An Energy Efficient Soft SIMD Microarchitecture and Its Application on Quantized CNNs

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

***Abstract*—The ever-increasing computational complexity and energy consumption of today's applications, such as Machine Learning (ML) algorithms, not only strain the capabilities of the underlying hardware but also significantly restrict their wide deployment at the edge. Addressing these challenges, novel architecture solutions are required by leveraging opportunities exposed by algorithms, e.g., robustness to small-bitwidth operand quantization and high intrinsic data-level parallelism. However, traditional Hardware Single Instruction Multiple Data (Hard SIMD) architectures only support a small set of operand bitwidths, limiting performance improvement. To fill the gap, this manuscript introduces a novel pipelined processor microarchitecture for arithmetic computing based on the Software-defined SIMD (Soft SIMD) paradigm that can define arbitrary SIMD modes through control instructions at run-time. This microarchitecture is optimized for parallel fine-grained fixed-point arithmetic, such as shift/add. It can also efficiently execute sequential shift-add-based multiplication over SIMD subwords, thanks to zero-skipping and Canonical Signed Digit (CSD) coding. A lightweight repacking unit allows changing subword bitwidth dynamically. These features are implemented within a tight energy and area budget. An energy consumption model is established through post-synthesis for performance assessment. We select heterogeneously quantized Convolutional Neural Networks (CNNs) from the ML domain as the benchmark and map it onto our microarchitecture. Experimental results showcase that our approach dramatically outperforms traditional Hard SIMD Multiplier-Adder regarding area and energy requirements. In particular, our microarchitecture occupies up to 59.9% less area than a Hard SIMD that supports fewer SIMD bitwidths, while consuming up to 50.1% less energy on average to execute heterogeneously quantized CNNs.**

***Index Terms*—Energy Efficient Computing, Software-defined Single Instruction Multiple Data, Data-level Parallelism, Canonic Signed Digit Coding, Heterogeneously Quantized Convolutional Neural Networks.**

I. INTRODUCTION

Single Instruction Multiple Data (SIMD) is a well-known approach to enhance hardware computing efficiency through data-level parallelism [1] [2] when executing a ubiquitous computational pattern in many domains ranging from scientific computing [3] to machine learning (ML) [4] [5]. SIMD implementations employ wide registers and processing units that support operations with subwords separated by multiplexers on the datapath. However, typical Hardware SIMD (Hard SIMD) architectures can only support a small set of subword sizes, such as 8-, 16-, 32-, 64-bit [6], because (i) the subword separation relies on multiplexers and can not be extended to more SIMD modes once the hardware implementation is fixed, and (ii) adding many SIMD modes (e.g., 3-, 4-, 6-bit, etc.) is costly in terms of hardware (e.g., area, speed).

These downsides are significantly critical to further improving computation performance, especially for algorithms that have (i) intrinsic redundancy to support aggressive quantization, (ii) a high degree of data-level parallelism, and (iii) varying accuracy requirements in different computation phases. For these cases, supporting operations with flexible bitwidths can fully leverage the advantages of quantization and parallelism to promote computation performance.

For example, Convolutional Neural Networks (CNNs) exhibit a high degree of robustness against operand approximation, allowing them to maintain sufficient accuracy when quantized with fixed-point data representation. Thus, quantized CNNs have been widely adopted in edge inference, which is a scenario with constrained resources and energy budget [7] [8], to significantly increase energy efficiency and reduce resource requirements, such as 8-/16-bit quantization in [9]–[15]. CNNs can further be heterogeneously quantized to arbitrarily bitwidths, such as 3-, 4-, 6-bit, etc. [16], and result in more model compression and data-level parallelism improvement. However, this approach has not yet been explored since most hardware resources do not allow for such small-bitwidth operations.

To overcome the limited bitwidths support of the Hard SIMD approach, we therefore present a novel pipelined processor microarchitecture for arithmetic computing based on the Software-defined SIMD (Soft SIMD) paradigm. Soft SIMD, detailed in Section II-A, divides the boundaries of the subword at the software level through control instructions, enabling support for operations at arbitrary bitwidths with little added cost, opening the way for the highly efficient use of data-level parallelism down to very small bitwidths.

The proposed microarchitecture features a fine-grained configurability of SIMD bitwidths that can be defined at run-time. It can adapt to any desired computation precision in targeted domains, such as different layers in quantized CNNs. It also allows seamless transitions among different SIMD formats using a dedicated Data Pack Unit (DPU). Furthermore, this microarchitecture can efficiently perform parallel shift/add operations, the basis of arithmetic computing, as well as shift-

Pengbo Yu, Flavio Ponzina, Alexandre Levisse, Giovanni Ansaloni, and David Atienza, are with the Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Email: {pengbo.yu, flavio.ponzina, alexandre.levisse, giovanni.ansaloni, david.atienza}@epfl.ch.

Mohit Gupta, Dwaipayan Biswas, and Francky Catthoor, are with Interuniversity Microelectronics Centre (IMEC), Leuven, Belgium. Email: mohitgupta.imec@gmail.com and {Dwaipayan.Biswas, Francky.Catthoor}@imec.be.

add-based parallel multiplication that uses Canonic Signed Digit (CSD) coding for the multiplier to reduce iteration cycles. We evaluate this microarchitecture with an example benchmark in the ML domain, the quantized CNNs, which have been quantized heterogeneously per layer under a maximum accuracy degradation constraint of 1%. The result shows that our Soft SIMD microarchitecture significantly outperforms the Hard SIMD alternative on both energy and area.

In summary, our contributions are as follows:

- We introduce a novel pipelined processor microarchitecture for arithmetic computing based on Soft SIMD. It supports a fine-grained configurability of SIMD bitwidths at run-time through instruction control and has very high and flexible data-level parallelism with little added cost.
- We apply CSD coding and shift-add iteration to efficiently perform parallel 2's complement multiplications under the Soft SIMD paradigm. We also explore the multiplication performance and resource requirement versus the maximum bit shift range when using CSD multiplier.
- We present a Data Pack Unit to repack the data size by bridging different SIMD formats, and show its design trade-off between flexibility and hardware cost.
- We choose heterogeneously quantized CNNs as an example benchmark set from the ML domain to evaluate the area, energy, and performance of our microarchitecture through a complete post-synthesis exploration. We show that our design requires up to 59.9% less area and 50.1% (on average) less energy than a Hard SIMD alternative.
- This work also fills the gap in supporting heterogeneously quantized CNNs from both software and hardware levels.

The paper proceeds as follows. In Section II, we introduce the fundamental concepts that provide the basis for our design choice. Then, the proposed microarchitecture is detailed in Section III. Next, the experimental setup is given in Section IV. Finally, the analysis of the results follows in Section V, and the key conclusions are summarized in Section VI.

II. BACKGROUND

A. Soft SIMD

The implementation of typical Hard SIMD entails using wide registers hosting a fixed number of subwords with corresponding sizes, and processing units performing vector operations among subwords, such as the Arithmetic Logic Unit. Hard SIMD is constrained in supporting many SIMD bitwidths since it relies on the multiplexers on the datapath to separate the adjacent subwords and control the carry propagation in arithmetic operations.

In contrast, Soft SIMD supports basic operations (e.g., addition, shift) with variable bitwidths and composes them to derive more complex operations, such as multiplication. In this way, Soft SIMD allows the partitioning of data registers into subword bit fields of arbitrary size. It can achieve flexible SIMD configurations by the control instructions at run-time on the software level (hence the name) with a small hardware cost. Thus, it is particularly appealing when a strongly heterogeneous fine-grained control of bitwidths is beneficial, the employed bitwidths are small (<16-bit), and the desired data representation precision changes in different computation phases. These features are common in many domains, such as heterogeneously quantized CNNs in the ML domain.

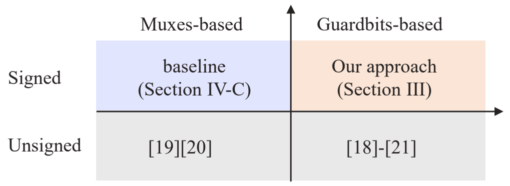


Fig. 1. Topology of Soft SIMD design. Our approach supports signed arithmetic, such as required computations for convolutional and fully connected layers of CNNs, while employing a low-overhead guardbits-based solution.

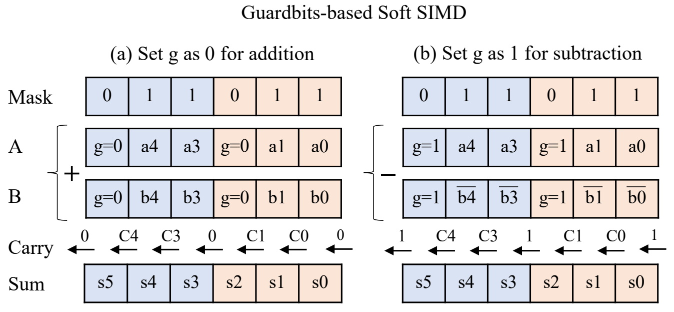

Guardbits-based Soft SIMD
(a) Set g as 0 for addition
(b) Set g as 1 for subtraction
Mask
A
B
Carry
Sum
0 1 1 0 1 1
g=0 a4 a3 g=0 a1 a0
g=0 b4 b3 g=0 b1 b0
+
0 C4 C3 0 C1 C0 0
s5 s4 s3 s2 s1 s0
g=1 a4 a3 g=1 a1 a0
g=1 $\overline{b4}$ $\overline{b3}$ g=1 $\overline{b1}$ $\overline{b0}$
−
1 C4 C3 1 C1 C0 1

Fig. 2. Soft SIMD can adapt the subword bitwidth to the data size requirements by instruction control at the software level, therefore maximizing data-level parallelism and word space utilization. Here is an example of arithmetic operations based on Soft SIMD. In the 3-bit subwords, the guardbits are (a) set as '0' in addition, (b) set as '1' in subtraction, to avoid overflows among subwords and guarantee correct carry propagation.

As opposed to our novel design, previous Soft SIMD implementations [17]–[21] only support unsigned arithmetic, limiting their applicability. We categorize them as unsigned Soft SIMD in Figure 1. Moreover, they neither explore the strategy and hardware-cost vs. flexibility trade-off of conversion between different SIMD bitwidths, nor assess their effectiveness on ML workloads. In contrast, our proposed signed Soft SIMD approach supports 2's complement operations (addition/subtraction/shift/multiplication/etc.) that are fundamental for running CNNs.

There are two implementation methods for signed or unsigned Soft SIMD. A straightforward solution is to insert multiplexers for all bit positions on the datapath (e.g., adder carry propagation chain) to support arbitrary SIMD bitwidth [19] [20]. However, by doing so, a large number of multiplexers are placed on the critical path, resulting in lower performance. Another striking alternative is to use guardbits to separate subwords on the datapath [18]–[21]. As indicated in Figure 1, we herein adopt the second approach for our proposed microarchitecture in Section III while using the first one to build an alternative as a baseline in Section IV-C.

In the rest of the paper, we only consider signed Soft SIMD implementations and their application to CNNs. For brevity, we name the baseline **Muxes-based Soft SIMD** and our proposed method **Guardbits-based Soft SIMD**, respectively.

As the example of the Guardbits-based Soft SIMD arithmetic operations shown in Figure 2, operands A and B have guardbits marked and identified by the mask vector ('0' for guardbits and '1' for non-guardbits), which is decoded from control instructions and defines the SIMD mode on the software level at run-time. During addition or subtraction, the guardbits are set to '0' or '1' by the logic operation to

TABLE I
CHARACTERISTICS OF OUR PROPOSED DESIGN AND THE RELATED STATE OF THE ART

	[9], [10]	[11]	[12]	[13], [14]	Our design
Subword Separation	multiplexers	multiplexers	multiplexers	guardbits	guardbits
SIMD Mode	8,16 bits	16 bits	8 bits	8 bits	arbitrary bits, e.g., 3,4,6,8,12,16,24 bits
Multiplication Method	bit-serial, multiple shifts	bit-serial	bit-serial	bit-parallel	bit-serial, CSD coding, multiple shifts
Multiplicand Operand Size	8,16 bits	16 bits	8 bits	8 bits	arbitrary bits, e.g., 3,4,6,8,12,16,24 bits
Multiplier Operand Size	1-16 bits	1-16 bits	1-16 bits	8 bits	1-16 bits
Average Cycles for N-bit multiplier	N/2	N	N	1 + extra cycles	N/3
Accumulation Overflow Management	no	no	no	yes	yes

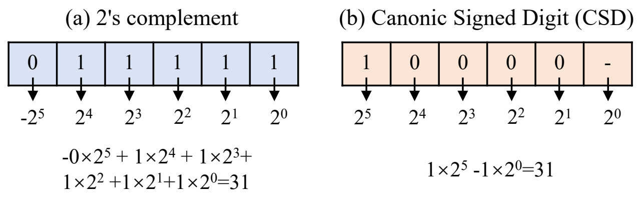

(a) 2's complement
0
1
1
1
1
1
-2^5 2^4 2^3 2^2 2^1 2^0
$-0\times2^5 + 1\times2^4 + 1\times2^3+$ $1\times2^2 + 1\times2^1+1\times2^0=31$
(b) Canonic Signed Digit (CSD)
1
0
0
0
0
-
2^5 2^4 2^3 2^2 2^1 2^0
$1\times2^5 - 1\times2^0=31$

Fig. 3. Reduce the number of non-zero bits from (a) 2's complement to (b) Canonic Signed Digit (CSD).

ensure the correct propagation of the carry between adjacent subwords. Similarly, shift and multiplication operations can be performed on the basis of guardbits, as detailed in Section III-B. Contrary to Muxes-based Soft SIMD, the timing overhead (hence the hardware cost) of Guardbits-based Soft SIMD is minimal, since the extra logic operations introduced are not on the critical path.

B. Canonical Signed Digit Coding

CSD representation [22] is a binary encoding method to reduce the number of non-zero bits for 2's complement. It employs three symbols for each digit: '1', '0', and '-', where '-' indicates that the bit value is -1. Figure 3 gives an example of how the CSD number is generated. The 2's complement 31 ("011111") in CSD notation equals ("10000-", $32 - 1 = 31$), reducing non-zero bits from five to two by eliminating two or more consecutive '1' iteratively from right to left. In CSD formats, $\frac{2}{3}$ of the digits are zero on average [23].

As detailed in Section III-B, our proposed microarchitecture performs multiplications as a sequence of arithmetic shifts and additions. Since additions are only executed when the multiplier bit is non-zero, multiple multiplier bits can be processed in one clock cycle for bit patterns with trailing zeros such as "10", "100", "1000", etc. CSD coding can significantly increase the frequency of such bit patterns for the multiplier operands. Therefore, we apply it to the Soft SIMD paradigm, speeding up shift-add-based 2's complement multiplication.

The CSD encodings are generated by the compiler when the multipliers are known constants or by an online decoder if they are unknown variables. Strategies for online methods of computing CSD multiplier operands have been proposed in [24]–[29]. They are compatible with our design with a negligible hardware cost. For example, a 16-bit CSD decoder only requires $431\mu m^2$ under 130nm technology in [29], which is 21% of the 48-bit Soft SIMD area under 28nm technology in this work. For advanced technology nodes (e.g., 28nm) and a larger datapath (e.g., 192-bit), the area overhead is less than 5%. This work focuses on the constant multipliers and considers CSD multiplier operands as inputs to our proposed microarchitecture.

C. Quantization of CNN models

Quantization is a class of optimization methods highly investigated in the literature and largely employed in Edge Machine Learning [30] [31] [32]. In general terms, quantization restricts the set of representable values of a baseline data representation (e.g., 32-bit floating-point) to more compact integer formats. In doing so, it inevitably introduces an approximation of the original representation. However, the intrinsic redundancy and robustness of AI models like CNNs allow quantization methods to be practical tools to reduce computing complexity and required resources, with minimal impact on the precision of a floating-point baseline [15], [33].

The most common approach to quantizing CNN models is that of uniform quantization, where the same quantization scheme is applied to weights and activations of all convolutional and fully-connected layers [32]. Typical uniform quantization levels employ 8- or 16-bit activations and 8-bit weights, without degrading the accuracy of the model [9] [15] [34]. A more recent quantization strategy is that of heterogeneous quantization, which does not restrict the whole application to use the same quantization scheme [16] [35]. In particular, this quantization strategy is generally applied to CNN models per layer, with the most robust layers represented with more compact formats [9]. This approach suits hardware-software co-design strategies well [10] [36] since it can (i) enable more fine-grained optimizations of the CNN models, leveraging the robustness of each layer to maximize the achieved compression, and (ii) adapt to the underlying resources by allowing, in each layer, quantization schemes that can be effectively exploited in hardware to get maximum efficiency gains.

Thus, heterogeneously quantized CNNs are employed in this work as an example benchmark to evaluate the performance of the proposed microarchitecture, detailed in Section IV-A.

D. Motivation and related works

Taking the accelerator for quantized CNNs as an example, we compare the differences between our proposed Soft SIMD paradigm and related state of the art, as summarized in Table I. Similar to our design, bit-serial (shift-add-based) multiplication is also used in the works of [9]–[12], and they support a broad range of multiplier operand size as well. However, the multiplicand operand size can only be either 8- or 16-bit, while our design provides super flexible multiplicand bitwidths. Hence, these works only release the quantization space of weights (multipliers) but can not leverage that of activations (multiplicands) due to the limited operand size, restricting further acceleration for quantized CNNs. Moreover, the average multiplication cycles for N-bit multiplier is N

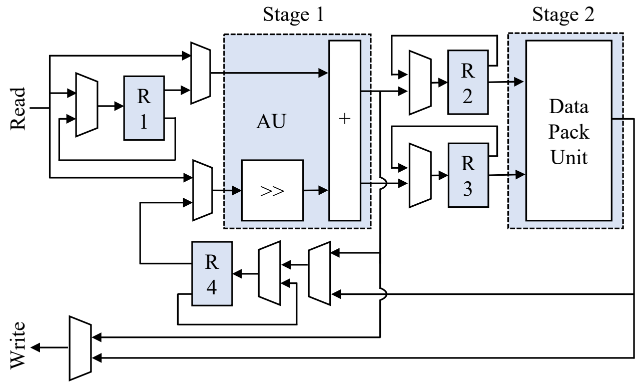


Fig. 4. Block scheme of the proposed Soft SIMD microarchitecture. The first stage is an Arithmetic Unit (AU), and the second stage is a Data Pack Unit (DPU). R1 to R4 are registers.

in [11] [12], N/2 in [9] [10], while just N/3 in our design thanks to CSD coding, as detailed in Figure 14. Finally, our work explores how to prevent overflow during accumulation, which has not been discussed in these works.

The work in [13] [14] also presents a method to realize SIMD operations that map operands in non-conflicting bit ranges and use guardbits to separate subwords. Nonetheless, their approach requires extra cycles and resources for operand packing and output correction (e.g., two shifts, one addition, one multiplication and one output correction for an 8-bit*8-bit multiplication with two subwords), making it difficult to scale beyond a few typical supported SIMD formats.

III. Arithmetic Microarchitecture

This section details the implementation of the proposed pipelined processor microarchitecture for arithmetic computing based on Guardbits-based Soft SIMD. First, Section III-A shows the overview of the block scheme. Then, Section III-B introduces how to build the Arithmetic Unit (AU, the first stage of the pipeline) and perform arithmetic operations such as addition, shift, and multiplication. Then, Section III-C presents the data conversion strategy among different SIMD bitwidths and the trade-off between flexibility and cost for the Data Pack Unit (DPU, the second stage of the pipeline) to perform such conversions. Finally, Section III-D shows a mapping example with the proposed pipeline microarchitecture.

A. Microarchitecture Overview

A block scheme of the proposed microarchitecture is illustrated in Figure 4. It consists of two pipeline stages, AU and DPU, respectively. Four registers are employed at their boundaries. The register R1 stores the data fetched from the memory when such data needs to be used multiple times. The R2 and R3 are pipeline registers between the two stages. The register R4 keeps the output from the first or second stage as intermediate values for data reuse in the first stage. The datapath can be interfaced with the typical memory hierarchy or with Very Wide Registers (VWR) [37] [38] for high computational throughput.

The first stage (AU) is dedicated to parallel iterative shift-add operations, sequentially performing multiplications or accumulations. It can flexibly support different SIMD bitwidths by being configured on the software level at run-time. According to different functional requirements, the input of the first stage can be read from registers R1 and R4, or memory. The output of the first stage can be written (i) to register R4 as the input for the first stage in the next cycle, (ii) to register R2 or R3 as the input for the second stage, or (iii) back to memory. Details of the first stages are introduced in Section III-B.

The second stage (DPU) is instead devoted to repacking data, namely, translating values between bitwidths across computation phases to adopt different SIMD modes. In the second stage, the input is read from register R2 or/and R3, and the output can be written (i) to register R4 as the next cycle's input in the first stage or (ii) back to memory. The second stage can be bypassed if no data conversion is required. Details of the second stage are described in Section III-C.

B. Arithmetic Unit

This section details the first pipeline stage (AU) by (i) Soft SIMD supported shift and addition/subtraction, (b) shift-add-based multiplication, and (iii) overflow management and word space utilization using guardbits.

1) Soft SIMD Supported Shift and Addition: As illustrated in Figure 5(a), the AU consists of a shifter and an adder in series. Figure 5(b) shows that the shifter is a logarithmic topology circuit consisting of multiple layers. The number of layers dictates the maximum bit-shift that can be performed in a single operation. For example, a two-layer logarithmic shifter would allow a maximum bit-shift of 3 (as a combination of shift 1-bit and shift 2-bit, i.e., 1+2), while a three-layer design would allow 7 (1+2+4).

In order to show the details of the shifter and adder more clearly, we use the 1-bit schematic as an example, as presented in Figure 5(c,d). In the two schematics, signal 'A_i' and 'B_i' are two input operands, signal 'V_i' is the mask vector to define SIMD mode and identify guardbits ('0' represents guardbits while '1' non-guardbits), signal '$Ctrl_i$' is the control of the shifter, and signal 'M_i' is the combinational logic output of 'A_i', 'B_i', and 'V_i'. **The following shift and add operations use one guardbit in each subword,** similar to the example in Figure 2.

As Figure 5(c) shows, the 1-bit shifter consists of a multiplexer. The shifter control signal, '$Ctrl_i$', executes an AND operation with signal 'V_i' to generate the selection signal for the multiplexer, which identifies the guardbits when shifting. Thus, if 'V_i=1', the shifter performs regular right shift (selection signal is '1', '$Shift_i$' $<=$ 'A_{i+1}') for non-guardbits, while the sign bit extension (selection signal is '0', '$Shift_i$' $<=$ 'A_i') for guardbits to prevent being affected by adjacent subwords.

The adder, depicted in Figure 5(d), consists of a standard adder cell and peripheral logic gates. When performing addition, both the operands 'A_i' and 'B_i' perform an AND operation with 'V_i' to ensure that the carry-out signal of each guardbit, which is the carry-in for the least significant bit (LSB) of the subsequent subword, is '0' (0 + 0). When performing subtraction, they perform an OR operation with the inverse of 'V_i' to ensure that the carry-out signal of each guardbit is instead '1' (1 + 1), as required by signed arithmetic in 2's complement. The final XOR operation checks and guarantees the correctness of the output. The logic gates

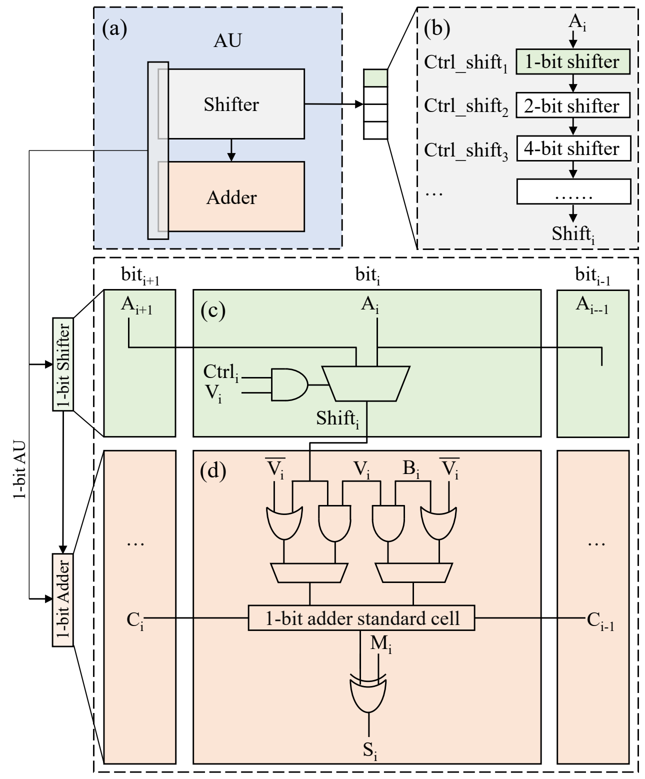


Fig. 5. Block schemes of the first pipeline stage. (a) The AU consists of a shifter and an adder. (b) The shifter has a logarithmic topology, consisting of multiple layers, and can support multiple shifts. (c) 1-bit shifter: it implements right shift for non-guardbits and sign extension for guardbits. (d) 1-bit adder: it consists of a standard adder cell and peripheral logic gates. The adder supports Soft SIMD addition/subtraction by configuring guardbits as ‘0’/‘1’, respectively.

around the standard adder cell are not in the carry chain, hence do not affect the critical path much. Therefore, the required hardware consumption is less than in the case of Muxes-based Soft SIMD, especially when timing path constraints are stringent, such as for higher frequencies, larger-bitwidth operands, or larger shifter numbers, as reported in Section V.

2) Shift-add-based Multiplication: Based on the shift and addition circuits introduced above, the AU supports the following operations, where A and B are two input operands:

(i) (A>>) $+/-$ B $\rightarrow$ addition or subtraction between A (possibly right-shifted) and B.

(ii) (-A>>) $+/-$ B $\rightarrow$ addition or subtraction between -A (possibly right-shifted) and B.

B can be set to 0, resulting in:

(iii) (A>>) $+/-$ 0 and (iv) (-A>>) $+/-$ 0.

Thus, the AU allows the execution of multiplications by decomposing them in iterative shifts-adds. In this work, fixed-point $Q1.X$ 2’s complement representation is selected as the data format, i.e., with one leading bit for the integer parts and the rest devoted to the fractional part. The multiplier is encoded to CSD notation to reduce the number of non-zero bits and required iteration cycles. Several multiplicands with the same multiplier are organized in subwords with $Q1.X$ representation, separated by one guardbit, enabling data-level parallelism.

The employed algorithm is shown in Figure 6, presenting an example of a $Q1.7$ multiplier and two 8-bit multiplicands stored in subwords, themselves represented as $Q1.6$ values

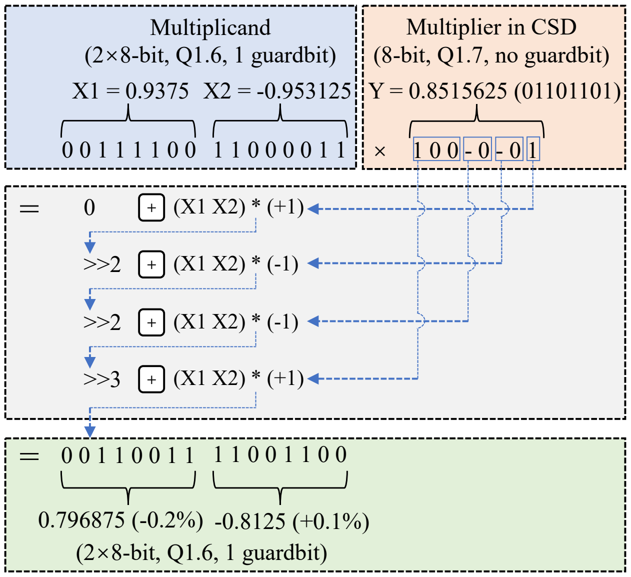


Fig. 6. Example of iterative shift-add-based multiplication. Multiplicand operands are in two subwords with Q1.X 2’s complement format (X=6 in the example, with 1 guardbit), and a multiplier in Q1.Y CSD format (Y=7 in the example). At each step, partial results are computed by right shift and addition/subtraction. Multiple bit-fields of the multipliers containing trailing zeros can be processed in one cycle. The output is also in Q1.X (X=6) 2’s complement format (with 1 guardbit). A total of three cycles are needed since the shift-add corresponding to the 1st and 2nd non-zero bits can be merged in one cycle.

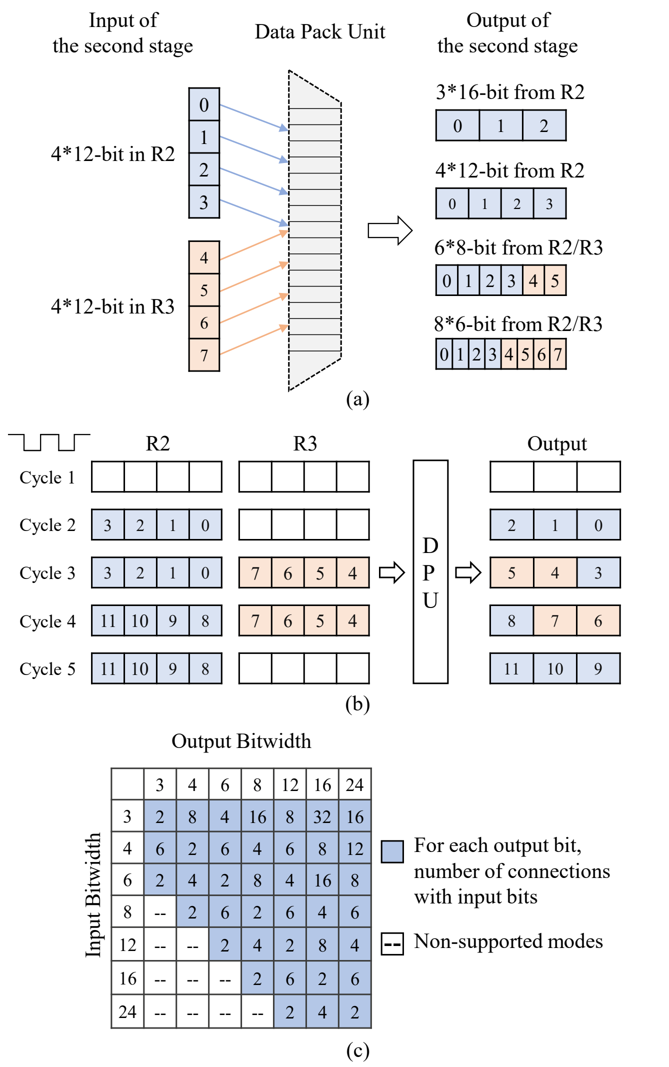
(with 1 guardbit). In each cycle, from right to left, the non-zero bits of the multiplier are processed, and right shift and addition/subtraction are performed, as dictated by bit values, until the multiplication result is computed. Due to CSD coding, multiplier bit patterns containing trailing zeros, such as “-0” and “100” in this example, can be processed in one cycle. Thus, the amount of cycles for this 8-bit multiplier example decreases from eight to three.

It can be seen that the bitwidth of the multiplication results and those of the multiplicands are the same, $Q1.6$ values (with 1 guardbit); hence, truncation errors occur. However, the errors are negligible even for very constrained bitwidths, for example, approximately 0.2% in the example in Figure 6. Wider bitwidths would result in even lower truncation errors, while narrower ones would increase parallelism and performance. We showcase in Section IV-A an accuracy-driven methodology that navigates this trade-off.

3) Overflow Management and Word Space Utilization:

For the shift, addition, and multiplication operations introduced above, guardbits are used to (i) separate subwords, (ii) ensure correct shift operations in each subword, and (iii) guarantee the correct carry generation and propagation in addition or subtraction operations between any two adjacent subwords and avoid possible overflows. It can be noted that guardbits are only required when data traverses the first stage (AU) of this pipeline microarchitecture.

If the first stage output needs to be reused as an input in the next cycle through datapath loops, the most significant bit (MSB) of each subword is used as a guardbit, which does not encode numerical values, as shown in Figure 6. Conversely, if the first stage output is used as input for the second stage or written back to memory, no guardbits are required, and the MSB encodes numerical values again.



	3	4	6	8	12	16	24
3	2	8	4	16	8	32	16
4	6	2	6	4	6	8	12
6	2	4	2	8	4	16	8
8	--	2	6	2	6	4	6
12	--	--	2	4	2	8	4
16	--	--	--	2	6	2	6
24	--	--	--	--	2	4	2

Fig. 7. (a) Example of the data repacking with a 48-bit word size. The registers R2 and R3 have four 12-bit subwords (0/1/2/3 in R2, and 4/5/6/7 in R3). After repacking, the SIMD format can be converted to 16-bit with three subwords, 8-bit with six subwords, 6-bit with eight subwords, or still 12-bit with four subwords. (b) For repacking from 12- to 16-bit, the output subwords can be fetched from different input positions. (c) Number of connections with input bits for each output bit. '–' indicates non-supported packing modes.

Therefore, guardbits do not reduce word space utilization nor increase storage requirements. The only requirement is the range of the input data for the first stage. For example, for N-bit operands, the input can be represented at a maximum with (N-1)-bit, to avoid the possible overflow after addition, and the N_{th} bit (MSB) is temporarily used as a guardbit.

Since each addition increases the bitwidth of the result by 1-bit, for multiple additions such as the multiply–accumulate (MAC) operations, the second stage (DPU) is used to repack the data to a larger bitwidth before arithmetic operations in the first stage. We detailed this procedure in Section III-C.

C. Data Pack Unit

The role of the second pipeline stage (DPU) is to bridge across different SIMD formats. To this end, a crossbar-based DPU similar to [39] is employed to repack words from register R2 or/and R3 into other supported bitwidths. The resulting resized words are then written to register R4 as the input of the first stage (AU) or back to memory.

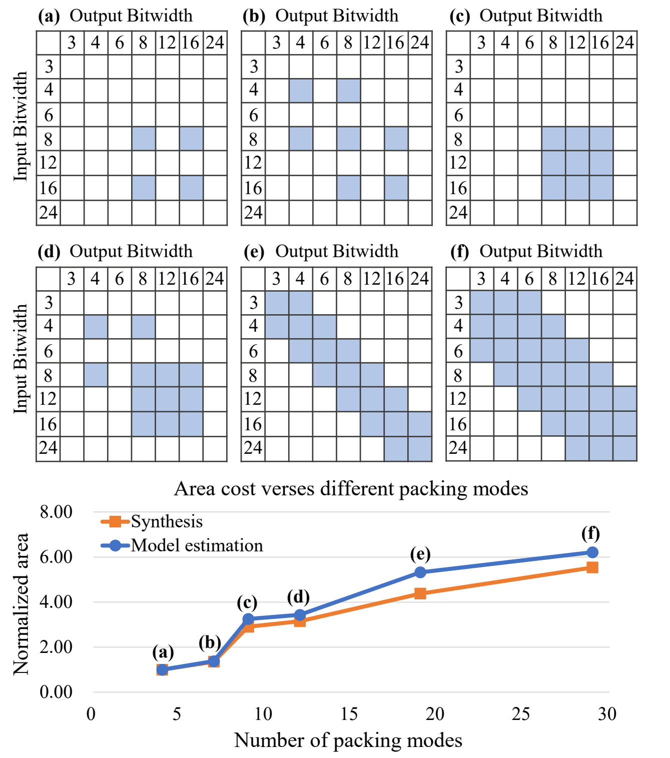


Fig. 8. Top: Different DPU configurations supporting (a) 2 SIMD modes, 4 packing modes, (b) 3 SIMD modes, 7 packing modes, (c) 3 SIMD modes, 9 packing modes, (d) 4 SIMD modes, 12 packing modes, (e) 7 SIMD modes, 19 packing modes, (f) 7 SIMD modes, 29 packing modes. Bottom: Normalized area cost of (a) ∼ (f) under logic synthesis and lightweight model evaluation.

Figure 7(a) shows an example of the repacking operation. In this example, we consider R2 and R3 as 48-bit words composed of four 12-bit subwords. Each subword can be resized to 6-, 8-, 12-, or 16-bit through the DPU, which manages increase and decrease in subword size. If the subword size increases, the sign bit of input subword data is extended, which provides a larger bitwidth to ensure that overflow does not occur in the following arithmetic operations in the first stage (AU). Conversely, if the subword size decreases, the input subword data is truncated with the required MSBs. In addition, subword sizes can stay the same when traversing the DPU, such as when R2 or/and R3 are used to store data temporarily for reducing read/write to memory.

Output subwords can be fetched from different input positions. For example, Figure 7(b) shows the possible four cases when repacking from 12- to 16-bit[1]. From a hardware perspective, the number of such cases reflects how many input bits are connected to each output bit in DPU, hence the connectivity/complexity of the data packing mode, and it is positively correlated to hardware cost.

As an illustrative example, in Figure 7(c), we consider 48-bit registers with 3-, 4-, 6-, 8-, 12-, 16-, and 24-bit SIMD modes, and show their connectivity. Some data packing modes are not supported due to the presence of only two input registers (R2 and R3). For example, converting from 12- to 3-bit would require four input registers and hence cannot be done in a single DPU traversal. The result also shows that the required connectivity for some data packing patterns is very large, such

[1]For simplicity, only the cases when starting data packing operations from R2 are presented here. The number of cases would double when symmetrical cases starting with R3 are considered.

TABLE II
A MAPPING EXAMPLE ON THE PROPOSED SOFT SIMD MICROARCHITECTURE

Cycle	Read	R1	R4	Stage 1	R2	R3	Stage 2	Write
1	B[11:0]			$\text{Res}_B[11:0] \leftarrow 0 \pm \text{B}[11:0]$				
2		B[11:0]	$\text{Res}_B[11:0]$	$\text{Res}_B[11:0] \leftarrow \text{Res}_B[11:0] >> \pm \text{B}[11:0]$				
3		B[11:0]	$\text{Res}_B[11:0]$	$\text{Res}_B[11:0] \leftarrow \text{Res}_B[11:0] >> \pm \text{B}[11:0]$				
4	B[23:12]			$\text{Res}_B[23:12] \leftarrow 0 \pm \text{B}[23:12]$	$\text{Res}_B[11:0]$			
5		B[23:12]	$\text{Res}_B[23:12$	$\text{Res}_B[23:12] \leftarrow \text{Res}_B[23:12] >> \pm \text{B}[23:12$	$\text{Res}_B[11:0]$			
6		B[23:12]	$\text{Res}_B[23:12$	$\text{Res}_B[23:12] \leftarrow \text{Res}_B[23:12] >> \pm \text{B}[23:12$	$\text{Res}_B[11:0]$		$\text{Res}_B[7:0]$	
7	$\text{Res}_A[7:0]$		$\text{Res}_B[7:0]$	$\text{Sum}[7:0] \leftarrow \text{Res}_A[7:0] + \text{Res}_B[7:0]$	$\text{Res}_B[11:0]$	$\text{Res}_B[23:12]$	$\text{Res}_B[15:8]$	Sum[7 : 0]
8	$\text{Res}_A[15:8]$		$\text{Res}_B[15:8]$	$\text{Sum}[15:8] \leftarrow \text{Res}_A[15:8] + \text{Res}_B[15:8]$		$\text{Res}_B[23:12]$	$\text{Res}_B[23:16]$	Sum[15 : 8]
9	$\text{Res}_A[23:16]$		$\text{Res}_B[23:16]$	$\text{Sum}[23:16] \leftarrow \text{Res}_A[23:16] + \text{Res}_B[23:16]$				Sum[23 : 16]

as from 3- to 8-bit or 16-bit, which means that such operations are very costly in hardware.

Hence, while the DPU could theoretically support conversion from/to any bitwidth, in practice, design complexity (hence area, energy, etc.) must be accounted for, which varies with the number and the selections of data packing modes [39] [40]. For instance, replacing a 3- to 16-bit repack with a 2-step 3- to 4-bit and 4- to 16-bit reduces design complexity. Also, since the DPU is mainly composed of multiplexers that can be reused by different data packing modes, choosing the modes with higher logic share is beneficial, such as more than 50% between 4- to 8-bit and 8- to 16-bit conversions.

To exemplify this, Figure 8(top: a$\sim$f) describes six possible choices having different flexibility and hardware requirements, ranging from supporting 4, 7, 9, 12, 19, and 29 data packing modes. Since the DPU is combinational logic and not the timing bottleneck of the datapath, we focus on the area cost in the following. Figure 8(bottom) presents the normalized area cost of (a) $\sim$ (f) based on the two following approaches:

(i) Logic synthesis under TSMC 28nm technology.

(ii) Calculate the number of required multiplexers based on the connectivity values in Figure 7(c), considering the multiplexers reuse among different data packing modes.

The result shows that supporting more SIMD and data packing modes can incur high area penalties. Nonetheless, as the number of modes increases, the area growth rate falls because of the multiplexers reuse among different modes. Also, the results obtained by logic synthesis and model estimation are very close, which indicates that we can use lightweight model estimation to evaluate the hardware resource requirement of different packing choices. In the design investigated in Section IV-B, we select the choice in Figure 8(e) because it supports all the required packing modes for MAC operations in CNNs while occupying 21.5% less area than the design in Figure 8(f).

D. Mapping Example

Table II shows a mapping example of performing multiplication, repack, and accumulation on our microarchitecture. We assume the register size is 48-bit, and the subword size can be 4-, 6-bit, etc. Operands A and B are 4-bit multiplicands with 24 subwords, while operands m1 and m2 are common multipliers. The operation is : Sum $<=$ A $\times$ m1 + B $\times$ m2. For simplicity, we only show the details of shift-add-based multiplication and repacking of Res_B[23:0] $<=$ B[23:0] $\times$ m2. The multiplication input operands and result are both 4-bit, so 12 subwords can be executed in parallel. The multiplicand (B[11:0] or B[23:12]) is fetched from memory in the start cycle of each multiplication, followed by shift-add iterations through registers R1 and R4 via the local datapath loop under control signals converted from the multiplier m2. The multiplication results (Res_B[11:0] and Res_B[23:12]) are then temporarily stored in registers R2 and R3 for repacking to 6-bit (Res_B[7:0], Res_B[15:8], and Res_B[23:16]) to prevent possible addition overflows. The same operations are performed to obtain Res_A [23:0], and the repacked outputs (Res_A[7:0], Res_A[15:8], and Res_A[23:16]) are already stored in memory. Finally, Sum[7:0], Sum[15:8] and Sum[23:16] are obtained.

Similar to this example, through careful mapping, our pipeline microarchitecture can ensure that the first stage (AU) is fully and seamlessly utilized to provide better performance/latency, while the second stage (DPU) is only activated when repack operations are required to save energy cost.

IV. EXPERIMENTAL SETUP

This section details the settings for our experiments. Section IV-A first introduces the selected CNN benchmarks, their quantization schemes, and the mapping strategy onto the proposed Guardbit-based Soft SIMD microarchitecture. Then Section IV-B summarizes the assigned parameters for the proposed microarchitecture. Section IV-C lists the two baselines, and Section IV-D specifies the synthesis configuration for Soft SIMD implementations. Finally, Section IV-E presents the power consumption model for performance evaluation.

A. CNN Benchmarks Selection

1) CNNs Quantization: To effectively leverage the computing capabilities of the proposed microarchitecture illustrated in Section III, we adopt a hardware-aware heterogeneous method to optimize the CNN benchmarks based on per-layer quantization, similar to the methodology in [41]. This strategy aims to reduce the number of shift-add operations as much as possible (hence the bitwidth of activations and weights), resulting in speed-ups and energy cost reductions, while keeping accuracy drops below a user-defined constraint.

An overview of the adopted optimization approach is depicted in Figure 9. Its input is a trained CNN model uniformly quantized to 16-bit activations and 8-bit weights [9] [10] [15]. Then, the first step of the optimization loop measures the potential reduction of the shift-add operation numbers when reducing the quantization level of either activations or weights in each layer. Following a greedy approach, the optimization action leading to the highest reduction is selected and applied to the model. After reducing the bitwidth of the target operands (i.e., either activations or weights) in the target layer, 20 retraining epochs are run. Finally, the accuracy of the obtained model is measured. If it satisfies the user-defined accuracy threshold, the optimization loop continues. Otherwise, the applied optimization is discarded and will not be considered in future iterations. This process ends when no further optimization actions can be performed. Note that, while this process

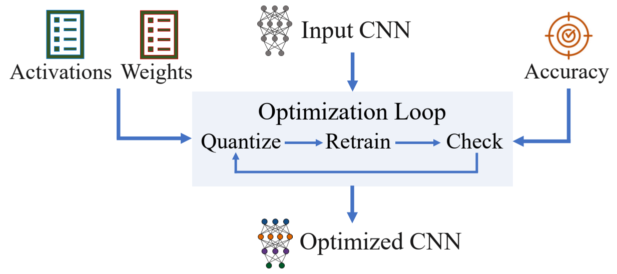
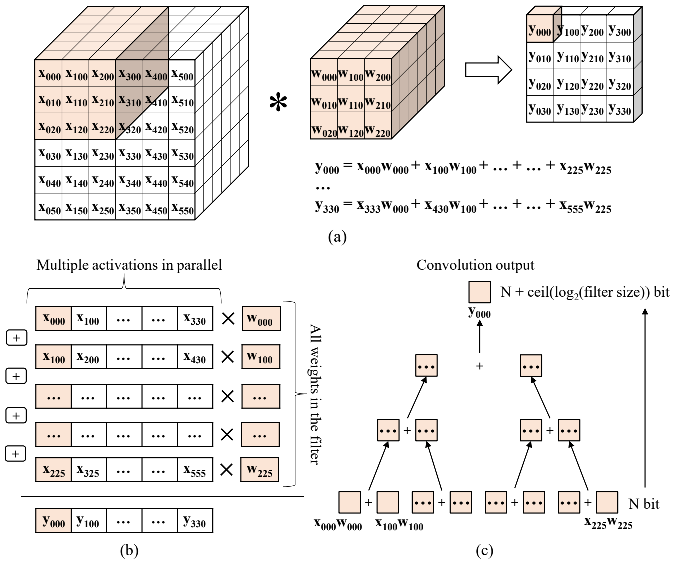


Fig. 9. The method to quantize CNN models heterogeneously. The optimization loop adjusts the bitwidth of activations and weights to maximize shift-add cycle-reduction while abiding by bitwidth and accuracy constraints.



$y_{000} = x_{000}w_{000} + x_{100}w_{100} + \dots + \dots + x_{225}w_{225}$
$y_{330} = x_{333}w_{000} + x_{430}w_{100} + \dots + \dots + x_{555}w_{225}$
(a)
Multiple activations in parallel
All weights in the filter
(b)
Convolution output
N + ceil(log$_2$(filter size)) bit
N bit
(c)

Fig. 10. (a) In the example, the input size is 6×6×6, the filter size is 3×3×6, and the output size is 4×4×1. (b) The word in each row contains multiple activations (multiplicands) from different convolutions and shares the same weight (multiplier) from the filter. The multiplication results are then accumulated in the column to get the convolution results. The activation bitwidth determines the maximum degree of parallelism. (c) All the multiplication results from the same column are accumulated with repacking to larger bitwidths constantly to avoid overflows. The number of repacking times is logarithmically related to the filter size.

induces a training time overhead, it only has to be performed once (offline) for a given network, training set, and accuracy threshold. Moreover, its complexity only scales linearly with the number of network layers, allowing it to converge in a reasonable time even for models having tens of layers.

2) CNNs Mapping: Figure 10 gives a mapping example of the basic convolution operation. As shown in Figure 10(a), the filter size is 3×3×6, and the input and output sizes are 6×6×6 and 4×4×1, respectively. In this setting, each convolution operation requires 54 multiplications and 53 additions. In Figure 10(b), each row represents a word containing serval subwords (multiplicands) and the corresponding weight from the filter (multiplier), while each column describes the accumulation of multiplication results in each convolution. To perform multiplications, multiple activations (up to the maximum amount of possible subwords) in different convolutions are mapped together on the same word and multiplied with the shared weight. Thus, the final output also contains multiple convolution results in a word.

As for accumulation (detailed in Figure 10(c)), operations are organized into a binary tree, in which the number of layers is logarithmically related to the size of the filter (e.g., for 53 additions, 6 layers). The first layer of the binary tree corresponds to the multiplication results of all the subwords in the same column (e.g., 54 subwords in the first column in Figure 10(b)), and the last layer is the convolution output (e.g., y_{000} in the first column in Figure 10(b)). Since each addition increases the subword size, the DPU stage is used to repack the data to a larger bitwidth to avoid possible overflows.

TABLE III
ACCURAY AND SHIFT-ADD-REDUCTION OF CNN BENCHMARKS

	UQ CNN Accuracy	HQ CNN Accuracy	HQ CNN Shift-add Reduction Percentage
LeNet5	74.35%	73.42%	70.40%
AlexNet	50.69%	49.75%	45.25%
VGG16	66.57%	66.63%	68.01%
ResNet20	62.66%	61.88%	87.88%
MobileNet	62.20%	61.30%	61.44%
ResNext	69.31%	68.51%	66.12%

Fully connected layers are mapped with the same strategy, as they can be executed via sequences of MAC operations [10].

It can also be noted that a smaller activation bitwidth means higher word-level parallelism, and a smaller weight bitwidth means fewer shift-add-based multiplication iteration cycles. The previous heterogeneous quantization methodology considers these two aspects comprehensively.

3) Target Benchmarks: We consider six popular edge CNN models with different complexity: LeNet5 [42], AlexNet [43], VGG16 [44], ResNet20 [45], MobileNet [46], and ResNext [47]. Accuracy is reported on the CIFAR-10 dataset [48] for LeNet5 and the CIFAR-100 dataset [48] for the other CNN benchmarks.

CNN benchmarks are first uniformly quantized with 16-bit activations and 8-bit weights (named as **UQ** in the following), as done in [9] [10] [15]. They are then quantized heterogeneously using the methodology presented in Figure 9 (named as **HQ** in the following). The maximum accuracy drop threshold is set as 1.0%. Here, UQ is used as the baseline CNN implementations as well as a reference to contrast with HQ. Note that HQ can only be supported by an architecture that features flexible subword sizes, as is the case in Soft SIMD.

Across experiments, multiplication operations need a maximum bitwidth of 16 bits, while the accumulation outputs require a maximum of 24 bits to ensure that overflow does not occur. Therefore, we consider the following subword bitwidth sets: 3, 4, 6, 8, 12, 16, and 24 bits. For weights, bitwidth varies from 1 to 16 bits. Both activations and weights are scaled to (-1,1) and are represented in Q1.X notation.

Figure 11 presents the HQ bitwidths of activations and weights in each layer for all CNN benchmarks. Table III lists the inference accuracy of UQ and HQ CNNs, as well as the shift-add reduction percentage, which is 66.52% on average and up to 87.88% for the ResNet20 case.

B. Guardbits-based Soft SIMD Instantiation

As a template instantiation for the proposed Guardbits-based Soft SIMD pipeline microarchitecture described in Section III, we consider: (i) the support for heterogeneous subwords width of 3, 4, 6, 8, 12, 16, and 24 bits, which can fully exploit the proposed microarchitecture, and (ii) a word size of 48 bits, which is the least common multiple of all SIMD modes to ensure full utilization of the word space.

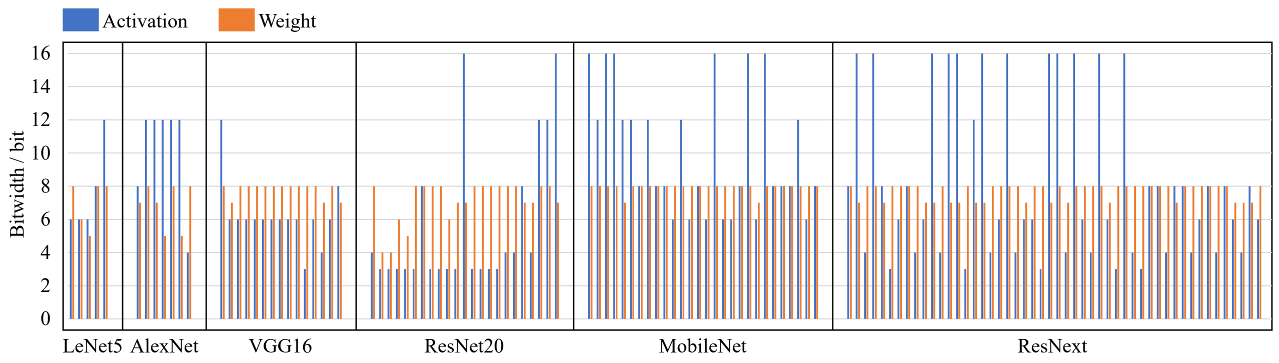


Fig. 11. The activation (blue bar) and weight (orange bar) bitwidth of each layer in corresponding heterogeneously quantized CNN benchmarks.

TABLE IV
CONFIGURATIONS OF GUARDBITS-BASED SOFT SIMD INSTANTIATION

	Configurations
SIMD Modes	[3, 4, 6, 8, 12, 16, 24] bits
Word bitwidth	48 bits
Packing modes	3 to 4, 4 to 6, 6 to 8, 8 to 12, 12 to 16, 16 to 24 bits 4 to 3, 6 to 4, 8 to 6, 12 to 8, 16 to 12, 24 to 16 bits
Shifter range	$0 \sim 3$, $0 \sim 7$

TABLE V
THE PROPOSED METHOD AND TWO BASELINES

	Guardbits-based Soft SIMD (Proposed)	Muxes-based Soft SIMD	Hard SIMD Multiplier-Adder
Word Bitwidth	48	48	48
SIMD Modes	3,4,6,8,12,16,24	3,4,6,8,12,16,24	8,16,24

It should be noted that only the second stage (DPU) of the pipeline is impacted by the choice of supported subwords, while the first stage (AU) is only influenced by the word size. In this work, available data packing modes reflect the ones illustrated in Figure 8(e): 3- to 4-bit, 4- to 3-bit, 4- to 6-bit, 6- to 4-bit, 6- to 8-bit, 8- to 6-bit, 8- to 12-bit, 12- to 8-bit, 12- to 16-bit, 16- to 12-bit, 16- to 24-bit, 24- to 16-bit, and other seven modes that keep the original bitwidth.

Finally, we consider two implementations for the logarithmic shifter in the first pipeline stage (AU), having two or three layers. In the first case, up to 3-bit right-shifts can be performed in a single operation, while up to 7-bit right-shifts in the second. The two cases are named **shift3** and **shift7**, respectively, in the following. As most operand bitwidths are below 16-bit, shifting more than 7-bit is rarely required, so we do not explore even more complex shifters.

All design parameters are summarized in Table IV. Such choices abide by the characteristics of the target scenarios (CNN benchmarks), but are not an intrinsic limitation of our hardware/software methodology or microarchitectural template.

C. Baselines

In order to evaluate the performance of the proposed Guardbits-based Soft SIMD microarchitecture, we propose two baselines in Table V : (i) Muxes-Based Soft SIMD microarchitecture, to compare the performance of different Soft SIMD implementation methods, (ii) Hard SIMD Multiplier-Adder, to compare the performance of the microarchitecture based on Soft SIMD versus Hard SIMD.

The Muxes-Based Soft SIMD baseline uses multiplexers to separate subwords instead of guardbits. It has the same features as the proposed Guardbits-based Soft-SIMD microarchitecture in Section III, but employs a different mechanism for implementing subword separation. The hardware implementation of Muxes-Based Soft SIMD baseline is similar to [9] [10] [11] [12], which performs shift-add-based arithmetic operations under 8- or 16-bit subword size for quantized neural network acceleration. Such an approach is here extended to the SIMD modes in Table V for greater flexibility, and CSD coding is adopted to speed multiplication.

On the other hand, the Hard SIMD Multiplier-Adder baseline is a typical microarchitecture that can perform parallel multiplication and addition in one cycle under limited SIMD modes. Here, we select 8-, 16-, and 24-bit SIMD modes since more modes bring a large area and energy increase. Indeed, even with this reduced set of supported bitwidths, this implementation requires vastly more area and energy with respect to Soft SIMD alternatives (see Section V).

D. Technology and Synthesis Configuration

The hardware implementations of the proposed microarchitecture and two baselines use a 28nm technology node and regular voltage standard cell library from TSMC, with a maximum operating frequency of 1.0GHz.

One particularity for the Guardbits-based and Muxes-based Soft SIMD microarchitecture synthesis is that, by default, the worst critical path length is dictated by the adder in the AU, which starts from bit 0 to bit 47 in this work. Nonetheless, no Soft SIMD configuration can traverse all word bits in practice, as we constrain the maximum operand width to 24-bit in this work. Such mismatch can cause an overly constrained optimization during synthesis, which impacts area and energy. To avoid it, we enforce specific timing constraints in the synthesis tool, ensuring that the worst critical path length of the adder corresponds to the supported SIMD modes (in this work, 16- and 24-bit, as the corresponding subwords delimitations are overlapping). Again, this does not restrict our methodology, as such an approach can be applied to any Soft SIMD microarchitecture synthesis.

E. Power Consumption Model

To evaluate the energy performance of the three implementations in Section IV-C, we develop a power consumption model containing the energy consumed by basic operations. For both Guardbits-based and Muxes-based Soft SIMD microarchitecture, we characterize the energy consumption of shift-add operations under different bitwidths and that of data packing operations. For the Hard SIMD Multiplier-Adder,

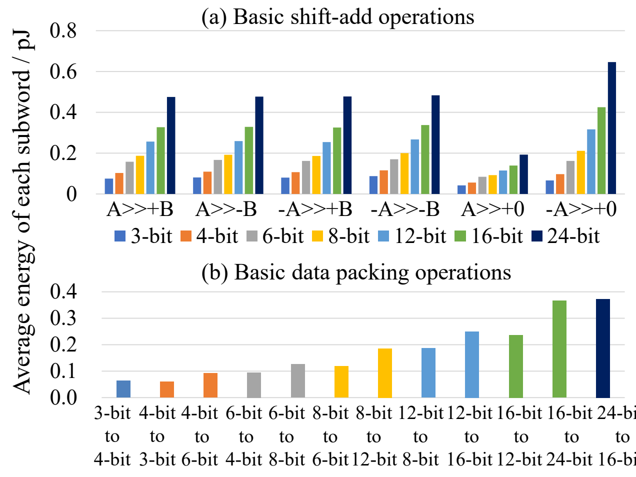


Fig. 12. The average energy cost of basic operations per subword at 1.0GHz for Guardbits-based Soft SIMD microarchitecture. (a) Shift-add operations, (b) Data packing operations.

similarly, we extract the energy cost of additions and multiplications for each supported bitwidth. Such energy extraction is based on the post-synthesis simulation at the register transfer level (RTL).

Figure 12 shows the energy consumption of all basic operations for the proposed Guardbits-based Soft SIMD microarchitecture at 1.0GHz. The decrease in energy consumption per subword operation is significant when the operands have a smaller bitwidth, such as 3-, 4-, 6-bit, since it results in higher data-level parallelism. The energy cost of data packing operations, which exhibits the same trend as shift-add operations, is shown in Figure 12(b).

We use this energy characterization to evaluate the energy cost of executing quantized CNN benchmarks. To this end, in the case of Soft SIMD architectures (both Guardbits-based and Muxes-based), we first map the CNN benchmarks to the microarchitecture as a sequence of shift-add and data packing operations (as explained in Section IV-A). Then, we multiply the number of basic operations by the corresponding energy cost to determine the overall energy consumption. A similar approach is followed for the Hard SIMD case, but considering multiplications and additions as basic operations.

V. Results

Section V-A first presents the area performance comparison. Then, Section V-B contrast the multiplication speed and energy performance. Finally, Section V-C analyzes the performance of running CNN benchmarks. In the following, we compare the performance of each 48-bit datapath for the three implementations in Table V. We consider the same operating conditions (e.g., voltage, frequency) and workload (CNN benchmarks) for all of them and explore trade-offs between area, latency, and energy consumption.

A. Area Evaluation

Figure 13 compares the area of the proposed Guardbits-based Soft SIMD microarchitecture and of two baselines when synthesized at 200MHz and 1.0GHz, respectively.

First, the comparison shows that the area increase brought by rising the maximum shift bits from 3 to 7 is marginal (while

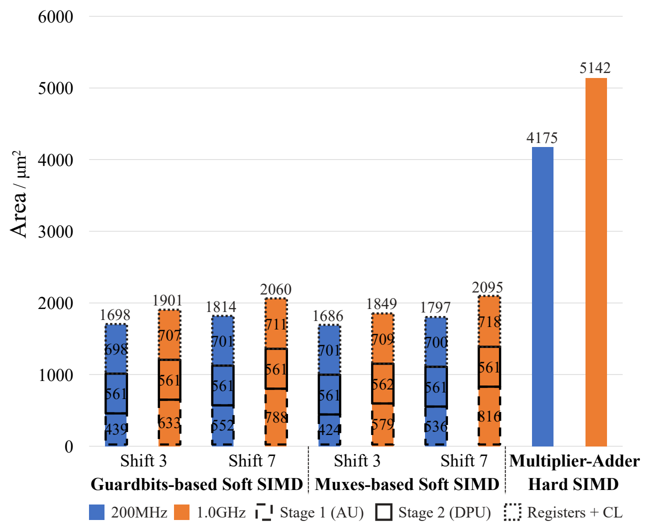


Fig. 13. When operating at 200MHz and 1.0GHz, synthesis area of (i) Guardbits-based Soft SIMD microarchitecture ('shift3' and 'shift7'), (ii) Muxes-based Soft SIMD microarchitecture ('shift3' and 'shift7'), and (iii) Hard SIMD Multiplier-Adder. The area is composed of three parts: Arithemetic Unit (the first stage, AU), the Data Pack Unit (the second stage, DPU), and the Registers and other Combinational Logic (R+CL).

the added flexibility does induce speed-ups when performing multiplications, as discussed in Section V-B). Besides, the area of the second stage (DPU) remains almost constant while that of the first stage (AU), which is the critical path, increases significantly with tighter timing constraints. The area of the registers and other combinational logic (R+CL) also fluctuates slightly.

Then, as expected, since Guardbits-based and Muxes-based Soft SIMD implementations do not require a complex combinatorial multiplier, they take up considerably less area (for 'shift7' at 1.0GHz, 59.9% and 59.3%, respectively) with respect to the Hard SIMD Multiplier-Adder, even if the latter supports fewer SIMD modes (8-, 16-, and 24-bit).

Finally, the Guardbits-based Soft SIMD microarchitecture scales better (i.e., has a smaller area increase) for tighter timing constraints than the Muxes-based Soft SIMD alternative, as it has a shorter critical path. For example, the area cost of Muxes-based Soft SIMD microarchitecture is always smaller than that of Guardbits-based Soft SIMD microarchitecture for 'shift3' configurations. However, its area cost becomes the larger one for 'shift7' cases at 1.0GHz. Also, for a constant frequency constraint, when increasing the shift from 3 to 7, the Guardbits-based Soft SIMD microarchitecture area grows by 6.8% and 8.4% at 200MHz and 1.0GHz, respectively, while for Muxes-based Soft SIMD microarchitecture 6.6% and 13.3%. These again prove a better resilience to tighter timing constraints for the Guardbits-based Soft SIMD microarchitecture.

B. Speed and Energy Evaluation of Iterative Multiplication

1) Soft SIMD Multiplication Cycle Count: The shifter range ('shift3' or 'shift7') affects not only the area cost but also the iterative amount cycles to complete a shift-add-based multiplication with CSD coding on both Guardbits-based and Muxes-based Soft SIMD microarchitecture. The average cycles (for 'shift3', 'shift7', and a theoretical infinitely-wide shifter with CSD coding, and the work in [9]–[12] without CSD coding) to finish a multiplication are presented in Figure 14.

Thanks to the CSD coding, the trendline slope of our design drops by 18.0% (‘shift3’) and 35.4% (‘shift7’) with respect to that of [9] [10], and by 59.0% (‘shift3’) and 67.7% (‘shift7’) with respect to that of [11] [12]. For the ‘shift7’ case, its trendline slope is almost the same as that of the infinitely-wide shifter, and the average cycle count difference between the two is less than 0.30 cycles. In contrast, for the ‘shift3’ case, its trendline slope increases by 27.3% compared with that of the theoretical scenario, resulting in less multiplication speed, especially for the larger-bitwidth multiplier. For example, the ‘shift3’ case requires an average of 0.38 more cycle count for the 8-bit multiplier and 1.09 more for the 16-bit multiplier than the ‘shift7’ case.

2) Soft SIMD Multiplication Energy: The impact of the shifter range on multiplication energy consumption is further contrasted on both Guardbits-based and Muxes-based Soft SIMD microarchitecture in Figure 15. Therein, we select 8-bit multiplicand as an example, and describe the relationship between the average multiplication energy cost per subword and the bitwidth of the multiplier for four Soft SIMD configurations (Guardbits-based or Muxes-based, ‘shift3’ or ‘shift7’). The energy cost of the Hard SIMD Multiplier is omitted as it is much larger (from 3-bit to 16-bit, ×4.8∼×2.0 times) than Soft SIMD alternatives.

First, the results indicate that Guardbits-based and Muxes-based Soft SIMD microarchitectures adopting ‘shift7’ is always better than those with ‘shift3’ when the multiplier bitwidth is larger than 6-bit. For multipliers bitwidth lower than 6-bit, the four Soft SIMD configurations have a minimal difference. Since Guardbits-based and Muxes-based Soft SIMD microarchitectures in ‘shift7’ configuration have advantages on both speed and energy consumption with only a few area increases, they are chosen for further explorations in the following.

Moreover, Guardbits-based Soft SIMD microarchitecture always consumes less energy than Muxes-based Soft SIMD microarchitecture, and the energy advantage becomes more significant as the multiplier bitwidth increases. The reason is that the critical path growth of Guardbits-based Soft SIMD microarchitecture is much lower than that of Muxes-based Soft SIMD microarchitecture. Thus, it has more prominent advantages for larger bitwidths, tighter timing constraints, and higher operating frequencies.

In addition, Guardbits-based Soft SIMD microarchitecture with ‘shift7’ has the minimum energy consumption when the multiplier bitwidth is larger than 6-bit. Also, it has the smallest trendline slope, again proving its best energy efficiency.

3) Soft and Hard SIMD Multiplication Energy Comparison: In Figure 16, 8-bit multiplicand by 8-bit multiplier is selected as the example, and the average energy consumption per subword of Guardbits-based Soft SIMD microarchitecture, Muxes-based Soft SIMD microarchitecture, and Hard SIMD Multiplier-Adder, are compared, considering target clock frequencies from 200MHz to 1.0GHz.

As expected, Guardbits-based and Muxes-based Soft SIMD implementations consume at least 41.0% and 37.5% less energy than Hard SIMD Multiplier using complex combinational logic. Moreover, the energy consumption of Hard SIMD Multiplier grows 6.6 and 6.1 times faster with frequency than that of Guardbits-based and Muxes-based Soft SIMD microarchitecture. Also, consistent with the previous analy-

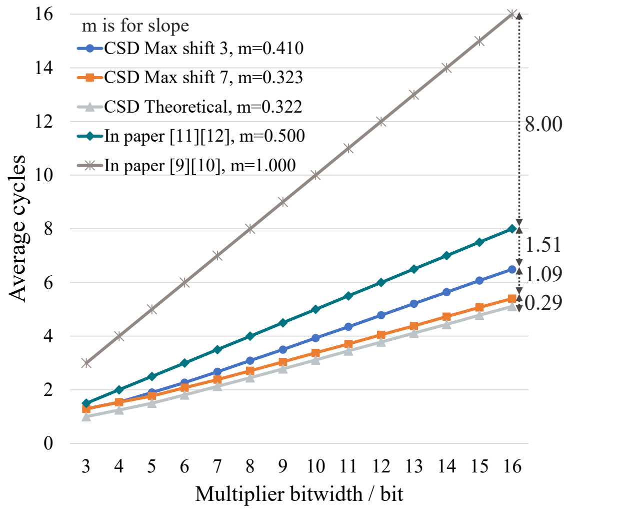


Fig. 14. Under different shifter ranges, the average cycles to finish a shift-add-based multiplication on the Guardbits-based or Muxes-based Soft SIMD microarchitecture verses the multiplier bitwidth (in CSD coding). The lower trendline slope means fewer cycles for multiplication.

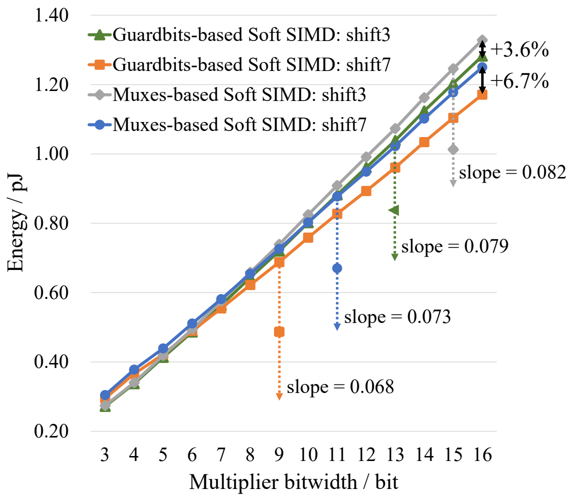


Fig. 15. Average multiplication energy cost per subword at 1.0GHz. The multiplicand is 8-bit, and the multiplier varies from 3- to 16-bit. The energy cost of the Hard SIMD Multiplier is not plotted as it is far greater than these.

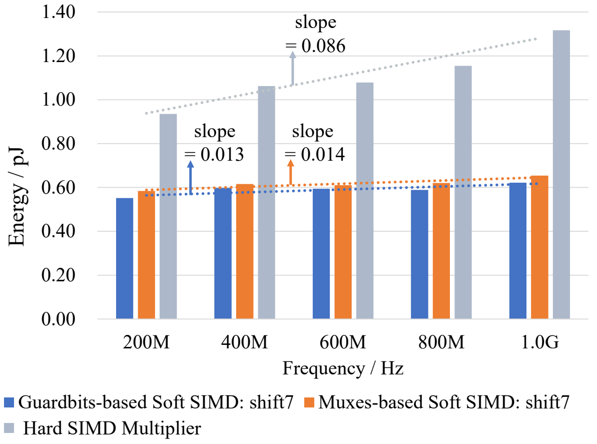


Fig. 16. 8bit * 8bit multiplication energy consumption per subword at different frequencies for proposed Guardbits-based Soft SIMD microarchitecture and two baselines.

sis, the energy consumption of Guardbits-based Soft SIMD microarchitecture is always lower than that of Muxes-based Soft SIMD microarchitecture for all frequencies.

C. *CNN Benchmarks Energy Cost Evaluation*

The energy consumption and execution time of the selected CNN benchmarks (LeNet5, AlexNet, VGG16, ResNet20, MobileNet, and ResNext) have been evaluated using the CNNs mapping method detailed in Section IV-A and the power consumption model illustrated in Section IV-E. We consider both uniformly quantized (UQ) and heterogeneously quantized (HQ) CNN benchmarks.

Table VI and VII present the normalized energy consumption and execution time to run per inference of different CNN benchmarks on the proposed Guardbits-based Soft SIMD microarchitecture and two baselines (Muxes-based Soft SIMD microarchitecture and Hard SIMD Multiplier-Adder) when operating at 1.0GHz.

Guardbits-based Soft SIMD microarchitecture consumes 9.9% (UQ) / 9.2% (HQ) less energy than Muxes-based Soft SIMD microarchitecture on average, while having the same execution time. Therefore, we only compare the energy and execution time performance of CNN benchmarks running on Guardbits-based Soft SIMD microarchitecture and Hard SIMD Multiplier-Adder in the following.

First, as Figure 17(a) presents, Guardbits-based Soft SIMD microarchitecture can significantly reduce energy consumption compared to Hard SIMD Multiplier-Adder. Moreover, due to using shift-add iterations to replace one-cycle multiplier, Guardbits-based Soft SIMD microarchitecture has an execution time increase for most CNN benchmarks, as shown in Figure 17(b). Specifically, for UQ CNN benchmarks, our proposed microarchitecture has an average energy reduction of 38.4% and execution time increase of 76.1% compared to Hard SIMD Multiplier-Adder. Since all UQ CNN benchmarks have 16-bit activations and 8-bit weights, their energy and execution time reduction has a slight difference of less than 1.0% across benchmarks. Conversely, for HQ CNN benchmarks, our proposed microarchitecture has an average energy reduction of 50.1% (ranging from 41.2% to 66.2%) and execution time increase of 31.5% (ranging from -14.5% to 58.7%). Their performance difference is due to the different quantization schemes adopted in each CNN benchmark (hence different activation and weight bitwidths, see Figure 11). Besides, from the perspective of TOPS/W, Guardbits-based Soft SIMD microarchitecture has an average increase of 63.2% (ranging from 60.0% to 68.8%) for UQ CNN benchmarks and 107.9% for HQ CNN benchmarks (ranging from 69.5% to 194.1%) with respect to Hard SIMD Multiplier-Adder.

Moreover, Guardbits-based Soft SIMD microarchitecture performs better with heterogeneously than uniformly quantized CNN benchmarks because they can fully use small bitwidth activations (i.e., higher data-level parallelism) and weights (i.e., fewer iterative cycles for multiplication). For instance, the average execution time overhead drops from 76.1% (UQ) to 31.5% (HQ), while the average energy reduction rises from 38.4% (UQ) to 50.1% (HQ). Especially for ResNet20 (HQ), Guardbits-based Soft SIMD saves 66.2% energy cost and reduces 14.5% the execution time because its activations and weights in many layers can be quantized to very small bitwidths (see Figure 11).

In addition, Figure 17(c) presents the energy-delay (execution time) product reduction. UQ CNN benchmarks exhibit an increase in this metric (ranging from 7.9% to 8.7%, and by 8.4% on average). However, by co-optimizing from the hardware and software aspects, HQ alternatives can reduce the energy-delay product (ranging from 6.5% and up to 71.1% for different quantization schemes, and by 32.3% on average).

TABLE VI
NORMALIZED ENERGY COST PER CNN INFERENCE AT 1.0GHZ

	Guardbit-based Soft SIMD		**Muxes-based Soft SIMD**		**Hard SIMD**	
	UQ	HQ	UQ	HQ	UQ	HQ
LeNet5	1.38	1.00	1.52	1.10	2.24	1.70
AlexNet	1.53	1.00	1.68	1.10	2.48	2.22
VGG16	2.14	1.00	2.35	1.09	3.48	1.84
ResNet20	3.73	1.00	4.10	1.07	6.06	2.96
MobileNet	1.40	1.00	1.54	1.10	2.27	1.72
ResNext	2.18	1.00	2.39	1.09	3.53	2.04

TABLE VII
NORMALIZED EXECUTION-TIME PER CNN INFERENCE AT 1.0GHZ

	Guardbit-based Soft SIMD		**Muxes-based Soft SIMD**		**Hard SIMD**	
	UQ	HQ	UQ	HQ	UQ	HQ
LeNet5	1.45	1.00	1.45	1.00	0.82	0.63
AlexNet	1.64	1.00	1.64	1.00	0.93	0.81
VGG16	2.45	1.00	2.45	1.00	1.39	0.73
ResNet20	4.28	1.00	4.28	1.00	2.44	1.17
MobileNet	1.46	1.00	1.46	1.00	0.83	0.64
ResNext	2.37	1.00	2.37	1.00	1.35	0.78

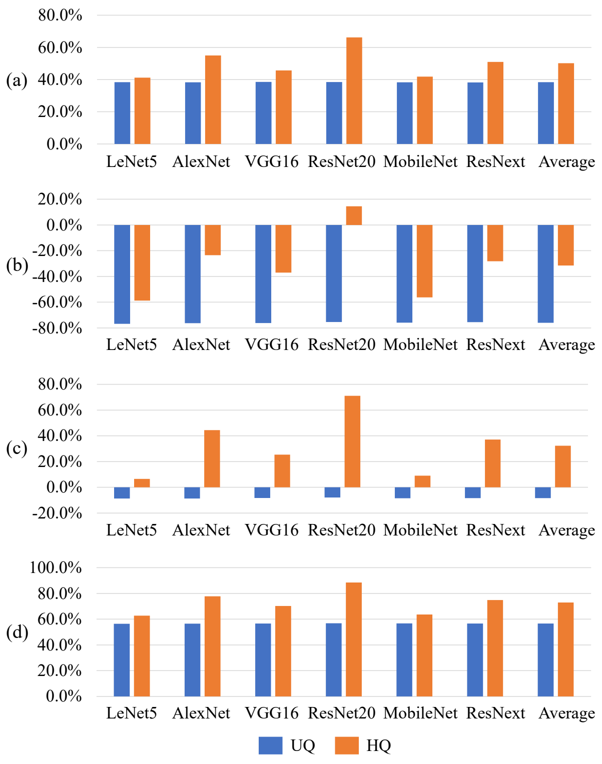


Fig. 17. When running per inference of different CNN benchmarks at 1.0GHz, the normalized (a) energy consumption reduction, (b) execution time (delay) reduction, (c) energy-delay product reduction, (d) area-energy-delay product reduction, by Guardbits-based Soft SIMD microarchitecture relative to Hard SIMD Multiplier-Adder. The reduction is defined as $(Cost_{Hard\ SIMD} - Cost_{Soft\ SIMD})/Cost_{Hard\ SIMD}$.

In summary, we compare all metrics employed (area, energy consumption, and execution time) by the energy-delay-area product in Figure 17(d). These results show that our proposed Soft SIMD microarchitecture significantly improves concerning a Hard SIMD alternative for both UQ and HQ CNN benchmarks. Specifically, for UQ CNN benchmarks, the energy-delay-area product decreases 56.6% on average, with less than 0.4% difference between benchmarks. For HQ CNN benchmarks, the energy-delay-area product declines from 62.7% to 88.5%, with an average of 72.9%.

VI. CONCLUSION

Typical Hardware SIMD resources can only support a small set of subword sizes, limiting their performance on fully quantized and highly data-level parallel algorithms, especially for small-bitwidth arithmetic operations. Filling this gap, our work has introduced a novel pipeline microarchitecture for arithmetic computing based on the Software-defined SIMD paradigm, which can flexibly support arbitrary SIMD modes at run-time through control instructions. This microarchitecture efficiently executes shift/add operations and shift-add-based multiplication/accumulation operations in highly parallel using guardbits. CSD coding is used to feature the skipping of trailing '0' digits in multiplier operands, providing multiplication operations with further energy gains and savings in execution time. A Data Pack Unit is introduced to bridge across different SIMD formats and seamlessly support a wide range of quantization levels. We choose heterogeneously quantized CNNs from the ML domain as the benchmark and map it onto this microarchitecture for performance evaluation. Experiments showcase that our Soft SIMD microarchitecture significantly outperforms Hard SIMD alternative when executing CNNs inferences, highlighting energy gains of 50.1% on average while requiring 59.9% less area.

VII. ACKNOWLEDGMENTS

This research was partially supported by EC H2020 FVLL-MONTI 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.

REFERENCES

[1] D. Kusswurm, "Advanced vector extensions (avx)," in *Modern X86 Assembly Language Programming*. Springer, 2014, pp. 327–349.

[2] C. J. Hughes, *Single-instruction multiple-data execution*. Springer Nature, 2022.

[3] M. Hassaballah, S. Omran, and Y. B. Mahdy, "A review of simd multimedia extensions and their usage in scientific and engineering applications," *The Computer Journal*, vol. 51, no. 6, pp. 630–649, 2008.

[4] S.-J. Lee, S.-S. Park, and K.-S. Chung, "Efficient simd implementation for accelerating convolutional neural network," in *Proceedings of the 4th International Conference on Communication and Information Processing*, 2018, pp. 174–179.

[5] S. Basalama, A. Panahi, A.-T. Ishimwe, and D. Andrews, "Spar-2: A simd processor array for machine learning in iot devices," in *2020 3rd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2020, pp. 141–147.

[6] Arm, "Neon™ version: 1.0 programmer's guide," 2013.

[7] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[8] W. Li, H. Hacid, E. Almazrouei, and M. Debbah, "A review and a taxonomy of edge machine learning: Requirements, paradigms, and techniques," *arXiv preprint arXiv:2302.08571*, 2023.

[9] F. Ponzina, M. Rios, G. Ansaloni, A. Levisse, and D. Atienza, "A flexible in-memory computing architecture for heterogeneously quantized cnns," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 164–169.

[10] M. Rios, F. Ponzina, A. Levisse, G. Ansaloni, and D. Atienza, "Bit-line computing for cnn accelerators co-design in edge ai inference," *IEEE Transactions on Emerging Topics in Computing*, 2023.

[11] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[12] L.-C. Hsu, C.-T. Chiu, K.-T. Lin, H.-H. Chou, and Y.-Y. Pu, "Essa: An energy-aware bit-serial streaming deep convolutional neural network accelerator," *Journal of Systems Architecture*, vol. 111, p. 101831, 2020.

[13] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on xilinx devices," *White Paper*, 2016.

[14] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, "Designing fast convolutional engines for deep learning applications," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 753–756.

[15] M. v. Baalen, A. Kuzmin, S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga *et al.*, "Fp8 versus int8 for efficient deep learning inference," *arXiv preprint arXiv:2303.17951*, 2023.

[16] R. Goyal, J. Vanschoren, V. Van Acht, and S. Nijssen, "Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms," *arXiv preprint arXiv:2102.02147*, 2021.

[17] S. Balakrishnan and S. K. Nandy, "Arbitrary precision arithmetic-simd style," in *Proceedings Eleventh International Conference on VLSI Design*. IEEE, 1998, pp. 128–132.

[18] S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr, "Softsimd-exploiting subword parallelism using source code transformations," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[19] F. Catthoor, P. Raghavan, A. Lambrechts, M. Jayapala, A. Kritikakou, and J. Absar, *Ultra-low energy domain-specific instruction-set processors*. Springer Science & Business Media, 2010.

[20] G. Psychou, R. Fasthuber, F. Catthoor, J. Hulzink, and J. Huisken, "Sub-word handling in data-parallel mapping," in *ARCS 2012*. IEEE, 2012, pp. 1–7.

[21] R. Fasthuber, F. Catthoor, P. Raghavan, and F. Naessens, "Energy-efficient communication processors," *Springer, New York, NY*, vol. 10, pp. 978–1, 2013.

[22] A. Avizienis, "Signed-digit numbe representations for fast parallel arithmetic," *IRE Transactions on electronic computers*, no. 3, pp. 389–400, 1961.

[23] A. K. Oudjida, "Binary arithmetic for finite-word-length linear controllers: Mems applications," Ph.D. dissertation, Besançon, 2014.

[24] A. Peled, "On the hardware implementation of digital signal processors," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 1, pp. 76–86, 1976.

[25] A. Herrfeld and S. Hentschke, "Look-ahead circuit for csd-code carry determination," *Electronics Letters*, vol. 31, no. 6, pp. 434–435, 1995.

[26] C. Koc, "Parallel canonical recoding," *Electronics Letters*, vol. 32, no. 22, pp. 2063–2065, 1996.

[27] Y. Tanaka, "Efficient signed-digit-to-canonical-signed-digit recoding circuits," *Microelectronics Journal*, vol. 57, pp. 21–25, 2016.

[28] M. Faust, O. Gustafsson, and C.-H. Chang, "Fast and vlsi efficient binary-to-csd encoder using bypass signal," *Electronics letters*, vol. 47, no. 1, pp. 18–20, 2011.

[29] G. A. Ruiz and M. Granda, "Efficient canonic signed digit recoding," *Microelectronics journal*, vol. 42, no. 9, pp. 1090–1097, 2011.

[30] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.

[31] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

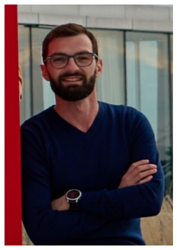
[32] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[33] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[34] F. Ponzina, “Hardware-software co-design methodologies for edge ai optimization,” EPFL, Tech. Rep., 2023.
[35] A. M. Abdelmoniem and M. Canini, “Towards mitigating device heterogeneity in federated learning via adaptive model quantization,” in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 96–103.
[36] C. N. Coelho Jr, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, “Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors,” *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.
[37] P. Raghavan, A. Lambrechts, M. Jayapala, F. Catthoor, D. Verkest, and H. Corporaal, “Very wide register: An asymmetric register file organization for low power embedded processors,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.
[38] B. W. Denkinger, M. Peón-Quirós, M. Konijnenburg, D. Atienza, and F. Catthoor, “Vwr2a: a very-wide-register reconfigurable-array architecture for low-power embedded devices,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 895–900.
[39] P. Raghavan, S. Munaga, E. R. Ramos, A. Lambrechts, M. Jayapala, F. Catthoor, and D. Verkest, “A customized cross-bar for data-shuffling in domain-specific simd processors,” in *International Conference on Architecture of Computing Systems*. Springer, 2007, pp. 57–68.
[40] F. Catthoor and D. Verkest, “Semi custom design: A case study on simd shufflers,” in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation: 17th International Workshop, PATMOS 2007, Gothenburg, Sweden, September 3-5, 2007, Proceedings*, vol. 4644. Springer Science & Business Media, 2007, p. 433.
[41] F. Ponzina, M. Rios, A. Levisse, G. Ansaloni, and D. Atienza, “Overflow-free compute memories for edge ai acceleration,” *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–23, 2023.
[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
[44] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
[45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[46] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
[47] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.
[48] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.

Pengbo Yu received the M.Sc. degree in Integrated Circuit Engineering from Tsinghua University, Beijing, China, in 2020. He is currently a Ph.D. student at the Embedded Systems Laboratory of EPFL, Switzerland. His research interests include the design of circuits and systems and energy-efficient computing.

Flavio Ponzina received the M.Sc. degree in Computer Engineering from Politecnico di Torino, Italy, in 2018, and the Ph.D degree in Electronic Engineering from EPFL, Switzerland, in 2023. He is currently a postdoctoral researcher at University of California, San Diego, United States. His main research interests include low power architectures and AI-based systems optimization.



Alexandre Levisse received his Ph.D. degree in Electrical Engineering from CEA-LETI and Aix-Marseille University, France, in 2017. From 2018 to 2021, he was a postdoctoral researcher at the Embedded Systems Laboratory of EPFL. From 2021, he works as a scientist in EPFL. His research interests include CAD, design flows, circuits and architectures for emerging memory and transistor technologies as well as in-memory computing and accelerators.

Mohit Gupta received the M.Tech. degree in electronics engineering from AMU, Aligarh, India,in 2014, and the Ph.D. degree from the Katholieke Universiteit Leuven, Belgium, in 2022. Since 2020, he has been an Research and Development Researcher with IMEC, Leuven, and working on non-volatile memory design and in/near memory computing with SRAM and NVMs.

Dwaipayan Biswas received the M.Sc. degree in system on chip and the Ph.D. degree in electrical engineering from University of Southampton (UoS), Southampton, U.K., in 2011 and 2015, respectively. From 2015 to 2016, he was a Postdoctoral Research Fellow with UoS. In 2016, he joined IMEC, as a researcher on digital IC design for biomedical applications. Currently, he is leading a team of 10 researchers working on the System Technology Co-optimization (STCO) program at IMEC, Leuven, Belgium. His current research focuses on exploring the interception of imec’s advanced semiconductor technology on future compute system challenges. He has authored several peer-reviewed journals, conferences, and edited a book. His research interests include low-power VLSI design, advanced technology for system optimization.

Giovanni Ansaloni is a senior researcher at the Embedded Systems Laboratory of EPFL. He previously worked as a Post-Doc at the University of Lugano (USI, CH) between 2015 and 2020, and at EPFL between 2011 and 2015. He received a Ph.D. degree in Informatics from USI in 2011. His research efforts focus on domain-specific and ultra-low-power architectures and algorithms for edge computing systems, including hardware and software optimization techniques.

David Atienza (Fellow, IEEE) received the Ph.D degree in computer science and engineering from UCM, Spain, and IMEC, Belgium, in 2005. He is a professor of electrical and computer engineering, heads the Embedded Systems Laboratory (ESL), and is the scientific director of the EcoCloud Center for Sustainable Computing at EPFL. His research interests include system-level design methodologies for high-performance MPSoC and low-power edge AI architectures. He is a co-author of more than 400 papers in peer-reviewed international journals and conferences, one book, and 14 patents in these fields. He is an ACM Fellow.

Francky Catthoor (Fellow, IEEE) received the Ph.D degree in EE from the Katholieke Univ. Leuven, Belgium, in 1987. Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000 he is strongly involved in other activities with IMEC including co-exploration of applications, computer architecture, deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photo-voltaic modules combined with renewable energy systems, all with IMEC Leuven, Belgium. Currently, he is an IMEC senior fellow. He is also a part-time full professor with the EE department of the KU Leuven.