SA4: A Comprehensive Analysis and Optimization of Systolic Array Architecture for 4-bit Convolutions

Geng Yang[1], Jie Lei[2], Zhenman Fang[3], Jiaqing Zhang[1], Junrong Zhang[1], Weiying Xie*[1], Yunsong Li[1]
[1]*Xidian University,* [2]*University of Technology Sydney,* [3]*Simon Fraser University*
{gengyang, jgzhang_2, zhangunrong}@stu.xidian.edu.cn,
jie.lei@uts.edu.au, zhenman@sfu.ca, wyxie@xidian.edu.cn, ysli@mail.xidian.edu.cn
* indicates the corresponding author of the paper.

Abstract—**Many studies have demonstrated that 4-bit precision quantization can maintain accuracy levels comparable to those of floating-point deep neural networks (DNNs). Thus, it has sparked a keen interest in the efficient acceleration of such compressed DNNs, especially 4-bit convolutions, on edge devices. However, we observe that conventional systolic array (SA) architectures, widely adopted for DNN acceleration, fail to fully exploit the high computational density benefits of 4-bit DSP packing.**

In this paper, we conduct the first comprehensive analysis of the integration of modern DSP packing techniques (specifically, 4-bit fully DSP packing) into the 4-bit systolic array design for convolutions. First, we introduce a row-temporal weight stationary 4-bit SA dataflow that complements the loop execution order inherent in 4-bit fully DSP packing in conventional SAs, which is called BaseSA. Next, we analyze the performance and resource efficiency of BaseSA, and identify two inefficiencies in the integration: 1) excessive LUT resource utilization that constraints the overall SA size, and 2) large latency gap to the theoretical optimum, due to various stalls in data supplies. To overcome these obstacles, we propose SA4: an HLS-based, customizable, and ultra-efficient hierarchical SA architecture optimized for 4-bit convolutions. The core unit in SA4 is a delicately designed cost-effective SA unit (SAU), which 1) replaces the costly buffer-based data suppliers for activations and weights with shift-register-based ones, 2) replaces LUT-intensive FIFO connections between SA PEs (processing elements) with registers, and 3) replaces the finite state machines (FSM) and data unpacking logic inside each PE with a global FSM inside each SAU and a data splitter shared by a column of PEs. While such an SAU can only support a small spatial size for an SA due to its delicate design, we further scale it out using an array of SAUs. Experimental results show that our proposed SA4 achieves 1153.2 GOPS on the AMD-Xilinx Ultra96-V2 FPGA, with a 13.8× increase in GOPS/DSP efficiency and a 49× increase in GOPS/kLUTs efficiency compared to a straightforward SA and 4-bit DSP packing integration. Our SA4 project is open sourced here: https://github.com/Michaela1224/SA4.

I. Introduction

The remarkable achievements of diverse DNNs—such as CNNs [1], Transformers [2], and diffusion models [3]—in a wide range of real-world applications [4], [5], have spurred an increasing demand for their inference deployment on edge devices, particularly embedded FPGAs [6], [7]. Within these DNNs, the efficient hardware acceleration of the convolutional layers has consistently garnered significant attention. To alleviate deployment challenges on resource-constrained edge devices, the adoption of 4-bit quantization for both weight and activation has proven to be an indispensable and effective model compression technique [8]–[13]. This approach yields negligible accuracy loss while simultaneously reducing memory footprint and increasing computational efficiency.

On one hand, the systolic array (SA) architecture, comprised of identical processing elements (PEs) with neighbor-to-neighbor interconnects, has been widely utilized for hardware acceleration of convolutional layers [14]–[20]. This architectural choice facilitates natural data reuse by allowing local data transfers between adjacent PEs, while also mitigating the timing issues associated with massive parallel computations.

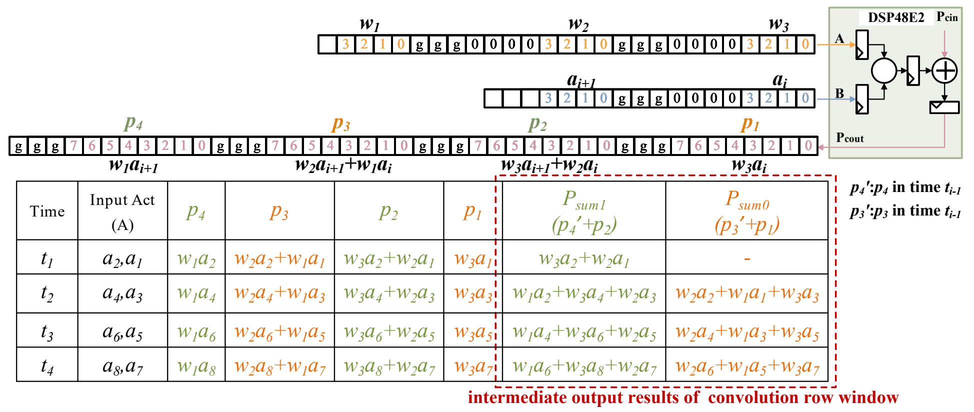
On the other hand, DSP packing is a well-established method that is closely associated with model quantization. It efficiently packs low bit-width multiplications into high bit-width DSPs, significantly improving DSP efficiency on FPGAs [21]. The latest work [22] leverages the window-based multiply-accumulate computation rules of convolutions to parallelize six 4-bit multiplications and two additions onto a single DSP48E2, thus maximizing DSP computational efficiency on the chip. This technique is called 4-bit fully DSP packing (4bF packing) in this paper.

In this paper, we aim to address a key question: *How to combine the effective 4-bit fully DSP packing and widely used systolic array architecture together in high-level synthesis (HLS) to take advantage of both to accelerate 4-bit convolutions used in CNNs, Transformers, and diffusion models?*

Unfortunately, most existing SA architectures [14], [20], [23]–[26] only support floating-point, 16-bit or 8-bit data precisions, which neglect the exploration of the hardware performance improvements brought by 4-bit quantization. More importantly, the latest 4bF packing relies on a specific loop execution order (detailed in Section II-A) that imposes constraints on the SA dataflow mapping. To the best of our knowledge, no existing SA-based work has implemented the integration of this cutting-edge 4bF packing method.

To tackle this challenge, we first analyze the widely-used weight stationary (WS) SA dataflow under the constraint of the loop execution order for 4bF packing. Then, we introduce a row-temporal WS SA dataflow tailored to 4bF packing, called BaseSA, which aims to maximize the data reuse and minimize the on-chip storage. However, after an in-depth analysis of this straightforward integration, we identify the following resource and performance inefficiencies in BaseSA.

i) LUT Resource Bottleneck: As will be analyzed in Section III-C, for the BaseSA with 10×10 PEs, its DSP utilization is only around 28% (on the AMD/Xilinx Ultra96-V2 FPGA); but its LUT utilization is already 74%, which limits the size of BaseSA. A detailed breakdown shows that, the buffer-based multi-level IFM (input feature map) and weight data loaders, in conjunction with FIFO interconnects between PEs,



Time	Input Act (A)	p_4	p_3	p_2	p_1	P_{sum1} $(p_4'+p_2)$	P_{sum0} $(p_3'+p_1)$
t_1	a_2,a_1	w_1a_2	$w_2a_2+w_1a_1$	$w_3a_2+w_2a_1$	w_3a_1	$w_3a_2+w_2a_1$	-
t_2	a_4,a_3	w_1a_4	$w_2a_4+w_1a_3$	$w_3a_4+w_2a_3$	w_3a_3	$w_1a_2+w_3a_4+w_2a_3$	$w_2a_2+w_1a_1+w_3a_3$
t_3	a_6,a_5	w_1a_6	$w_2a_6+w_1a_5$	$w_3a_6+w_2a_5$	w_3a_5	$w_1a_4+w_3a_6+w_2a_5$	$w_2a_4+w_1a_3+w_3a_5$
t_4	a_8,a_7	w_1a_8	$w_2a_8+w_1a_7$	$w_3a_8+w_2a_7$	w_3a_7	$w_1a_6+w_3a_8+w_2a_7$	$w_2a_6+w_1a_5+w_3a_7$

Fig. 1: Working principle of 4bF packing [22].

collectively consume 59% of these LUT resources. Moreover, to accommodate 4bF packing, each PE internally requires additional data splitters and adders, resulting in increased LUT consumption. As a result, the PE sets, each with its own finite state machine (FSM), consume another 35% of those LUT resources. As the SA size increases, LUT resource consumption escalates, ultimately making LUTs the primary resource bottleneck, overshadowing the expected DSP utilization.

ii) Large Latency Gap: Due to the reliance of the buffer-based multi-level data loaders used in conventional SAs, which require specific data reuse patterns to hide the data supply latencies to feed the PEs, BaseSA exhibits notably poor latency performance when adapting to our row-temporal WS dataflow. Specifically, there is 1) a sequential data supply latency to propagate the data onto a column of PEs, and 2) a Ping-Pong buffer switching overhead. These result in a latency gap exceeding 87% compared to the theoretical optimal latency.

To address the above challenges, we propose a cost-effective SA unit (SAU). First, we replace the buffer-based multi-level IFM and weight data loaders with the shift-register-based IFM and weight fetchers. This not only addresses the latency discrepancies as data can now be continuously supplied into PEs independent of the data reuse patterns, but also reduces the LUT usage by about 20%. Second, we replace the data unpacking logic inside each PE with a shared data splitter that is shared by a column of PEs, which reduces the LUT usage by another 10%. Finally, we replace the FIFOs connecting neighbor PEs with registers, and replace the FSMs within each PE with a global FSM, which further reduces the LUT usage by about 28%. In total, for a SA with 4×4 PEs, our SAU reduces about 58% of the LUT usages compared to BaseSA, eliminating the LUT resource bottleneck.

However, as will be analyzed in Section IV-A3, there is a limitation of the spatial size of the SAU design, due to 1) the need to efficiently support small spatial size convolutions, and 2) DSP multiply-accumulate data overflow caused by our shared data splitter design. To overcome this limitation, we introduce a two-level hierarchical SA design, named SA4. SA4 organizes the original large SA into an array of SAU units, each with an SAU size of 4×4, ensuring minimal latency gap (0.3%) across various convolution spatial sizes.

Experimental results demonstrate that our proposed SA4 achieves 1153.2 GOPS for the 4-bit convolutions on AMD-Xilinx Ultra96-V2 FPGA, with 13.8× higher GOPS/DSP efficiency and 49× higher GOPS/kLUTs efficiency compared to the straightforward integration of SA and 4bF packing. Our case study for the 4-bit UltraNet [1] deployment using SA4 demonstrates that, on the same Ultra96-V2 FPGA, we achieve a 3.7× FPS (frames per second) improvement compared to the original UltraNet [1] implementation that uses one DSP to pack two 4-bit multiplications, and a 2.3× FPS improvement compared to an all-on-chip fully-pipelined UltraNet implementation [22] that uses the same 4bF packing.

In summary, this paper makes the following contributions:

1. A comprehensive analysis of resource and performance inefficiencies in a straightforward integration of widely used systolic array architecture and 4-bit fully DSP packing.
2. The first customizable, ultra-efficient FPGA-based systolic array architectural template for 4-bit convolutions, with the efficient integration of 4-bit fully DSP packing.
3. Superior performance and performance/resource efficiency of SA4 over prior 4-bit convolution designs.

II. Background and Related Work

A. 4-bit Fully DSP Packing for Convolution

To harness the advantages of 4-bit precision quantization and translate them into tangible hardware performance enhancements, HiKonv [22] proposed a groundbreaking concept: the mapping of six 4-bit multiplications and two additions of one convolutional layer into a single AMD-Xilinx DSP48E2. This achievement was accomplished by combining the 4-bit DSP overpacking proposed by Liu et al. [27] with the sliding-window-based multiply-accumulate rule designed for convolutional layers. This technique, called 4-bit fully DSP packing (4bF packing) in our paper, led to a substantial improvement in on-chip DSP utilization.

As depicted in Fig. 1, each 4bF packing computation involves sequentially reading two 4-bit unsigned input activations (a_i, a_{i+1}) along the column dimension (C) of input feature map (IFM) and three 4-bit signed weights (w_1, w_2, w_3) along the column dimension (K_c) of the filter kernel. These data are then meticulously packed and supplied to the DSP48E2 for 18×27-bit multiplication. g stands for the guard bits, set at a value of 3, to ensure output remains correct when packing six 4-bit multiplications together for one DSP48E2. Subsequently, the 44-bit output from the DSP is split into four 11-bit data (p_1,p_2,p_3,p_4), which are then reorganized to obtain the final row windowed accumulation results for convolution.

The table in Fig. 1 provides an example of the data reorganization process where the input activations $a_1 - a_8$ along the column dimension under the same weights (w_1,w_2,w_3). Specifically, the current p_1 and the previous step $p_{3'}$ are accumulated to produce one intermediate output result p_{sum0} along the column dimension (C) of the output feature map (OFM). Similarly, the current p_2 and the previous $p_{4'}$ are accumulated to obtain another intermediate output result p_{sum1}. To the best of our knowledge, there is currently no work that has seamlessly integrated this efficient 4-bit DSP packing method into more versatile SA architectures.

B. Systolic Array for Convolution

The Systolic Array (SA) architecture, characterized by its highly parallel and pipelined PEs operating in lock-step, wave-like fashion, has gained significant attention for accelerating convolution operations on FPGAs [14], [18], [20], [25], [26]. The traditional dataflow patterns in SA architecture [17] are categorized into weight stationary (WS) [15], [28], output stationary (OS) [19], [24], [25], and input stationary (IS) [14], based on different loop unrolling strategies (i.e., data reuse across different dimensions). State-of-the-art (SOTA) FPGA-based SA framework—AutoSA [14]—explores space-time transformation, array partitioning, latency hiding, and SIMD vectorization to generate a SA design that minimizes the latency within available resources. It has reported the best performance results in automatic SA generation field [23]. However, most existing SA architectures only support floating-point, 16-bit, or 8-bit data precision. Unlike previous work, this paper delves into the efficient integration of 4-bit data quantization and modern 4bF packing into the SA architecture for the first time, and provides an ultra-efficient 4-bit SA architecture customized for 4-bit convolutions.

III. BaseSA with 4bF Packing and Its Limitations

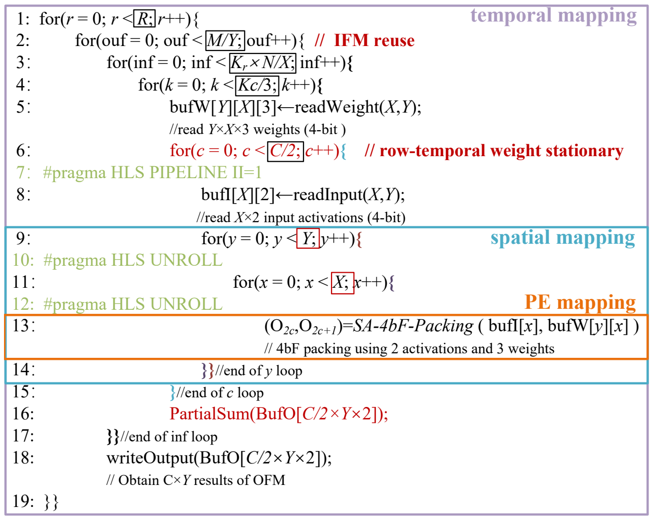
A. Dataflow Mapping Analysis

Different dataflow patterns essentially entail mapping data reuse from different dimensions (i.e., fine-grained spatial mapping) of one convolutional layer onto the locally connected PEs. Moreover, specific dataflow patterns further determine off-chip memory access to maximize tile data reuse (i.e., coarse-grained temporal mapping) for IFM/Weight/OFM loaded into on-chip buffers. The WS dataflow, characterized by its small on-chip IFM buffer and continuous result output, is extensively utilized in convolutional layers [14], [29], [30], making it the starting point for our analysis. The explanations of the variables involved are summarized in Table I.

Unlike the traditional WS-based dataflow, the 4bF packing introduces dependencies among input activations across column dimension C shown in the table of Fig. 1, thereby constraining the order of IFM data feeding into the SA. By

TABLE I: Design variable and explanation

Variables	Explanation
N	Number of input channels of IFM
M	Number of output channels of OFM
(R,C)	Number of rows/columns of IFM/OFM
(K_r,K_c)	Filter kernel size
(X,Y)	Size of systolic array
A	2 4-bit activations for one 4bF packing
W	3 4-bit filter weights for one 4bF packing



temporal mapping
spatial mapping
PE mapping

```
1:  for($r$ = 0; $r$ < $\boxed{R;}$ $r$++){
2:      for(ouf = 0; ouf < $\boxed{M/Y;}$ ouf++){  // IFM reuse
3:          for(inf = 0; inf < $\boxed{K_r \times N/X;}$ inf++){
4:              for($k$ = 0; $k$ < $\boxed{Kc/3;}$ $k$++){
5:                  bufW[$Y$][$X$][3]←readWeight($X$,$Y$);
                    //read $Y \times X \times 3$ weights (4-bit )
6:                  for($c$ = 0; $c$ < $\boxed{C/2;}$ $c$++){   // row-temporal weight stationary
7:  #pragma HLS PIPELINE II=1
8:                      bufI[$X$][2]←readInput($X$,$Y$);
                        //read $X \times 2$ input activations (4-bit)
9:                      for($y$ = 0; $y$ < $\boxed{Y;}$ $y$++){
10: #pragma HLS UNROLL
11:                         for($x$ = 0; $x$ < $\boxed{X;}$ $x$++){
12: #pragma HLS UNROLL
13:                             ($O_{2c}$,$O_{2c+1}$)=SA-4bF-Packing ( bufI[$x$], bufW[$y$][$x$] )
                                // 4bF packing using 2 activations and 3 weights
14:                     }}//end of $y$ loop
15:                 }//end of $c$ loop
16:                 PartialSum(BufO[$C/2 \times Y \times 2$]);
17:         }}//end of inf loop
18:         writeOutput(BufO[$C/2 \times Y \times 2$]);
            // Obtain $C \times Y$ results of OFM
19: }}
```

Fig. 2: Pseudo code for row-temporal weight stationary (WS) SA dataflow with 4bF packing.

carefully considering this constraint, we construct the row-temporal WS SA dataflow as illustrated in Fig. 2. The spatial size X and Y of SA correspond to the parallelism mapping of IFM across the input channel dimension N and the output channel dimension M of filter weights, respectively.

For fine-grained spatial mapping (line 9-14), $Y \times X \times 3$ weights (line 5) are loaded into $Y \times X$ PEs, with each PE possessing 3 weights. $X \times 2$ input activations (line 8) can be reused by Y filter weights. Within each PE (line 13), 3 4-bit weights and 2 4-bit input activations undergo the 4bF packing shown in Fig. 1. For coarse-grained temporal mapping, the weights loaded into each PE can be reused $C/2$ times (line 6). Finally, after $Kr \times (N/X) \times (Kc/3)$ loop iterations (line 3-4), SA generates $C \times Y$ OFM results (line 18).

For each retrieval of $C \times M$ OFM results (line 2-19), due to the IFM reuse in line 2, only $K_r \times C \times N$ input activations of IFM and $C \times Y$ intermediate results of output feature maps (line 16) need to be stored. We indicate that only $K_r + 1$ rows of IFM buffer with N channels are required to enable the WS-based SA to operate at full capacity.

Meanwhile, due to the typically higher number of channels in modern convolutions, setting parallelism on both input and output channels ensures that the SA maintains high PE utilization across convolutional layers with diverse spatial sizes. Due to its row-wise weight stationary, this paper terms this SA dataflow as the ***row-temporal WS dataflow***.

B. BaseSA Architecture

Fig. 3 presents an overview of our BaseSA microarchitecture, which is a straightforward integration of widely used SA

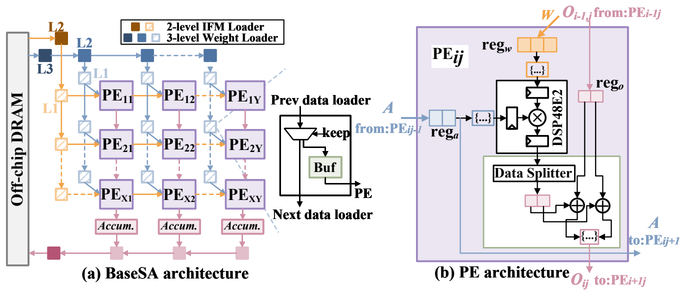


Fig. 3: Overview of BaseSA architecture: straightforward integration of widely used SA architecture and 4bF packing in our 4-bit row-temporal WS dataflow.

architecture (e.g., AutoSA [14]) and 4bF packing in our 4-bit row-temporal WS dataflow. Fig. 3(a) shows a typical SA architecture designed for the WS dataflow (e.g., AutoSA [14]), which is adapted for 4-bit data precision. It consists of multi-level data loaders, including three-level weight loaders (L1, L2, L3 highlighted in blue) and two-level IFM loaders (L1, L2 highlighted in orange), along with identical PEs connected locally to each other via FIFO buffers. The L1/L2/L3-level data loader was inspired by a daisy-chain architecture previously employed [31]. As shown to the right of Fig. 3(a), they selectively retain the data in a Buf required by associated PEs from the input sequential data stream and pass the remaining data to the adjacent data loader. Differing from the L3/L2-level data loader, the L1-level data loader, directly connected to the PEs, facilitates double buffering to overlap tiling data transfer with PE computations.

The configuration of multi-level data loaders for both IFM and weights within the SA architecture depends on whether data reuse occurs among the $X \times Y$ PEs. As observed in the fine-grained spatial mapping of the row-temporal WS dataflow in Fig. 2, the parallel $X \times Y$ PEs possess their own weights. Translating this insight into the corresponding BaseSA architecture in Fig. 3(a) entails configuring 1 L3-level loader, Y L2-level loaders, and $X \times Y$ L1-level loaders for the weights to continuously feed 3 weights to each PE. Conversely, $X \times 2$ input activations can be reused across Y filter weights. BaseSA therefore requires 1 L2-level IFM loader and X L1-level IFM loaders to manage activations supply.

In Fig. 3(b), we delve into the intricate operations of each PE_{ij}, where i and j denote the position of the PE within the array. Each PE_{ij} receives 2 input activations (A) from the left adjacent PE_{ij-1} or the L1-level IFM loader, 3 weights (W) from the L1-level weight loader, and the partial accumulated result (O_{i-1j}) from the top adjacent PE. Processing begins by dispatching A and W to the DSP48E2 for 18×27-bit multiplication as shown in Fig. 1. Subsequently, the high-bit-width result from the DSP48E2 undergoes data splitting and reorganization within the data splitter as depicted in the table of Fig. 1, resulting in two intermediate results along the column dimension (C) of OFM. Finally, these two results are accumulated and concatenated with O_{i-1j}, yielding the output O_{ij}. This final output is then directed into a FIFO for transmission to PE_{i+1j}.

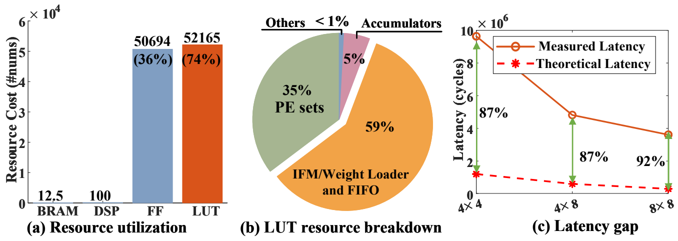


Fig. 4: (a) and (b) shows the resource utilization for BaseSA in Fig. 3 with the sizes of 10×10. (c) shows latency gap with different SA sizes ($X \times Y$) on BaseSA. The latency test adopts the convolution layer of the same input and output size ($M = N = 64$, $R = C = 56$, $K_r = K_c = 3$).

C. Quantitative Resource and Performance Analysis

Through a quantitative analysis, we observe that the BaseSA design, with a straightforward integration of 4-bit SA and 4bF packing, performed significantly below expectations due to LUT resource bottleneck and large latency gap.

i) LUT Resource Bottleneck. Fig. 4(a) provides an overview of resource consumption for the maximum feasible BaseSA size (10×10) deployable on the Ultra96-V2 edge device. Strikingly, instead of being constrained by DSPs, the performance is hampered by LUT resource consumption, accounting for 74% of the total. To decipher the underlying reasons for this resource bottleneck, Fig. 4(b) offers further insight into LUT consumption proportions across different modules. Notably, more than half of resource consumption (59%) is attributed to multi-level data loaders and FIFO interconnection overhead.

A closer examination of BaseSA's microarchitecture, as depicted in Fig. 3(a), reveals that each PE and single-level (L1/L2/L3) data loader is instantiated separately, each equipped with its independent FSM. This approach results in a substantial resource overhead in LUT, particularly owing to the extensive instantiation of multi-level data loaders, including $X \times Y$ L1-level weight loaders. Furthermore, costly FIFOs are employed for local interconnections between neighbor PEs as well as across multi-level data loaders, compounding LUT resource consumption. Such a heavy LUT resource overhead ultimately limits the ability to incorporate more PEs, constraining the SA size on edge devices.

On the other hand, despite leveraging DSP48E2 to accommodate six 4-bit multiplications and two additions, PE sets still consume 35% of the LUT resources shown in Fig. 4(b). This is due to the additional data splitter and adders, as illustrated in Fig. 3(b), which are employed to obtain the final results of the sliding window calculations. Intuitively, the LUT consumption by PE sets increases linearly as the SA size scales up.

ii) Large Latency Gap. Fig. 4(c) provides a comprehensive view of the latency performance across various SA sizes under the same convolution layer. An alarming observation is the considerable gap between actual and theoretical latency, surpassing an unacceptable 87% gap. The primary reason behind this phenomenon is discontinuous data supply to the PEs by BaseSA's multi-level data loaders.

As shown in Fig. 5, due to the neighbor-to-neighbor com-

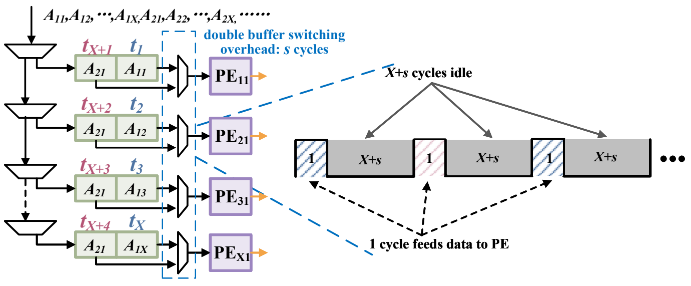


Fig. 5: Latency gap due to the L1-level IFM loader in Fig. 3(a).

munication pattern in SAs, it takes X cycles to propagate the input (IFM) data to the L1-level data loader for the bottom PE. That is, every L1-level data loader takes X cycles to load one data for the corresponding PE computation. Achieving continuous data supply to the PEs with double buffering in the L1-level data loader in Fig. 3(a) necessitates that current data stored in the buffer is reused within the PE for a duration longer than the time required to load the next tile of data. However, in the row-temporal WS dataflow, there is no data reuse during a single load of the input activations loaded into the L1-level IFM loader buffer. Moreover, the latency overhead induced by double buffer switching, approximately s clock cycles, is an inherent challenge that cannot be mitigated.

Although there is $C/2$ reuse for single-load weights (line 6 in Fig. 2), the time needed to transfer $X \times Y$ weights W into the buffer by the L1-level weight loaders increases with the SA size. This makes it challenging to hide the weight data supply latency, particularly when processing convolutions with small spatial dimensions (i.e., smaller C). Detailed analysis will be provided in Sections IV-A3 and V-B.

IV. Design and Implementation of SA4

To address the LUT resource bottleneck and the large latency gap in BaseSA presented in Section III-C, we first present our proposed cost-effective SA unit (SAU) for the efficient integration of 4-bit SA and 4bF packing, and analyze the constraints on the SAU size in Section IV-A. Next, in Section IV-B, we scale out our SA4 design by introducing a two-level hierarchical architecture that uses an array of SAU units to achieve near-theoretical latency across various convolutional spatial sizes.

A. Cost-Effective SAU

Fig. 6 represents the microstructure of our proposed SAU, comprising of shift-register-based IFM fetcher and weight fetcher, $X \times Y$ identical PEs with 4bF packing, and Y data splitters shared among each column of PEs. Firstly, moving away from the multi-level data loaders in BaseSA, the proposed IFM/weight fetcher is based on the shift-register control and ensures a continuous supply of weights and activations without specific data reuse constraints, thus achieving close to theoretical latency while consuming low LUT resources. Secondly, instead of configuring one data splitter and adders within each PE in Fig. 3, each column of PEs shares one data splitter, which is responsible for splitting and reorganizing the high-bit-width results of DSP48E2. Thirdly, without separately

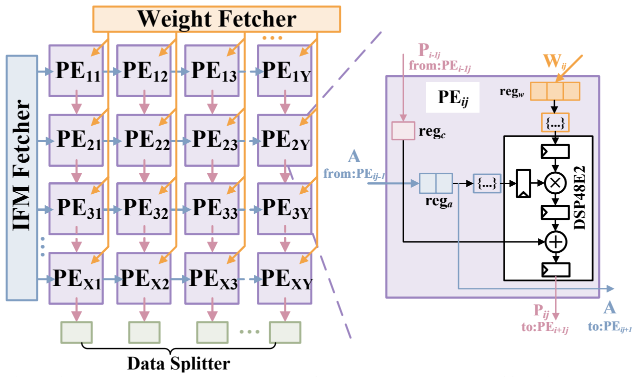


Fig. 6: Hardware structure of proposed cost-effective SAU.

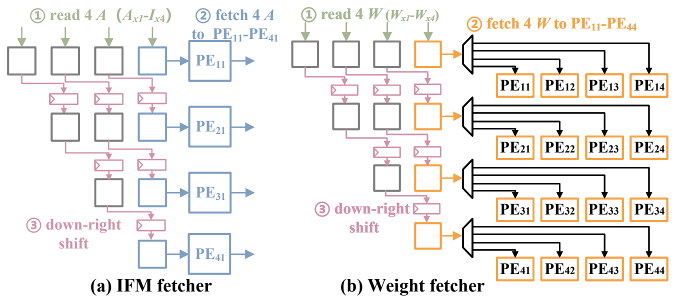
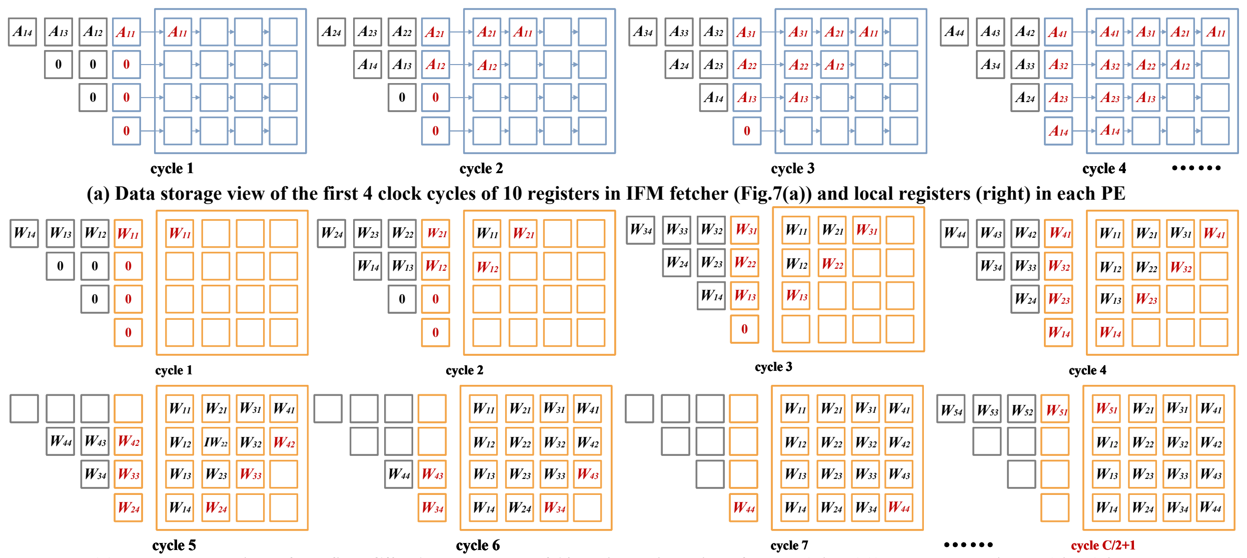


Fig. 7: Hardware structure of IFM fetcher and weight fetcher with $X = 4$ and $Y = 4$ as an example. The ten boxes refer to the data registers.

instantiating each module with FIFO connection in BaseSA, a global FSM and the register-based local interconnection are utilized to oversee the entire structure.

1) IFM Fetcher: The IFM fetcher, shown in Fig. 7(a), is tasked to continuously supply the X parallel input data A to $PE_{11} - PE_{X1}$ in a systolic-like skewed mode. As depicted in the figure, we use $X = 4$ as an example to demonstrate the working mechanism of this module. In each cycle, four packed input data A ($A_{x1} - A_{x4}$) along input channel dimension (N) are firstly cached into the top 4 registers. Then, the four A from the rightmost four registers are fed into the local registers within $PE_{11} - PE_{41}$, while the previous data in $PE_{11} - PE_{41}$ are passed rightwards along the rows of SA. Finally, the values in the ten registers are shifted to the lower-right direction, and new packed input data ($A_{x+1,1} - A_{x+1,4}$) is received from the top. Fig. 8(a) illustrates the data storage view of 10 registers and PE registers in the first four clock cycles. Instead of the two-level IFM loader in BaseSA with FIFO interconnections shown in Fig. 3, our shift-register-based IFM fetcher not only significantly saves LUT resource overhead, but also ensures continuous supply of input activations to the leftmost PEs, even when there is no data reuse for a single feed.

2) Weight Fetcher: As shown in Fig. 7(b), the weight fetcher has a similar structure to the IFM fetcher, with the addition of X selectors to ensure that the $X \times Y$ W are promptly fetched into the local registers of PE_{11}-PE_{XY}. We take $X = 4, Y = 4$ as an example to explain how this module works. Fig. 8(b) illustrates the variation in cached values within the local registers of the weight fetcher over the first $C/2+1$ clock cycles. In the first 4 clock cycles, each


(a) Data storage view of the first 4 clock cycles of 10 registers in IFM fetcher (Fig.7(a)) and local registers (right) in each PE

(b) Data storage view of the first C/2 +1 clock cycles of 10 registers in weight fetcher (Fig.7(b)) and local registers (right) in each PE

Fig. 8: Data supply from IFM fetcher (a) and weight fetcher (b) to each PE within proposed SAU with $X = 4$ and $Y = 4$.

4 W ($W_{y1} - W_{y3}$) are read into the topmost 4 registers in each clock cycle. In the first 7 (i.e., $X + Y - 1$) clock cycles, the data from the rightmost side, highlighted in orange, is loaded into PE_{11}-PE_{44} through selectors (highlighted in red).

Since a single load of weights can be reused for a row of input activations (shown in line 6 of Fig. 2), the weight values in the local registers of each PE remain stationary from cycle 7 to cycle $C/2$. In cycle $C/2+1$, the operations from cycle 1 to cycle 7 are repeated to refresh buffered weight values within each PE. Note that we do not have to set aside specific time for loading weights. Our design makes the most out of the systolic array's startup rules. During the initial $X+Y-1$ clock cycles of PE computation, we concurrently fill in the required weight data. This simple but effective design achieves a continuous weight supply, eliminating the need for three-level weight loaders and their extensive LUT resource overhead in BaseSA.

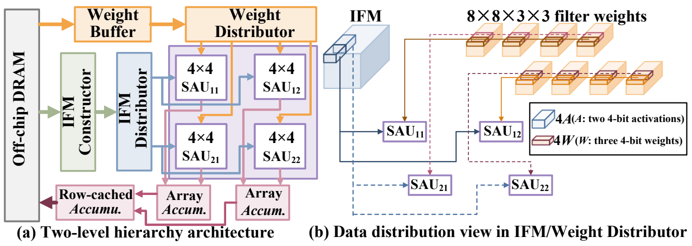
3) Constraints on SAU Size: Revisiting the weight supply scheduling of the weight fetcher in Fig. 8(b), we observe that it always requires Y cycles (from cycle 1 to cycle 4 in our example) to load the $Y \times X$ W into the top four registers (on the left of the figure) for processing a row ($C/2$) of input activations. That is, it needs to meet the following constraints to ensure the continuous weight supply to each PE in our row-temporal WS dataflow, i.e., hiding the Y cycles weight load latency by the $C/2$ cycles computation latency:

$$Y \leq C/2 \tag{1}$$

To make sure small spatial-sized convolutions (i.e., smaller C, as small as $C = 8$) can sustain continuous data supply to each PE, a small $Y \leq 4$ is needed.

Meanwhile, when utilizing the 4bF packing, DSP48E2 in each PE (Fig. 6) experiences guard bit (g in Fig. 1) failures when performing more than four consecutive multiply-accumulate operations, resulting in data overflow. That is, with a shared data splitter by a column of PEs, the row size of SAU is limited to:

$$X \leq 4 \tag{2}$$



(a) Two-level hierarchy architecture (b) Data distribution view in IFM/Weight Distributor

Fig. 9: Overall architecture of SA4. SA size with $X = 8$ and $Y = 8$ is provided as an example.

Note if an SAU with the size of $X > 4$ is employed, it has to equip an independent data splitter within each PE like BaseSA in Fig. 3(b), leading to extra LUT resource overhead.

Combining the constraints in Eqs. 1 and 2, we decide to use $X = Y = 4$ in our SAU design.

B. Two-Level Hierarchical Architecture

To address the size limitation of our SAU unit, we propose a two-level hierarchical SA design as illustrated in Fig. 9(a). The 2D array comprises $(X/4) \times (Y/4)$ parallel SAUs, with an SAU size of 4×4. Within each SAU, its PEs are interconnected with economical registers, forming the L1-level hierarchy. The L2-level hierarchy consists of the IFM constructor, IFM/weight distributor, SAU sets, and two-level accumulators, interconnected with each other via FIFOs.

1) IFM Constructor: The IFM constructor in Fig. 9 is responsible for continuously reading high-bit-width input activations (maximum available bit-width of the device, e.g., 256 bits), packed along input channel dimension, from off-chip DRAM. It then splits the high-bit-width data into $(X \times 2 \times 4)$-bit data packets to match the 2D array's size. Finally, we maintain this bit-width and utilize the equipped $K_r + 1$ rows of input activation cache with N channels to generate the continuous data stream, which adapts to sliding window computation of the convolution layer and then feeds into the IFM distributor. According to our row-temporal WS dataflow, the raw IFM data (without any off-chip preprocessing) only need to be read from off-chip once and then all output results for OFMs can be calculated on-chip in a pipelined fashion.

2) IFM/Weight Distributor: The IFM distributor is responsible for distributing the constructed input data into the $X/4 \times Y/4$ SAU sets in parallel shown in Fig. 9(b). Taking $X = 8$ and $Y = 8$ as an example, the first 4 out of 8 As along the input channel are fed to SAU_{11} and SAU_{12} in the same row, while SAU_{21} and SAU_{22} receive the remaining 4 As. Similar to the IFM distributor, the first 4 out of 8 filters are fed to SAU_{11} and SAU_{21} in the same column, while SAU_{12} and SAU_{22} receive the remaining 4 filters. Note each SAU only receives the corresponding channel inside a filter instead of an entire filter to match the IFM data.

3) Two-Level Accumulator: To adapt our two-level hierarchical architecture, this module consists of an array accumulator (array $accumu.$) and a row-cached accumulator (row-cached $accumu.$) in Fig. 9(a). The array accumulator accumulates the partial results from the data splitter within SAUs along the column direction, with four partial results in four columns of each SAU, and then feeds them into the row-cached accumulator. The row-cached accumulator receives $Y \times 2$ partial results from the 2D array, and it performs $K_r \times N/X \times K_c/3$ accumulations by a row cache to obtain $C \times Y$ OFMs shown in Fig. 2. Without waiting for the 2D array to complete all calculations, this module can produce the final results of OFM in a row-granularity pipelined fashion.

V. Evaluation

A. Experimental Setup

Our SA4 architecture is synthesized and implemented using AMD-Xilinx Vitis HLS and Vivado tools. For hardware performance evaluation, we employ the AMD-Xilinx Ultra96-V2 embedded FPGA board, which is equipped with 70,560 LUTs and 360 DSP slices. Note that all the designs operate at a frequency of 300 MHz, and hardware resource utilization data is extracted from the Vivado post-implementation report.

B. Comparison to 4-bit BaseSA

We compare our SA4 with BaseSA design described in Section III, which is a straightforward integration of the widely used SA architecture and 4bF packing. For a quantitative and step-by-step analysis of the proposed architecture's advantages, in addition to $BaseSA$, we replaced the 2-level IFM loaders of BaseSA with our IFM fetcher, denoted as *BaseSA-IFM Fetcher*. Building on *BaseSA-IFM Fetcher*, we further replaced the 3-level weight loaders of BaseSA with our weight fetcher, denoted as *BaseSA-IFM/W Fetcher*. Finally, we apply the two-level hierarchy concept to divide the original SA with a size of $X \times Y$ into 4×4 PE sets, referred to as *BaseSA-IFM/W Fetcher-Hier*. All BaseSA variants retain FIFO local interconnects and independent FSM control for each PE. That is, the main difference between *BaseSA-IFM/W Fetcher-Hier* and our SA4 is that SA4 uses registers for PE connections and a global FSM inside one SAU.

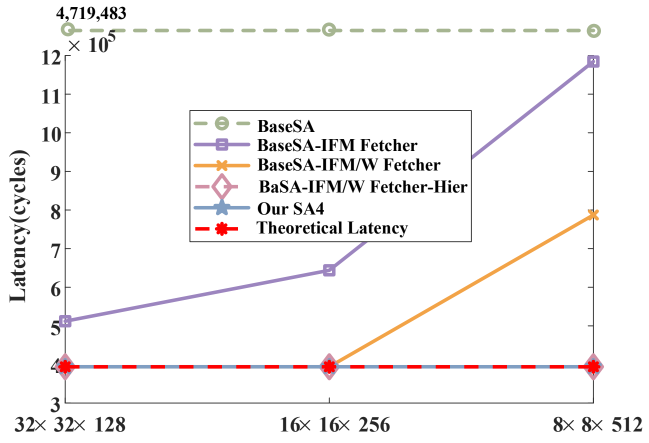


Fig. 10: Latency performance for different convolution sizes ($M = N$,$K_r = K_c = 3$) with the same SA size of 8×8.

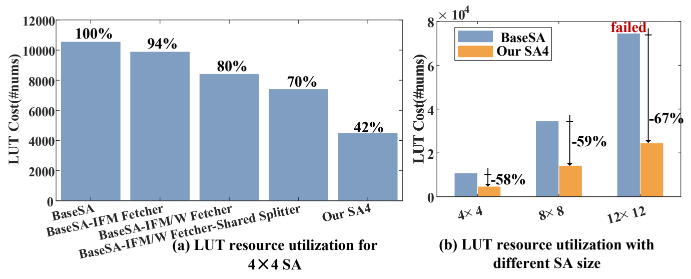
1) Comparison for Latency: As shown in Fig. 10, *BaseSA* exhibits a significant gap between measured and theoretical latency, with errors exceeding 92% with SA size of 8×8. As described in Section III-C, this is due to L1-level IFM loader, which relies on data reuse within the double buffer to hide data loading latency. For the row-temporal WS dataflow, there is no data reuse in the buffer for a single IFM data load, resulting in non-continuous IFM data being supplied to the PEs shown in Fig. 5. That is, the multi-level data loading mechanism in *BaseSA* becomes ineffective when there is no data reuse for the cached data in a single load.

BaseSA-IFM Fetcher only replaces the original 2-level IFM loader with the proposed IFM fetcher and remains 3-level weight loaders. As shown in Fig. 10, the latency is significantly improved for large convolutional spatial sizes (32×32 and 16×16), with the gap between measured and theoretical latency reduced to less than 38%.

With the adoption of our weight fetcher, the latency overhead incurred by double buffer switching in the L1-level weight loader is further mitigated, reducing the latency gap of *BaseSA-IFM/W Fetcher* to 0.3% for large convolutional spatial sizes (32×32 and 16×16). However, as the spatial size of the convolutional layer decreases, the latency error deteriorates again. In the case of a size of $8 \times 8 \times 512$, the latency gap is at 50% for *BaseSA-IFM/W Fetcher*. The reason for this phenomenon is similar to what was discussed in Section IV-A3. The 3-level weight loader leads to discontinuities in weights updates within PEs when the SAU spatial size $Y = 8$ is larger than the column size ($C/2 = 8/2 = 4$) of the convolutional layer.

To address this problem, we adopt the proposed two-level hierarchical strategy to split the large SA into a parallel execution set of 4×4 PEs in *BaseSA-IFM/W Fetcher-Hier*. With this optimization, the latency performance of small-sized convolutions is also ensured.

Finally, with the integration of the proposed IFM fetcher, weight fetcher, and the hierarchical design with 4×4 SAU sets, the proposed SA4 achieves a negligible latency gap to the theoretical optimum with various convolutional spatial sizes.



(a) LUT resource utilization for 4×4 SA

(b) LUT resource utilization with different SA size

Fig. 11: LUT resource utilization comparison.

TABLE II: Overall performance comparison

Architecture	LUT	DSP	GOPS	GOPS/DSP	GOPS/kLUTs
BaseSA (10 $\times$ 10)	52,165 (74%)	100 (28%)	26.0	0.26	0.5
Our SA4 (12 $\times$ 12)	24,239 (34%)	144 (40%)	521.8	3.62	21.5
Our SA4 (16 $\times$ 20)	47,060 (67%)	320 (89%)	1153.2	3.60	24.5

2) Comparison for Resource Consumption: Fig. 11 (a) highlights the resource advantage of our SA4 by illustrating the step-by-step optimization of the baseline resource-intensive designs in 4×4 BaseSA. Firstly, when replacing the 1 L2-level and X L1-level IFM loaders in BaseSA with the proposed IFM fetcher, the LUT consumption of *BaseSA-IFM Fetcher* decreases by 6%. Second, when further replacing the Y L2-level and $X \times Y$ L1-level weight loaders in BaseSA with the proposed weight fetcher, the LUT consumption of *BaseSA-IFM/W Fetcher* significantly decreases, with another reduction of 14%. Third, when the column-shared data splitter is enabled, the resource consumption of *BaseSA-IFM/W Fetcher-Shared Splitter* is further reduced by another 10%. Finally, when our SA4 further employs a global FSM within the 4×4 SAU set and facilitates communication between PEs through cost-effective registers, the resource consumption for the 4×4 SA4 is further reduced by another 28%. In the end, our SA4 only consumes 42% LUTs of the original BaseSA.

Fig. 11(b) further presents the LUT resource utilization of BaseSA and the proposed SA4 with different SA sizes. Compared to the original BaseSA, our two-level hierarchical architecture based on cost-effective SAU sets achieves a reduction of more than 58% in overall LUT resource consumption. It is noteworthy that the 12×12 BaseSA cannot be deployed on the Ultra96-V2 due to excessively high LUT consumption.

3) Comparison for GOPS: Table II displays the maximum SA size ($X \times Y$ in the first column) and the corresponding performance achieved on Ultra96-V2. The performance of *BaseSA* is bottlenecked by LUT resource consumption and large latency gap, as described in the above subsections. With the similar SA size, our SA4 (12×12) achieves a $20\times$ improvement in GOPS compared to *BaseSA* (10×10), with $43\times$ higher GOPS/kLUTs efficiency. Finally, our SA4 with size of 16×20 achieves 1153.2 GOPS with 89% DSP resource utilization, which achieves $13.8\times$ higher GOPS/DSP efficiency and $49\times$ higher GOPS/kLUTs efficiency compared to BaseSA. Note additions in data splitters shown in Fig. 6 and Fig. 3(b) and accumulators shown in Fig. 9 are performed using LUTs.

TABLE III: Performance result of board-level deployment for CNN model(AMD-Xilinx Ultra96-V2 with batch size=1)

Architecture	IoU	Freq (MHz)	LUT	DSP	FPS	FPS/kLUTs	FPS/DSP
UltraNet [1]	0.656	-	43k (61%)	360 (100%)	248	5.8	0.7
UltraNet-HiKonv [22]		-	48k (68%)	327 (91%)	401	8.4	1.2
UltraNet-SA4		300	37k (52%)	252 (70%)	909	24.6	3.6

C. CNN Deployment with SA4

Table III presents the performance results of CNN model deployment based on the proposed SA4. The based model is the 4-bit UltraNet [1] for object detection, which includes 9 convolutional layers and 4 max-pooling layers. The spatial size of 9 convolutions ranges from 160×320 to 10×20. In addition to SA4 with a size of 16×8, we also included the first convolution with 8-bit input precision for the original RGB image, max-pooling, and non-linear units with fused BatchNorm in Ultra96-V2. Compared to the original UltraNet implementation [1] that uses one DSP for packing two 4-bit multiplications, SA4 achieves a $3.7\times$ FPS improvement with $4.2\times$ higher FPS/kLUTs and $5.1\times$ higher FPS/DSP efficiency. Compared to the all-on-chip fully-pipelined structure implementation that uses the same 4bF packing [22], SA4 achieves a $2.3\times$ FPS improvement, with $2.9\times$ higher FPS/kLUTs and $3\times$ higher FPS/DSP efficiency. Note none of these prior studies integrate 4bF packing into SA architectures as we did. In addition, our proposed SA4 also enables layer-wise reuse to make it more versatile for larger CNN models, which is also the focus of our future work.

VI. Conclusion

Extensive research has confirmed that 4-bit quantization precision strikes the optimal balance between DNN model accuracy and hardware performance. In this paper, we have presented the first comprehensive analysis and optimizations to efficiently integrate the latest 4-bit fully DSP packing technique into the widely used systolic array architectures to accelerate 4-bit convolutions in HLS. More specifically, we have designed and implemented the cost-effective SA unit (SAU) to address the LUT resource bottleneck and large latency gap, and further scaled it out using a two-level hierarchical SA architecture. Deployed on the AMD-Xilinx Ultra96-V2 FPGA board, our SA4 achieves 1153.2 GOPS for 4-bit convolutions, exhibiting $13.8\times$ higher GOPS/DSP and $49\times$ higher GOPS/kLUTs efficiency over a straightforward SA and 4bF packing integration. Our case study deployed for the 4-bit UltraNet using SA4 also demonstrates a $2.3\times$ to $3.7\times$ FPS improvement over prior studies.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant 62071360, 62322117, 62371365, and the National Key R&D Program of China under Grant 2023YFE0208100.

References

[1] K. Zhang, J. Guo, B. Song, W. Zhang, and Z. Bao, "Ultranet: A fpga-based object detection for the dac-sdc 2020," 2020. [Online]. Available: https://github.com/heheda365/ultranet

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3836–3847.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] A. Sohrabizadeh, J. Wang, and J. Cong, "End-to-end optimization of deep learning applications," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 133–139.

[6] C. Hao, J. Dotzel, J. Xiong, L. Benini, Z. Zhang, and D. Chen, "Enabling design methodologies and future trends for edge ai: Specialization and codesign," *IEEE Design & Test*, vol. 38, no. 4, pp. 7–26, 2021.

[7] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 40–50.

[8] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[9] Z. Liu, K.-T. Cheng, D. Huang, E. P. Xing, and Z. Shen, "Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4942–4952.

[10] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019.

[11] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1lBj2VFPS

[12] K. Yamamoto, "Learnable companding quantization for accurate low-bit neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 5029–5038.

[13] Z. Bao, K. Zhan, W. Zhang, and J. Guo, "Lsfq: A low precision full integer quantization for high-performance fpga-based cnn acceleration," in *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2021, pp. 1–6.

[14] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 93?104.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[16] R. Xu, S. Ma, Y. Wang, and Y. Guo, "Cmsa: Configurable multi-directional systolic array for convolutional neural networks," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 494–497.

[17] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 58–68.

[18] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[19] S. Das, A. Roy, K. K. Chandrasekharan, A. Deshwal, and S. Lee, "A systolic dataflow based accelerator for cnns," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[20] S. Basalama, A. Sohrabizadeh, J. Wang, L. Guo, and J. Cong, "Flexcnn: An end-to-end framework for composing cnn accelerators on fpga," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 2, pp. 1–32, 2023.

[21] T. Han, T. Zhang, D. Li, G. Liu, L. Tian, D. Xie, and Y. S. Shan, "Convolutional neural network with int4 optimization on xilinx devices," *Xilinx White Paper, WP521*, 2020.

[22] X. Liu, Y. Chen, P. Ganesh, J. Pan, J. Xiong, and D. Chen, "Hikonv: High throughput quantized convolution with novel bit-wise management and computation," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, p. 140–146.

[23] S. Basalama, J. Wang, and J. Cong, "A comprehensive automated exploration framework for systolic array designs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[24] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2019.

[25] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[26] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based cnns on fpgas," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–4.

[27] J. Sommer, M. A. Özkan, O. Keszocze, and J. Teich, "Dsp-packing: Squeezing low-precision arithmetic into fpga dsp blocks," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 160–166.

[28] "Nvdla deep learning accelerator," 2017. [Online]. Available: http://nvdla.org

[29] X. Cai, Y. Wang, X. Ma, Y. Han, and L. Zhang, "Deepburning-seg: Generating dnn accelerators of segment-grained pipeline architecture," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1396–1413.

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

[31] D. J. Moss, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "A customizable matrix multiplication framework for the intel harpv2 xeon+fpga platform: A deep learning case study," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 107–116.