# DBFS: Dynamic Bitwidth-Frequency Scaling for Efficient Software-defined SIMD

Pengbo Yu*, Flavio Ponzina†, Alexandre Levisse*, Dwaipayan Biswas‡, Giovanni Ansaloni*, David Atienza*, Francky Catthoor‡

*EPFL, Lausanne, Switzerland, {pengbo.yu, alexandre.levisse, giovanni.ansaloni, david.atienza}@epfl.ch

†UCSD, La Jolla, USA, fponzina@ucsd.edu

‡IMEC, Leuven, Belgium, {Dwaipayan.Biswas, Francky.Catthoor}@imec.be

***Abstract*—Machine learning algorithms such as Convolutional Neural Networks (CNNs) are characterized by high robustness towards quantization, supporting small-bitwidth fixed-point arithmetic at inference time with little to no degradation in accuracy. In turn, small-bitwidth arithmetic can avoid using area-and-energy-hungry combinational multipliers, employing instead iterative shift-add operations. Crucially, this approach paves the way for very efficient data-level-parallel computing architectures, which allow fine-grained control of the operand bitwidths at run time to realize heterogeneous quantization schemes. For the first time, we herein analyze a novel scaling opportunity offered by shift-add architectures, which emerges from the relation between the bitwidth of operands and their effective critical path timing at run time. Employing post-layout simulations, we show that significant operating frequency increases can be achieved (by as much as 4.13× in our target architecture) at run time, with respect to the nominal design-time frequency constraint. Critically, by exploiting the ensuing Dynamic Bitwidth-Frequency Scaling (DBFS), *speedups* of up to 73% are achieved in our experiments when executing quantized CNNs, with respect to an alternative solution based on a combinational multiplier-adder that occupies 2.35× more area and requires 51% more energy.**

***Index Terms*—Low power architecture, Edge machine learning, Software-defined SIMD, Dynamic Bitwidth-Frequency Scaling (DBFS).**

## I. Introduction

With the rising number of edge devices and the rapid growth of edge Machine Learning (ML), the computing requirement of ultra-low power platforms has seen a dramatic increase [1].

One practical approach to tackling this challenge is to optimize ML models via quantization, and then use hardware accelerators supporting deeply quantized arithmetic. Indeed, due to their intrinsic robustness, the operands (weights and activations) of ML algorithms such as Convolutional Neural Networks (CNNs) can be converted from floating-point to small-bitwidth fixed-point arithmetic with a negligible impact on accuracy, down to a handful of bits (e.g., 3-bit, 4-bit) [2]–[4]. In this scenario, multiply-accumulates (MACs), the key operation in ML algorithms, can be performed using shifts and additions instead of conventional combinational multipliers, resulting in area-and-energy-efficient implementations [3]–[5]. Moreover, shift-add accelerators can perform multiplications between a single scalar value and a vector of multiplicands in parallel, a form of Single Instruction Multiple Data (SIMD) processing that can be harnessed for key ML kernels such as convolutional and fully connected layers.

A particularly interesting embodiment of the shift-add SIMD paradigm is that of Software-defined SIMD (Soft SIMD), in which dedicated bit positions (named guardbits) are employed to partition a word in several subwords, each storing a binary value [6]–[10]. The benefit of such an approach is two-fold. First, very little hardware overhead is required to support SIMD processing. Second, Soft SIMD resource requirements only marginally scale with the number of supported SIMD modes (i.e., subword bitwidths). This characteristic is particularly beneficial when heterogeneous quantization strategies are employed [10]–[12], in which different bitwidths are supported in different computation phases, e.g., in each layer of a CNN.

Against this backdrop, the key observation driving this paper's contribution is that the critical path timing of Soft SIMD architecture at run time is strongly dependent on the subword bitwidths, with small-bitwidth operations allowing higher operating frequencies than large-bitwidth ones. Hence, a new opportunity emerges, namely of adjusting the operating frequency in dependence on the operand bitwidths at run time. Herein, for the first time, the effectiveness of this strategy, which we name Dynamic Bitwidth-Frequency Scaling (DBFS), is explored.

To this end, we apply the proposed DBFS method on a Soft SIMD hardware [10] that supports multiple operand sizes. The variation in critical path timing across different operand bitwidths is leveraged to set appropriate operating frequencies accordingly. We showcase that DBFS can effectively enhance its execution time efficiency while preserving energy efficiency. Indeed, post-layout (28nm, 0.9V) simulation shows that DBFS can achieve a maximum increase in the operating frequency of ×4.13 (for 3-bit operands) with respect to the nominal value imposed as a design constraint, which considers the entire SIMD word instead of individual subwords. Crucially, this benefit is achieved without negatively increasing operating voltage or modifying hardware, thus not impacting energy efficiency. When employed to execute inference on a collection of heterogeneously quantized CNN benchmarks, DBFS-based Soft SIMD simultaneous leads to a reduction of

This research was partially supported by EC H2020 FVLLMONTI project (GA No. 101016776) and by the ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR, and in part by a joint research grant for ESL-EPFL by IMEC.

up to 51% in energy cost and 73% in execution time compared to a typical multiplier-adder implementation that lacks DBFS support.

The key contributions of this work are as follows :

- We investigate the scaling between operand bitwidth and operating frequency of fixed-point quantization operand, and we propose DBFS as a method to fully leverage the benefits of quantization.
- We demonstrate that DBFS can achieve up to 4.13× higher operating frequency for small-bitwidth operand than the nominal value set at design time, without increasing voltage or energy consumption. It enhances execution time efficiency and throughput of shift-add-based architecture while maintaining high energy efficiency.
- Through post-layout simulations, we showcase that employing DBFS can decrease by up to 51% in energy consumption and 73% in execution time when performing inference on heterogeneously quantized CNNs.

## II. Background and Related Works

### A. Frequency-based Energy Optimization

A straightforward approach to optimize energy costs with operating frequency is Dynamic Voltage Frequency Scaling (DVFS) [13]. DVFS has a programmable clock generator and an adjustable voltage generator, and it dynamically lowers the operating voltage and clock frequency based on specific workloads and hardware performance. Due to the concurrent reduction of both frequency and voltage, although the execution time rises, the overall energy cost is reduced. Thermal throttling employs similar methods to DVFS, like reducing the operating frequency and voltage, but it is passively triggered to ensure the processor temperature stays within acceptable ranges [14] [15]. Our DBFS approach also manipulates the clock frequency, but in conjunction with the operand bitwidths instead of the voltage supply. By doing so, and as opposed to DVFS, DBFS can operate at higher-than-nominal frequencies and does not require a tunable voltage supply.

A further related strategy is Dynamic Duty Cycle Modulation (DDCM) [16], which selectively enables/disables the clock as required by applications. Similarly to DBFS, DDCM does not require voltage adjustments. However, it operates at a higher granularity, adjusting the number of clock cycles instead of the operating clock periods. As with DVFS, DDCM cannot exceed the nominal operating frequency.

Finally, dynamic clock adjustment [17] [18] is a tunable clocking strategy that targets general-purpose processor pipelines. It analyzes the timing of different instructions being executed on a processor pipeline stages, and dynamically sets the operating frequency accordingly. Our proposed DBFS focuses instead on the timing differences resulting from the operand bitwidths of arithmetic operations. Our stance has a much higher leeway for optimization: the authors of [17] report clock frequency boost of up to 38% (28nm, from 494MHz to 680MHz in their design), while DBFS achieves speedups of 313% (28nm, from 200MHz to 826MHz) when reducing operand bitwidths down to 3-bit.

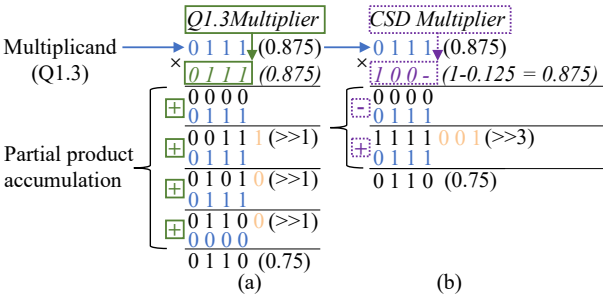


Fig. 1. Example of shift-add-based multiplication. The multiplicand is Q1.3 format, and the multiplier is (a) Q1.3, (b) Q1.3 + canonical signed digit (CSD) encoding. The multiplication result is also in Q1.3 format.

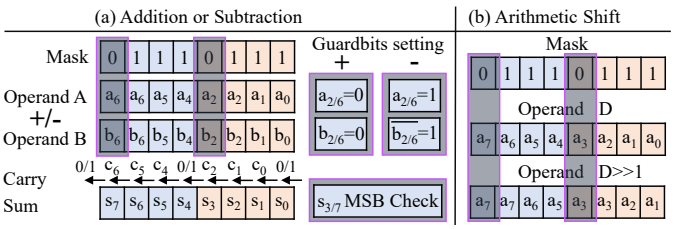


Fig. 2. Soft SIMD operations: (a) addition or subtraction and (b) arithmetic shift. Soft SIMD can adapt to arbitrary SIMD size through control signals (mask) at run time, providing more data-level parallelism. It assigns guardbits as either '0' or '1' for addition or subtraction operations to guarantee correct carry propagation.

### B. Shift-add-based Edge Accelerator

DBFS is experimented on Soft SIMD pipelines, as reported in Section III-A, which perform MACs in fixed-point arithmetic serially using shifts and additions. The shift-add-based multiplication is presented in Figure 1(a). Notice that, when employing 1 bit for the integer part and $X$ bits for the fractional part (commonly indicated as Q1.X notation, and ranges in (-1,1) ), the multiplication result is guaranteed not to overflow. Moreover, the result of an N-bit × M-bit multiplication can be stored in N bits by truncating the less significant bits (LSBs). In more detail, it begins from the LSB of the multiplier, accumulating partial products for each multiplier bit, then shifting the accumulator right by 1-bit iteratively until the multiplication result is achieved. In the naive implementation in Figure 1(a), on average, such a process requires N shift-adds for N-bit multiplier operand. Nonetheless, this number is averagely reduced to N/3 by allowing multiple bit-shift when processing bit fields with trailing zeros (e.g., "100"), and by adopting Canonical Signed Digit (CSD) [19] to encode the multiplier operand [10]. CSD encoding comprises three symbols (positive: '1', zero: '0', and negative: '-') for each bit and maximizes the likelihood of '0' bits. This optimized implementation is shown in Figure 1(b).

### C. Shift-add Topology with Soft SIMD

Soft SIMD supports data-level-parallel signed arithmetic among operands stored in subwords, implementing addition/subtraction and shift, which are then chained to realize more complex ones such as multiplications, dot-products, and convolutions [6]–[10].

The boundary between subwords is dictated by guardbits, determined at run time by a mask. For N-bit fixed-point addition/subtraction, the information representation of the

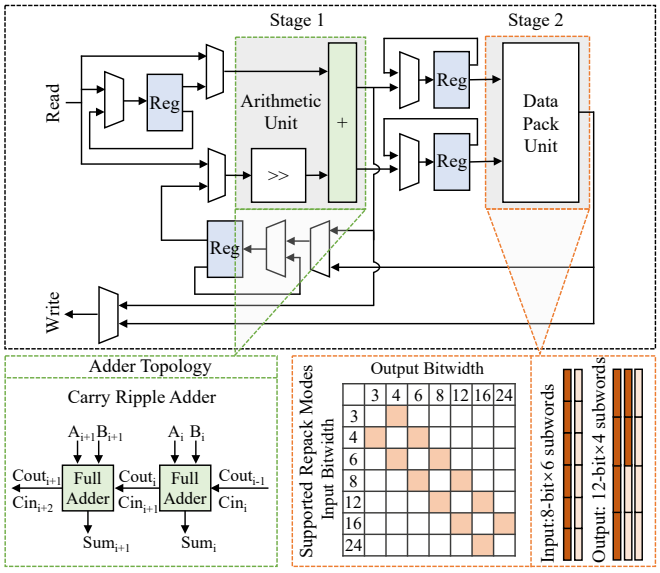


Fig. 3. The computing microarchitecture overview. It consists of an Arithmetic Unit (AU) and a Data Pack Unit (DPU). AU consists of a shifter and an adder, which implements signed Soft SIMD shifts and additions/subtractions, by controlling the guardbits of the operands. DPU converts the operand bitwidth to a larger/equal/smaller size.

operands is confined to N-1 bits to prevent result overflow, allowing the $N_{th}$ bit to be used as a guardbit temporarily during computation. During addition, the guardbit positions of the operands are both set to '0', as shown in Figure 2(a). Conversely, for subtraction, guardbits of operands are set to '1', so that the carry-in of the next subword is equal to '1' to implement 2's complement arithmetic properly (A-B = A + $\bar{B}$ + 1). Shift operations are also conditioned by guardbits, ensuring that sign extensions occur in guardbit positions, while regular shifts are performed for non-guardbit positions, as shown in Figure 2(b). In this way, independence among subwords can be enforced, particularly ensuring that overflows cannot propagate from one subword to neighboring ones.

## III. DBFS on the Arithmetic Microarchitecture

### A. Microarchitecture Overview

As a target for our DBFS methodology, we consider a Soft SIMD pipelines similar to the one in [10], whose block scheme is presented in Figure 3. It comprises two stages, with the first stage being the Arithmetic Unit (AU) and the second stage being the Data Pack Unit (DPU). Four registers are used for data reuse and localization through feedback datapath loops.

The AU consists of a shifter and an adder, and it supports Soft SIMD operations as illustrated in Figure 2. The adder is based on a carry ripple topology. The shifter has a logarithmic topology and supports a maximum of 7-bit right shifts.

The DPU can repack data to smaller, equal or larger subwords by employing multiplexers. Notably, its structure presents a shallow critical path, which is not dependent on the subword bitwidths. Nonetheless, it does not represent a bottleneck for DBFS, even when very small bitwidths are considered. In our target applications, the DPU is employed during the accumulations, where it dynamically increases the

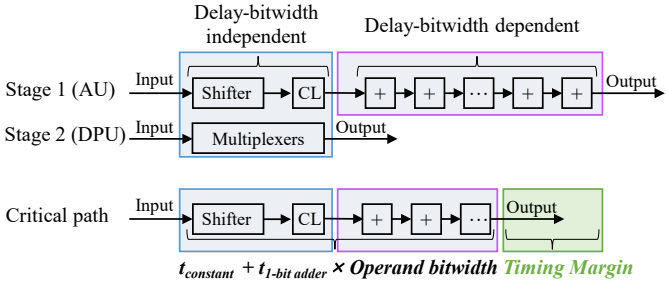


Fig. 4. Delay analysis of the two stages in the arithmetic microarchitecture.

subword bitwidths, ensuring that the accumulation result never overflows. In this work, we consider repacking between 12 Soft SIMD modes as shown in Figure 3.

### B. DBFS Implementation

The adder in the AU pipeline stage is the focus of our DBFS strategy. It employs a carry ripple topology. Such choice minimizes area requirements, while at the same time allowing to place the critical path only along the carry propagation chain. In turn, since the carry chain does not propagate across subword boundaries at run time, a carry ripple implementation takes full advantage of DBFS, ultimately allowing the Soft SIMD pipeline to achieve higher performance than a combinatorial alternative performing multiplications in a single cycle.

DBFS relies on the difference between the nominal ***design-time*** critical path and the actual ***run-time*** critical path, where the former depends on the datapath width ($W_{DP}$) and the latter on the subword width ($W_{SubW}$). The design-time critical path of the AU in Soft SIMD pipeline is expressed as follows:

$$t_{\text{Shifter}} + t_{\text{CL}} + W_{DP} \times t_{1-\text{bit}-\text{adder}} \quad (1)$$

Where $t_{\text{CL}}$ refers to the delay of peripheral combinational logic, such as the multiplexers routing the AU inputs/outputs.

As illustrated in Figure 4, the timing path delays at run time can be classified as operand-bitwidth-dependent and operand-bitwidth-independent. The delay of the shifter and peripheral combinational logic is almost constant with different operand bitwidths, while the delay of the adder rises with the increase of operand bitwidth. Thus, the run-time critical timing path is redefined as :

$$t_{\text{constant}} + W_{SubW} \times t_{1-\text{bit}-\text{adder}} \quad (2)$$

Hence, if the operating clock frequency is determined by design-time consideration alone, a large amount of timing margin is not exploited. DBFS instead exploits run-time information to determine operating frequency and increase performance. Also, given that the hardware remains unchanged, the total energy consumption stays constant despite the increased operating frequency, thereby preserving energy efficiency. Thus, this approach does not involve a conventional energy-delay trade-off.

We show in Section V-B that, as the operand bitwidth decreases, the increase in operating frequency facilitated by DBFS becomes more significant. In turn, very small bitwidth operands (down to 3-bit, 4-bit) are very commonly employed in deeply quantized ML models such as CNNs for edge inference, where they have shown to incur in very small accuracy degradation with respect to floating-point equivalents [12].

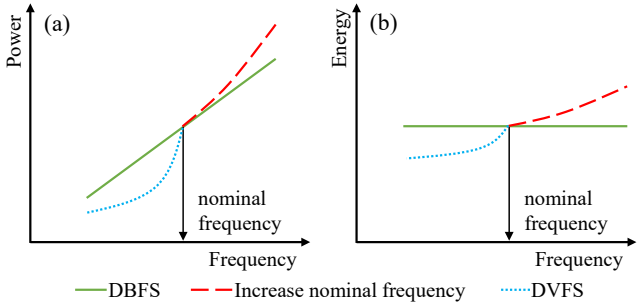


Fig. 5. The trend of (a) power and (b) energy cost versus operating frequency of our proposed DBFS method and other approaches.

Figure 5 qualitatively illustrates the power and energy trends with respect to frequency of our proposed DBFS method and state of the art alternatives, namely DVFS and varying the timing constraint at design time. As the design-time timing constraint rises, the corresponding power increases super-linearly, increasing energy consumption. DBFS instead enables high operating frequency on hardware designed with a lower timing constraint through scaling between bitwidth and frequency. This strategy leads to approximately linear growth in power value, while energy consumption remains essentially unchanged. Moreover, traditional DVFS can only operate at frequencies lower than the designed frequency, by decreasing both the operating frequency and voltage. Note that nothing prevents DBFS and DVFS to be applied concurrently. We reserve the exploration of the joint DBFS/DVFS design space as future work.

## IV. Experimental Setup

### A. Microarchitecture Parameters

As a test vehicle for our experiments, we consider a Soft SIMD pipeline microarchitecture as described in Section III-A, having a datapath width of 48 bits and supporting subwords of [3, 4, 6, 8, 12, 16, 24] bits. The DPU in the pipeline second stage supports conversions between adjacent subword sizes. Synthesis and place and route are performed considering a 28nm CMOS technology library characterized at 0.9V and adopting a 200MHz design-time frequency constraint.

### B. Baselines

As a first baseline, termed "Hard SIMD" in the following, we implemented a state-of-the-art SIMD-based combinational multiplier and adder [20], which can support 8, 16, and 24 bits of multiplication or addition operations in each cycle. We assume this baseline has the same microarchitecture topology as our adopted one and supports data conversion of 8 to 16, 16 to 8, 16 to 24, and 24 to 16 bits. As detailed in Section V-A, while having less flexibility in terms of supported SIMD modes, this implementation requires 2.35× more area, as combinatorial multipliers are area-hungry.

The second baseline only differs from our implementation by waiving the support from DBFS. That is, it has the same microarchitecture as described in Section III-A, but operates at the frequency dictated by its design-time critical path, irrespective of the subword size.

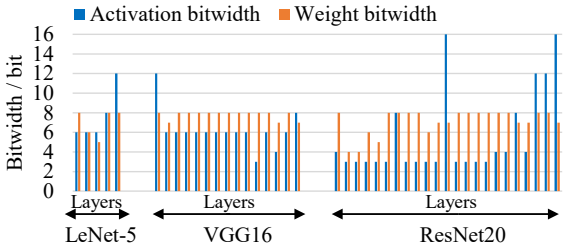


Fig. 6. Heterogeneous quantization of CNN benchmarks.

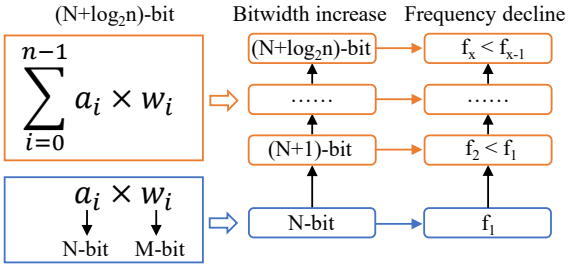


Fig. 7. Mapping example of dot product operations with DBFS.

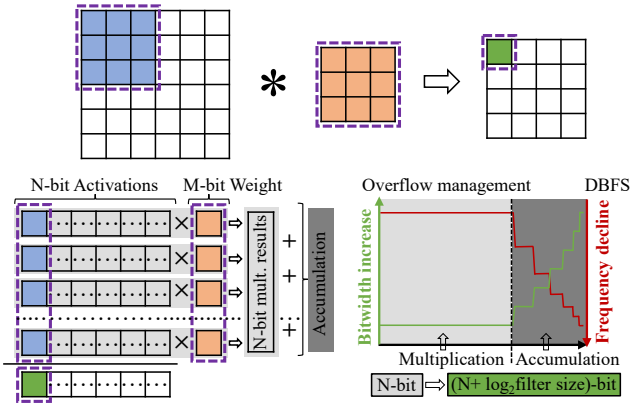


Fig. 8. Mapping strategy of convolution layers with DBFS.

### C. Benchmark CNNs

To assess the benefit of DBFS from an application perspective, we targeted a collection of edge CNN models comprising LeNet-5 [21], VGG16 [22] and ResNet20 [23]. These were heterogeneously quantized, adopting more aggressive quantizations for the layers showing a higher degree of robustness, according to the quantization-aware training methodology presented in [11]. The resulting implementations, described in Figure 6, employ representations of weights and activations ranging from 3 to 16 bits. They exhibit less than 1% accuracy degradation on the employed dataset (CIFAR-10 dataset [24] in the case of LeNet-5, CIFAR-100 dataset [24] for VGG16 and ResNet20), with respect to floating-point equivalents.

### D. Mapping CNN Layers with DBFS

The dominating computational pattern of ML applications like CNNs is the dot-product: $\sum_{i=0}^{n-1} a_i w_i$, in which multiple MACs are executed in sequence, since convolutions and fully connected layers can be expressed in terms of this pattern. Figure 7 exemplifies how the execution of dot products is optimized with DBFS. Assuming that the $a_i$ and $w_i$ operands

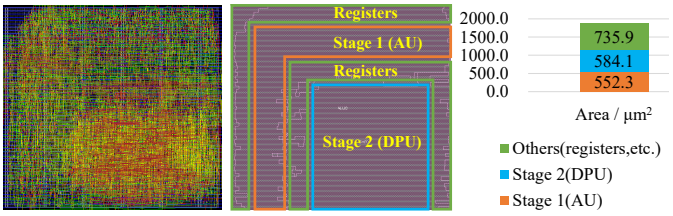


Fig. 9. Layout of adopted microarchitecture

are encoded in N and M bits, respectively, by using the shift-add-based multiplication depicted in Figure 1, the result of $a_i w_i$ is also N-bit. During accumulation, the bitwidth size of the results increases logarithmically with the number of accumulated values $n$, requiring the second stage (DPU) to dynamically repack the operands to a larger size to prevent overflows. During dynamic adjustment of the operand bitwidth, the corresponding operating frequency must be adjusted accordingly. Hence, at run time two phases can be identified: (i) during the multiplication phase, the frequency and bitwidth are constant, (ii) while in the accumulation phase, the bitwidth must increase while the frequency decreases.

The experiments in Section V-D consider the mapping of all convolutional and fully connected layers of each CNN on the DBFS microarchitecture and on the baselines. As for convolutional layers, mapping is performed as in Figure 8. First, the im2col transformation [25] is used to reshape the convolution as a sequence of dot-products. Then these are performed for different input activations in parallel (on different subwords), accumulating along filter weights. A similar strategy is employed for fully connected layers, skipping the im2col transformation as their mathematical expression is already in terms of dot-products. Note that the multiplication phase occupies a larger share of the execution time with respect to the accumulation phase.

## V. Results

### A. Hardware Implementation

The layout of the Soft SIMD datapath, presented in Figure 9, occupies 1872.3 $\mu$m$^2$ in the target technology. On the contrary, the Hard SIMD baseline requires 2.35× times larger area, which takes 4407.9 $\mu$m$^2$ after place and route, even if less subword bitwidths are supported. In more detail, the largest portion of area in our implementation is taken up by registers, along with other peripheral logic gates, comprising 39.3% of the total area. In contrast, in the case of Hard SIMD, the multiplier alone occupies 78.4% of the area.

### B. Frequency Adaptive Performance

Figure 10(a) illustrates the maximum delay of the first stage (AU) and the second stage (DPU) versus different operand sizes in the applied microarchitecture. As analyzed in Section III-B, the maximum delay of the second stage is almost constant, and it can be easily adjusted to always be smaller than the first stage without extra hardware cost. Regarding the maximum delay of the first stage, it grows almost linearly with the operand bitwidth. We observe a tiny difference in the delay of each bit of the carry ripple adder, resulting

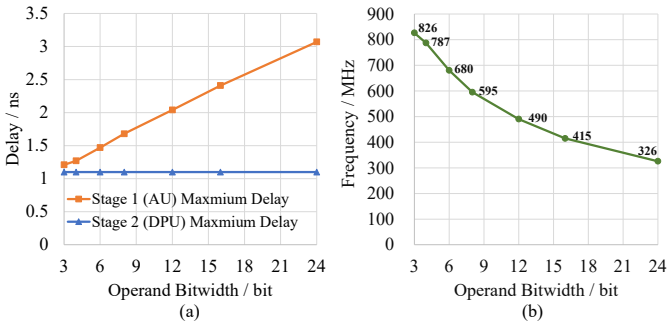


Fig. 10. (a) The maximum delay of stage 1 and stage 2 versus operand bitwidth. (b) The maximum operating frequency versus operand bitwidth.

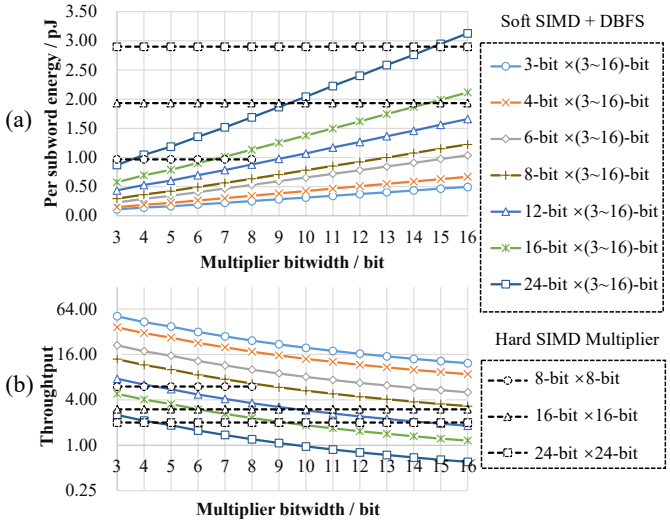


Fig. 11. (a) Average multiplication energy cost per subword, and (b) multiplication throughput. The energy is based on post-layout simulation. The throughput is defined as the number of multiplications completed within 1 design clock period (5ns).

in a slightly higher than expected critical path for the 3-bit configuration. Figure 10(b) illustrates the maximum operating frequency corresponding to different operand bitwidths when utilizing DBFS. For instance, for 3-bit operands, the operating frequency can be raised to 826MHz, ×4.13 times the nominal frequency corresponding to the design synthesis/place-and-route timing constraint. Thus, DBFS fully harnesses the timing margin resulting from the decreased operand size thanks to quantization. Performance gains can be harnessed even for larger bitwidth settings: in the case of 24-bits subwords, a run-time frequency of 326MHz can be achieved for a design-time timing constraint of 200MHz. Additionally, in more advanced wire-dominated technology nodes [26], the timing margin and corresponding DBFS range resulting from differences in operand bitwidths will be more significant, and we reserve this as a topic for future exploration.

### C. Multiplications performance

Before analyzing entire applications, we herein comparatively evaluate the energy and throughput of Soft SIMD + DBFS when performing multiplications, as a) this is the most critical operation in ML algorithms and b) a shift-add-based implementation could be at a disadvantage with respect to a combinatorial solution for throughput. The results are generated from post-layout random multiplication simulations.

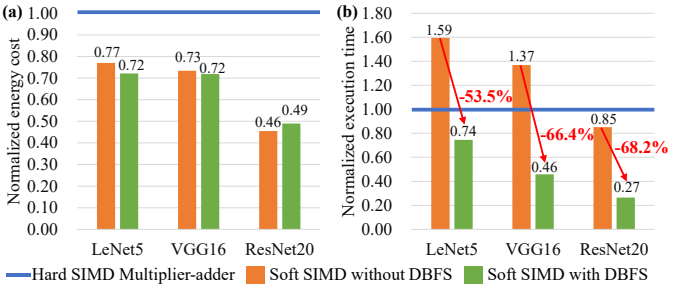


Fig. 12. When running heterogeneously quantized CNNs, (a) the normalized energy cost and (b) the normalized execution time of our adopted design with/without DBFS and the baseline.

As shown in Figure 11(a), an energy analysis clearly favors Soft SIMD with DBFS. For example, our design only require 19.6% of the energy of the Hard SIMD baseline to perform a 4-bit × 4-bit multiplication. The energy gain increases as the multiplicand or/and multiplier size decreases, highlighting the benefit of our approach for deeply quantized ML algorithms.

Results in Figure 11(b) further show that, counterintuitively, the increased flexibility of Soft SIMD combined with DBFS achieves instead comparable, and often better, performance in throughput than 1-cycle Hard SIMD multiplier that can not support DBFS due to much longer and more complex critical path. This is particularly evident for smaller multiplicand bitwidths (3, 4, 6, 8 bits) or/and small-size multipliers (<8 bits). On the other hand, Hard SIMD performs better for the 24-bit multiplicand case, but such large bitwidths are very rarely required in CNNs, as they are employed only at the very last accumulation operations and only in large layers.

### D. CNNs Performance

Figure 12 shows the normalized energy cost and execution time per inference on the considered benchmarks of the Soft SIMD microarchitecture, with and without DBFS, normalized to the Hard SIMD baseline. Regarding energy consumption, Soft SIMD behaves similarly either when utilizing DBFS or not, achieving a reduction ranging from 23% to 54%, mainly because it does not require an energy-hungry combinational multiplier and offers highly flexible data-level parallelism.

On the contrary, in terms of execution time, DBFS is a clear differentiator. Indeed, Soft SIMD without frequency scaling may incur in slowdowns of up to 59% with respect to Hard SIMD. However, if DBFS is employed, the adopted design achieves speedups ranging from 26% to 73%, which tend to increase for deeper CNNs such as ResNet20, as these are more robust towards quantization.

## VI. Conclusion

This work has introduced Dynamic Bitwidth-Frequency Scaling as a novel strategy for optimizing fixed-point and deeply quantized arithmetic, demonstrating its effectiveness on benchmark Convolutional Neural Networks. Our approach exploits the critical path timing margin resulting from quantized operands in shift-add-based architectures, using it to scale operating frequencies versus operand bitwidths above the timing constraint defined at design time. We applied DBFS on a Soft SIMD microarchitecture, which supports highly flexible data-level parallelism. Experimental results highlight that, for small-bitwidth operands, DBFS can operate at up to 4.13 times its design-time frequency constraint. When executing inference on heterogeneously quantized CNNs, DBFS leads to up to 73% reduction in execution time and 51% reduction in energy consumption with respect to an equivalent architecture using a combinatorial multiplier-adder, which also requires 2.35× more area.

## References


[1] R. Singh and S. S. Gill, "Edge AI: a survey," *Internet Things Cyber-Phys. Syst.*, 2023.

[2] M. van Baalen *et al.*, "FP8 versus INT8 for efficient deep learning inference," *arXiv:2303.17951*, 2023.

[3] F. Ponzina *et al.*, "A flexible in-memory computing architecture for heterogeneously quantized CNNs," in *ISVLSI*, 2021.

[4] M. Rios *et al.*, "Bit-Line Computing for CNN Accelerators Co-Design in Edge AI Inference," *IEEE Trans. Emerging Top. Comput.*, 2023.

[5] L.-C. Hsu *et al.*, "Essa: An energy-aware bit-serial streaming deep convolutional neural network accelerator," *J. Syst. Archit.*, 2020.

[6] S. Kraemer *et al.*, "SoftSIMD-Exploiting Subword Parallelism Using Source Code Transformations," in *DATE*, 2007.

[7] F. Catthoor *et al.*, *Ultra-low energy domain-specific instruction-set processors*. Springer, 2010.

[8] G. Psychou *et al.*, "Sub-word handling in data-parallel mapping," in *ARCS*, 2012.

[9] R. Fasthuber *et al.*, "Energy-efficient communication processors," *Springer*, 2013.

[10] P. Yu *et al.*, "An Energy Efficient Soft SIMD Microarchitecture and Its Application on Quantized CNNs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2024.

[11] F. Ponzina *et al.*, "Overflow-free compute memories for edge AI acceleration," *ACM Trans. Embedded Comput. Syst.*, 2023.

[12] R. Goyal *et al.*, "Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms," *arXiv:2102.02147*, 2021.

[13] S. K. Panda *et al.*, "Energy-efficient computation offloading with DVFS using deep reinforcement learning for time-critical IoT applications in edge computing," *IEEE Internet Things J.*, 2022.

[14] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *CASES*, 2007.

[15] M. Bao *et al.*, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *DAC*, 2009.

[16] Bhalachandra *et al.*, "Using dynamic duty cycle modulation to improve energy efficiency in high performance computing," in *IEEE IPDPS Workshop*, 2015.

[17] J. Constantin *et al.*, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *DATE*, 2015.

[18] ——, "DynOR: A 32-bit microprocessor in 28 nm FD-SOI with cycle-by-cycle dynamic clock adjustment," in *ESSCIRC*, 2016.

[19] A. Avizienis, "Signed-digit numbe representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, 1961.

[20] Arm, "NEON™ Version: 1.0 Programmer's Guide," 2013.

[21] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[23] K. He *et al.*, "Deep residual learning for image recognition," in *IEEE/CVF CVPR*, 2016.

[24] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.

[25] K. Chellapilla *et al.*, "High performance convolutional neural networks for document processing," in *IWFHR*, 2006.

[26] V. Huang *et al.*, "From interconnect materials and processes to chip level performance: Modeling and design for conventional and exploratory concepts," in *IEEE IEDM*, 2020.