STAR: Sum-Together/Apart Reconfigurable Multipliers for Precision-Scalable ML Workloads

Edward Manca, Luca Urbinati, and Mario R. Casu
Department of Electronics and Telecommunication, *Politecnico di Torino*, Turin, Italy
{edward.manca, luca.urbinati, mario.casu}@polito.it

***Abstract*—To achieve an optimal balance between accuracy and latency in Deep Neural Networks (DNNs), precision-scalability has become a paramount feature for hardware specialized for Machine Learning (ML) workloads. Recently, many precision-scalable (PS) multipliers and multiply-and-accumulate (MAC) units have been proposed. They are mainly divided in two categories, Sum-Apart (SA) and Sum-Together (ST), and have been always presented as alternative implementations. Instead, in this paper, we introduce for the first time a new class of PS Sum-Together/Apart Reconfigurable multipliers, which we call STAR, designed to support both SA and ST modes with a single reconfigurable architecture. STAR multipliers could be useful in MAC units of CPU or hardware accelerators, for example, enabling them to handle both 2D Convolution (in ST mode) and Depth-wise Convolution (in SA mode) with a unique PS hardware design, thus saving hardware resources. We derive four distinct STAR multiplier architectures, including two derived from the well-known Divide-and-Conquer and Sub-word Parallel SA and ST families, which support 16, 8 and 4-bit precision. We perform an extensive exploration of these architectures in terms of power, performance, and area, across a wide range of clock frequency constraints, from 0.4 to 2.0 GHz, targeting a 28-nm CMOS technology. We identify the Pareto-optimal solutions with the lowest area and power in the low-frequency, mid-frequency, and high-frequency ranges. Our findings allow designers to select the best STAR solution depending on their design target, either low-power and low-area, high performance, or balanced.**

***Index Terms*—Machine Learning, Mixed Precision, Reconfigurable Circuits, Precision Scalable Multipliers**

I. Introduction

To strike an optimal balance between accuracy and latency in a Deep Neural Network (DNN), its various layers might need to be quantized with different precision [1]. For this reason, a variety of precision-scalable (PS) multipliers [2], [3] and multiply-and-accumulate (MAC) units [4] have been recently proposed to be used in hardware specialized for Machine-Learning (ML) workloads.

Considering PS multiplier architectures [4], there are two main categories: *Sum-Apart* (SA) (also known as *Sum-Separate* [5]) and *Sum-Together* (ST). Both of them divide the two full-precision input operands into low-precision sub-words, and then perform low-precision multiplications between couples of sub-words in parallel. The ST multipliers sum internally the results of the low-precision multiplications, without requiring an external adder. Instead, the SA ones keep these results separate, i.e., *apart* [5]. Since these multipliers work with multiple sub-words at low precision, they have a higher throughput compared to standard full-precision multipliers, resulting in a faster DNN inference when used in PS MAC units [4]. For example, a MAC unit based on the ST multiplier saves $N-1$ MAC additions, thus reducing the overall latency up to $1/N$, where N is the number of low-precision multiplications performed in parallel. For this reason, PS multipliers already started to be used in hardware accelerators [3], [5], [6], [7], [8] and microprocessors [9], [10] to speedup DNN inference tasks.

Until now SA multipliers and ST multipliers have been proposed as alternative implementations. However, we believe that there is a case for PS multipliers that support both SA and ST modes in a single design, which we call *Sum-Together/Apart Reconfigurable* (*STAR*) multipliers.

These have potential applications in various scenarios, such as within the MAC units of RISC-V cores or in PS hardware accelerators. In fact, the possibility to reconfigure application-specific accelerators enables a more efficient utilization of hardware resources, as they can be dynamically shared to perform different tasks. For instance, in Fig. 1, we show how a STAR-based MAC unit can be used in a single hardware accelerator to enable the support for both 2D and Depth-wise (DW) Convolutions. For 2D Convolution, the STAR multiplier is configured in ST mode to multiply and accumulate pairs of low-precision feature maps (light blue) and weights (orange) by reading them channel-wise (Fig. 1(a)) [3], [8]. An external accumulator further sums up the partial results of the STAR unit until the entire input tensors are scanned and the specific element of the output tensor (green) is computed. For DW Convolution, instead, the STAR multiplier is configured in SA mode to perform multiple low-precision products in parallel (without internal accumulation) between features and weights belonging to different channels, according to the DW Convolution algorithm, Fig. 1(b). In this case the external accumulator is reconfigured, depending on the precision, to keep the accumulated results of the multiplications in $N=2$ or 4 separate elements. STAR could also be used in ST mode for Fully-Connected layers, as described in [3], [5].

With this paper, we make these main contributions to the state of the art (SoA) of PS multipliers:

- We propose for the first time STAR, a new class of PS multipliers that can be reconfigured to operate either in

Acknowledgment: This work was partially supported by the Key Digital Technologies Joint Undertaking under the REBECCA Project with Grant Agreement 101097224, receiving support from the European Union, Greece, Germany, Netherlands, Spain, Italy, Sweden, Turkey, Lithuania, Switzerland.

979-8-3503-4859-0/DATE24/© 2024 EDAA

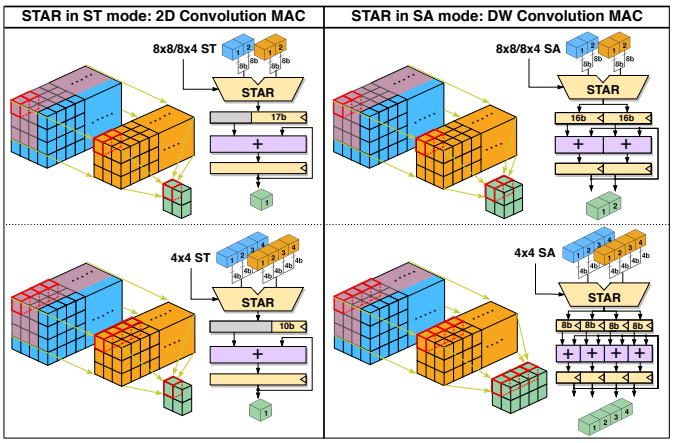

Fig. 1. STAR-enabled reconfigurable MAC for 2D/DW Convolution.

TABLE I
OPERATING MODES OF STAR.

CONFIG	STAR output
16×16	$O_{[31:0]} = A_{[15:0]} \times B_{[15:0]}$
16×8	$O_{[31:0]} = A_{[15:0]} \times B_{[7:0]}$
4×4 ST	$O_{[21:12]} = A_{[3:0]} \times B_{[15:12]} + A_{[7:4]} \times B_{[11:8]} + A_{[11:8]} \times B_{[7:4]} + A_{[15:12]} \times B_{[3:0]}$
8×8 ST	$O_{[24:8]} = A_{[7:0]} \times B_{[15:8]} + A_{[15:8]} \times B_{[7:0]}$
8×4 ST	$O_{[24:8]} = A_{[7:0]} \times B_{[11:8]} + A_{[15:8]} \times B_{[3:0]}$
4×4 SA	$O_{[31:24]} = A_{[15:12]} \times B_{[15:12]}$ $O_{[15:8]} = A_{[7:4]} \times B_{[7:4]}$ $O_{[23:16]} = A_{[11:8]} \times B_{[11:8]}$ $O_{[7:0]} = A_{[3:0]} \times B_{[3:0]}$
8×8 SA	$O_{[31:16]} = A_{[15:8]} \times B_{[15:8]}$ $O_{[15:0]} = A_{[7:0]} \times B_{[7:0]}$
8×4 SA	$O_{[31:16]} = A_{[15:8]} \times B_{[11:8]}$ $O_{[15:0]} = A_{[7:0]} \times B_{[3:0]}$

SA mode or ST mode.

- We derive four STAR multiplier architectures, which support $N = 1$ full-precision multiplication, $N = 2, 4$ parallel multiplications in SA mode, and $N = 2, 4$ parallel multiplications with accumulation (i.e. dot-products) in ST mode, with operands at $16/N$ bits. Two of these architectures are based on the well-known *Divide-and-Conquer* (D&C) and *Sub-word Parallel* (SWP) families [4]. In particular, for the latter we consider a Baugh-Wooley (BW) approach, for which we provide a detailed explanation. The third solution is a 3-way approach inspired from [9] and consisting of three mutually exclusive datapaths with one 16-bit, two 8-bit, and four 4-bit multipliers each. The fourth one consists of two separate multipliers, one SA and one ST, with multiplexed outputs.
- We perform an exploration to determine the best solutions in terms of Power, Performance and Area (PPA) targeting a 28-nm technology. We vary the clock frequency target in a large range so as to determine the best low-power and low-area solution, the best high-performance one, and the best solution for mid-range PPA.

II. Related Work

Recently, all the SoA ST multipliers and PS MAC units were reviewed and benchmarked in [2], [3], [4], including SWP [5], [9] and D&C architectures [11], [12]. Regarding hardware acceleration, in [6] a BW SA was integrated in a convolutional neural network processor. In [7] and [8] two accelerators for DW Convolution and 2D Convolution, leveraging ST-based PS MAC units, were proposed, and their speedup was demonstrated against accelerators relying on standard non-ST MAC units. In [5] the benefits of SA and ST approaches, applied to a BW architecture, were analyzed by comparing two fully-connected (FC) engines in terms of energy consumption, speed, and area utilization. The results highlighted pros and cons of both approaches without declaring a single winner for a given objective. Even RISC-V processors make use of sub-word parallel multipliers and low-precision dot-product units to accelerate quantized neural networks [9], [10]. Building on the well-established effectiveness and benefits of SA and ST multipliers demonstrated in previous research, this paper focuses on the description of STAR architectures and their comparison in terms of PPA. While many SA and ST multipliers appeared in the literature, we are the first to introduce the concept of a STAR multiplier.

A non-recent paper [13] proposed a fixed-point SWP DSP unit capable of operating in SA and ST mode, although at that time these two terms had not been introduced yet. Moreover, it includes extra logic unnecessary for both STAR multipliers and ML workloads (i.e. configurable adders, output saturation logic). For these reasons we exclude it from our analysis.

III. STAR Architectures

The four proposed STAR multiplier architectures are shown in Fig. 2, while the supported configuration modes are reported in Tab. I. A and B are the 16-bit input operands, O is the 32-bit output, and $CONFIG$ is the control signal used to select the operating mode (not shown in Fig. 2 for better readability).

In addition to symmetric configurations (i.e., 16×16, 8×8, and 4×4), we also support asymmetric ones (i.e. 16×8 and 8×4), because they help reduce the memory footprint by allowing an efficient packing of the lowest-precision operands (e.g., DNN weights) without the need to sacrifice precision in the other operands (e.g., DNN activations). For this reason they are present in SoA ML accelerators [4] and microprocessors [9], [10], and are used in commercial ML software tools, like TFLite Micro[1]. Asymmetric configurations are implemented by properly sign-extending operand B, as shown in Fig. 2 in the dashed-line blocks. Should the support for asymmetric configurations be unnecessary, its removal is immediate.

Before delving into the details of STAR architectures, it is worth recalling the taxonomy of [4], which further divides the SA and ST multipliers into SWP and D&C classes. The first class comprises multipliers that can work in full- or

[1] An example of TFLite Micro kernel for a 2D-convolution with asymmetric configurations can be found at https://github.com/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/kernels/conv.cc

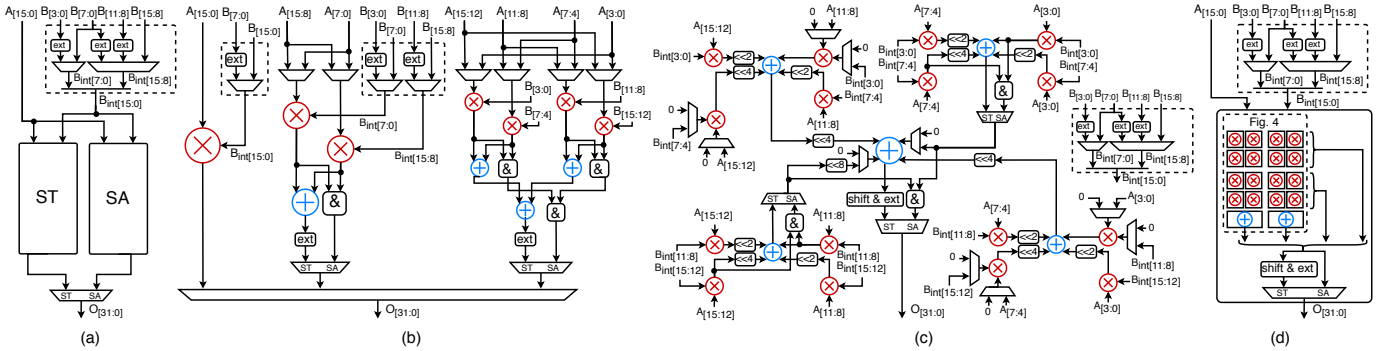

Fig. 2. The proposed STAR architectures: (a) naive, (b) 3-way, (c) D&C, (d) SWP (BW).

reduced-precision mode by selectively gating their arithmetic logic cells, like BW multipliers [5], [6] and Booth-based ones [2]. Instead, the elements of the second class are composed by many low-precision multipliers (e.g. 4-bit) that can be combined by means of shift-add logic to form higher precision multipliers (e.g. 16-bit), like [11] and [12].

The STAR architecture in Fig. 2(a) is based on a naive approach, which consists of multiplexing the outputs of two multiplier components, one SA and one ST, using *CONFIG* as control signal. We created two RTL implementations using this architecture: one called *ST+SA SWP (BW)*, where the two components are SWP multipliers inspired from the ST and SA BW schemes introduced in [5]; the other one called *ST+SA D&C*, where the two components are two D&C multipliers inspired from [11], a variant of [12] that fixes the bitwidth of the two input registers to avoid the explosion of the input memory bandwidth [4]. In particular, we re-implemented [11] with 4-bit multipliers as basic building blocks.

The STAR architecture in Fig. 2(b) is called *3-way* and is inspired by the dot-product unit of the RISC-V core of [9]. It consists of three datapaths activated by signal *CONFIG* in a mutually exclusive way: one 16-bit multiplier for configurations 16×16 and 16×8, two 8-bit multipliers and one adder for configurations 8×8 and 8×4, and four 4-bit multipliers and three adders for configuration 4×4. The blocks named *ext*, located after the last adders of the 8-bit and 4-bit datapaths, sign-extend the low-precision outputs to 32 bits in case of ST operations, while the "&" blocks concatenate them in case of SA operations. We created only one instance of this architecture, letting the logic synthesizer choose the best implementation of the various multipliers.

The third and fourth STAR architectures in Fig. 2(c)-(d) are based on the D&C and SWP PS architectures [4], respectively. The first one, named *STAR D&C*, is derived from [11], while the second one is named *STAR SWP (BW)* because we employed a BW approach. The blocks named *shift & ext* are used to align the output of ST operations to the least significant position [5] and to extend its sign until 32 bits.

In Sec. IV we provide a detailed explanation of the design of STAR SWP (BW), while for the other STAR architectures the diagrams in Fig. 2 are already self-explanatory.

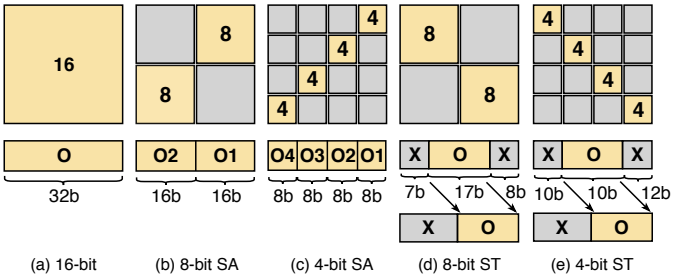

Fig. 3. STAR SWP (BW) operating modes. Partial product matrix (top square) and output (bottom rectangle). Right shift for ST modes only (d)-(e).

IV. STAR SWP (BW) Design

To support all the previously defined operations, STAR SWP (BW) operates in five different modes, which are illustrated in Figs. 3(a)-(e). The upper square in each subfigure represents the BW partial products matrix (PPM), while the lower part shows the 32-bit output. According to the selected operating mode, some partial products (PPs) become active (yellow squares) and contribute to generate the valid output bits (yellow), whereas other PPs are inactive (grey squares) and do not contribute to the final 32-bit result.

The PPM of STAR SWP (BW) is detailed in Fig.4(a). Like any standard BW, each block computes the partial product (PP) between a different pair of bits of the two 16-bit input operands using an AND gate. Then, through a Full Adder (FA), it compresses the output of the AND gate together with the input sum S_i and carry C_i bits coming from the previous row of PPs, and it provides the output sum S_o and carry C_o bits to the blocks of the next row of the PPM. The sixteen S_o bits exiting from the right-most column of the PPM represent the least significant part of the multiplier's output. The most significant part is instead obtained by compressing, through a 16-bit Ripple-Carry Adder (RCA), the S_o and C_o output bits exiting from the last row of the PPM. To deal with signed numbers, a standard BW inverts the PPs of the left-most column and of the last row [14]. This is accomplished by substituting the AND gate with a NAND gate, in each of these blocks. Moreover, to perform the BW algorithm as in [14], the addition of logic 1s is required: this is done via S_i

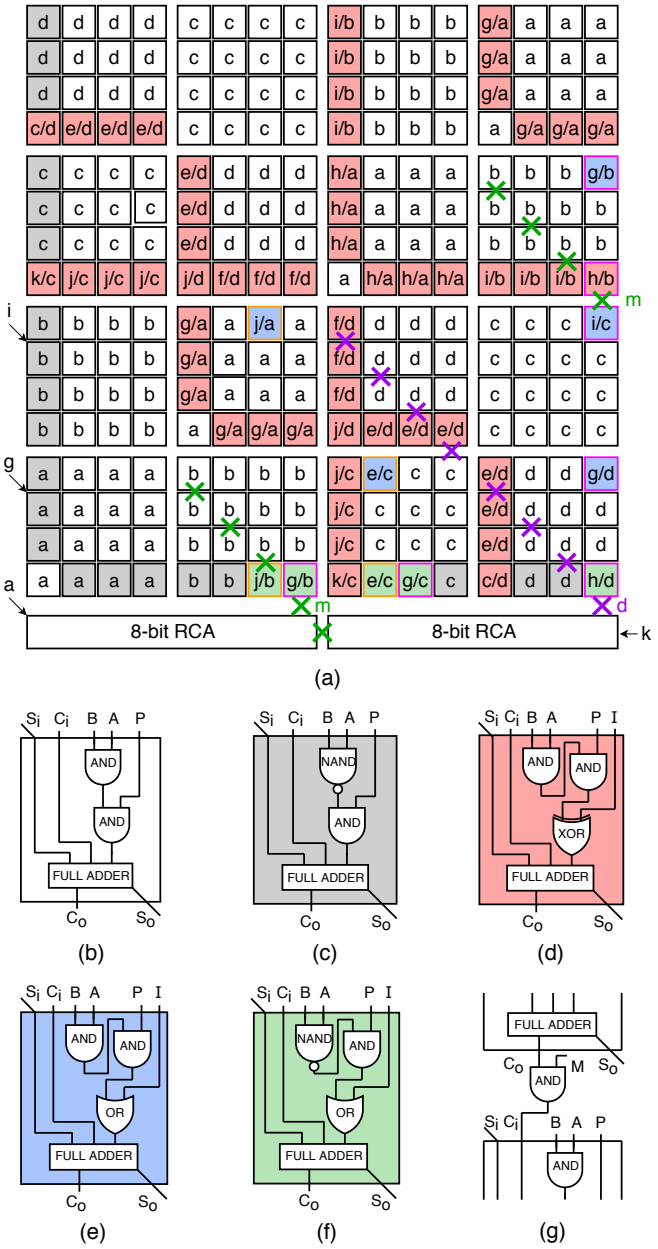

Fig. 4. STAR SWP (BW) PPM with two 8-bit RCAs (a), four versions of PPM blocks (b-f), and carry propagation blocking strategy (g).

inputs of the first row and the left-most column.

To derive STAR SWP (BW) from a standard BW multiplier, we added few logic gates to each block of the PPM, resulting in the five different versions shown in Figs. 4(b)-(f). The reconfiguration of these blocks is achieved through the binary control signals P and I, which are generated from a decoding logic acting on $CONFIG$, the signal controlling the operating mode as in Tab. I. Each block of the PPM receives specific binary values for P and I, according to the letters reported in the block itself (see Fig. 4(a)): when a block contains a single letter, it receives only logic values for P (in fact, white and grey blocks do not have input I); when a block contains two letters, it receives logic values for I and P, respectively (e.g., c/d means $I=$c, $P=$d).

As shown in Fig. 3, depending on the selected operating mode, the output of each block requires potentially to be turned off in order not to contribute to the final result. For this reason, we added an AND gate in all blocks to control the propagation of PPs towards the internal FAs, gating it ($P=0$) or letting it pass ($P=1$). The white and grey blocks shown in Figs. 4(b)-(c), are almost identical to those of a standard BW, with the exception of this AND gate.

When dealing with signed operations at reduced precision, we need to guarantee the PP inversion in the left-most and bottom blocks of all the 8-bit and 4-bit sub-precision multipliers inside the PPM array, as well. These blocks are the red ones in Fig. 4(a), with details of the internal logic in Fig. 4(d): we added an XOR gate to invert the internal PP ($I=1$), or leave it unchanged ($I=0$).

In red blocks we can also force the input of their internal FAs to be logic 1, regardless the actual value of the PP, using a proper control signal configuration (i.e. $P=0$ and $I=1$). This feature can be exploited to add the 1s required by the BW algorithm in place of the S_i inputs of the PPM blocks in the top row and left-most column, thus saving resources and reducing the propagation path of these inputs. However, the red blocks do not cover all the positions that would be required by the insertion of the logic 1s. For this reason we have blue blocks (details in Fig. 4(e)), and green blocks (see Fig. 4(f)), in which an OR gate can force the input of the FA to be a logic 1 (when $I=1$).

In case of SA operations, the carry chains connecting the MSB bit of a sub-word result to the LSB of the next one need to be interrupted. For this reason, we added a few AND gates in selected positions to stop the propagation with control signal $M=0$, as shown in Fig. 4(g). These AND gates also affect the carry chain of the 16-bit RCA, which is halved in two independent 8-bit RCAs when needed. The "X" symbols in Fig. 4(a) mark the positions of these AND gates. Notice the letter associated to each diagonal of "X" symbols, which corresponds to the signal associated to the M inputs of the AND gates of that diagonal. Like P and I, M is derived from $CONFIG$ by the decoding logic and the letters m and d in Fig. 4(a) correspond to the logic values associated to the green and violet "X" symbols, respectively.

In the following we explain in detail how P, I and M enable the reconfiguration of STAR SWP (BW):

1) 16-bit multiplications: For 16x16 and 16x8 multiplications, the PPM is configured to work as in a standard full-precision BW multiplier [14]. In particular, all the blocks are configured with $P=1$ ($\{a,b,c,d\}=1$), and all but the red blocks in the left-most column and the last row are configured with $I=0$ ($\{e,f,g,h,i,j\}=0$). These red blocks instead have to invert the PPs, having $I=1$ ($\{c,k\}=1$). Finally, the addition of logic 1s, required by signed operations [14], is introduced through the carry-in signal of the right 8-bit RCA ($k=1$) and through the most significant FA of the left 8-bit RCA ($a=1$), as clear in Fig. 4(a).

2) SA multiplications: In SA mode, the STAR SWP (BW) multiplier is configured to operate as two or four sub-word parallel BW multipliers.

In the 4×4 configuration, the yellow squares of the PPM in Fig.3(c) correspond to the four groups of 4 x 4 = 16 blocks (64 blocks in total) in the right-to-left diagonal of the PPM array in Fig. 4(a). These four diagonal squares can be seen as four independent 4-bit BW multipliers. The 16 blocks inside each of these 4-bit multipliers are configured with $P=1$ ($a=1$) and $I=0$ ($j=0$), except for the red blocks in the left-most column and in the last row, having $I=1$ ($\{g,h\}=1$). The other blocks in the grey squares in Fig. 3(c) are gated, thus have $P=0$ ($\{b,c,d\}=0$), and become inactive. To perform the addition of 1s required by the BW algorithm [14], we use some of these inactive blocks, since they do not need to propagate the internal PPs when in SA mode. In particular, we used the red, blue and green blocks with magenta borders in Fig. 4(a), which can add a 1 in the FAs chain with $I=1$ ($\{g,h\}=1$, $i=0$). All the other inactive blocks have $I=0$ ($\{c,e,f,i,j,k\}=0$) thus they do not add any 1. In positions where no inactive blocks are available, we exploit the S_i inputs of the blocks in the left-most column of the PPM and the input of the most significant FA of the left-most 8-bit RCA ($\{a,g\}=1$, $i=0$). Finally, to keep the four 8-bit multiplication results separated in the final 32-bit result, we have to interrupt the propagation of the carry-out bits in all the positions marked by the green and violet "X" symbols in Fig.4(a) ($\{d,m\}=0$), as explained before.

In the 8×8 and 8×4 cases, the two yellow squares of the PPM in Fig.3(b) correspond to the two groups of 8 x 8 = 64 blocks (128 blocks in total) in the right-to-left diagonal of the array in Fig.4(a). Also in this case, these two squares can be seen as independent 8-bit BW multipliers. The 64 blocks inside each 8-bit BW are configured with $P=1$ ($\{a,b\}=1$) and $I=0$ ($\{g,j\}=0$), while the red blocks in the left-most column and the last row receive $I=1$ ($\{h,i\}=1$). Inactive blocks require $P=0$ ($\{c,d\}=0$). To add logic 1s, we use again the blue and green blocks present in inactive positions, with magenta border in Fig. 4(a), by setting $I=1$ ($\{i,h\}=1$, $g=0$), while other blocks have $I=0$ ($\{c,e,f,i,j,k\}=0$). When no inactive blocks are available, we use the S_i inputs of the blocks in the left-most column and of the most significant FA of the left 8-bit RCA ($\{a,i\}=1$, $g=0$).

Finally, to maintain the two independent 16-bit multiplication results, we gate the carry-out bits in all the positions marked by the violet "X" symbols in Fig.4(a) ($d=0$, $m=1$).

3) ST multiplications: In ST mode, STAR SWP (BW) always operates in sub-word parallel mode, but the active yellow squares of the PPM are mirrored with respect to SA mode, as shown in Figs.3(d)-(e). This implies that the output sum bits S_o of one yellow square, propagating diagonally from left to right, can be used as inputs for the next yellow square, hence allowing the addition of the low-precision multiplications and ultimately achieving a dot-product operation.

Like in SA mode, blocks belonging to the yellow squares can be seen as independent BW multipliers. These blocks are configured with $P=1$ and $I=0$, except for the red blocks in the left-most column and in the last row, having $I=1$ (precise signal assignment is now trivial and left to the reader). The blocks belonging to the grey squares are gated, thus have $P=0$ to make them inactive. As in SA mode, we used some of these inactive blocks to accomplish the 1s addition required for signed operations. In this case, we used blue and green blocks with orange borders, as visible in Fig. 4(a), which propagate, given $I=1$, a logic 1 to the internal FAs. The remaining inactive blocks receive $I=0$.

As explained in Sec. III, ST operations need to align the final result to the LSB position [4]. Indeed, the *shift & ext* block in Fig. 2(d) is used to right-shift the result by 8 bits for 8x8 and 8x4 operations (Figs. 3(d)), and by 12 bits for 4x4 operation (Figs. 3(e)).

V. Experimental Results

To rank the STAR solutions and establish the best PPA trade-off, we synthesized their RTL description after adding I/O registers using Synopsys Design Compiler and targeting a 28-nm CMOS technology. Figs. 5(a)-(b) report the results of area and power vs clock period, respectively, obtained by varying the target clock frequency from 0.4 to 2 GHz. The clock period also takes the reconfiguration time into account, allowing to change configuration at every cycle. Pareto-optimal points represent the solutions with lowest area or power for a given target clock period. The reported power is an average of eight values obtained when the multipliers are configured in 16-bit mode (16×16 and 16×8), ST mode (8×8, 8×4, and 4×4 ST) and SA mode (8×8, 8×4, and 4×4 SA). For simplicity, the power was determined by applying random input bits evenly distributed between zero and one. While this may not be representative of realistic ML workloads, it still allows for a correct relative comparison. In the future, we plan to use distributions derived from quantized DNNs to provide more accurate estimates of absolute power consumption.

In Fig. 5(a) we observe that STAR SWP is Pareto-optimal in the low frequency range up to 800 MHz (1.25 ns) with runners-up STAR D&C and STAR 3-way. This is because in this range the BW implementation manages to effectively share the logic gates among all the operating modes in the best possible way, while the other solutions have redundant gates that result in larger area. The lower area, and so also lower capacitance, results in lower power for a given clock period, making STAR SWP Pareto-optimal also in power vs clock period up to 900 MHz (1.11 ns), as clear in Fig. 5(b).

In the range 0.9-1.1 GHz both STAR D&C and STAR 3-way become Pareto-optimal in terms of area vs clock period, with STAR SWP third-best. This is because the BW implementation has longer critical paths, which is well known for BW multipliers in general [14] and results in larger gates to satisfy tighter frequency constraints.

In the same middle range, more precisely between 1.0 and 1.2 GHz, the STAR D&C solution manages to obtain the best trade-off in power vs clock period.

In the upper frequency range, STAR 3-way emerges without any contenders and achieves the best area and power. This is

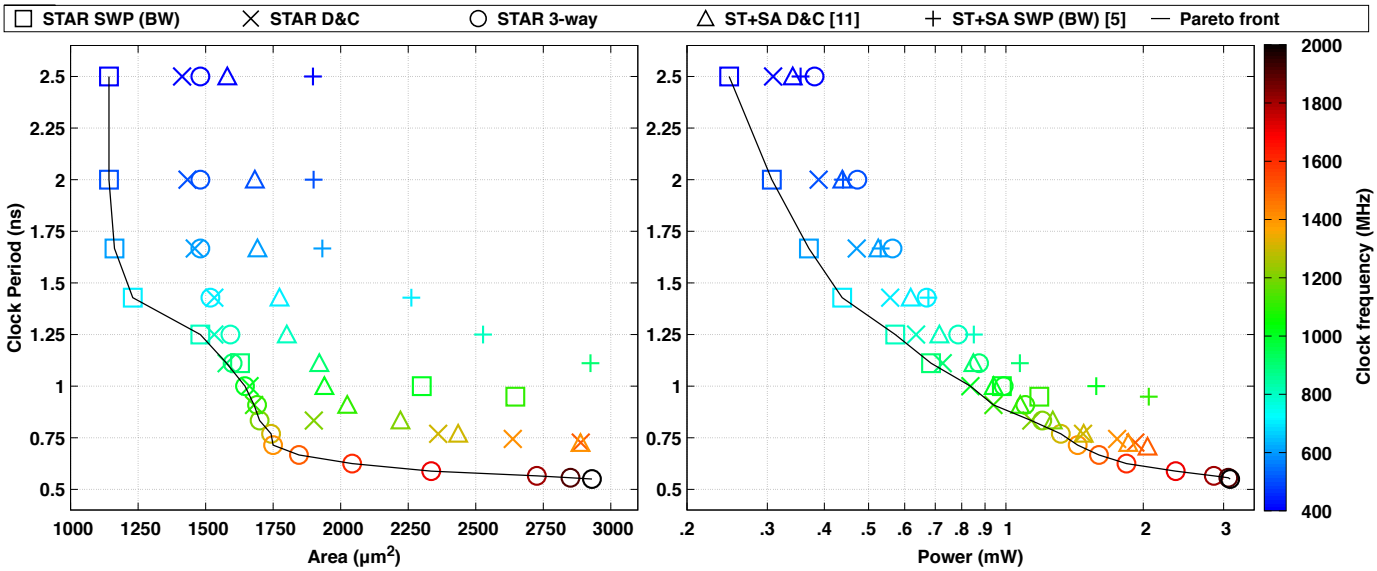

Fig. 5. PPA comparison of STAR architectures.

because the synthesizer freely selects the best implementation for the three types of multipliers, recovering area in the smallest 4x4 ones with shorter paths and using effectively gate sizing for the larger 16×16 one.

As expected, naively combining ST and SA solutions (ST+SA) is not effective and results in Pareto-dominated solutions both in area and power vs clock period. For a fair comparison, we did not analyze ST-only or SA-only designs because they do not support all the STAR configurations.

To summarize, the results of this exploration allow designers to choose the best implementation according to the design target. For low-power and low-area, a STAR SWP solution is the most appropriate implementation. At the other end of the spectrum, for high-performance designs, the best choice is STAR 3-way. In the middle, also STAR D&C can become competitive with STAR 3-way, especially in terms of power.

VI. Conclusion

In this paper we proposed STAR, a new PS multiplier that fuses the SA and ST approaches in a single architecture. We explored different STAR implementations based on the so-called SWP and D&C approaches, showing that SWP obtains least area and power at low frequency targets, while D&C is appropriate for middle range frequency targets. In the high-performance range, instead, a solution that we called STAR 3-way and that combines three architectures dedicated to different precision, emerges as the best. Our results allow designers to select the best STAR solution for their target.

We are now developing STAR-based accelerators for DNN layers, and have already successfully tested a RISC-V core with a PS MAC unit based on a STAR SWP multiplier [15].

References

[1] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Proc. IEEE WACV*, 2016, pp. 1–8.

[2] L. Urbinati and M. R. Casu, "A reconfigurable multiplier/dot-product unit for precision-scalable deep learning applications," in *Proc. of SIE 2022*. Springer Nature Switzerland, 2023, pp. 9–14.

[3] L. Urbinati and M. R. Casu, "Design-space exploration of mixed-precision dnn accelerators based on sum-together multipliers," in *Proc. IEEE PRIME*, 2023, pp. 377–380.

[4] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 9, no. 4, pp. 697–711, 2019.

[5] L. Mei, et al., "Sub-word parallel precision-scalable MAC engines for efficient embedded dnn inference," in *Proc. AICAS*, 2019, pp. 6–10.

[6] 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.

[7] L. Urbinati and M. R. Casu, "A reconfigurable depth-wise convolution module for heterogeneously quantized DNNs," in *Proc. IEEE ISCAS*, 2022, pp. 128–132.

[8] L. Urbinati and M. R. Casu, "A reconfigurable 2d-convolution accelerator for dnns quantized with mixed-precision," in *Proc. ApplePies*. Springer, 2022, pp. 210–215.

[9] M. Gautschi, et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. VLSI Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct 2017.

[10] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "Xpulpnn: Enabling energy efficient and flexible inference of quantized neural networks on risc-v based iot end nodes," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 3, pp. 1489–1505, 2021.

[11] R. Lin, "Reconfigurable parallel inner product processor architectures," *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 261–272, April 2001.

[12] H. Sharma, et al., "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE ISCA*, 2018, pp. 764–775.

[13] X. Zhang, Z. Li, and Q. Zheng, "Design of a configurable fixed-point multiplier for digital signal processor," in *Proc. IEEE PRIMEASIA*, 2009, pp. 217–220.

[14] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design*, 4th ed. Reading, MA: Addison-Wesley, 2011, ch. 11.

[15] 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.