Accelerating Quantized DNN Layers on RISC-V with a STAR MAC Unit

Accelerating Quantized DNN Layers on RISC-V with a STAR MAC Unit

Edward Manca[0009−0006−1342−2677], Luca Urbinati[0000−0001−5317−1960], and Mario R. Casu[0000−0002−1026−0178]

Dept. Electronics and Telecommunications, Politecnico di Torino, Torino, Italy
{luca.urbinati,mario.casu}@polito.it

Abstract. To support quantized neural networks in low-end CPUs, we propose STAR MAC, a reconfigurable multiply-and-accumulate unit based on a modified Baugh-Wooley architecture that operates at a variable reduced precision. We integrated it in a small RISC-V processor called *Ibex* obtaining an acceleration up to 5.8× in Fully-Connected (FC) layers, 3.7× in 2D-Convolution (2DConv) layers, and 2.8× in Depth-Wise Convolution (DWConv) layers, with respect to the original Ibex core (*Orig.*), and up to 4.5× in FC layers, 3.0× in 2DConv layers, and 2.3× in DWConv layers, against a modified Ibex core supporting standard 32-bit MAC operations (*Orig.+MAC*). Area and power in a 28-nm technology with 200 and 600 MHz target clock frequency are 0.015 and 0.017 mm^2, and 1.5 and 4.3 mW, respectively, with a limited overhead within 10% and 3% with respect to *Orig.*, and within 3% and 3% against *Orig.+MAC*.

Keywords: Variable-precision MAC Unit · RISC-V · Deep Learning

1 Introduction

Deep Neural Networks (DNNs) can run in devices with very low power, memory, and silicon resources, thanks to quantization and training methods that allow using low-precision computation at negligible accuracy loss. To fully leverage this paradigm in low-end CPUs, there is a need to support low-precision arithmetic instructions with dedicated Multiply-and-Accumulate (MAC) units.

The Sum Apart (SA) and Sum Together (ST) approaches exploit a typical multiplier to perform sub-word parallel multiplications or multiplication and additions, respectively [1]. Being valid in different contexts, we merge them in our new Sum-Together/Apart Reconfigurable (STAR) MAC unit based on a Baugh-Wooley (BW) multiplier [2]. We embed it in the Ibex core, a small RISC-V, formerly *zero-riscy* [3], adding MAC instructions with reduced precision.

Close to our work in terms of supported operations, but different in goal and implementation, is the *Ri5cy* processor [4]. In fact, Ri5cy aims to high performance using many low-precision functional units (FUs) to support specific operations at reduced precision, while we aim to reuse the same FU via reconfiguration to reduce resources. The *XMPI* processor [5] follows the approach of

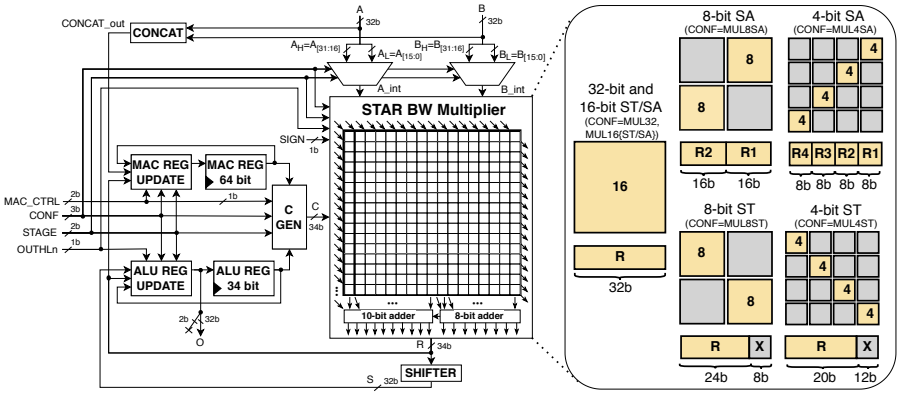

Fig. 1. The STAR MAC unit implemented in the Ibex core (left side) and the five operating modes of the STAR multiplier (right side), where the top square of each configuration is the BW PPM and the bottom rectangle is the multiplier's output.

[4] of more dedicated resources, but adds support for operations at even lower precision. Different approaches than SA/ST, for low-precision operation MAC units in RISC-V cores, are *divide-and-conquer* [6] and *bit-serial* [7]. However, the first one results in a larger multiplier structure than the BW one, while the second one, being serial in nature, is inherently low-performance.

2 Hardware Design

The Multiplier/Divider (*MULT/DIV*) unit of the Ibex core [3] is a multiplication and division block controlled by a state machine. In this paper we use the *Small* version of the Ibex processor[1], which uses the *Fast MULT/DIV* unit to extend the base RISC-V instruction set with the RV32M ISA extension. In the original design, the Fast *MULT/DIV* unit computes 32-bit MUL operations (namely MUL, MULH, MULHSU, and MULHU) in 3/4 clock cycles by iteratively using a behaviorally described 17-bit multiplier along with a 34-bit adder and the ALU register *ALU REG* defined in the ALU unit.

In this work we replaced this 17-bit multiplier with our sub-word parallel Sum-Together/Apart Reconfigurable (STAR) multiplier, described in detail in Sec. 2.1, obtaining the proposed STAR MAC unit shown in the left side of Fig. 1. The STAR multiplier executes one 16-bit multiplication, or N=2, 4 parallel low-precision multiplications with 16/N-bit operands for both SA/ST operating modes. The STAR MAC unit still supports all RV32M instructions and preserves the iterative approach of the original *MULT/DIV* unit, requiring multiple clock cycles—i.e., *iteration cycles (ICs)*—to complete all the instructions. Moreover, it

[1] The Ibex processor version used in our work can be found at: https://github.com/lowRISC/ibex/tree/8db89a9dfc0cb08371d079cfc76e83d9ffc66480

Table 1. New MAC instructions and number of required clock cycles.

Instruction	Operation	Clock cycles
MAC	$O_{[31:0]} = A_{[31:0]} \times B_{[31:0]} + C_{[31:0]}$	3
MAC16ST	$O_{[31:0]} = A_{[15:0]} \times B_{[31:16]} + A_{[31:16]} \times B_{[15:0]} + C_{[31:0]}$	2
MAC8ST	$O_{[23:0]} = A_{[7:0]} \times B_{[31:24]} + A_{[15:8]} \times B_{[23:16]} + A_{[23:16]} \times B_{[15:8]} + A_{[31:24]} \times B_{[7:0]} + C_{[31:8]}$	2
MAC4ST	$O_{[19:0]} = A_{[3:0]} \times B_{[31:28]} + A_{[7:4]} \times B_{[27:24]} + A_{[11:8]} \times B_{[23:20]} + A_{[15:12]} \times B_{[19:16]} + A_{[19:16]} \times B_{[15:12]} + A_{[23:20]} \times B_{[11:8]} + A_{[27:24]} \times B_{[7:4]} + A_{[31:28]} \times B_{[3:0]} + C_{[31:12]}$	2
MAC16SA	$O_{[15:0]} = A_{[15:0]} \times B_{[31:16]} + C_{[31:0]}$	2
MAC8SA	$O_{[15:0]} = A_{[7:0]} \times B_{[23:16]} + C_{[15:0]}$; $O_{[31:16]} = A_{[15:8]} \times B_{[31:24]} + C_{[31:16]}$	2
MAC4SA	$O_{[7:0]} = A_{[3:0]} \times B_{[19:16]} + C_{[7:0]}$; $O_{[15:8]} = A_{[7:4]} \times B_{[23:20]} + C_{[15:8]}$ $O_{[23:16]} = A_{[11:8]} \times B_{[27:24]} + C_{[23:16]}$; $O_{[31:24]} = A_{[15:12]} \times B_{[31:28]} + C_{[31:24]}$	2
MAC16SAH	$O_{[31:0]} = A_{[31:16]} \times B_{[15:0]} + C_{[63:32]}$	2
MAC8SAH	$O_{[15:0]} = A_{[23:16]} \times B_{[7:0]} + C_{[47:32]}$; $O_{[31:16]} = A_{[31:24]} \times B_{[15:8]} + C_{[63:48]}$	2
MAC4SAH	$O_{[7:0]} = A_{[19:16]} \times B_{[3:0]} + C_{[39:32]}$; $O_{[15:8]} = A_{[23:20]} \times B_{[7:4]} + C_{[47:40]}$ $O_{[23:16]} = A_{[27:24]} \times B_{[11:8]} + C_{[55:48]}$; $O_{[31:24]} = A_{[31:28]} \times B_{[15:12]} + C_{[63:56]}$	2
MACSET	MAC $REG_{[63:0]} = \{A_{[31:0]}, B_{[31:0]}\}$	1

supports MAC operations, which were not available in the original *MULT/DIV* unit: we added a standard 32-bit MAC instruction, which takes 3 clock cycles, and custom low-precision ST/SA MAC instructions (MACxSA and MACxST, $x \in \{4, 8, 16\}$), which exploit the STAR multiplier and take 2 clock cycles. All the new instructions are reported in Table 1, where A and B are the two-input operands, C is the value to accumulate, and O is the output of the STAR MAC unit. Low-precision instructions support signed operands only. We also defined the equivalent MAC instructions (MACH, MACHSU, and MACHU) of the corresponding MUL instructions (MULH, MULHSU, and MULHU). However, since these six instructions will not be used in the our benchmarks of quantized DNN layers of Sec. 3, we do not comment any further on them.

2.1 STAR multiplier

The core of our STAR MAC unit is the 16-bit STAR multiplier, whose BW architecture is depicted in Fig. 2. It consists of a 16×16 partial product matrix (PPM) that works as a typical BW multiplier [2]: each block receives a pair of bits from the two input operands, computes the partial product (PP) between them using an AND gate; then it generates the output sum *So* and carry *Co* bits by compressing the PP, along with the input sum *Si* and input carry *Ci* bits, using a Full Adder (FA). The sum bits propagate diagonally, while the carry bits propagates vertically, connecting all the blocks of the PPM (not shown in Fig. 2(a) for better clarity).

To support low-precision SA and ST operating modes, in addition to the standard full-precision 16-bit multiplication, the PPM requires to be reconfigured

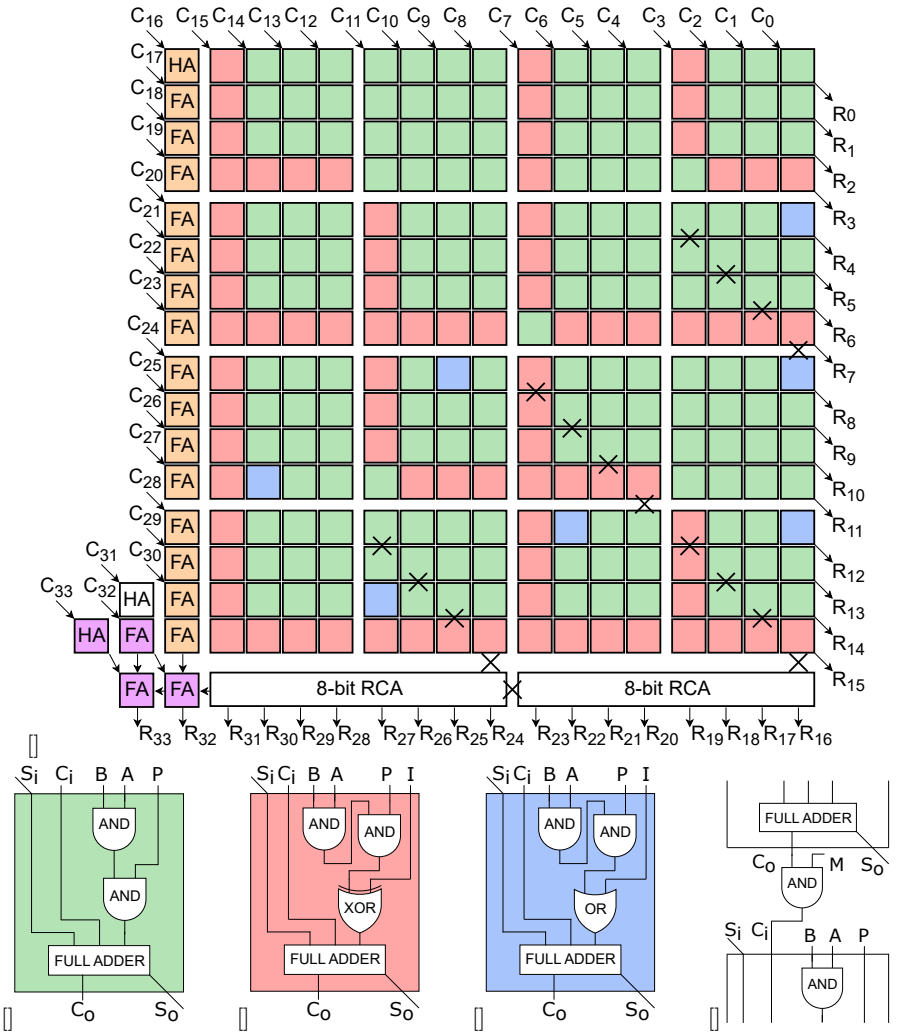

Fig. 2. PPM of the STAR BW multiplier (a), three versions of PPM blocks (b-d), and carry propagation blocking strategy (e).

in one of the five ways illustrated in the right box of Fig. 1, where the top square of each configuration represents the PPM and the bottom rectangle corresponds to the final 32-bit result R of the STAR multiplier. The PPs in the yellow areas of the PPM contribute to generate the yellow bits of R, while the PPs in the grey areas are gated and do not contribute to it. To obtain this behavior, we created three configurable blocks (green, red and blue, Fig. 2(b)-(d)) by adding few logic gates controlled by signals P (*propagate*) and I (*invert*), and we placed them in specific positions in the STAR PPM.

As shown in Fig. 1, depending on the STAR configuration, any block of the PPM could be potentially be gated. Thus, each block of the STAR PPM must have the possibility of blocking the PP propagation. For this reason all the three configurable blocks have an AND gate which stops the propagation of the PP towards the FA when $P = 0$.

To support signed operations, we need to take into account two aspects. The first is that the PPs belonging to the blocks in the left-most column and bottom row of a typical BW PPM have to be inverted. Since in any ST/SA configuration of STAR each yellow square of Fig. 1 behaves like an independent low-precision BW multiplier, we need to guarantee the PP inversion in all the left and bottom blocks of these sub-precision multipliers, as well. For this task, we conceived the red block (Fig. 2(c)), which inverts the output of the PP when $I = 1$ via an XOR gate. The second aspect is the insertion of logic 1s in specific positions of the PPM to accomplish the BW algorithm [2]. Typically, the insertion of these 1s occurs through the *Si* inputs of the top and left-most blocks of the BW PPM. However, we decided to use those inputs to add the third operand of the MAC operation, i.e. the 34-bit signal C. Hence, when possible, we generate these 1s using the inactive red blocks (with $P = 0$ and $I = 1$) inside the PPM. When not possible, we inserted the blue blocks (Fig. 2(d)), which use an OR gate to force the PP to be logic 1 (again with $P = 0$ and $I = 1$).

To guarantee that R is separated in two or four independent subwords for 8-bit and 4-bit SA operations, we have to interrupt the propagation of the carry between two consecutive low-precision multiplications. Therefore, we inserted a few AND gates, controlled by signal M (Fig. 2(e)), between two vertically adjacent blocks, as shown in Fig. 2(a) by the X symbols. We also added an AND gate between the two 8-bit Ripple-Carry Adders (RCAs).

To compute MULH, MULHU, and MULHSU 32-bit instructions, as well as MACH, MACHU, and MACHSU, the STAR MAC unit requires 4 cycles. Thus, to prevent overflow while adding the intermediate multiplication results, the RCA on the left needs to be extended to 34 bits (see the two pink FAs at the bottom left corner of Fig. 2(a)). Moreover, the column of yellow Full-Adders (FA) on the left side of the PPM is necessary to add the two most significant bits of the 34-bit RCA, which in some cases have to be sign extended (namely, for MULH instructions at cycle 2 and 3, for MULHSU instructions at cycle 3). The final output becomes the concatenation of the eighteen bits at the output of the RCA (R [33:16]), which compresses the *So* and carry *Co* bits of the last row of the PPM, and the sixteen *So* bits exiting from the rightmost column of the PPM (R [15:0]). As already stated in Sec. 1, MULH instructions are not used in the experiments of this paper, thus further details on this matter are reserved for future work.

Finally, signals P and I are generated by a combinational logic that uses the following control signals: *CONF*, a 3-bit signal indicating the actual operation type (SA/ST at 32/16/8/4 bits); *OUTHLn*, a single bit indicating which part (32-bit high/low) of the 64-bit final result has to be returned by the instruction (note: SA operations can have multiple separate results in the 32-bit high/low of

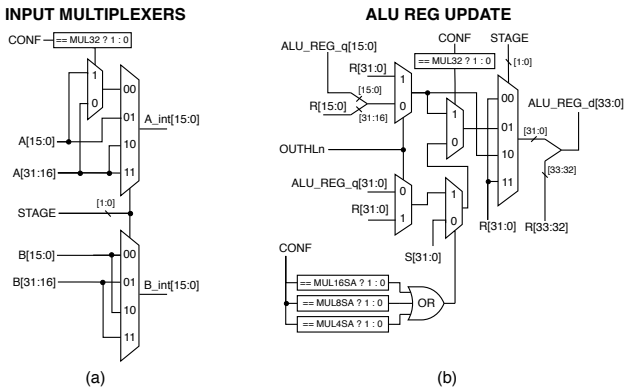

Fig. 3. Schematics describing the input multiplexers (of Fig. 1) (a) and the *ALU REG UPDATE* block (b). These are actually described in Verilog and synthesized, thus the resulting logic might be different yet functionally equivalent to these schematics.

the result, according to Table 1); *SIGN*, a bit indicating if the instruction works with signed or unsigned operands; *STAGE*, 2 bits indicating the current clock cycle of the Finite State Machine (FSM) inside the *MULT/DIV* unit. Moreover, the first three signals come directly from the instruction opcode.

2.2 STAR MAC unit

Like in the original *MULT/DIV* unit, the 32-bit input operands A and B come from the register file. The proper upper and lower 16-bit chunks from A and B are selected by a couple of multiplexers depending on the IC. However, in our MAC STAR unit these multiplexers are controlled not only by *STAGE*, but also by *CONF* because we have to deal with ST and SA instructions (Fig. 3(a)).

For MUL instructions, at each IC the combinational logic *ALU REG UPDATE* in Fig. 1 selects which subword of R to store in the register *ALU REG* for the next iteration. In fact, the content of *ALU REG* is used by the multiplier in the subsequent cycle to continue the iterative multiplication process. *ALU REG UPDATE* takes as inputs the multiplier's output R, the $SHIFTER$' output S and the content of *ALU REG*, and is controlled by *STAGE*, $OUTHLn$ and *CONF* (the first two are present also in the original *MULT/DIV* unit), as reported in Fig. 3(b).

For MAC instructions, while *ALU REG* serves the same purpose as in the MUL instructions case, which is to temporarily store R to use it in the next IC, we introduced the 64-bit accumulation register *MAC REG* along with the combinational logic *MAC REG UPDATE* responsible for its updating. In particular, for 32-bit MAC instructions the result is accumulated in *MAC REG* up to 32-bit precision, while for 32-bit MACH, MACHSU, MACHU instructions the result is accumulated up to 64-bit precision. For SA instructions, instead, *MAC REG* stores the accumulated results in separate data chunks of 32 (MAC16SA),

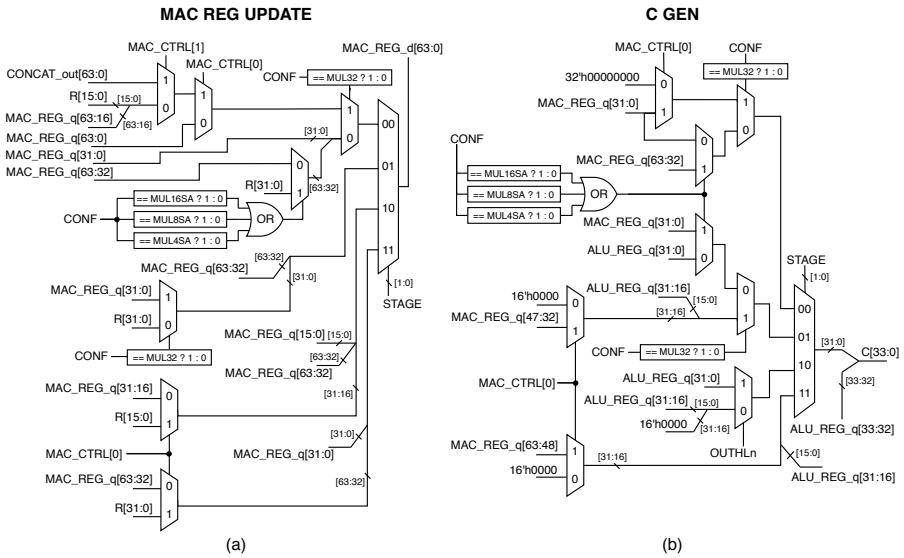

Fig. 4. Schematics of *MAC REG UPDATE* (a) and *C GEN* (b) blocks. These are actually described in Verilog and synthesized, thus the resulting logic might be different yet functionally equivalent to these schematics.

16 (MAC8SA), or 8 bits (MAC4SA). For ST instructions results are accumulated and stored with a precision of 32 (MAC16ST), 24 (MAC8ST), and 20 bits (MAC4ST). *MAC REG UPDATE*, whose schematic is reported in Fig. 4(a), behaves similarly to *ALU REG UPDATE*: it takes R, the content of *ALU REG* and the concatenation of A and B (i.e. the output of the *CONCAT* block, called *CONCAT_out*) to update the register itself according to *STAGE*, *CONF* and *MAC_CTRL* control signals. MACSET is commonly used to initialize *MAC REG*, with the value of *CONCAT_out*, at the beginning of a software routine that requires a series of MAC instructions.

For both MUL and MAC instructions, the third input C to the STAR BW multiplier comes from the combinational logic *C GEN*. Through control signals *MAC_CTRL*, *STAGE* and *CONF*, *C GEN* selects *ALU REG* in case of standard MUL operations, or the proper sub-words of *MAC REG* and *ALU REG* in case of MAC operations, as shown in the schematic of Fig. 4(b). *MAC_CTRL* is a 2-bit signal which indicates if the current instruction is a MAC (01b), a MACSET (11b), or a MUL (00b). The final output O of the STAR MAC unit corresponds to the 32 LSBs of the output of *ALU REG UPDATE*, which can be: R for all non-ST operations and for the MAC16ST operation, or S for MAC8ST and MAC4ST operations. In fact, in this latter case the *SHIFTER* is used to get rid of the 8 or 12 invalid LSBs of R (as reported in grey in Fig. 1, right side, and in Fig. 3(b)).

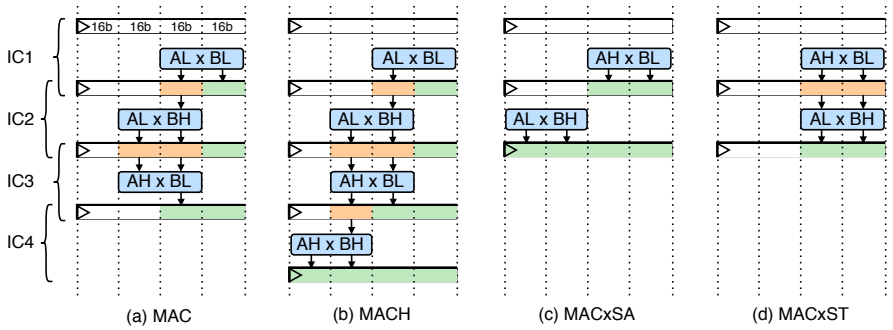

Fig. 5. Visual representation of the ICs for the MAC instructions of the STAR MAC unit.

Fig.,5 illustrates the sequence in which operands are sent to the STAR multiplier at each IC, for MAC (Fig. 5(a)), MACH (Fig. 5(b)), MACxSA (Fig. 5(c)) and MACxST (Fig. 5(d)) instructions. The blue rectangle shows the 16-bit subwords of A and B processed by the STAR multiplier at the corresponding IC. The sharp-edge rectangle is a "virtual" register used only for the sake of this explanation, which contains the bits generated by STAR at each IC. The green bits are already the final bits of the result, while the orange bits represent partial results that, through the third input C, are accumulated by STAR at the next IC with the product between A and B. In the STAR MAC unit the green bits are actually stored in *MAC REG*, while the orange bits are stored in *ALU REG*.

3 Experimental Results

The possibility to perform parallel low-precision MAC operations, due to the STAR MAC unit, enables the acceleration of quantized DNN layers on the Ibex core. We call *STAR-based* the new Ibex with the STAR MAC unit. To have a fair comparison in area, power and execution latency of DNN layers, we decided to slightly modify the original Ibex (*Orig.*) to add the support to MAC instructions, resulting in a third Ibex implementation (*Orig. + MAC*).

We synthesized all the three Ibex versions on a 28-nm technology (0.9 V) at 200 and 600 MHz, two tight constraints for this processor [3]. The results of area and estimated power are reported in Table 2. As we can see, *STAR-based* has limited area and power overheads compared to *Orig.*; when compared to *Orig. + MAC*, the area overhead decreases even further.

Then, we evaluated the performance of *STAR-based* vs *Orig. + MAC* on these quantized layers: fully-connected (FC), 128-256 input and 32 output neurons; 2D convolutional (2DConv), 32-128 input channels (inch.), 8x8 feature map (fmap.), 3x3 kernel (kern.) and 4 output channels (outch.); depth-wise convolutional (DWConv), 16-64 channels (ch.), 16x16 feature map size and 3x3 kernel. We

Table 2. Logic synthesis results of *Orig.*, *Orig. + MAC* and *STAR-based*.

Ibex type @ clk. freq. [MHz]	**Area [µm^2]**	**Power [mW]**	**Ibex type @ clk. freq. [MHz]**	**Area [µm^2]**	**Power [mW]**
Orig. @ 200	14241	1.46	*Orig.* @ 600	15135	4.17
Orig.+MAC @ 200	14658 (+2.9%)	1.50 (+2.3%)	*Orig.+MAC* @ 600	15588 (+3.0%)	4.27 (+2.6%)
STAR-based @ 200	15299 (+7.4%)	1.50 (+2.3%)	***STAR-based*** @ 600	16528 (+9.2%)	4.29 (+2.9%)

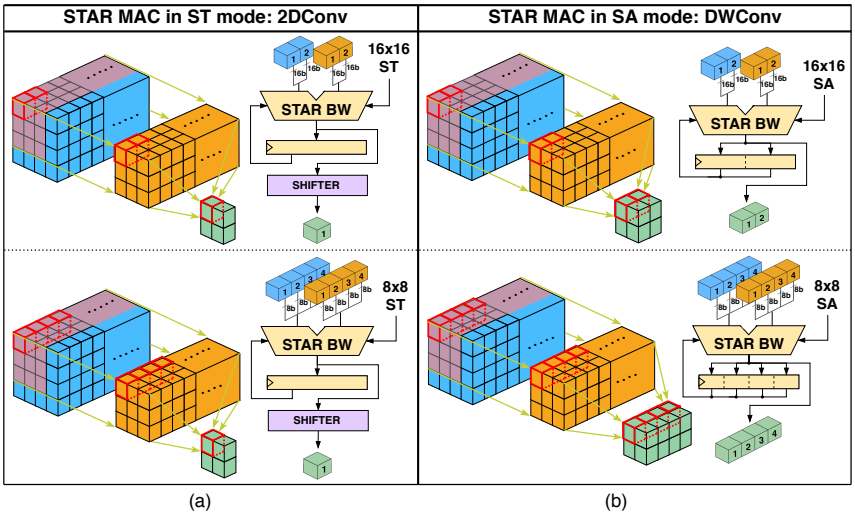

Fig. 6. Two examples of STAR MAC used in 2DConv and DWConv layers: MAC16{ST/SA} in the upper part, MAC8{ST/SA} in the lower part.

assume that the low-precision operands are already packed/prearranged in memory in such a way that the instructions listed in Table 1 can be directly executed without incurring any costs due to operand reordering. To exploit the acceleration provided by STAR at reduced precision, we coded these layers with an output-stationary dataflow [8]. We used MACxST instructions for FC/2DConv and MACxSA instructions for DWConv ($x \in \{4, 8, 16\}$) because the former layers require the accumulation along the input channels/neurons dimension, while the latter does not [9]. In this regard, Fig. 6 shows two examples, for 2DConv (MAC16ST and MAC8ST, Fig. 6a) and DWConv (MAC16SA and MAC8SA, Fig. 6b), on how low-precision input features (light blue) and weights (orange) are read from the corresponding tensors and packed in the two 32-bit input registers of the STAR MAC unit to produce the output features (green). Both the

Table 3. (a) Average speedup (i.e. ratio between clock cycles) of *STAR-based* vs *Orig.* and (b) of *STAR-based* vs *Orig. + MAC*, for three DNN layers for different features and weights bitwidths.

DNN layers @ Instruction	**Avg. Speedup *STAR-based* vs. *Orig.* (y = 16, 8, 4)**	**Avg. Speedup *STAR-based* vs. *Orig.+MAC* (y = 16, 8, 4)**
FC (128-256 input, 32 output) @ MACyST	2.0x, 3.3x, 5.8x	1.7x, 2.7x, 4.5x
2DConv (32-128 inch., 8x8 fmap., 3x3 kern., 4 outch.) @ MACyST	1.6x, 2.3x, 3.7x	1.4x, 1.9x, 3.0x
DWConv (16-64 ch., 16x16 fmap., 3x3 kern.) @ MACySA	1.4x, 1.9x, 2.8x	1.3x, 1.6x, 2.3x

approaches can be easily extended to MAC4{ST/SA} instructions and to the FC layer [9]. Furthermore, to have a fair comparison between the *STAR-based* and *Orig.,+,MAC* approaches, we ensured that the code of each DNN layer for both Ibex cores featured an equal number of memory accesses. As a result, the comparison exclusively highlights the computational advantages stemming from the new STAR MAC unit.

[table]skip=4pt The results of the average speedup, i.e. the ratio between the clock cycles, of *STAR-based* vs *Orig. + MAC* for these three DNN layers at different features and weights precision is reported in the last column of Table 3. For the 16, 8, and 4-bit cases, respectively, the average speedup for FC is 1.7×, 2.7× and 4.5×; for 2DConv is 1.4×, 1.9× and 3.0×; for DWconv is 1.3×, 1.6× and 2.3×. We also report the average speedup of *STAR-based* vs *Orig.* for the same DNN layers in the second column of Table 3. We ensured, once again, that the code of these DNN layers had the same number of memory accesses for both processors for a fair comparison. For this comparison, the average speedup for the 16, 8, and 4-bit cases is, respectively, 2.0×, 3.3× and 5.8× for FC; 1.6×, 2.3× and 3.7× for 2DConv; is 1.4×, 1.9× and 2.8× for DWconv.

4 Conclusions

Our proposed STAR MAC represents a promising solution for enabling quantized neural networks on low-end CPUs. With substantial acceleration gains for typical DNN layers and minimal overhead in terms of area and power consumption, it offers a viable option for efficient deep learning inference in resource-constrained environments.

References

1. Camus, V., Mei, L., Enz, C., and Verhelst, M.: Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-

Network Processing. IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JESTCS), **9**(4), 697–711 (2019).
2. Weste, N. H. E., and Harris, D. M.: CMOS VLSI Design. 4th edn. Addison-Wesley (Pearson), Boston (2011).
3. Schiavone, P. et al.: Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In: Proceedings 27th IEEE International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 1–8. IEEE, Thessaloniki, Greece (2017).
4. Gautschi, M. et al.: Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, **25**(10), 2700–2713 (2017).
5. Ottavi, G., Garofalo, A., Tagliavini, G., Conti, F., Benini, L., and Rossi, D.: A mixed-precision RISC-V processor for extreme-edge DNN inference. In: Proceedings IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 512–517. IEEE, Limasson, Cyprus (2020).
6. Devic, G., France-Pillois, M., Salles, J., Sassatelli, G., and Gamatié, A.: Highly-adaptive mixed-precision MAC unit for smart and low-power edge computing. In: Proceedings 19th IEEE International New Circuits and Systems Conference (NEWCAS), pp. 1–4. IEEE, Toulon, France (2021).
7. RK, R., Sinha, S., and Rao, N.: Variable Bit-Precision Vector Extension for RISC-V Based Processors. In: Proceedings 14th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 114–121. IEEE, Singapore (2021).
8. Chen, Y. -H., Emer, J., and Sze, V.: Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In: Proc. ISCA, pp. 367–379. ACM/IEEE, Seoul, South Korea (2016).
9. Urbinati, L., and Casu, M. R.: Design-Space Exploration of Mixed-precision DNN Accelerators based on Sum-Together Multipliers. In: Proceedings 18th International Conference on PhD Research in Microelectronics and Electronics (PRIME), pp. 377–380. IEEE, Valencia, Spain (2023).