Sub-Word Parallel Precision-Scalable MAC Engines for Efficient Embedded DNN Inference

Linyan Mei[1], Mohit Dandekar[1], Dimitrios Rodopoulos[2], Jeremy Constantin[2],
Peter Debacker[2], Rudy Lauwereins[2], Marian Verhelst[1]
[1]MICAS, Dept. of Electrical Engineering, KU Leuven, Belgium
[2]imec, Belgium
linyan.mei@esat.kuleuven.be, nmorph@imec.be

***Abstract*—To enable energy-efficient embedded execution of Deep Neural Networks (DNNs), the critical sections of these workloads, their multiply-accumulate (MAC) operations, need to be carefully optimized. The SotA pursues this through run-time precision-scalable MAC operators, which can support the varying precision needs of DNNs in an energy-efficient way. Yet, to implement the adaptable precision MAC operation, most SotA solutions rely on separately optimized low precision multipliers and a precision-variable accumulation scheme, with the possible disadvantages of a high control complexity and degraded throughput. This paper, first optimizes one of the most effective SotA techniques to support fully-connected DNN layers. This mode, exploiting the transformation of a high precision multiplier into independent parallel low-precision multipliers, will be called the Sum Separate (SS) mode. In addition, this work suggests an alternative low-precision scheme, i.e. the implicit accumulation of multiple low precision products within the multiplier itself, called the Sum Together (ST) mode. Based on the two types of MAC arrangements explored, corresponding architectures have been proposed to implement DNN processing. The two architectures, yielding the *same throughput*, are compared in different working precisions (2/4/8/16-bit), based on Post-Synthesis simulation. The result shows that the proposed ST-Mode based architecture outperforms the earlier SS-Mode by up to $\times$1.6 on Energy Efficiency (TOPS/W) and $\times$1.5 on Area Efficiency (GOPS/mm^2).**

***Index Terms*—Digital Design, Deep Neural Network, Inference, Multiplier, Scalable Precision, Synthesis, Energy Efficiency.**

I. Introduction

Deep learning currently plays a crucial role in the rapid development of artificial intelligence (AI). It enables tremendous performance improvements in various machine learning applications, such as computer vision [1]–[3], speech recognition [4], [5], natural language processing [6], etc. Moving deep neural network (DNN) inference from the cloud to embedded devices is a trend today to get rid of the limitation of wireless bandwidth and to protect users' privacy. However, running DNN-based applications on battery-constrained and memory-limited systems puts extremely high demand on both throughput and energy efficiency for embedded platforms [7].

A good approach to alleviate this stress is to compute at a reduced computational precision, using appropriate per-layer or per-channel quantization of activations and weights, which can largely save resources without excessive accuracy loss [8], [9]. Yet, it has been shown that there is a wide variety of precision requirements among (non-binary) neural networks [10], [11], ranging from 2 to 16 bit. As such, DNN hardware platforms should better support run-time precision scalability towards energy-efficient and application-flexible embedded DNN execution. Targeting this goal, the design of the precision-variable multiply-accumulate (MAC) operator, representing the dominant DNN compute kernel, should be carefully optimized.

Most SotA solutions rely on optimized low precision multipliers with a separately optimized precision-variable accumulation scheme to implement the adaptable precision MAC operation, which might lead to low throughput and high control complexity. This paper, in contrast, uses a single high precision multiplier deployed in parallel low precision modes. The main contributions of this work are as such:

1) We enhance the Baugh-Wooley multiplier with two alternative sub-word parallellization schemes: Sum Separate (SS) inspired from [12], and Sum Together (ST) designed by us. Hybrid combinations of the two are also explored.
2) Based on the two multiplier configurations, SS mode and ST mode, corresponding processing element (PE)-array-level architectures have been proposed to support the DNN inference execution.
3) A detailed comparison between the SS-based and ST-based architectures is made under different precision (2/4/8/16-bit) modes, taking throughput, energy efficiency, and area efficiency into consideration.

II. Related Work

Recently, several multiplier architectures were introduced in the SotA [12]–[15] to support run-time precision adjustment. In general, we can divide precision-configurable MAC units into two categories, temporal-based and spatial-based. Temporal-based structures realize precision tuning through iterative sequencing of operations, adding more resolution in each step, as in [14]. While spatial-based methods implement variable computing precision 1) through an agglomeration of simple multiplier units connected together via a network of adders/shifters, as in [13], [15], or 2) via architectural submapping and rearrangement, changing signal flow patterns and gating or activating certain circuit parts when bitwidth changes, as has been explored in [12] and in this paper.

A. Overview of SotA Architectures

UNPU [14] exploits the temporal-based approach, using bit-serial PEs to realize precision-configurable MAC operations for every kind of neural network. It can support fully variable

weight from 1 to 16 bit precision and enjoy an optimized area budget. However, in such serial approaches, throughput and energy efficiency could be sub-optimal [16]. Moreover, it only efficiently supports bitwidth adaptation of one of the two multiplier inputs, weight in this case.

For spatial-based approach, a lookup-table (LUT)-based reconfigurable multiplier has been used in DNPU processor [13]. It supports precision-scalable multiplication by utilizing a LUT to firstly get partial products and then feed them into a configurable shift-add pipeline. However, this MAC engine can only maximize its benefits when the weights are quite stationary so that it doesn't need to frequently update the LUT, which is not the case for FC-NN and LSTM.

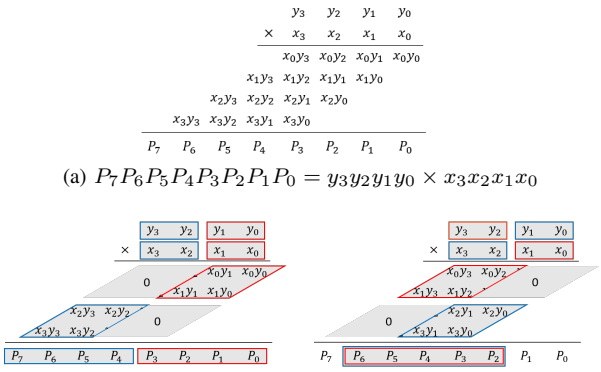
BitFusion [15] recently emerged as a spatial-based bitwidth-flexible accelerator. It consists of an array of bit-level processing elements, each executing 2-bit multiplications and then fusing their results through configurable adder/shifter networks. This architecture can achieve good hardware utilization rate and throughput for different bitwidth settings but at the cost of the overhead coming from relatively complex interconnect network and control logic. Besides, the variability of its input bandwidth complicates the memory fetch pattern, degrading the compatibility when applying it to different system topologies .

All previously discussed approaches configure into a high resolution multiplier from multiple optimized low resolution building blocks. In contrast, the architecture described in [12] modifies a full precision Baugh-Wooley array multiplier to support variable-precision operation by architectural submapping and rearrangement. The MAC unit operates normally in full precision mode with the option of trading the precision for more parallel low-precision processing and energy saving.

B. Key Takeaways

It is important to limit interconnect complexity and control logic required for the precision adjustment. [12] has proven that variable precision can be supported in regular multiplier with very limited overhead. Its operating principle is (for a 4-bit full precision example) depicted in Fig. 1b. As can be seen in 1b, its implementation only requires little overhead in breaking a few accumulation paths in low precision mode. Dynamic power savings are achieved in low-precision modes, due to the increased parallel processing, the reduced switching activity and the shorter critical path.

This paper continues on the basic idea of enabling sub-word parallel operations in a full-precision multiplier. Yet, we explored an alternative internal configuration, named the "Sum-Together" (ST) mode, to extract more parallel processing with reduced precision. We compared this mode to the approach introduced in [12], which we denoted by "Sum Separate" (SS) mode, that we have enhanced for FC-NNs.



(a) $P_7P_6P_5P_4P_3P_2P_1P_0 = y_3y_2y_1y_0 \times x_3x_2x_1x_0$

(b) $P_7P_6P_5P_4 = y_3y_2 \times x_3x_2$, $P_3P_2P_1P_0 = y_1y_0 \times x_1x_0$

(c) $P_6P_5P_4P_3P_2 = y_3y_2 \times x_1x_0 + y_1y_0 \times x_3x_2$

Fig. 1: Decomposing a (a) 4-bit multiplication through (b) Sum Separate (SS) and (c) Sum Together (ST) methodology.

III. Design Concepts

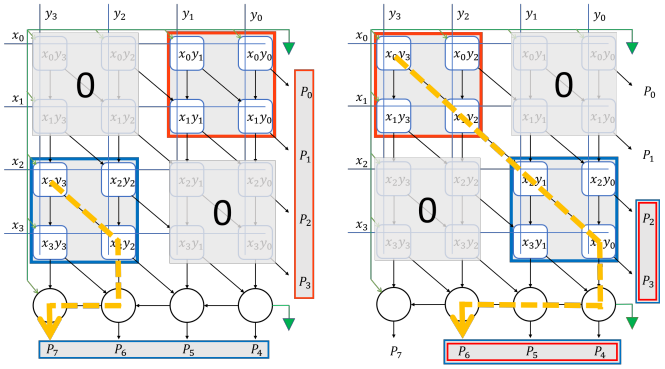
A. Sum Together (ST) mode operating principle

The objective of the ST mode is to decompose a high resolution multiplication into several reduced precision multiplications, whose resulting products are directly accumulated inside of the multiplier. Its principle is (for a simple 4-bit full precision example) depicted in Fig. 1c. Such configuration saves the necessity of using additional adders to sum up these products as it uses the multiplier array cells to implicitly perform the addition. As a result, ST mode saves significant register activity and MAC output bandwidth, as can be seen in the implementation diagram in Fig. 2b, which could possibly bring large system-level savings.

To explore these savings, we will propose the PE-array-level engines to implement the full DNN processing kernel of both approaches, SS and ST respectively. Both designs consider computing the matrix-vector product of weight matrix $W_{M\times N}$ and activation vector $A_{N\times 1}$ that is typical to the DNN processing in inference mode. To be able to make an on-par comparison, both processing engines will be conceived towards identical throughput, making use of 16 parallel multipliers with a 16-bit maximum precision.

B. Sum Separate (SS) Engine

The SS-mode configuration of the multiplier enables the computation of $16/m$ products independently at m-bit precision with m = {2,4,8,16}. We construct a PE as depicted in Fig. 3,



(a) $P_7P_6P_5P_4 = y_3y_2 \times x_3x_2$, $P_3P_2P_1P_0 = y_1y_0 \times x_1x_0$

(b) $P_6P_5P_4P_3P_2 = y_3y_2 \times x_1x_0 + y_1y_0 \times x_3x_2$

Fig. 2: The modified 4-bit multiplier performs two 2-bit multiplications in (a) SS and (b) ST Mode.

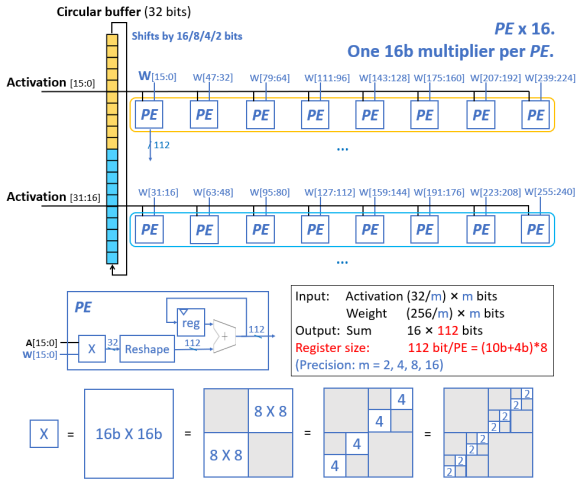


Fig. 3: The Architecture of SS Engine

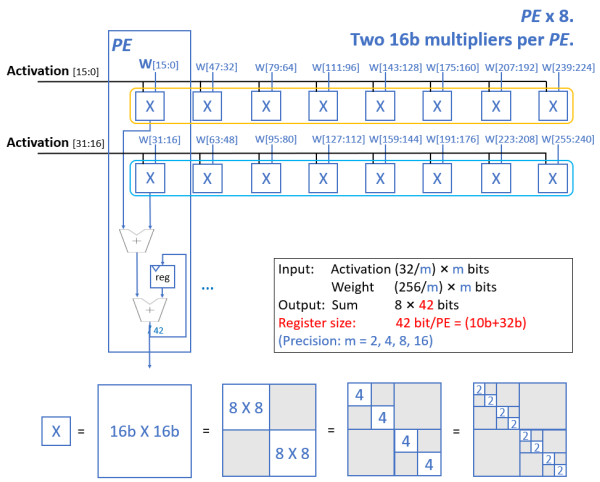


Fig. 4: The Architecture of ST Engine

and deploy it 16-wise parallel with following enhancements for efficient execution of fully-connected layers. The elements of $A_{N\times 1}$ are loaded 32 bits at a time into a circular input buffer, from which the top 16 bits broadcast to the input of 8 PEs and the bottom 16 bits correspondingly. The $W_{M\times N}$ elements are loaded 256 bits every clock cycle and split over the PEs as depicted in Fig. 3. As such, each PE gets a pair of $16/m$ inputs of m bit precision each that correspond to a partial product of the matrix vector multiply. Every clock cycle, the scheme loads $W_{M\times N}$ new weight elements, while rotating the circular input buffer by m shifts, thus computing $16 \times 16/m$ outputs in N cycles. Every PE contains a register to accumulate the partial sums in every precision mode. Yet, to support independent parallel result accumulation in the worst case, precision mode m = 2, the buffer requires enough accumulation headroom bits for each result. For example, enabling 10-bit accumulation margin needs a register length requirement of 112 bits.

C. Sum Together (ST) Engine

The ST-mode configuration of the multiplier also enables the computation of $16/m$ products at m bit precision in parallel, albeit the output is a sum of the set of products. The ST engine, as depicted in Fig. 4, consists of two sets of 8 ST-mode multipliers, each pair operating in tandem, the results of which are added together and finally accumulated in the output register. The elements of $A_{N\times 1}$ are loaded 32 bits at a time and the top 16 bits broadcast to the top set of multipliers and the bottom 16 bits correspondingly. The $W_{M\times N}$ elements are loaded 256 bits at a time and split over the multipliers as depicted in Fig. 4. As such, each PE gets a pair of $16/m$ inputs of m bit precision that correspond to a partial product of the matrix vector multiply similar to the previous scheme. This scheme iterates by loading new elements of both the matrix and vector in each cycle and compute 8 outputs in $N/(2*16/m)$ cycles. The difference however is that the $16/m$ products implicitly add within the array multiplier and thus this scheme vectorizes over the matrix/vector element differently. This prevents the need for a circular buffer, and avoids separate accumulation overhead bits for all low-precision products. Still targeting 10-bit accumulation margin, the output register can now operate with just 42 bits.

D. Additional Hybrid Sub-Mapping Ideas

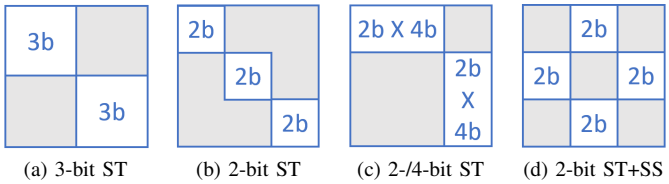
The ST topology can easily be extended towards other multiplication precision. As indicated in the 6-bit multiplier in Fig. 5, one 6-bit multiplication can be decomposed into two 3-bit or three 2-bit multiplications. Moreover, it can also support two 2-bit/4-bit asymmetrical multiplications. Finally, also hybrid approaches of the ST and SS are possible, as in Fig. 5(d), where four 2-bit multiplications are summed together pairwise, enabling benefits of both approaches.

IV. Results and Comparison

A. Comparing ST and SS Engines at Architectural Level

Comparing SS and ST multipliers, in full-precision mode, they work the same. In lower precision modes, they are configured into 2 different diagonal patterns. The basic trade-off is that SS multiplier has higher switching activity (since the bottom-right triangle region of SS-mode multiplier need to pass on several products to the right-bottom edge of the array multiplier, thus cannot be fully switched off), while ST multiplier has longer critical path.

Comparing SS and ST engines, as summarized in Table I, their throughputs are the same and scale geometrically with the precision. Both engines utilize the weight load bandwidth to the fullest. However, they differ in the read rate for activation, with the SS method requiring less access owing to the local circular buffer. The output write rate overall is the same between the two methods, yet at different periodicity. Another main difference,



(a) 3-bit ST (b) 2-bit ST (c) 2-/4-bit ST (d) 2-bit ST+SS

Fig. 5: Four examples of configuring a 6-bit array multiplier

TABLE I: Comparing ST and SS Engines at Architectural Level

Matrix-Vector Multiplication $W_{M\times N} \times A_{N\times 1}$ @ 16/8/4/2-bit precision					
Method	**Precision Bits**	**Total Compute Cycles (rows) × (columns) × (inner loop)**	**Activation read rate (bits / cycles)**	**Weight read rate (bits / cycles)**	**Output write rate (bits / cycles)**
SS	16	M/16 × N/2 × 2 = MN/16	32b / 2	256b / 1	16 ×16b / N
	8	M/32 × N/4 × 4 = MN/32	32b / 4	256b / 1	32 × 8b / N
	4	M/64 × N/8 × 8 = MN/64	32b / 8	256b / 1	64 × 4b / N
	2	M/128 × N/16 × 16 = MN/128	32b / 16	256b / 1	128 × 2b / N
ST	16	M/8 × N/2 × 1 = MN/16	32b / 1	256b / 1	8 × 16b / (N/2)
	8	M/8 × N/4 × 1 = MN/32	32b / 1	256b / 1	8 × 8b / (N/4)
	4	M/8 × N/8 × 1 = MN/64	32b / 1	256b / 1	8 × 4b / (N/8)
	2	M/8 × N/16 × 1 = MN/128	32b / 1	256b / 1	8 × 2b / (N/16)

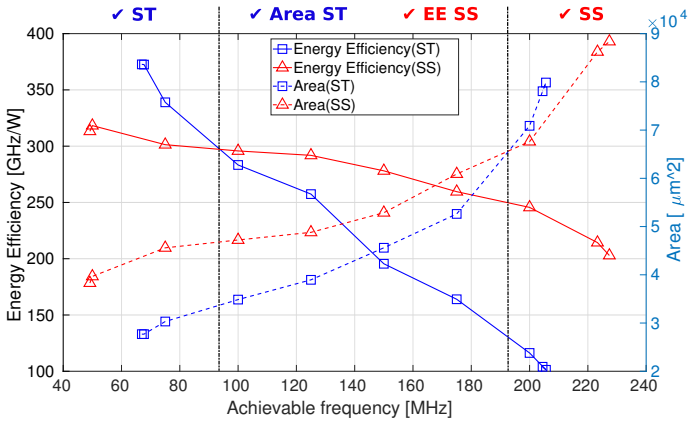


Fig. 6: Energy Efficiency vs. Speed vs. Area

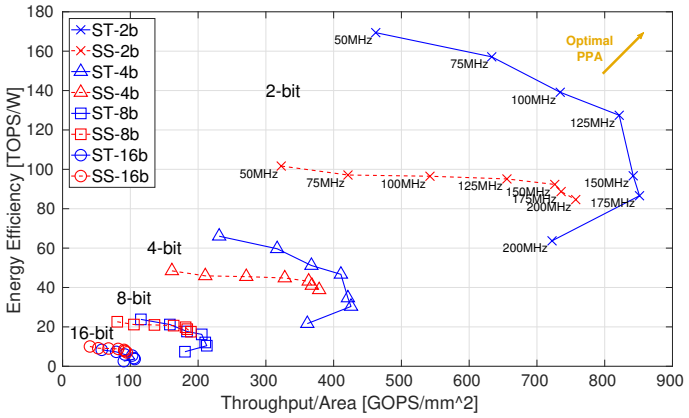


Fig. 7: Energy efficiency vs. Area efficiency

which is not depicted in the table, is the data memory placement. SS requires significantly more complex addressing scheme while ST requires only linear addressing. The simplicity of the addressing scheme for ST may yield significantly simpler memory hierarchy and access logic, potentially yielding energy gains. However, the simulation of this work does not include memory hierarchy and chooses to focus on the PPA of the two proposed MAC engines.

B. Comparing ST and SS Engines from PPA aspect

The two proposed precision-scalable MAC engines were synthesized in 40nm Low Power technology for various operating frequency targets. Fig. 6 captures the average energy efficiency, which is extracted from the simulation of every possible precision mode with equal weight, vs area vs target frequency for each engines, which allows to summarize the general trade-off between the two architectures.

Fig. 7 shows a more interesting results with the energy efficiency vs area efficiency curves of specific precision modes. At lower precision mode, the ST clearly outperforms SS, while both yield similar performance at full precision. This is primarily due to the higher multiplier switching activity and the energy spent on the internal wide registers in SS. The curves also indicate the most optimal PPA trade-off operating points. Another interesting observation is that the scaling of the energy efficiency (TOPS/W) is not by a factor of 2 between precision mode, but higher. This comes from two factors, the increased number of operations in parallel yielding a factor of 2; and the reduced switching activity yielding a factor < 2 of energy saving. These two effects compound into a super linear energy efficiency gain as a function of precision.

C. Comparison with SotA

The differentiating aspect of this work with the SotA, is in the use of a single high precision multiplier in sub-word parallel mode with internal accumulation. While enabling parallelism, this allows to decouple scalability from the macro architecture and avoids the need for complex adder shifter networks and control logic as seen in [13], [14], [15]. A similar ST mode could potentially be deployed in [13], by disabling the shifters in their multipliers to achieve similar savings in this design. The energy savings due to dynamic voltage scaling in low precision mode, first reported in [12], can also be applied to this work to yield further energy savings.

V. Conclusion

In this work, we have proposed two MAC engines built around sub-word parallel mappings of an array multiplier to target DNN inference processing with run-time precision scalability. Performance measures of both design have been extensively compared, as well as contrasted with the SotA approaches. The results yield optimal frequency operating points, trading energy efficiency vs. area efficiency, as well as design decision guidelines based on application level requirements.

Acknowledgments

This work is partially supported by the imec Industrial Affiliation Program (IIAP) on Machine Learning technology.

REFERENCES

[1] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation," *ArXiv e-prints*, Nov. 2017.

[2] R. Hu, P. Dollár, K. He, T. Darrell, and R. Girshick, "Learning to Segment Every Thing," *ArXiv e-prints*, Nov. 2017.

[3] C. Zou, A. Colburn, Q. Shan, and D. Hoiem, "LayoutNet: Reconstructing the 3D Room Layout from a Single RGB Image," *ArXiv e-prints*, Mar. 2018.

[4] Z. Wang, J. L. Roux, and J. R. Hershey, "Multi-channel deep clustering: Discriminative spectral and spatial embeddings for speaker-independent speech separation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 1–5.

[5] C. Xu, W. Rao, X. Xiao, E. S. Chng, and H. Li, "Single channel speech separation with constrained utterance level permutation invariant training using grid lstm," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 6–10.

[6] D. Yogatama, P. Blunsom, C. Dyer, E. Grefenstette, and W. Ling, "Learning to Compose Words into Sentences with Reinforcement Learning," *ArXiv e-prints*, Nov. 2016.

[7] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, Fall 2017.

[8] C. Sakr and N. Shanbhag, "An analytical method to determine minimum per-layer precision of deep neural networks," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 1090–1094.

[9] B. Moons, B. D. Brabandere, L. V. Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2016, pp. 1–8.

[10] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[11] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," *ArXiv e-prints*, Apr. 2016.

[12] 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 *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 246–247.

[13] D. Shin, J. Lee, J. Lee, and H. Yoo, "14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 240–241.

[14] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "Unpu: A 50.6tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 218–220.

[15] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," *ArXiv e-prints*, Dec. 2017.

[16] V. Camus, C. Enz, and M. Verhelst, "Survey of precision-scalable multiply-accumulate units for neural-network processing," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, March 2019.