=F, respectively, while the black star marks the minimum-area configuration.

The trends are similar across all memories. The conventional configuration used by many SA designs, with SRs and a single mem-block (S=T, M=1), is always dominated by S=T configurations with M=4 or M=8. Thus, when using SRs, partitioning the memory into four or eight mem-blocks and implementing part of the skewing through constant address offsets provides a better hybrid solution. In contrast, relying only on address offsets (M=16), as in [19], gives the worst results, as fine granularity of the memory replicates addressing logic, which requires more area than that saved by not using

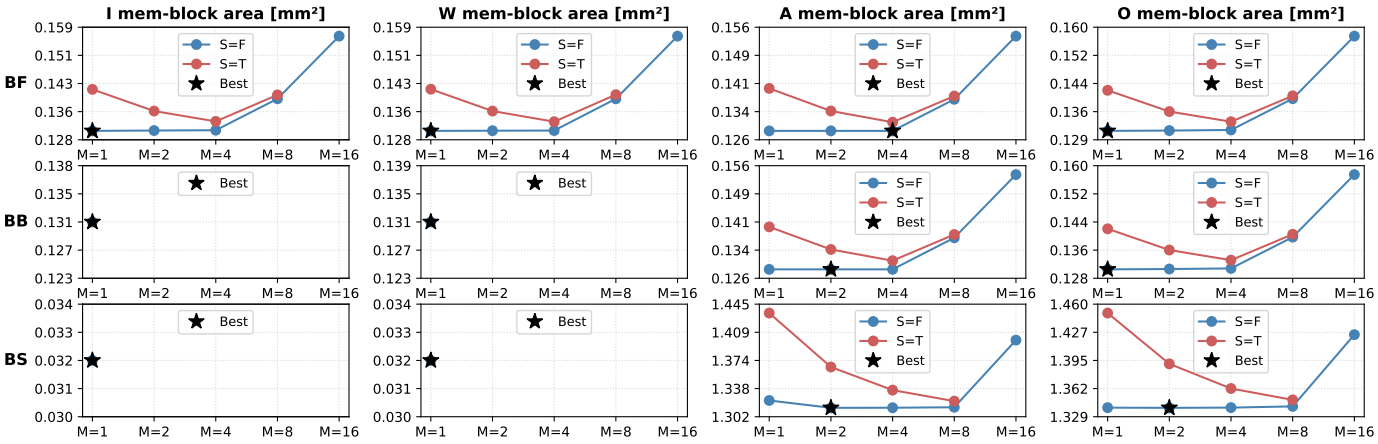

Fig. 6. Area exploration for different memory solutions of the BF, BB, and BS architecture.

TABLE III
Memory bitwidth, size, area (A) and power (P) for the best configurations.

	I (M, S) BW/size [bits]/[KB]	W (M, S) BW/size [bits]/[KB]	A (M, S) BW/size [bits]/[KB]	O (M, S) BW/size [bits]/[KB]	Memory A [mm^2] P [mW]	Total A [mm^2] P [mW]
BF:	(1,F) 512 / 8	(1,F) 512 / 8	(4,F) 512 / 8	(1,F) 512 / 8	0.521 240.36	0.946 313.71
BB:	(1,F) 512 / 8	(1,F) 512 / 8	(2,F) 512 / 8	(1,F) 512 / 8	0.521 240.30	0.778 312.67
BS:	(1,F) 128 / 2	(1,F) 128 / 2	(2,F) 5120 / 80	(2,F) 5120 / 80	2.717 1252.43	2.968 1320.29

SRs. In general, the minimum-area configurations use skewing encoded in the SRAM (S=F), where the area remains almost constant between M=1 and M=4. An explanation is that the memory capacity required to store one data tile can also accommodate invalid skewing entries without requiring larger SRAM macros, with area reduction coming from SRs removal.

Using the minimum-area configuration for each memory, we synthesized the three PS-SAs, reporting results in Tab. III. We report post-synthesis area and power, with power estimated using random inputs with evenly distributed zeros and ones. The BB-based PS-SA achieves the best result, followed by BF, which has a comparable memory area but a larger PS-SA. The BS-based design instead requires larger memories, due to the large bitwidth of partial sums (A values in Tab. I), leading to an unbalanced area distribution between memory and PS-SA.

To summarize, while the present study focuses primarily on post-synthesis area minimization, a natural extension is to complement these results with post-layout characterization and a more detailed energy evaluation. In particular, the area-optimal configurations use S=F, where skewing is encoded in the SRAM data layout. This may introduce an energy trade-off because each SRAM access can also include invalid entries, whereas with S=T all transferred data are useful. A future extension will therefore quantify this effect through memory-level simulation and assess how these additional SRAM accesses impact the energy consumption of the entire system. This will provide a more complete picture of the area-energy trade-offs in the design of a memory-system for PS-SA.

V. Conclusions

This work systematically investigated local memory-system design for BF-, BB-, and BS-based PS-SAs under a common weight-stationary baseline. Starting from the precision-dependent bitwidth requirements of inputs, weights, and partial sums, we developed a parametric memory system with separate memories for each data type, dedicated UPK/PKR units, and three complementary mechanisms for implementing the skewing delay: dedicated SRs, skewing encoded in the SRAM data layout, and memory partitioning with constant read/write address offsets. We explored the design space of all memories, synthesizing a total of 76 configurations. Results show that the conventional monolithic memory with SRs, corresponding to M=1 and S=T, is not area-optimal. When SRs are retained, partitioning the memory into four or eight mem-blocks, and implementing part of the skewing through address offsets, reduces the overall area. More importantly, all minimum-area configurations use S=F with a limited number of mem-blocks, M=1 to M=4. After independently optimizing the four memories of each PS-SA, the BB-based architecture achieves the smallest total area, equal to 0.778 mm^2. Conversely, the high activation bitwidth required by the BS PS-PE results in a considerably larger memory system, making the overall implementation less area-efficient.

A natural extension of this study is to complement the present single-tile execution and post-synthesis analysis with many-tile execution, post-layout characterization and activity-based energy evaluation. In particular, when the skewing is encoded in the SRAM data layout, this approach will allow us to quantify the impact of invalid entries both from the SRAM size in a many-tile execution scheme, and the energy overhead associated with reading and writing invalid entries, thus providing a more complete characterization of the area-energy-size trade-offs in PS-SA memory-system design.

REFERENCES

[1] R. Xu, S. Ma, Y. Guo, and D. Li, "A survey of design and optimization for systolic array-based dnn accelerators," *ACM Comput. Surv.*, vol. 56, no. 1, Aug. 2023.

[2] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[3] R. Raj, S. Banerjee, N. Chandra, Z. Wan, J. Tong *et al.*, "Scale-sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2025, pp. 186–200.

[4] A. C. Yüzügüler, C. Sönmez, M. Drumond, Y. Oh, B. Falsafi *et al.*, "Scale-out systolic arrays," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, Mar. 2023.

[5] A. Gutermann, A. Serdyuk, F. Paraskevas, H. T. Kiesa, F. Lesniak *et al.*, "Improving ai accelerator performance through co-designing neural networks and systolic hw," in *2025 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2025, pp. 1–7.

[6] M. Rakka, M. E. Fouda, P. Khargonekar, and F. Kurdahi, "A review of state-of-the-art mixed-precision neural network frameworks," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–20, Apr. 2024.

[7] L. Mei, M. Dandekar, D. Rodopoulos, J. Constantin, P. Debacker *et al.*, "Sub-word parallel precision-scalable MAC engines for efficient embedded dnn inference," in *Proc. AICAS*, 2019, pp. 6–10.

[8] E. Manca, L. Urbinati, and M. R. Casu, "Star: Sum-together/apart reconfigurable multipliers for precision-scalable ml workloads," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE)*. Valencia, Spain: IEEE, 2024, pp. 1–6.

[9] Y. Liu, S. Ullah, and A. Kumar, "Bitwise systolic array architecture for runtime-reconfigurable multi-precision quantized multiplication on hardware accelerators," in *2025 26th International Symposium on Quality Electronic Design (ISQED)*, 2025, pp. 1–9.

[10] A. Othman, A. Kamaleldin, and D. Göhringer, "Amper-x: Adaptive mixed-precision risc-v core for embedded applications," in *2024 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2024, pp. 1–7.

[11] E. Manca, L. Urbinati, and M. R. Casu, "Accelerating quantized dnn layers on risc-v with a star mac unit," in *Proc. of SIE 2023*. Springer Nature Switzerland, 2024, pp. 43–53.

[12] E. Manca, L. Urbinati, and M. R. Casu, "Accelerating mixed-precision qnn inference on risc-v mcus with the star-mac unit," *IEEE Access*, vol. 13, pp. 208 533–208 548, 2025.

[13] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *Proc. IEEE ISSCC*, 2017, pp. 246–247.

[14] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[15] S. Ryu, H. Kim, W. Yi, E. Kim, Y. Kim *et al.*, "Bitblade: Energy-efficient variable bit-precision hardware accelerator for quantized neural networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, 2022.

[16] Y. Cao, M. Wen, J. Shen, and Z. Li, "Bitshare: An efficient precision-scalable accelerator with combining-like-terms gemm," in *2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2024, pp. 36–44.

[17] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 769–774.

[18] S. Zouzoula, M. A. Maleki, M. W. Azhar, and P. Trancoso, "Scratchpad memory management for deep learning accelerators," in *Proceedings of the 53rd International Conference on Parallel Processing*, ser. ICPP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 629–639. [Online]. Available: https://doi.org/10.1145/3673038.3673115

[19] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang *et al.*, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 214–225.

[20] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian *et al.*, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783